TensorForce Documentation

Release 0.3.3

reinforce.io

Mar 18, 2018

Contents:

TensorForce is an open source reinforcement learning library focused on providing clear APIs, readability and modularisation to deploy reinforcement learning solutions both in research and practice. TensorForce is built on top on TensorFlow.

CHAPTER 1

Quick start

For a quick start, you can run one of our example scripts using the provided configurations, e.g. to run the TRPO agent on CartPole, execute from the examples folder:

```
python examples/openai_gym.py CartPole-v0 -a examples/configs/ppo.json -n examples/
↪configs/mlp2_network.json
```

In python, it could look like this:

```
 # examples/quickstart.py

import numpy as np

from tensorforce.agents import PPOAgent
from tensorforce.execution import Runner
from tensorforce.contrib.openai_gym import OpenAIGym

# Create an OpenAIgym environment
env = OpenAIGym('CartPole-v0', visualize=True)

# Network as list of layers
network_spec = [
    dict(type='dense', size=32, activation='tanh'),
    dict(type='dense', size=32, activation='tanh')
]

agent = PPOAgent(
    states_spec=env.states,
    actions_spec=env.actions,
    network_spec=network_spec,
    batch_size=4096,
    # BatchAgent
    keep_last_timestep=True,
    # PPOAgent
    step_optimizer=dict(
        type='adam',
```

```
        learning_rate=1e-3
    ),
    optimization_steps=10,
    # Model
    scope='ppo',
    discount=0.99,
    # DistributionModel
    distributions_spec=None,
    entropy_regularization=0.01,
    # PGModel
    baseline_mode=None,
    baseline=None,
    baseline_optimizer=None,
    gae_lambda=None,
    # PGLRModel
    likelihood_ratio_clipping=0.2,
    summary_spec=None,
    distributed_spec=None
)

# Create the runner
runner = Runner(agent=agent, environment=env)


# Callback function printing episode statistics
def episode_finished(r):
    print("Finished episode {ep} after {ts} timesteps (reward: {reward})".format(ep=r.
↪episode, ts=r.episode_timestep,

↪reward=r.episode_rewards[-1]))
    return True


# Start learning
runner.run(episodes=3000, max_episode_timesteps=200, episode_finished=episode_
↪finished)
runner.close()

# Print statistics
print("Learning finished. Total episodes: {ep}. Average reward of last 100 episodes:
↪{ar}.".format(
    ep=runner.episode,
    ar=np.mean(runner.episode_rewards[-100:]))
)
```

1.1 Agent and model overview

A reinforcement learning agent provides methods to process states and return actions, to store past observations, and to load and save models. Most agents employ a `Model` which implements the algorithms to calculate the next action given the current state and to update model parameters from past experiences.

> Environment <-> Runner <-> Agent <-> Model

Parameters to the agent are passed as a `Configuration` object. The configuration is passed on to the `Model`.

1.1.1 Ready-to-use algorithms

We implemented some of the most common RL algorithms and try to keep these up-to-date. Here we provide an overview over all implemented agents and models.

Agent / General parameters

`Agent` is the base class for all reinforcement learning agents. Every agent inherits from this class.

class tensorforce.agents.**Agent**(*states*, *actions*, *batched_observe=True*, *batching_capacity=1000*)

Bases: `object`

Base class for TensorForce agents.

__init__(*states*, *actions*, *batched_observe=True*, *batching_capacity=1000*)

Initializes the agent.

Parameters **states** – States specification, with the following attributes (required):

Parameters **actions** – Actions specification, with the following attributes (required):

Parameters

- **batched_observe** (*bool*) – Specifies whether calls to model.observe() are batched, for improved performance (default: true).
- **batching_capacity** (*int*) – Batching capacity of agent and model (default: 1000).

act(*states*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)

Return action(s) for given state(s). States preprocessing and exploration are applied if configured accordingly.

Parameters

- **states** (*any*) – One state (usually a value tuple) or dict of states if multiple states are expected.
- **deterministic** (*bool*) – If true, no exploration and sampling is applied.
- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).
- **fetch_tensors** (*list*) – Optional String of named tensors to fetch

Returns Scalar value of the action or dict of multiple actions the agent wants to execute. (fetched_tensors) Optional dict() with named tensors fetched

static from_spec(*spec*, *kwargs*)

Creates an agent from a specification dict.

initialize_model()

Creates the model for the respective agent based on specifications given by user. This is a separate call after constructing the agent because the agent constructor has to perform a number of checks on the specs first, sometimes adjusting them e.g. by converting to a dict.

observe(*terminal*, *reward*)

Observe experience from the environment to learn from. Optionally pre-processes rewards Child classes should call super to get the processed reward EX: terminal, reward = super()...

Parameters

- **terminal** (*bool*) – boolean indicating if the episode terminated after the observation.
- **reward** (*float*) – scalar reward that resulted from executing the action.

reset()
Reset the agent to its initial state (e.g. on experiment start). Updates the Model's internal episode and time step counter, internal states, and resets preprocessors.

restore_model(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters
- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

save_model(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters
- **directory** (*str*) – Optional checkpoint directory.
- **append_timestep** (*bool*) – Appends the current timestep to the checkpoint file if true. If this is set to True, the load path must include the checkpoint timestep suffix. For example, if stored to models/ and set to true, the exported file will be of the form models/model.ckpt-X where X is the last timestep saved. The load path must precisely match this file name. If this option is turned off, the checkpoint will always overwrite the file specified in path and the model can always be loaded under this path.

Returns Checkpoint path were the model was saved.

Model

The `Model` class is the base class for reinforcement learning models.

class tensorforce.models.**Model**(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*, *variable_noise*, *states_preprocessing*, *actions_exploration*, *reward_preprocessing*)

Bases: `object`

Base class for all (TensorFlow-based) models.

__init__(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*, *variable_noise*, *states_preprocessing*, *actions_exploration*, *reward_preprocessing*)
Model.

Parameters
- **states** (*spec*) – The state-space description dictionary.
- **actions** (*spec*) – The action-space description dictionary.
- **scope** (*str*) – The root scope str to use for tf variable scoping.
- **device** (*str*) – The name of the device to run the graph of this model on.
- **saver** (*spec*) – Dict specifying whether and how to save the model's parameters.
- **summarizer** (*spec*) – Dict specifying which tensorboard summaries should be created and added to the graph.
- **execution** (*spec*) – Dict specifying whether and how to do distributed training on the model's graph.
- **batching_capacity** (*int*) – Batching capacity.

- **variable_noise** (*float*) – The stddev value of a Normal distribution used for adding random noise to the model's output (for each batch, noise can be toggled and - if active - will be resampled). Use None for not adding any noise.
- **states_preprocessing** (*spec / dict of specs*) – Dict specifying whether and how to preprocess state signals (e.g. normalization, greyscale, etc..).
- **actions_exploration** (*spec / dict of specs*) – Dict specifying whether and how to add exploration to the model's "action outputs" (e.g. epsilon-greedy).
- **reward_preprocessing** (*spec*) – Dict specifying whether and how to preprocess rewards coming from the Environment (e.g. reward normalization).

act(*states*, *internals*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)

Does a forward pass through the model to retrieve action (outputs) given inputs for state (and internal state, if applicable (e.g. RNNs))

Parameters

- **states** (*dict*) – Dict of state values (each key represents one state space component).
- **internals** (*dict*) – Dict of internal state values (each key represents one internal state component).
- **deterministic** (*bool*) – If True, will not apply exploration after actions are calculated.
- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).

Returns

- Actual action-outputs (batched if state input is a batch).

Return type tuple

create_operations(*states*, *internals*, *actions*, *terminal*, *reward*, *deterministic*, *independent*)

Creates output operations for acting, observing and interacting with the memory.

get_component(*component_name*)

Looks up a component by its name.

Parameters **component_name** – The name of the component to look up.

Returns The component for the provided name or None if there is no such component.

get_components()

Returns a dictionary of component name to component of all the components within this model.

Returns (dict) The mapping of name to component.

get_savable_components()

Returns the list of all of the components this model consists of that can be individually saved and restored. For instance the network or distribution.

Returns List of util.SavableComponent

get_summaries()

Returns the TensorFlow summaries reported by the model

Returns List of summaries

get_variables(*include_submodules=False*, *include_nontrainable=False*)

Returns the TensorFlow variables used by the model.

Parameters

- **include_submodules** – Includes variables of submodules (e.g. baseline, target network) if true.
- **include_nontrainable** – Includes non-trainable variables if true.

Returns List of variables.

initialize(*custom_getter*)
Creates the TensorFlow placeholders and functions for this model. Moreover adds the internal state placeholders and initialization values to the model.
Parameters **custom_getter** – The custom_getter_ object to use for tf.make_template when creating TensorFlow functions.

observe(*terminal*, *reward*)
Adds an observation (reward and is-terminal) to the model without updating its trainable variables.
Parameters
- **terminal** (*bool*) – Whether the episode has terminated.
- **reward** (*float*) – The observed reward value.

Returns The value of the model-internal episode counter.

reset()
Resets the model to its initial state on episode start. This should also reset all preprocessor(s).
Returns Current episode, timestep counter and the shallow-copied list of internal state initialization Tensors.
Return type tuple

restore(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).
Parameters
- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

restore_component(*component_name*, *save_path*)
Restores a component's parameters from a save location.
Parameters
- **component_name** – The component to restore.
- **save_path** – The save location.

save(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.
Parameters
- **directory** – Optional checkpoint directory.
- **append_timestep** – Appends the current timestep to the checkpoint file if true.

Returns Checkpoint path where the model was saved.

save_component(*component_name*, *save_path*)
Saves a component of this model to the designated location.
Parameters
- **component_name** – The component to save.
- **save_path** – The location to save to.

Returns Checkpoint path where the component was saved.

setup()
Sets up the TensorFlow model graph and initializes (and enters) the TensorFlow session.

tf_action_exploration(*action*, *exploration*, *action_spec*)
Applies optional exploration to the action (post-processor for action outputs).
Parameters

- **action** (*tf.Tensor*) – The original output action tensor (to be post-processed).
- **exploration** (Exploration) – The Exploration object to use.
- **action_spec** (*dict*) – Dict specifying the action space.

Returns The post-processed action output tensor.

tf_actions_and_internals(*states*, *internals*, *deterministic*)

Creates and returns the TensorFlow operations for retrieving the actions and - if applicable - the posterior internal state Tensors in reaction to the given input states (and prior internal states).

Parameters

- **states** (*dict*) – Dict of state tensors (each key represents one state space component).
- **internals** – List of prior internal state tensors.
- **deterministic** – Boolean tensor indicating whether action should be chosen deterministically.

Returns

1. dict of output actions (with or without exploration applied (see `deterministic`))
2. list of posterior internal state Tensors (empty for non-internal state models)

Return type tuple

tf_observe_timestep(*states*, *internals*, *actions*, *terminal*, *reward*)

Creates the TensorFlow operations for performing the observation of a full time step's information.

Parameters

- **states** (*dict*) – Dict of state tensors (each key represents one state space component).
- **internals** – List of prior internal state tensors.
- **actions** – Dict of action tensors.
- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.

Returns The observation operation.

MemoryAgent

BatchAgent

Deep-Q-Networks (DQN)

class tensorforce.agents.**DQNAgent**(*states*, *actions*, *network*, *batched_observe=True*, *batching_capacity=1000*, *scope='dqn'*, *device=None*, *saver=None*, *summarizer=None*, *execution=None*, *variable_noise=None*, *states_preprocessing=None*, *actions_exploration=None*, *reward_preprocessing=None*, *update_mode=None*, *memory=None*, *optimizer=None*, *discount=0.99*, *distributions=None*, *entropy_regularization=None*, *target_sync_frequency=10000*, *target_update_weight=1.0*, *double_q_model=False*, *huber_loss=None*)

Bases: *tensorforce.agents.learning_agent.LearningAgent*

Deep Q-Network agent (Mnih et al., 2015).

__init__(*states*, *actions*, *network*, *batched_observe=True*, *batching_capacity=1000*, *scope='dqn'*, *device=None*, *saver=None*, *summarizer=None*, *execution=None*, *variable_noise=None*, *states_preprocessing=None*, *actions_exploration=None*, *reward_preprocessing=None*, *update_mode=None*, *memory=None*, *optimizer=None*, *discount=0.99*, *distributions=None*, *entropy_regularization=None*, *target_sync_frequency=10000*, *target_update_weight=1.0*, *double_q_model=False*, *huber_loss=None*)

Initializes the DQN agent.

Parameters **update_mode** – Update mode specification, with the following attributes:

Parameters

- **memory** (*spec*) – Memory specification, see core.memories module for more information (default: {type='replay', include_next_states=true, capacity=1000*batch_size}).
- **optimizer** (*spec*) – Optimizer specification, see core.optimizers module for more information (default: {type='adam', learning_rate=1e-3}).
- **target_sync_frequency** (*int*) – Target network sync frequency (default: 10000).
- **target_update_weight** (*float*) – Target network update weight (default: 1.0).
- **double_q_model** (*bool*) – Specifies whether double DQN mode is used (default: false).
- **huber_loss** (*float*) – Huber loss clipping (default: none).

Normalized Advantage Functions

class tensorforce.agents.**NAFAgent**(*states*, *actions*, *network*, *batched_observe=True*, *batching_capacity=1000*, *scope='naf'*, *device=None*, *saver=None*, *summarizer=None*, *execution=None*, *variable_noise=None*, *states_preprocessing=None*, *actions_exploration=None*, *reward_preprocessing=None*, *update_mode=None*, *memory=None*, *optimizer=None*, *discount=0.99*, *distributions=None*, *entropy_regularization=None*, *target_sync_frequency=10000*, *target_update_weight=1.0*, *double_q_model=False*, *huber_loss=None*)

Bases: *tensorforce.agents.learning_agent.LearningAgent*

Normalized Advantage Function agent (Gu et al., 2016).

__init__(*states*, *actions*, *network*, *batched_observe=True*, *batching_capacity=1000*, *scope='naf'*, *device=None*, *saver=None*, *summarizer=None*, *execution=None*, *variable_noise=None*, *states_preprocessing=None*, *actions_exploration=None*, *reward_preprocessing=None*, *update_mode=None*, *memory=None*, *optimizer=None*, *discount=0.99*, *distributions=None*, *entropy_regularization=None*, *target_sync_frequency=10000*, *target_update_weight=1.0*, *double_q_model=False*, *huber_loss=None*)

Initializes the NAF agent.

Parameters **update_mode** – Update mode specification, with the following attributes:

Parameters

- **memory** (*spec*) – Memory specification, see core.memories module for more information (default: {type='replay', include_next_states=true, capacity=1000*batch_size}).
- **optimizer** (*spec*) – Optimizer specification, see core.optimizers module for more information (default: {type='adam', learning_rate=1e-3}).
- **target_sync_frequency** (*int*) – Target network sync frequency (default: 10000).
- **target_update_weight** (*float*) – Target network update weight (default: 1.0).
- **double_q_model** (*bool*) – Specifies whether double DQN mode is used (default: false).
- **huber_loss** (*float*) – Huber loss clipping (default: none).

Deep-Q-learning from demonstration (DQFD)

class tensorforce.agents.**DQFDAgent**(*states*, *actions*, *network*, *batched_observe=True*, *batching_capacity=1000*, *scope='dqfd'*, *device=None*, *saver=None*, *summarizer=None*, *execution=None*, *variable_noise=None*, *states_preprocessing=None*, *actions_exploration=None*, *reward_preprocessing=None*, *update_mode=None*, *memory=None*, *optimizer=None*, *discount=0.99*, *distributions=None*, *entropy_regularization=None*, *target_sync_frequency=10000*, *target_update_weight=1.0*, *huber_loss=None*, *expert_margin=0.5*, *supervised_weight=0.1*, *demo_memory_capacity=10000*, *demo_sampling_ratio=0.2*)

Bases: *tensorforce.agents.learning_agent.LearningAgent*

Deep Q-learning from demonstration agent (Hester et al., 2017).

__init__(*states*, *actions*, *network*, *batched_observe=True*, *batching_capacity=1000*, *scope='dqfd'*, *device=None*, *saver=None*, *summarizer=None*, *execution=None*, *variable_noise=None*, *states_preprocessing=None*, *actions_exploration=None*, *reward_preprocessing=None*, *update_mode=None*, *memory=None*, *optimizer=None*, *discount=0.99*, *distributions=None*, *entropy_regularization=None*, *target_sync_frequency=10000*, *target_update_weight=1.0*, *huber_loss=None*, *expert_margin=0.5*, *supervised_weight=0.1*, *demo_memory_capacity=10000*, *demo_sampling_ratio=0.2*)

Initializes the DQFD agent.

Parameters **update_mode** – Update mode specification, with the following attributes:

Parameters

- **memory** (*spec*) – Memory specification, see core.memories module for more information (default: {type='replay', include_next_states=true, capacity=1000*batch_size}).
- **optimizer** (*spec*) – Optimizer specification, see core.optimizers module for more information (default: {type='adam', learning_rate=1e-3}).
- **target_sync_frequency** (*int*) – Target network sync frequency (default: 10000).
- **target_update_weight** (*float*) – Target network update weight (default: 1.0).
- **huber_loss** (*float*) – Huber loss clipping (default: none).
- **expert_margin** (*float*) – Enforced supervised margin between expert action Q-value and other Q-values (default: 0.5).
- **supervised_weight** (*float*) – Weight of supervised loss term (default: 0.1).
- **demo_memory_capacity** (*int*) – Capacity of expert demonstration memory (default: 10000).
- **demo_sampling_ratio** (*float*) – Runtime sampling ratio of expert data (default: 0.2).

import_demonstrations(*demonstrations*)

Imports demonstrations, i.e. expert observations. Note that for large numbers of observations,

set_demonstrations is more appropriate, which directly sets memory contents to an array an expects a different layout.

Parameters **demonstrations** – List of observation dicts

pretrain(*steps*)

Computes pre-train updates.

Parameters **steps** – Number of updates to execute.

Vanilla Policy Gradient

class tensorforce.agents.**VPGAgent**(*states*, *actions*, *network*, *batched_observe=True*, *batching_capacity=1000*, *scope='vpg'*, *device=None*, *saver=None*, *summarizer=None*, *execution=None*, *variable_noise=None*, *states_preprocessing=None*, *actions_exploration=None*, *reward_preprocessing=None*, *update_mode=None*, *memory=None*, *optimizer=None*, *discount=0.99*, *distributions=None*, *entropy_regularization=None*, *baseline_mode=None*, *baseline=None*, *baseline_optimizer=None*, *gae_lambda=None*)

Bases: *tensorforce.agents.learning_agent.LearningAgent*

Vanilla policy gradient agent (Williams, 1992)).

__init__(*states*, *actions*, *network*, *batched_observe=True*, *batching_capacity=1000*, *scope='vpg'*, *device=None*, *saver=None*, *summarizer=None*, *execution=None*, *variable_noise=None*, *states_preprocessing=None*, *actions_exploration=None*, *reward_preprocessing=None*, *update_mode=None*, *memory=None*, *optimizer=None*, *discount=0.99*, *distributions=None*, *entropy_regularization=None*, *baseline_mode=None*, *baseline=None*, *baseline_optimizer=None*, *gae_lambda=None*)

Initializes the VPG agent.

Parameters **update_mode** – Update mode specification, with the following attributes:

Parameters

- **memory** (*spec*) – Memory specification, see core.memories module for more information (default: {type='latest', include_next_states=false, capacity=1000*batch_size}).
- **optimizer** (*spec*) – Optimizer specification, see core.optimizers module for more information (default: {type='adam', learning_rate=1e-3}).
- **baseline_mode** (*str*) – One of 'states', 'network' (default: none).
- **baseline** (*spec*) – Baseline specification, see core.baselines module for more information (default: none).
- **baseline_optimizer** (*spec*) – Baseline optimizer specification, see core.optimizers module for more information (default: none).
- **gae_lambda** (*float*) – Lambda factor for generalized advantage estimation (default: none).

Trust Region Policy Optimization (TRPO)

class tensorforce.agents.**TRPOAgent**(*states*, *actions*, *network*, *batched_observe=True*, *batching_capacity=1000*, *scope='trpo'*, *device=None*, *saver=None*, *summarizer=None*, *execution=None*, *variable_noise=None*, *states_preprocessing=None*, *actions_exploration=None*, *reward_preprocessing=None*, *update_mode=None*, *memory=None*, *discount=0.99*, *distributions=None*, *entropy_regularization=None*, *baseline_mode=None*, *baseline=None*, *baseline_optimizer=None*, *gae_lambda=None*, *likelihood_ratio_clipping=None*, *learning_rate=0.001*, *cg_max_iterations=20*, *cg_damping=0.001*, *cg_unroll_loop=False*, *ls_max_iterations=10*, *ls_accept_ratio=0.9*, *ls_unroll_loop=False*)

Bases: *tensorforce.agents.learning_agent.LearningAgent*

Trust Region Policy Optimization agent (Schulman et al., 2015).

__init__(*states*, *actions*, *network*, *batched_observe=True*, *batching_capacity=1000*, *scope='trpo'*, *device=None*, *saver=None*, *summarizer=None*, *execution=None*, *variable_noise=None*, *states_preprocessing=None*, *actions_exploration=None*, *reward_preprocessing=None*, *update_mode=None*, *memory=None*, *discount=0.99*, *distributions=None*, *entropy_regularization=None*, *baseline_mode=None*, *baseline=None*, *baseline_optimizer=None*, *gae_lambda=None*, *likelihood_ratio_clipping=None*, *learning_rate=0.001*, *cg_max_iterations=20*, *cg_damping=0.001*, *cg_unroll_loop=False*, *ls_max_iterations=10*, *ls_accept_ratio=0.9*, *ls_unroll_loop=False*)

Initializes the TRPO agent.

Parameters **update_mode** – Update mode specification, with the following attributes:

Parameters

- **memory** (*spec*) – Memory specification, see core.memories module for more information (default: {type='latest', include_next_states=false, capacity=1000*batch_size}).
- **optimizer** (*spec*) – TRPO agent implicitly defines a optimized-step natural-gradient optimizer.
- **baseline_mode** (*str*) – One of 'states', 'network' (default: none).
- **baseline** (*spec*) – Baseline specification, see core.baselines module for more information (default: none).
- **baseline_optimizer** (*spec*) – Baseline optimizer specification, see core.optimizers module for more information (default: none).
- **gae_lambda** (*float*) – Lambda factor for generalized advantage estimation (default: none).
- **likelihood_ratio_clipping** (*float*) – Likelihood ratio clipping for policy gradient (default: none).
- **learning_rate** (*float*) – Learning rate of natural-gradient optimizer (default: 1e-3).
- **cg_max_iterations** (*int*) – Conjugate-gradient max iterations (default: 20).

- **cg_damping** (*float*) – Conjugate-gradient damping (default: 1e-3).
- **cg_unroll_loop** (*bool*) – Conjugate-gradient unroll loop (default: false).
- **ls_max_iterations** (*int*) – Line-search max iterations (default: 10).
- **ls_accept_ratio** (*float*) – Line-search accept ratio (default: 0.9).
- **ls_unroll_loop** (*bool*) – Line-search unroll loop (default: false).

1.1.2 State preprocessing

The agent handles state preprocessing. A preprocessor takes the raw state input from the environment and modifies it (for instance, image resize, state concatenation, etc.). You can find information about our ready-to-use preprocessors here.

1.1.3 Building your own agent

If you want to build your own agent, it should always inherit from `Agent`. If your agent uses a replay memory, it should probably inherit from `MemoryAgent`, if it uses a batch replay that is emptied after each update, it should probably inherit from `BatchAgent`.

We distinguish between agents and models. The `Agent` class handles the interaction with the environment, such as state preprocessing, exploration and observation of rewards. The `Model` class handles the mathematical operations, such as building the tensorflow operations, calculating the desired action and updating (i.e. optimizing) the model weights.

To start building your own agent, please refer to this blogpost to gain a deeper understanding of the internals of the TensorForce library. Afterwards, have look on a sample implementation, e.g. the DQN Agent and DQN Model.

1.2 Environments

A reinforcement learning environment provides the API to a simulated or real environment as the subject for optimization. It could be anything from video games (e.g. Atari) to robots or trading systems. The agent interacts with this environment and learns to act optimally in its dynamics.

Environment <-> Runner <-> Agent <-> Model

class `tensorforce.environments.`**Environment**

Base environment class.

actions

Return the action space. Might include subdicts if multiple actions are available simultaneously.

Returns: dict of action properties (continuous, number of actions)

close()

Close environment. No other method calls possible afterwards.

execute(*actions*)

Executes action, observes next state(s) and reward.

Parameters **actions** – Actions to execute.

Returns (Dict of) next state(s), boolean indicating terminal, and reward signal.

static from_spec(*spec*, *kwargs*)

Creates an environment from a specification dict.

reset()

Reset environment and setup for new episode.

Returns initial state of reset environment.

seed(*seed*)
Sets the random seed of the environment to the given value (current time, if seed=None). Naturally deterministic Environments (e.g. ALE or some gym Envs) don't have to implement this method.

Parameters **seed**(*int*) – The seed to use for initializing the pseudo-random number generator (default=epoch time in sec).

Returns: The actual seed (int) used OR None if Environment did not override this method (no seeding supported).

states
Return the state space. Might include subdicts if multiple states are available simultaneously.

Returns: dict of state properties (shape and type).

1.2.1 Ready-to-use environments

OpenAI Gym

class tensorforce.contrib.openai_gym.**OpenAIGym**(*gym_id*, *monitor=None*, *monitor_safe=False*, *monitor_video=0*, *visualize=False*)
Bases: *tensorforce.environments.environment.Environment*

__init__(*gym_id*, *monitor=None*, *monitor_safe=False*, *monitor_video=0*, *visualize=False*)
Initialize OpenAI Gym.

Parameters

- **gym_id** – OpenAI Gym environment ID. See https://gym.openai.com/envs
- **monitor** – Output directory. Setting this to None disables monitoring.
- **monitor_safe** – Setting this to True prevents existing log files to be overwritten. Default False.
- **monitor_video** – Save a video every monitor_video steps. Setting this to 0 disables recording of videos.
- **visualize** – If set True, the program will visualize the trainings of gym's environment. Note that such visualization is probabily going to slow down the training.

OpenAI Universe

class tensorforce.contrib.openai_universe.**OpenAIUniverse**(*env_id*)
Bases: *tensorforce.environments.environment.Environment*

OpenAI Universe Integration: https://universe.openai.com/. Contains OpenAI Gym: https://gym.openai.com/.

__init__(*env_id*)
Initialize OpenAI universe environment.

Parameters **env_id** – string with id/descriptor of the universe environment, e.g. 'HarvestDay-v0'.

Deepmind Lab

class tensorforce.contrib.deepmind_lab.**DeepMindLab**(*level_id*, *repeat_action=1*, *state_attribute='RGB_INTERLACED'*, *settings={'width': '320', 'appendCommand': '', 'fps': '60', 'height': '240'}*)

Bases: *tensorforce.environments.environment.Environment*

DeepMind Lab Integration: https://arxiv.org/abs/1612.03801 https://github.com/deepmind/lab

Since DeepMind lab is only available as source code, a manual install via bazel is required. Further, due to the way bazel handles external dependencies, cloning TensorForce into lab is the most convenient way to run it using the bazel BUILD file we provide. To use lab, first download and install it according to instructions https://github.com/deepmind/lab/blob/master/docs/build.md:

```
git clone https://github.com/deepmind/lab.git
```

Add to the lab main BUILD file:

Clone TensorForce into the lab directory, then run the TensorForce bazel runner.

Note that using any specific configuration file currently requires changing the Tensorforce BUILD file to adjust environment parameters.

```
bazel run //tensorforce:lab_runner
```

Please note that we have not tried to reproduce any lab results yet, and these instructions just explain connectivity in case someone wants to get started there.

__init__(*level_id*, *repeat_action=1*, *state_attribute='RGB_INTERLACED'*, *settings={'width': '320', 'appendCommand': '', 'fps': '60', 'height': '240'}*)

Initialize DeepMind Lab environment.

Parameters

- **level_id** – string with id/descriptor of the level, e.g. 'seekavoid_arena_01'.
- **repeat_action** – number of frames the environment is advanced, executing the given action during every frame.
- **state_attribute** – Attributes which represents the state for this environment, should adhere to the specification given in DeepMindLabEnvironment.state_spec(level_id).
- **settings** – dict specifying additional settings as key-value string pairs. The following options are recognized: 'width' (horizontal resolution of the observation frames), 'height' (vertical resolution of the observation frames), 'fps' (frames per second) and 'appendCommand' (commands for the internal Quake console).

close()

Closes the environment and releases the underlying Quake III Arena instance. No other method calls possible afterwards.

execute(*actions*)

Pass action to universe environment, return reward, next step, terminal state and additional info.

Parameters **action** – action to execute as numpy array, should have dtype np.intc and should adhere to the specification given in DeepMindLabEnvironment.action_spec(level_id)

Returns dict containing the next state, the reward, and a boolean indicating if the next state is a terminal state

fps
An advisory metric that correlates discrete environment steps ("frames") with real (wallclock) time: the number of frames per (real) second.

num_steps
Number of frames since the last reset() call.

reset()
Resets the environment to its initialization state. This method needs to be called to start a new episode after the last episode ended.
Returns initial state

Unreal Engine 4 Games

class tensorforce.contrib.unreal_engine.**UE4Environment**(*host='localhost'*, *port=6025*, *connect=True*, *discretize_actions=False*, *delta_time=0*, *num_ticks=4*)
Bases: *tensorforce.contrib.remote_environment.RemoteEnvironment*, *tensorforce.contrib.state_settable_environment.StateSettableEnvironment*

A special RemoteEnvironment for UE4 game connections. Communicates with the remote to receive information on the definitions of action- and observation spaces. Sends UE4 Action- and Axis-mappings as RL-actions and receives observations back defined by MLObserver objects placed in the Game (these could be camera pixels or other observations, e.g. a x/y/z position of some game actor).

__init__(*host='localhost'*, *port=6025*, *connect=True*, *discretize_actions=False*, *delta_time=0*, *num_ticks=4*)
Parameters
- **host** (*str*) – The hostname to connect to.
- **port** (*int*) – The port to connect to.
- **connect** (*bool*) – Whether to connect already in this c'tor.
- **discretize_actions** (*bool*) – Whether to treat axis-mappings defined in UE4 game as discrete actions. This would be necessary e.g. for agents that use q-networks where the output are q-values per discrete state-action pair.
- **delta_time** (*float*) – The fake delta time to use for each single game tick.
- **num_ticks** (*int*) – The number of ticks to be executed in a single act call (each tick will repeat the same given actions).

discretize_action_space_desc()
Creates a list of discrete action(-combinations) in case we want to learn with a discrete set of actions, but only have action-combinations (maybe even continuous) available from the env. E.g. the UE4 game has the following action/axis-mappings:

```
{
'Fire':
    {'type': 'action', 'keys': ('SpaceBar',)},
'MoveRight':
    {'type': 'axis', 'keys': (('Right', 1.0), ('Left', -1.0), ('A', -
↪1.0), ('D', 1.0))},
}
```

-> this method will discretize them into the following 6 discrete actions:

```
[
[(Right, 0.0),(SpaceBar, False)],
[(Right, 0.0),(SpaceBar, True)]
[(Right, -1.0),(SpaceBar, False)],
[(Right, -1.0),(SpaceBar, True)],
[(Right, 1.0),(SpaceBar, False)],
[(Right, 1.0),(SpaceBar, True)],
]
```

execute(*actions*)
Executes a single step in the UE4 game. This step may be comprised of one or more actual game ticks for all of which the same given action- and axis-inputs (or action number in case of discretized actions) are repeated. UE4 distinguishes between action-mappings, which are boolean actions (e.g. jump or dont-jump) and axis-mappings, which are continuous actions like MoveForward with values between -1.0 (run backwards) and 1.0 (run forwards), 0.0 would mean: stop.

reset()
same as step (no kwargs to pass), but needs to block and return observation_dict
- stores the received observation in self.last_observation

translate_abstract_actions_to_keys(*abstract*)
Translates a list of tuples ([pretty mapping], [value]) to a list of tuples ([some key], [translated value]) each single item in abstract will undergo the following translation:

Example1: we want: "MoveRight": 5.0 possible keys for the action are: ("Right", 1.0), ("Left", -1.0) result: "Right": 5.0 * 1.0 = 5.0

Example2: we want: "MoveRight": -0.5 possible keys for the action are: ("Left", -1.0), ("Right", 1.0) result: "Left": -0.5 * -1.0 = 0.5 (same as "Right": -0.5)

1.3 Preprocessing

Often it is necessary to modify state input tensors before passing them to the reinforcement learning agent. This could be due to various reasons, e.g.:

- Feature scaling / input normalization,
- Data reduction,
- Ensuring the Markov property by concatenating multiple states (e.g. in Atari)

TensorForce comes with a number of ready-to-use preprocessors, a preprocessing stack and easy ways to implement your own preprocessors.

1.3.1 Usage

The

Each preprocessor implements three methods:

1. The constructor (`__init__`) for parameter initialization
2. `process(state)` takes a state and returns the processed state
3. `processed_shape(original_shape)` takes a shape and returns the processed shape

The preprocessing stack iteratively calls these functions of all preprocessors in the stack and returns the result.

Using one preprocessor

```
from tensorforce.core.preprocessing import Sequence

pp_seq = Sequence(4)  # initialize preprocessor (return sequence of last 4 states)

state = env.reset()  # reset environment
processed_state = pp_seq.process(state)  # process state
```

Using a preprocessing stack

You can stack multipe preprocessors:

```
from tensorforce.core.preprocessing import Preprocessing, Grayscale, Sequence

pp_gray = Grayscale()  # initialize grayscale preprocessor
pp_seq = Sequence(4)  # initialize sequence preprocessor

stack = Preprocessing()  # initialize preprocessing stack
stack.add(pp_gray)  # add grayscale preprocessor to stack
stack.add(pp_seq)  # add maximum preprocessor to stack

state = env.reset()  # reset environment
processed_state = stack.process(state)  # process state
```

Using a configuration dict

If you use configuration objects, you can build your preprocessing stack from a config:

```
from tensorforce.core.preprocessing import Preprocessing

preprocessing_config = [
    {
        "type": "image_resize",
        "width": 84,
        "height": 84
    }, {
        "type": "grayscale"
    }, {
        "type": "center"
    }, {
        "type": "sequence",
        "length": 4
    }
]

stack = Preprocessing.from_spec(preprocessing_config)
config.state_shape = stack.shape(config.state_shape)
```

The `Agent` class expects a *preprocessing* configuration parameter and then handles preprocessing automatically:

```
from tensorforce.agents import DQNAgent

agent = DQNAgent(config=dict(
    states=...,
```

```
    actions=...,
    preprocessing=preprocessing_config,
    # ...
))
```

1.3.2 Ready-to-use preprocessors

These are the preprocessors that come with TensorForce:

Standardize

Grayscale

ImageResize

Normalize

Sequence

1.3.3 Building your own preprocessor

All preprocessors should inherit from `tensorforce.core.preprocessing.Preprocessor`.

For a start, please refer to the source of the Grayscale preprocessor.

1.4 TensorForce: Details for "summary_spec" agent parameters

1.4.1 summarizer

TensorForce has the ability to record summary data for use with TensorBoard as well STDIO and file export. This is accomplished through dictionary parameter called "summarizer" passed to the agent on initialization.

"summarizer" supports the following optional dictionary entries:

Key	Value
directory	(str) Path to storage for TensorBoard summary data
steps	(int) Frequency in steps between storage of summary data
seconds	(int) Frequency in seconds to store summary data
labels	(list) Requested Export, See "*LABELS*" section
meta_dict	(dict) For used with label "configuration"

1.4.2 LABELS

Entry	Data produced
losses	Training total-loss and "loss-without-regularization"
total-loss	Final calculated loss value
variables	Network variables
inputs	Equivalent to: ['states', 'actions', 'rewards']
states	Histogram of input state space
actions	Histogram of input action space
rewards	Histogram of input reward space
gradients	Histogram and scalar gradients
gradients_histogram	Variable gradients as histograms
gradients_scalar	Variable Mean/Variance of gradients as scalar
regularization	Regularization values
configuration	See *Configuration Export* for more detail
configuration	Export configuration to "TEXT" tab in TensorBoard
print_configuration	Prints configuration to STDOUT

```
from tensorforce.agents import PPOAgent

# Create a Proximal Policy Optimization agent
agent = PPOAgent(
    states=...,
    actions=...,
    network=...,
    summarizer=dict(directory="./board/",
                    steps=50,
                    labels=['configuration',
                        'gradients_scalar',
                        'regularization',
                        'inputs',
                        'losses',
                        'variables']
                ),
    ...
)
```

1.4.3 Configuration Export

Adding the "configuration" label will create a "TEXT" tab in TensorBoard that contains all the parameters passed to the Agent. By using the additional "summarizer" dictionary key "meta_dict", custom keys and values can be added to the data export. The user may want to pass "Description", "Experiement #", "InputDataSet", etc.

If a key is already in use within TensorForce an error will be raised to notify you to change the key value. To use the custom feature, create a dictionary with keys to export:

```
from tensorforce.agents import PPOAgent

metaparams['MyDescription'] = "This experiment covers the first test ...."
metaparams['My2D'] = np.ones((9,9))   # 9x9 matrix of  1.0's
metaparams['My1D'] = np.ones((9))   # Column of 9   1.0's

# Create a Proximal Policy Optimization agent
agent = PPOAgent(
    states=...,
    actions=...,
    network=...,
    summarizer=dict(directory="./board/",
                        steps=50,
                        meta_dict=metaparams,  #Add custom keys to export
                        labels=['configuration',
                            'gradients_scalar',
                            'regularization',
                            'inputs',
                            'losses',
                            'variables']
                    ),
    ...
)
```

Use the "print_configuration" label to export the configuration data to the command line's STDOUT.

1.5 Runners

A "runner" manages the interaction between the Environment and the Agent. TensorForce comes with ready-to-use runners. Of course, you can implement your own runners, too. If you are not using simulation environments, the runner is simply your application code using the Agent API.

> Environment <-> Runner <-> Agent <-> Model

1.5.1 Ready-to-use runners

We implemented a standard runner, a threaded runner (for real-time interaction e.g. with OpenAI Universe) and a distributed runner for A3C variants.

Runner

This is the standard runner. It requires an agent and an environment for initialization:

```
from tensorforce.execution import Runner

runner = Runner(
    agent = agent,   # Agent object
    environment = env   # Environment object
)
```

A reinforcement learning agent observes states from the environment, selects actions and collect experience which is used to update its model and improve action selection. You can get information about our ready-to-use agents here.

The environment object is either the "real" environment, or a proxy which fulfills the actions selected by the agent in the real world. You can find information about environments here.

The runner is started with the `Runner.run(...)` method:

```
runner.run(
    episodes = int,  # number of episodes to run
    max_timesteps = int,  # maximum timesteps per episode
    episode_finished = object,  # callback function called when episode is finished
)
runner.close()
```

You can use the episode_finished callback for printing performance feedback:

```
def episode_finished(r):
    if r.episode % 10 == 0:
        print("Finished episode {ep} after {ts} timesteps".format(ep=r.episode + 1,␣
↪ts=r.timestep + 1))
        print("Episode reward: {}".format(r.episode_rewards[-1]))
        print("Average of last 10 rewards: {}".format(np.mean(r.episode_rewards[-
↪10:])))
    return True
```

Using the Runner

Here is some example code for using the runner (without preprocessing).

```
import logging

from tensorforce.contrib.openai_gym import OpenAIGym
from tensorforce.agents import DQNAgent
from tensorforce.execution import Runner

def main():
    gym_id = 'CartPole-v0'
    max_episodes = 10000
    max_timesteps = 1000

    env = OpenAIGym(gym_id)
    network_spec = [
        dict(type='dense', size=32, activation='tanh'),
        dict(type='dense', size=32, activation='tanh')
    ]

    agent = DQNAgent(
        states_spec=env.states,
        actions_spec=env.actions,
        network_spec=network_spec,
        batch_size=64
    )

    runner = Runner(agent, env)

    report_episodes = 10
```

```
    def episode_finished(r):
        if r.episode % report_episodes == 0:
            logging.info("Finished episode {ep} after {ts} timesteps".format(ep=r.
↪episode, ts=r.timestep))
            logging.info("Episode reward: {}".format(r.episode_rewards[-1]))
            logging.info("Average of last 100 rewards: {}".format(sum(r.episode_
↪rewards[-100:]) / 100))
        return True

    print("Starting {agent} for Environment '{env}'".format(agent=agent, env=env))

    runner.run(max_episodes, max_timesteps, episode_finished=episode_finished)
    runner.close()

    print("Learning finished. Total episodes: {ep}".format(ep=runner.episode))

if __name__ == '__main__':
    main()
```

1.5.2 Building your own runner

There are three mandatory tasks any runner implements: Obtaining an action from the agent, passing it to the environment, and passing the resulting observation to the agent.

```
# Get action
action = agent.act(state)

# Execute action in the environment
state, reward, terminal_state = environment.execute(action)

# Pass observation to the agent
agent.observe(state, action, reward, terminal_state)
```

The key idea here is the separation of concerns. External code should not need to manage batches or remember network features, this is that the agent is for. Conversely, an agent need not concern itself with how a model is implemented and the API should facilitate easy combination of different agents and models.

If you would like to build your own runner, it is probably a good idea to take a look at the source code of our Runner class.

1.6 tensorforce package

1.6.1 Subpackages

tensorforce.agents package

Submodules

tensorforce.agents.agent module

class tensorforce.agents.agent.**Agent**(*states*, *actions*, *batched_observe=True*, *batching_capacity=1000*)

Bases: object

Base class for TensorForce agents.

__init__(*states*, *actions*, *batched_observe=True*, *batching_capacity=1000*)

Initializes the agent.

Parameters **states** – States specification, with the following attributes (required):

Parameters **actions** – Actions specification, with the following attributes (required):

Parameters

- **batched_observe** (*bool*) – Specifies whether calls to model.observe() are batched, for improved performance (default: true).
- **batching_capacity** (*int*) – Batching capacity of agent and model (default: 1000).

act(*states*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)

Return action(s) for given state(s). States preprocessing and exploration are applied if configured accordingly.

Parameters

- **states** (*any*) – One state (usually a value tuple) or dict of states if multiple states are expected.
- **deterministic** (*bool*) – If true, no exploration and sampling is applied.
- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).
- **fetch_tensors** (*list*) – Optional String of named tensors to fetch

Returns Scalar value of the action or dict of multiple actions the agent wants to execute. (fetched_tensors) Optional dict() with named tensors fetched

close()

static from_spec(*spec*, *kwargs*)

Creates an agent from a specification dict.

initialize_model()

Creates the model for the respective agent based on specifications given by user. This is a separate call after constructing the agent because the agent constructor has to perform a number of checks on the specs first, sometimes adjusting them e.g. by converting to a dict.

last_observation()

observe(*terminal*, *reward*)
Observe experience from the environment to learn from. Optionally pre-processes rewards Child classes should call super to get the processed reward EX: terminal, reward = super()...

Parameters

- **terminal** (*bool*) – boolean indicating if the episode terminated after the observation.
- **reward** (*float*) – scalar reward that resulted from executing the action.

reset()
Reset the agent to its initial state (e.g. on experiment start). Updates the Model's internal episode and time step counter, internal states, and resets preprocessors.

restore_model(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters

- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

save_model(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters

- **directory** (*str*) – Optional checkpoint directory.
- **append_timestep** (*bool*) – Appends the current timestep to the checkpoint file if true. If this is set to True, the load path must include the checkpoint timestep suffix. For example, if stored to models/ and set to true, the exported file will be of the form models/model.ckpt-X where X is the last timestep saved. The load path must precisely match this file name. If this option is turned off, the checkpoint will always overwrite the file specified in path and the model can always be loaded under this path.

Returns Checkpoint path were the model was saved.

set_normalized_actions(*actions*)

set_normalized_states(*states*)

should_stop()

tensorforce.agents.batch_agent module

tensorforce.agents.constant_agent module

class tensorforce.agents.constant_agent.**ConstantAgent**(*states*, *actions*, *action_values*, *batched_observe=True*, *batching_capacity=1000*, *scope='constant'*, *device=None*, *saver=None*, *summarizer=None*, *distributed=None*)

Bases: *tensorforce.agents.agent.Agent*

Agent returning constant action values.

__init__(*states*, *actions*, *action_values*, *batched_observe=True*, *batching_capacity=1000*, *scope='constant'*, *device=None*, *saver=None*, *summarizer=None*, *distributed=None*)

Initializes the constant agent.

Parameters

- **action_values** (*value, or dict of values*) – Action values returned by the agent (required).
- **scope** (*str*) – TensorFlow scope (default: name of agent).
- **device** – TensorFlow device (default: none)
- **saver** – Saver specification, with the following attributes (default: none):

Parameters **summarizer** – Summarizer specification, with the following attributes (default: none):

Parameters **distributed** – Distributed specification, with the following attributes (default: none):

act(*states*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)

Return action(s) for given state(s). States preprocessing and exploration are applied if configured accordingly.

Parameters

- **states** (*any*) – One state (usually a value tuple) or dict of states if multiple states are expected.
- **deterministic** (*bool*) – If true, no exploration and sampling is applied.
- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).
- **fetch_tensors** (*list*) – Optional String of named tensors to fetch

Returns Scalar value of the action or dict of multiple actions the agent wants to execute. (fetched_tensors) Optional dict() with named tensors fetched

close()

from_spec(*spec*, *kwargs*)

Creates an agent from a specification dict.

initialize_model()

last_observation()

observe(*terminal*, *reward*)

Observe experience from the environment to learn from. Optionally pre-processes rewards Child classes should call super to get the processed reward EX: terminal, reward = super()...

Parameters

- **terminal** (*bool*) – boolean indicating if the episode terminated after the observation.
- **reward** (*float*) – scalar reward that resulted from executing the action.

reset()
Reset the agent to its initial state (e.g. on experiment start). Updates the Model's internal episode and time step counter, internal states, and resets preprocessors.

restore_model(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters

- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

save_model(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters

- **directory** (*str*) – Optional checkpoint directory.
- **append_timestep** (*bool*) – Appends the current timestep to the checkpoint file if true. If this is set to True, the load path must include the checkpoint timestep suffix. For example, if stored to models/ and set to true, the exported file will be of the form models/model.ckpt-X where X is the last timestep saved. The load path must precisely match this file name. If this option is turned off, the checkpoint will always overwrite the file specified in path and the model can always be loaded under this path.

Returns Checkpoint path were the model was saved.

set_normalized_actions(*actions*)

set_normalized_states(*states*)

should_stop()

tensorforce.agents.ddqn_agent module

tensorforce.agents.dqfd_agent module

class tensorforce.agents.dqfd_agent.**DQFDAgent**(*states*, *actions*, *network*, *batched_observe=True*, *batching_capacity=1000*, *scope='dqfd'*, *device=None*, *saver=None*, *summarizer=None*, *execution=None*, *variable_noise=None*, *states_preprocessing=None*, *actions_exploration=None*, *reward_preprocessing=None*, *update_mode=None*, *memory=None*, *optimizer=None*, *discount=0.99*, *distributions=None*, *entropy_regularization=None*, *target_sync_frequency=10000*, *target_update_weight=1.0*, *huber_loss=None*, *expert_margin=0.5*, *supervised_weight=0.1*, *demo_memory_capacity=10000*, *demo_sampling_ratio=0.2*)

Bases: *tensorforce.agents.learning_agent.LearningAgent*

Deep Q-learning from demonstration agent (Hester et al., 2017).

__init__(*states*, *actions*, *network*, *batched_observe=True*, *batching_capacity=1000*, *scope='dqfd'*, *device=None*, *saver=None*, *summarizer=None*, *execution=None*, *variable_noise=None*, *states_preprocessing=None*, *actions_exploration=None*, *reward_preprocessing=None*, *update_mode=None*, *memory=None*, *optimizer=None*, *discount=0.99*, *distributions=None*, *entropy_regularization=None*, *target_sync_frequency=10000*, *target_update_weight=1.0*, *huber_loss=None*, *expert_margin=0.5*, *supervised_weight=0.1*, *demo_memory_capacity=10000*, *demo_sampling_ratio=0.2*)

Initializes the DQFD agent.

Parameters **update_mode** – Update mode specification, with the following attributes:

Parameters

- **memory** (*spec*) – Memory specification, see core.memories module for more information (default: {type='replay', include_next_states=true, capacity=1000*batch_size}).
- **optimizer** (*spec*) – Optimizer specification, see core.optimizers module for more information (default: {type='adam', learning_rate=1e-3}).
- **target_sync_frequency** (*int*) – Target network sync frequency (default: 10000).
- **target_update_weight** (*float*) – Target network update weight (default: 1.0).
- **huber_loss** (*float*) – Huber loss clipping (default: none).
- **expert_margin** (*float*) – Enforced supervised margin between expert action Q-value and other Q-values (default: 0.5).
- **supervised_weight** (*float*) – Weight of supervised loss term (default: 0.1).
- **demo_memory_capacity** (*int*) – Capacity of expert demonstration memory (default: 10000).

- **demo_sampling_ratio** (*float*) – Runtime sampling ratio of expert data (default: 0.2).

act(*states*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)
Return action(s) for given state(s). States preprocessing and exploration are applied if configured accordingly.

Parameters

- **states** (*any*) – One state (usually a value tuple) or dict of states if multiple states are expected.
- **deterministic** (*bool*) – If true, no exploration and sampling is applied.
- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).
- **fetch_tensors** (*list*) – Optional String of named tensors to fetch

Returns Scalar value of the action or dict of multiple actions the agent wants to execute. (fetched_tensors) Optional dict() with named tensors fetched

close()

from_spec(*spec*, *kwargs*)
Creates an agent from a specification dict.

import_demonstrations(*demonstrations*)
Imports demonstrations, i.e. expert observations. Note that for large numbers of observations, set_demonstrations is more appropriate, which directly sets memory contents to an array an expects a different layout.

Parameters **demonstrations** – List of observation dicts

import_experience(*experiences*)
Imports experiences.

Parameters **experiences** –

initialize_model()

last_observation()

observe(*terminal*, *reward*)
Observe experience from the environment to learn from. Optionally pre-processes rewards Child classes should call super to get the processed reward EX: terminal, reward = super()...

Parameters

- **terminal** (*bool*) – boolean indicating if the episode terminated after the observation.
- **reward** (*float*) – scalar reward that resulted from executing the action.

pretrain(*steps*)
Computes pre-train updates.

Parameters **steps** – Number of updates to execute.

reset()
Reset the agent to its initial state (e.g. on experiment start). Updates the Model's internal episode and time step counter, internal states, and resets preprocessors.

restore_model(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters

- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

save_model(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters

- **directory** (*str*) – Optional checkpoint directory.
- **append_timestep** (*bool*) – Appends the current timestep to the checkpoint file if true. If this is set to True, the load path must include the checkpoint timestep suffix. For example, if stored to models/ and set to true, the exported file will be of the form models/model.ckpt-X where X is the last timestep saved. The load path must precisely match this file name. If this option is turned off, the checkpoint will always overwrite the file specified in path and the model can always be loaded under this path.

Returns Checkpoint path were the model was saved.

set_normalized_actions(*actions*)

set_normalized_states(*states*)

should_stop()

tensorforce.agents.dqn_agent module

class tensorforce.agents.dqn_agent.**DQNAgent**(*states*, *actions*, *network*, *batched_observe=True*, *batching_capacity=1000*, *scope='dqn'*, *device=None*, *saver=None*, *summarizer=None*, *execution=None*, *variable_noise=None*, *states_preprocessing=None*, *actions_exploration=None*, *reward_preprocessing=None*, *update_mode=None*, *memory=None*, *optimizer=None*, *discount=0.99*, *distributions=None*, *entropy_regularization=None*, *target_sync_frequency=10000*, *target_update_weight=1.0*, *double_q_model=False*, *huber_loss=None*)
Bases: *tensorforce.agents.learning_agent.LearningAgent*

Deep Q-Network agent (Mnih et al., 2015).

__init__(*states*, *actions*, *network*, *batched_observe=True*, *batching_capacity=1000*, *scope='dqn'*, *device=None*, *saver=None*, *summarizer=None*, *execution=None*, *variable_noise=None*, *states_preprocessing=None*, *actions_exploration=None*, *reward_preprocessing=None*, *update_mode=None*, *memory=None*, *optimizer=None*, *discount=0.99*, *distributions=None*, *entropy_regularization=None*, *target_sync_frequency=10000*, *target_update_weight=1.0*, *double_q_model=False*, *huber_loss=None*)
Initializes the DQN agent.

Parameters **update_mode** – Update mode specification, with the following attributes:

Parameters

- **memory** (*spec*) – Memory specification, see core.memories module for more information (default: {type='replay', include_next_states=true, capacity=1000*batch_size}).
- **optimizer** (*spec*) – Optimizer specification, see core.optimizers module for more information (default: {type='adam', learning_rate=1e-3}).
- **target_sync_frequency** (*int*) – Target network sync frequency (default: 10000).
- **target_update_weight** (*float*) – Target network update weight (default: 1.0).
- **double_q_model** (*bool*) – Specifies whether double DQN mode is used (default: false).
- **huber_loss** (*float*) – Huber loss clipping (default: none).

act(*states*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)
Return action(s) for given state(s). States preprocessing and exploration are applied if configured accordingly.

Parameters

- **states** (*any*) – One state (usually a value tuple) or dict of states if multiple states are expected.
- **deterministic** (*bool*) – If true, no exploration and sampling is applied.
- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).
- **fetch_tensors** (*list*) – Optional String of named tensors to fetch

Returns Scalar value of the action or dict of multiple actions the agent wants to execute. (fetched_tensors) Optional dict() with named tensors fetched

close()

from_spec(*spec*, *kwargs*)
Creates an agent from a specification dict.

import_experience(*experiences*)
Imports experiences.

Parameters **experiences** –

initialize_model()

last_observation()

observe(*terminal*, *reward*)
Observe experience from the environment to learn from. Optionally pre-processes rewards Child classes should call super to get the processed reward EX: terminal, reward = super()...

Parameters

- **terminal** (*bool*) – boolean indicating if the episode terminated after the observation.
- **reward** (*float*) – scalar reward that resulted from executing the action.

reset()
Reset the agent to its initial state (e.g. on experiment start). Updates the Model's internal episode and time step counter, internal states, and resets preprocessors.

restore_model(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters

- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

save_model(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters

- **directory** (*str*) – Optional checkpoint directory.
- **append_timestep** (*bool*) – Appends the current timestep to the checkpoint file if true. If this is set to True, the load path must include the checkpoint timestep suffix. For example, if stored to models/ and set to true, the exported file will be of the form models/model.ckpt-X where X is the last timestep saved. The load path must precisely match this file name. If this option is turned off, the checkpoint will always overwrite the file specified in path and the model can always be loaded under this path.

Returns Checkpoint path were the model was saved.

set_normalized_actions(*actions*)

set_normalized_states(*states*)

should_stop()

tensorforce.agents.dqn_nstep_agent module

class tensorforce.agents.dqn_nstep_agent.**DQNNstepAgent**(*states*, *actions*, *network*, *batched_observe=True*, *batching_capacity=1000*, *scope='dqn-nstep'*, *device=None*, *saver=None*, *summarizer=None*, *execution=None*, *variable_noise=None*, *states_preprocessing=None*, *actions_exploration=None*, *reward_preprocessing=None*, *update_mode=None*, *memory=None*, *optimizer=None*, *discount=0.99*, *distributions=None*, *entropy_regularization=None*, *target_sync_frequency=10000*, *target_update_weight=1.0*, *double_q_model=False*, *huber_loss=None*)
Bases: *tensorforce.agents.learning_agent.LearningAgent*

DQN n-step agent.

__init__(*states*, *actions*, *network*, *batched_observe=True*, *batching_capacity=1000*, *scope='dqn-nstep'*, *device=None*, *saver=None*, *summarizer=None*, *execution=None*, *variable_noise=None*, *states_preprocessing=None*, *actions_exploration=None*, *reward_preprocessing=None*, *update_mode=None*, *memory=None*, *optimizer=None*, *discount=0.99*, *distributions=None*, *entropy_regularization=None*, *target_sync_frequency=10000*, *target_update_weight=1.0*, *double_q_model=False*, *huber_loss=None*)

Initializes the DQN n-step agent.

Parameters **update_mode** – Update mode specification, with the following attributes:

Parameters

- **memory** (*spec*) – Memory specification, see core.memories module for more information (default: {type='latest', include_next_states=true, capacity=1000*batch_size}).
- **optimizer** (*spec*) – Optimizer specification, see core.optimizers module for more information (default: {type='adam', learning_rate=1e-3}).
- **target_sync_frequency** (*int*) – Target network sync frequency (default: 10000).
- **target_update_weight** (*float*) – Target network update weight (default: 1.0).
- **double_q_model** (*bool*) – Specifies whether double DQN mode is used (default: false).
- **huber_loss** (*float*) – Huber loss clipping (default: none).

act(*states*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)

Return action(s) for given state(s). States preprocessing and exploration are applied if configured accordingly.

Parameters

- **states** (*any*) – One state (usually a value tuple) or dict of states if multiple states are expected.
- **deterministic** (*bool*) – If true, no exploration and sampling is applied.
- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).
- **fetch_tensors** (*list*) – Optional String of named tensors to fetch

Returns Scalar value of the action or dict of multiple actions the agent wants to execute. (fetched_tensors) Optional dict() with named tensors fetched

close()

from_spec(*spec*, *kwargs*)

Creates an agent from a specification dict.

import_experience(*experiences*)

Imports experiences.

Parameters **experiences** –

initialize_model()

last_observation()

observe(*terminal*, *reward*)
Observe experience from the environment to learn from. Optionally pre-processes rewards Child classes should call super to get the processed reward EX: terminal, reward = super()...

Parameters

- **terminal** (*bool*) – boolean indicating if the episode terminated after the observation.
- **reward** (*float*) – scalar reward that resulted from executing the action.

reset()
Reset the agent to its initial state (e.g. on experiment start). Updates the Model's internal episode and time step counter, internal states, and resets preprocessors.

restore_model(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters

- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

save_model(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters

- **directory** (*str*) – Optional checkpoint directory.
- **append_timestep** (*bool*) – Appends the current timestep to the checkpoint file if true. If this is set to True, the load path must include the checkpoint timestep suffix. For example, if stored to models/ and set to true, the exported file will be of the form models/model.ckpt-X where X is the last timestep saved. The load path must precisely match this file name. If this option is turned off, the checkpoint will always overwrite the file specified in path and the model can always be loaded under this path.

Returns Checkpoint path were the model was saved.

set_normalized_actions(*actions*)

set_normalized_states(*states*)

should_stop()

tensorforce.agents.learning_agent module

class tensorforce.agents.learning_agent.**LearningAgent**(*states*, *actions*, *network*, *update_mode*, *memory*, *optimizer*, *batched_observe=True*, *batching_capacity=1000*, *scope='learning-agent'*, *device=None*, *saver=None*, *summarizer=None*, *execution=None*, *variable_noise=None*, *states_preprocessing=None*, *actions_exploration=None*, *reward_preprocessing=None*, *discount=0.99*, *distributions=None*, *entropy_regularization=None*)

Bases: *tensorforce.agents.agent.Agent*

Base class for learning agents, using as model a subclass of MemoryModel and DistributionModel.

__init__(*states*, *actions*, *network*, *update_mode*, *memory*, *optimizer*, *batched_observe=True*, *batching_capacity=1000*, *scope='learning-agent'*, *device=None*, *saver=None*, *summarizer=None*, *execution=None*, *variable_noise=None*, *states_preprocessing=None*, *actions_exploration=None*, *reward_preprocessing=None*, *discount=0.99*, *distributions=None*, *entropy_regularization=None*)

Initializes the learning agent.

Parameters **update_mode** – Update mode specification, with the following attributes (required):

Parameters

- **memory** (*spec*) – Memory specification, see core.memories module for more information (required).
- **optimizer** (*spec*) – Optimizer specification, see core.optimizers module for more information (required).
- **network** (*spec*) – Network specification, usually a list of layer specifications, see core.networks module for more information (required).
- **scope** (*str*) – TensorFlow scope (default: name of agent).
- **device** – TensorFlow device (default: none)
- **saver** – Saver specification, with the following attributes (default: none):

Parameters **summarizer** – Summarizer specification, with the following attributes (default: none):

Parameters **execution** – Distributed specification, with the following attributes (default: none):

Parameters

- **variable_noise** (*float*) – Standard deviation of variable noise (default: none).

- **states_preprocessing** (*spec, or dict of specs*) – States preprocessing specification, see core.preprocessors module for more information (default: none)
- **actions_exploration** (*spec, or dict of specs*) – Actions exploration specification, see core.explorations module for more information (default: none).
- **reward_preprocessing** (*spec*) – Reward preprocessing specification, see core.preprocessors module for more information (default: none).
- **discount** (*float*) – Discount factor for future rewards (default: 0.99).
- **distributions** (*spec / dict of specs*) – Distributions specifications, see core.distributions module for more information (default: none).
- **entropy_regularization** (*float*) – Entropy regularization weight (default: none).

act(*states*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)
Return action(s) for given state(s). States preprocessing and exploration are applied if configured accordingly.

Parameters

- **states** (*any*) – One state (usually a value tuple) or dict of states if multiple states are expected.
- **deterministic** (*bool*) – If true, no exploration and sampling is applied.
- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).
- **fetch_tensors** (*list*) – Optional String of named tensors to fetch

Returns Scalar value of the action or dict of multiple actions the agent wants to execute. (fetched_tensors) Optional dict() with named tensors fetched

close()

from_spec(*spec*, *kwargs*)
Creates an agent from a specification dict.

import_experience(*experiences*)
Imports experiences.

Parameters **experiences** –

initialize_model()
Creates the model for the respective agent based on specifications given by user. This is a separate call after constructing the agent because the agent constructor has to perform a number of checks on the specs first, sometimes adjusting them e.g. by converting to a dict.

last_observation()

observe(*terminal*, *reward*)
Observe experience from the environment to learn from. Optionally pre-processes rewards Child classes should call super to get the processed reward EX: terminal, reward = super()...

Parameters

- **terminal** (*bool*) – boolean indicating if the episode terminated after the observation.
- **reward** (*float*) – scalar reward that resulted from executing the action.

reset()
Reset the agent to its initial state (e.g. on experiment start). Updates the Model's internal episode and time step counter, internal states, and resets preprocessors.

restore_model(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters

- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

save_model(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters

- **directory** (*str*) – Optional checkpoint directory.
- **append_timestep** (*bool*) – Appends the current timestep to the checkpoint file if true. If this is set to True, the load path must include the checkpoint timestep suffix. For example, if stored to models/ and set to true, the exported file will be of the form models/model.ckpt-X where X is the last timestep saved. The load path must precisely match this file name. If this option is turned off, the checkpoint will always overwrite the file specified in path and the model can always be loaded under this path.

Returns Checkpoint path were the model was saved.

set_normalized_actions(*actions*)

set_normalized_states(*states*)

should_stop()

tensorforce.agents.memory_agent module

tensorforce.agents.naf_agent module

class tensorforce.agents.naf_agent.**NAFAgent**(*states*, *actions*, *network*, *batched_observe=True*, *batching_capacity=1000*, *scope='naf'*, *device=None*, *saver=None*, *summarizer=None*, *execution=None*, *variable_noise=None*, *states_preprocessing=None*, *actions_exploration=None*, *reward_preprocessing=None*, *update_mode=None*, *memory=None*, *optimizer=None*, *discount=0.99*, *distributions=None*, *entropy_regularization=None*, *target_sync_frequency=10000*, *target_update_weight=1.0*, *double_q_model=False*, *huber_loss=None*)
Bases: *tensorforce.agents.learning_agent.LearningAgent*

Normalized Advantage Function agent (Gu et al., 2016).

__init__(*states*, *actions*, *network*, *batched_observe=True*, *batching_capacity=1000*, *scope='naf'*, *device=None*, *saver=None*, *summarizer=None*, *execution=None*, *variable_noise=None*, *states_preprocessing=None*, *actions_exploration=None*, *reward_preprocessing=None*, *update_mode=None*, *memory=None*, *optimizer=None*, *discount=0.99*, *distributions=None*, *entropy_regularization=None*, *target_sync_frequency=10000*, *target_update_weight=1.0*, *double_q_model=False*, *huber_loss=None*)

Initializes the NAF agent.

Parameters **update_mode** – Update mode specification, with the following attributes:

Parameters

- **memory** (*spec*) – Memory specification, see core.memories module for more information (default: {type='replay', include_next_states=true, capacity=1000*batch_size}).
- **optimizer** (*spec*) – Optimizer specification, see core.optimizers module for more information (default: {type='adam', learning_rate=1e-3}).
- **target_sync_frequency** (*int*) – Target network sync frequency (default: 10000).
- **target_update_weight** (*float*) – Target network update weight (default: 1.0).
- **double_q_model** (*bool*) – Specifies whether double DQN mode is used (default: false).
- **huber_loss** (*float*) – Huber loss clipping (default: none).

act(*states*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)

Return action(s) for given state(s). States preprocessing and exploration are applied if configured accordingly.

Parameters

- **states** (*any*) – One state (usually a value tuple) or dict of states if multiple states are expected.
- **deterministic** (*bool*) – If true, no exploration and sampling is applied.
- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).
- **fetch_tensors** (*list*) – Optional String of named tensors to fetch

Returns Scalar value of the action or dict of multiple actions the agent wants to execute. (fetched_tensors) Optional dict() with named tensors fetched

close()

from_spec(*spec*, *kwargs*)

Creates an agent from a specification dict.

import_experience(*experiences*)

Imports experiences.

Parameters **experiences** –

initialize_model()

last_observation()

observe(*terminal*, *reward*)

Observe experience from the environment to learn from. Optionally pre-processes rewards Child classes should call super to get the processed reward EX: terminal, reward = super()...

Parameters

- **terminal** (*bool*) – boolean indicating if the episode terminated after the observation.
- **reward** (*float*) – scalar reward that resulted from executing the action.

reset()
Reset the agent to its initial state (e.g. on experiment start). Updates the Model's internal episode and time step counter, internal states, and resets preprocessors.

restore_model(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters

- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

save_model(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters

- **directory** (*str*) – Optional checkpoint directory.
- **append_timestep** (*bool*) – Appends the current timestep to the checkpoint file if true. If this is set to True, the load path must include the checkpoint timestep suffix. For example, if stored to models/ and set to true, the exported file will be of the form models/model.ckpt-X where X is the last timestep saved. The load path must precisely match this file name. If this option is turned off, the checkpoint will always overwrite the file specified in path and the model can always be loaded under this path.

Returns Checkpoint path were the model was saved.

set_normalized_actions(*actions*)

set_normalized_states(*states*)

should_stop()

tensorforce.agents.ppo_agent module

class tensorforce.agents.ppo_agent.**PPOAgent**(*states*, *actions*, *network*, *batched_observe=True*, *batching_capacity=1000*, *scope='ppo'*, *device=None*, *saver=None*, *summarizer=None*, *execution=None*, *variable_noise=None*, *states_preprocessing=None*, *actions_exploration=None*, *reward_preprocessing=None*, *update_mode=None*, *memory=None*, *discount=0.99*, *distributions=None*, *entropy_regularization=None*, *baseline_mode=None*, *baseline=None*, *baseline_optimizer=None*, *gae_lambda=None*, *likelihood_ratio_clipping=0.2*, *step_optimizer=None*, *subsampling_fraction=0.1*, *optimization_steps=50*)

Bases: *tensorforce.agents.learning_agent.LearningAgent*

Proximal Policy Optimization agent (Schulman et al., 2017).

__init__(*states*, *actions*, *network*, *batched_observe=True*, *batching_capacity=1000*, *scope='ppo'*, *device=None*, *saver=None*, *summarizer=None*, *execution=None*, *variable_noise=None*, *states_preprocessing=None*, *actions_exploration=None*, *reward_preprocessing=None*, *update_mode=None*, *memory=None*, *discount=0.99*, *distributions=None*, *entropy_regularization=None*, *baseline_mode=None*, *baseline=None*, *baseline_optimizer=None*, *gae_lambda=None*, *likelihood_ratio_clipping=0.2*, *step_optimizer=None*, *subsampling_fraction=0.1*, *optimization_steps=50*)

Initializes the PPO agent.

Parameters **update_mode** – Update mode specification, with the following attributes:

Parameters

- **memory** (*spec*) – Memory specification, see core.memories module for more information (default: {type='latest', include_next_states=false, capacity=1000*batch_size}).
- **optimizer** (*spec*) – PPO agent implicitly defines a multi-step subsampling optimizer.
- **baseline_mode** (*str*) – One of 'states', 'network' (default: none).
- **baseline** (*spec*) – Baseline specification, see core.baselines module for more information (default: none).
- **baseline_optimizer** (*spec*) – Baseline optimizer specification, see core.optimizers module for more information (default: none).
- **gae_lambda** (*float*) – Lambda factor for generalized advantage estimation (default: none).
- **likelihood_ratio_clipping** (*float*) – Likelihood ratio clipping for policy gradient (default: 0.2).
- **step_optimizer** (*spec*) – Step optimizer specification of implicit multi-step subsampling optimizer, see core.optimizers module for more information (default: {type='adam', learning_rate=1e-3}).
- **subsampling_fraction** (*float*) – Subsampling fraction of implicit subsampling optimizer (default: 0.1).

- **optimization_steps** (*int*) – Number of optimization steps for implicit multi-step optimizer (default: 50).

act(*states*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)
Return action(s) for given state(s). States preprocessing and exploration are applied if configured accordingly.

Parameters

- **states** (*any*) – One state (usually a value tuple) or dict of states if multiple states are expected.
- **deterministic** (*bool*) – If true, no exploration and sampling is applied.
- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).
- **fetch_tensors** (*list*) – Optional String of named tensors to fetch

Returns Scalar value of the action or dict of multiple actions the agent wants to execute. (fetched_tensors) Optional dict() with named tensors fetched

close()

from_spec(*spec*, *kwargs*)
Creates an agent from a specification dict.

import_experience(*experiences*)
Imports experiences.

Parameters **experiences** –

initialize_model()

last_observation()

observe(*terminal*, *reward*)
Observe experience from the environment to learn from. Optionally pre-processes rewards Child classes should call super to get the processed reward EX: terminal, reward = super()...

Parameters

- **terminal** (*bool*) – boolean indicating if the episode terminated after the observation.
- **reward** (*float*) – scalar reward that resulted from executing the action.

reset()
Reset the agent to its initial state (e.g. on experiment start). Updates the Model's internal episode and time step counter, internal states, and resets preprocessors.

restore_model(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters

- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

save_model(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters

- **directory** (*str*) – Optional checkpoint directory.
- **append_timestep** (*bool*) – Appends the current timestep to the checkpoint file if true. If this is set to True, the load path must include the checkpoint timestep suffix. For example, if stored to models/ and set to true, the exported file will be of the form models/model.ckpt-X where X is the last timestep saved. The load path must precisely match this file name. If this option is turned off, the checkpoint will always overwrite the file specified in path and the model can always be loaded under this path.

Returns Checkpoint path were the model was saved.

set_normalized_actions(*actions*)

set_normalized_states(*states*)

should_stop()

tensorforce.agents.random_agent module

class tensorforce.agents.random_agent.**RandomAgent**(*states*, *actions*, *batched_observe=True*, *batching_capacity=1000*, *scope='random'*, *device=None*, *saver=None*, *summarizer=None*, *distributed=None*)

Bases: *tensorforce.agents.agent.Agent*

Agent returning random action values.

__init__(*states*, *actions*, *batched_observe=True*, *batching_capacity=1000*, *scope='random'*, *device=None*, *saver=None*, *summarizer=None*, *distributed=None*)

Initializes the random agent.

Parameters

- **scope** (*str*) – TensorFlow scope (default: name of agent).
- **device** – TensorFlow device (default: none)
- **saver** – Saver specification, with the following attributes (default: none):

Parameters **summarizer** – Summarizer specification, with the following attributes (default: none):

Parameters **distributed** – Distributed specification, with the following attributes (default: none):

act(*states*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)

Return action(s) for given state(s). States preprocessing and exploration are applied if configured accordingly.

Parameters

- **states** (*any*) – One state (usually a value tuple) or dict of states if multiple states are expected.
- **deterministic** (*bool*) – If true, no exploration and sampling is applied.

- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).
- **fetch_tensors** (*list*) – Optional String of named tensors to fetch

Returns Scalar value of the action or dict of multiple actions the agent wants to execute. (fetched_tensors) Optional dict() with named tensors fetched

close()

from_spec(*spec*, *kwargs*)
Creates an agent from a specification dict.

initialize_model()

last_observation()

observe(*terminal*, *reward*)
Observe experience from the environment to learn from. Optionally pre-processes rewards Child classes should call super to get the processed reward EX: terminal, reward = super()...

Parameters

- **terminal** (*bool*) – boolean indicating if the episode terminated after the observation.
- **reward** (*float*) – scalar reward that resulted from executing the action.

reset()
Reset the agent to its initial state (e.g. on experiment start). Updates the Model's internal episode and time step counter, internal states, and resets preprocessors.

restore_model(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters

- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

save_model(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters

- **directory** (*str*) – Optional checkpoint directory.
- **append_timestep** (*bool*) – Appends the current timestep to the checkpoint file if true. If this is set to True, the load path must include the checkpoint timestep suffix. For example, if stored to models/ and set to true, the exported file will be of the form models/model.ckpt-X where X is the last timestep saved. The load path must precisely match this file name. If this option is turned off, the checkpoint will always overwrite the file specified in path and the model can always be loaded under this path.

Returns Checkpoint path were the model was saved.

set_normalized_actions(*actions*)

set_normalized_states(*states*)

should_stop()

tensorforce.agents.trpo_agent module

class tensorforce.agents.trpo_agent.**TRPOAgent**(*states*, *actions*, *network*, *batched_observe=True*, *batching_capacity=1000*, *scope='trpo'*, *device=None*, *saver=None*, *summarizer=None*, *execution=None*, *variable_noise=None*, *states_preprocessing=None*, *actions_exploration=None*, *reward_preprocessing=None*, *update_mode=None*, *memory=None*, *discount=0.99*, *distributions=None*, *entropy_regularization=None*, *baseline_mode=None*, *baseline=None*, *baseline_optimizer=None*, *gae_lambda=None*, *likelihood_ratio_clipping=None*, *learning_rate=0.001*, *cg_max_iterations=20*, *cg_damping=0.001*, *cg_unroll_loop=False*, *ls_max_iterations=10*, *ls_accept_ratio=0.9*, *ls_unroll_loop=False*)

Bases: *tensorforce.agents.learning_agent.LearningAgent*

Trust Region Policy Optimization agent (Schulman et al., 2015).

__init__(*states*, *actions*, *network*, *batched_observe=True*, *batching_capacity=1000*, *scope='trpo'*, *device=None*, *saver=None*, *summarizer=None*, *execution=None*, *variable_noise=None*, *states_preprocessing=None*, *actions_exploration=None*, *reward_preprocessing=None*, *update_mode=None*, *memory=None*, *discount=0.99*, *distributions=None*, *entropy_regularization=None*, *baseline_mode=None*, *baseline=None*, *baseline_optimizer=None*, *gae_lambda=None*, *likelihood_ratio_clipping=None*, *learning_rate=0.001*, *cg_max_iterations=20*, *cg_damping=0.001*, *cg_unroll_loop=False*, *ls_max_iterations=10*, *ls_accept_ratio=0.9*, *ls_unroll_loop=False*)

Initializes the TRPO agent.

Parameters **update_mode** – Update mode specification, with the following attributes:

Parameters

- **memory** (*spec*) – Memory specification, see core.memories module for more information (default: {type='latest', include_next_states=false, capacity=1000*batch_size}).
- **optimizer** (*spec*) – TRPO agent implicitly defines a optimized-step natural-gradient optimizer.
- **baseline_mode** (*str*) – One of 'states', 'network' (default: none).
- **baseline** (*spec*) – Baseline specification, see core.baselines module for more information (default: none).
- **baseline_optimizer** (*spec*) – Baseline optimizer specification, see core.optimizers module for more information (default: none).
- **gae_lambda** (*float*) – Lambda factor for generalized advantage estimation (default: none).

- **likelihood_ratio_clipping** (*float*) – Likelihood ratio clipping for policy gradient (default: none).
- **learning_rate** (*float*) – Learning rate of natural-gradient optimizer (default: 1e-3).
- **cg_max_iterations** (*int*) – Conjugate-gradient max iterations (default: 20).
- **cg_damping** (*float*) – Conjugate-gradient damping (default: 1e-3).
- **cg_unroll_loop** (*bool*) – Conjugate-gradient unroll loop (default: false).
- **ls_max_iterations** (*int*) – Line-search max iterations (default: 10).
- **ls_accept_ratio** (*float*) – Line-search accept ratio (default: 0.9).
- **ls_unroll_loop** (*bool*) – Line-search unroll loop (default: false).

act(*states*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)
Return action(s) for given state(s). States preprocessing and exploration are applied if configured accordingly.

Parameters

- **states** (*any*) – One state (usually a value tuple) or dict of states if multiple states are expected.
- **deterministic** (*bool*) – If true, no exploration and sampling is applied.
- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).
- **fetch_tensors** (*list*) – Optional String of named tensors to fetch

Returns Scalar value of the action or dict of multiple actions the agent wants to execute. (fetched_tensors) Optional dict() with named tensors fetched

close()

from_spec(*spec*, *kwargs*)
Creates an agent from a specification dict.

import_experience(*experiences*)
Imports experiences.

Parameters **experiences** –

initialize_model()

last_observation()

observe(*terminal*, *reward*)
Observe experience from the environment to learn from. Optionally pre-processes rewards Child classes should call super to get the processed reward EX: terminal, reward = super()...

Parameters

- **terminal** (*bool*) – boolean indicating if the episode terminated after the observation.
- **reward** (*float*) – scalar reward that resulted from executing the action.

reset()
Reset the agent to its initial state (e.g. on experiment start). Updates the Model's internal episode and time step counter, internal states, and resets preprocessors.

restore_model(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters

- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

save_model(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters

- **directory** (*str*) – Optional checkpoint directory.
- **append_timestep** (*bool*) – Appends the current timestep to the checkpoint file if true. If this is set to True, the load path must include the checkpoint timestep suffix. For example, if stored to models/ and set to true, the exported file will be of the form models/model.ckpt-X where X is the last timestep saved. The load path must precisely match this file name. If this option is turned off, the checkpoint will always overwrite the file specified in path and the model can always be loaded under this path.

Returns Checkpoint path were the model was saved.

set_normalized_actions(*actions*)

set_normalized_states(*states*)

should_stop()

tensorforce.agents.vpg_agent module

class tensorforce.agents.vpg_agent.**VPGAgent**(*states*, *actions*, *network*, *batched_observe=True*, *batching_capacity=1000*, *scope='vpg'*, *device=None*, *saver=None*, *summarizer=None*, *execution=None*, *variable_noise=None*, *states_preprocessing=None*, *actions_exploration=None*, *reward_preprocessing=None*, *update_mode=None*, *memory=None*, *optimizer=None*, *discount=0.99*, *distributions=None*, *entropy_regularization=None*, *baseline_mode=None*, *baseline=None*, *baseline_optimizer=None*, *gae_lambda=None*)
Bases: *tensorforce.agents.learning_agent.LearningAgent*

Vanilla policy gradient agent (Williams, 1992)).

__init__(*states*, *actions*, *network*, *batched_observe=True*, *batching_capacity=1000*, *scope='vpg'*, *device=None*, *saver=None*, *summarizer=None*, *execution=None*, *variable_noise=None*, *states_preprocessing=None*, *actions_exploration=None*, *reward_preprocessing=None*, *update_mode=None*, *memory=None*, *optimizer=None*, *discount=0.99*, *distributions=None*, *entropy_regularization=None*, *baseline_mode=None*, *baseline=None*, *baseline_optimizer=None*, *gae_lambda=None*)
Initializes the VPG agent.

Parameters **update_mode** – Update mode specification, with the following attributes:

Parameters

- **memory** (*spec*) – Memory specification, see core.memories module for more information (default: {type='latest', include_next_states=false, capacity=1000*batch_size}).
- **optimizer** (*spec*) – Optimizer specification, see core.optimizers module for more information (default: {type='adam', learning_rate=1e-3}).
- **baseline_mode** (*str*) – One of 'states', 'network' (default: none).
- **baseline** (*spec*) – Baseline specification, see core.baselines module for more information (default: none).
- **baseline_optimizer** (*spec*) – Baseline optimizer specification, see core.optimizers module for more information (default: none).
- **gae_lambda** (*float*) – Lambda factor for generalized advantage estimation (default: none).

act(*states*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)
Return action(s) for given state(s). States preprocessing and exploration are applied if configured accordingly.

Parameters

- **states** (*any*) – One state (usually a value tuple) or dict of states if multiple states are expected.
- **deterministic** (*bool*) – If true, no exploration and sampling is applied.
- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).
- **fetch_tensors** (*list*) – Optional String of named tensors to fetch

Returns Scalar value of the action or dict of multiple actions the agent wants to execute. (fetched_tensors) Optional dict() with named tensors fetched

close()

from_spec(*spec*, *kwargs*)
Creates an agent from a specification dict.

import_experience(*experiences*)
Imports experiences.

Parameters **experiences** –

initialize_model()

last_observation()

observe(*terminal*, *reward*)
Observe experience from the environment to learn from. Optionally pre-processes rewards Child classes should call super to get the processed reward EX: terminal, reward = super()...

Parameters

- **terminal** (*bool*) – boolean indicating if the episode terminated after the observation.
- **reward** (*float*) – scalar reward that resulted from executing the action.

reset()
Reset the agent to its initial state (e.g. on experiment start). Updates the Model's internal episode and time step counter, internal states, and resets preprocessors.

restore_model(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters

- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

save_model(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters

- **directory** (*str*) – Optional checkpoint directory.
- **append_timestep** (*bool*) – Appends the current timestep to the checkpoint file if true. If this is set to True, the load path must include the checkpoint timestep suffix. For example, if stored to models/ and set to true, the exported file will be of the form models/model.ckpt-X where X is the last timestep saved. The load path must precisely match this file name. If this option is turned off, the checkpoint will always overwrite the file specified in path and the model can always be loaded under this path.

Returns Checkpoint path were the model was saved.

set_normalized_actions(*actions*)

set_normalized_states(*states*)

should_stop()

Module contents

class tensorforce.agents.**Agent**(*states*, *actions*, *batched_observe=True*, *batching_capacity=1000*)
Bases: object

Base class for TensorForce agents.

__init__(*states*, *actions*, *batched_observe=True*, *batching_capacity=1000*)
Initializes the agent.

Parameters **states** – States specification, with the following attributes (required):

Parameters **actions** – Actions specification, with the following attributes (required):

Parameters

- **batched_observe** (*bool*) – Specifies whether calls to model.observe() are batched, for improved performance (default: true).
- **batching_capacity** (*int*) – Batching capacity of agent and model (default: 1000).

act(*states*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)
Return action(s) for given state(s). States preprocessing and exploration are applied if configured accordingly.

Parameters

- **states** (*any*) – One state (usually a value tuple) or dict of states if multiple states are expected.
- **deterministic** (*bool*) – If true, no exploration and sampling is applied.
- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).
- **fetch_tensors** (*list*) – Optional String of named tensors to fetch

Returns Scalar value of the action or dict of multiple actions the agent wants to execute. (fetched_tensors) Optional dict() with named tensors fetched

close()

static from_spec(*spec*, *kwargs*)
Creates an agent from a specification dict.

initialize_model()
Creates the model for the respective agent based on specifications given by user. This is a separate call after constructing the agent because the agent constructor has to perform a number of checks on the specs first, sometimes adjusting them e.g. by converting to a dict.

last_observation()

observe(*terminal*, *reward*)
Observe experience from the environment to learn from. Optionally pre-processes rewards Child classes should call super to get the processed reward EX: terminal, reward = super()...

Parameters

- **terminal** (*bool*) – boolean indicating if the episode terminated after the observation.
- **reward** (*float*) – scalar reward that resulted from executing the action.

reset()
Reset the agent to its initial state (e.g. on experiment start). Updates the Model's internal episode and time step counter, internal states, and resets preprocessors.

restore_model(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters

- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

save_model(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters

- **directory** (*str*) – Optional checkpoint directory.

- **append_timestep** (*bool*) – Appends the current timestep to the checkpoint file if true. If this is set to True, the load path must include the checkpoint timestep suffix. For example, if stored to models/ and set to true, the exported file will be of the form models/model.ckpt-X where X is the last timestep saved. The load path must precisely match this file name. If this option is turned off, the checkpoint will always overwrite the file specified in path and the model can always be loaded under this path.

Returns Checkpoint path were the model was saved.

set_normalized_actions(*actions*)

set_normalized_states(*states*)

should_stop()

class tensorforce.agents.**ConstantAgent**(*states*, *actions*, *action_values*, *batched_observe=True*, *batching_capacity=1000*, *scope='constant'*, *device=None*, *saver=None*, *summarizer=None*, *distributed=None*)

Bases: *tensorforce.agents.agent.Agent*

Agent returning constant action values.

__init__(*states*, *actions*, *action_values*, *batched_observe=True*, *batching_capacity=1000*, *scope='constant'*, *device=None*, *saver=None*, *summarizer=None*, *distributed=None*)

Initializes the constant agent.

Parameters

- **action_values** (*value, or dict of values*) – Action values returned by the agent (required).
- **scope** (*str*) – TensorFlow scope (default: name of agent).
- **device** – TensorFlow device (default: none)
- **saver** – Saver specification, with the following attributes (default: none):

Parameters **summarizer** – Summarizer specification, with the following attributes (default: none):

Parameters **distributed** – Distributed specification, with the following attributes (default: none):

act(*states*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)

Return action(s) for given state(s). States preprocessing and exploration are applied if configured accordingly.

Parameters

- **states** (*any*) – One state (usually a value tuple) or dict of states if multiple states are expected.
- **deterministic** (*bool*) – If true, no exploration and sampling is applied.
- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).
- **fetch_tensors** (*list*) – Optional String of named tensors to fetch

Returns Scalar value of the action or dict of multiple actions the agent wants to execute. (fetched_tensors) Optional dict() with named tensors fetched

close()

from_spec(*spec*, *kwargs*)
Creates an agent from a specification dict.

initialize_model()

last_observation()

observe(*terminal*, *reward*)
Observe experience from the environment to learn from. Optionally pre-processes rewards Child classes should call super to get the processed reward EX: terminal, reward = super()...

Parameters

- **terminal** (*bool*) – boolean indicating if the episode terminated after the observation.
- **reward** (*float*) – scalar reward that resulted from executing the action.

reset()
Reset the agent to its initial state (e.g. on experiment start). Updates the Model's internal episode and time step counter, internal states, and resets preprocessors.

restore_model(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters

- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

save_model(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters

- **directory** (*str*) – Optional checkpoint directory.
- **append_timestep** (*bool*) – Appends the current timestep to the checkpoint file if true. If this is set to True, the load path must include the checkpoint timestep suffix. For example, if stored to models/ and set to true, the exported file will be of the form models/model.ckpt-X where X is the last timestep saved. The load path must precisely match this file name. If this option is turned off, the checkpoint will always overwrite the file specified in path and the model can always be loaded under this path.

Returns Checkpoint path were the model was saved.

set_normalized_actions(*actions*)

set_normalized_states(*states*)

should_stop()

class tensorforce.agents.**RandomAgent**(*states*, *actions*, *batched_observe=True*, *batching_capacity=1000*, *scope='random'*, *device=None*, *saver=None*, *summarizer=None*, *distributed=None*)
Bases: *tensorforce.agents.agent.Agent*

Agent returning random action values.

__init__(*states*, *actions*, *batched_observe=True*, *batching_capacity=1000*, *scope='random'*, *device=None*, *saver=None*, *summarizer=None*, *distributed=None*)
Initializes the random agent.

Parameters

- **scope** (*str*) – TensorFlow scope (default: name of agent).
- **device** – TensorFlow device (default: none)
- **saver** – Saver specification, with the following attributes (default: none):

Parameters **summarizer** – Summarizer specification, with the following attributes (default: none):

Parameters **distributed** – Distributed specification, with the following attributes (default: none):

act(*states*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)
Return action(s) for given state(s). States preprocessing and exploration are applied if configured accordingly.

Parameters

- **states** (*any*) – One state (usually a value tuple) or dict of states if multiple states are expected.
- **deterministic** (*bool*) – If true, no exploration and sampling is applied.
- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).
- **fetch_tensors** (*list*) – Optional String of named tensors to fetch

Returns Scalar value of the action or dict of multiple actions the agent wants to execute. (fetched_tensors) Optional dict() with named tensors fetched

close()

from_spec(*spec*, *kwargs*)
Creates an agent from a specification dict.

initialize_model()

last_observation()

observe(*terminal*, *reward*)
Observe experience from the environment to learn from. Optionally pre-processes rewards Child classes should call super to get the processed reward EX: terminal, reward = super()...

Parameters

- **terminal** (*bool*) – boolean indicating if the episode terminated after the observation.
- **reward** (*float*) – scalar reward that resulted from executing the action.

reset()
Reset the agent to its initial state (e.g. on experiment start). Updates the Model's internal episode and time step counter, internal states, and resets preprocessors.

restore_model(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters

- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

save_model(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters

- **directory** (*str*) – Optional checkpoint directory.
- **append_timestep** (*bool*) – Appends the current timestep to the checkpoint file if true. If this is set to True, the load path must include the checkpoint timestep suffix. For example, if stored to models/ and set to true, the exported file will be of the form models/model.ckpt-X where X is the last timestep saved. The load path must precisely match this file name. If this option is turned off, the checkpoint will always overwrite the file specified in path and the model can always be loaded under this path.

Returns Checkpoint path were the model was saved.

set_normalized_actions(*actions*)

set_normalized_states(*states*)

should_stop()

class tensorforce.agents.**LearningAgent**(*states*, *actions*, *network*, *update_mode*, *memory*, *optimizer*, *batched_observe=True*, *batching_capacity=1000*, *scope='learning-agent'*, *device=None*, *saver=None*, *summarizer=None*, *execution=None*, *variable_noise=None*, *states_preprocessing=None*, *actions_exploration=None*, *reward_preprocessing=None*, *discount=0.99*, *distributions=None*, *entropy_regularization=None*)
Bases: *tensorforce.agents.agent.Agent*

Base class for learning agents, using as model a subclass of MemoryModel and DistributionModel.

__init__(*states*, *actions*, *network*, *update_mode*, *memory*, *optimizer*, *batched_observe=True*, *batching_capacity=1000*, *scope='learning-agent'*, *device=None*, *saver=None*, *summarizer=None*, *execution=None*, *variable_noise=None*, *states_preprocessing=None*, *actions_exploration=None*, *reward_preprocessing=None*, *discount=0.99*, *distributions=None*, *entropy_regularization=None*)
Initializes the learning agent.

Parameters **update_mode** – Update mode specification, with the following attributes (required):

Parameters

- **memory** (*spec*) – Memory specification, see core.memories module for more information (required).

- **optimizer** (*spec*) – Optimizer specification, see core.optimizers module for more information (required).
- **network** (*spec*) – Network specification, usually a list of layer specifications, see core.networks module for more information (required).
- **scope** (*str*) – TensorFlow scope (default: name of agent).
- **device** – TensorFlow device (default: none)
- **saver** – Saver specification, with the following attributes (default: none):

Parameters **summarizer** – Summarizer specification, with the following attributes (default: none):

Parameters **execution** – Distributed specification, with the following attributes (default: none):

Parameters

- **variable_noise** (*float*) – Standard deviation of variable noise (default: none).
- **states_preprocessing** (*spec, or dict of specs*) – States preprocessing specification, see core.preprocessors module for more information (default: none)
- **actions_exploration** (*spec, or dict of specs*) – Actions exploration specification, see core.explorations module for more information (default: none).
- **reward_preprocessing** (*spec*) – Reward preprocessing specification, see core.preprocessors module for more information (default: none).
- **discount** (*float*) – Discount factor for future rewards (default: 0.99).
- **distributions** (*spec / dict of specs*) – Distributions specifications, see core.distributions module for more information (default: none).
- **entropy_regularization** (*float*) – Entropy regularization weight (default: none).

act(*states*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)
Return action(s) for given state(s). States preprocessing and exploration are applied if configured accordingly.

Parameters

- **states** (*any*) – One state (usually a value tuple) or dict of states if multiple states are expected.
- **deterministic** (*bool*) – If true, no exploration and sampling is applied.
- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).
- **fetch_tensors** (*list*) – Optional String of named tensors to fetch

Returns Scalar value of the action or dict of multiple actions the agent wants to execute. (fetched_tensors) Optional dict() with named tensors fetched

close()

from_spec(*spec*, *kwargs*)
Creates an agent from a specification dict.

import_experience(*experiences*)
Imports experiences.

Parameters **experiences** –

initialize_model()
Creates the model for the respective agent based on specifications given by user. This is a separate call after constructing the agent because the agent constructor has to perform a number of checks on the specs first, sometimes adjusting them e.g. by converting to a dict.

last_observation()

observe(*terminal*, *reward*)
Observe experience from the environment to learn from. Optionally pre-processes rewards Child classes should call super to get the processed reward EX: terminal, reward = super()...

Parameters

- **terminal** (*bool*) – boolean indicating if the episode terminated after the observation.
- **reward** (*float*) – scalar reward that resulted from executing the action.

reset()
Reset the agent to its initial state (e.g. on experiment start). Updates the Model's internal episode and time step counter, internal states, and resets preprocessors.

restore_model(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters

- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

save_model(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters

- **directory** (*str*) – Optional checkpoint directory.
- **append_timestep** (*bool*) – Appends the current timestep to the checkpoint file if true. If this is set to True, the load path must include the checkpoint timestep suffix. For example, if stored to models/ and set to true, the exported file will be of the form models/model.ckpt-X where X is the last timestep saved. The load path must precisely match this file name. If this option is turned off, the checkpoint will always overwrite the file specified in path and the model can always be loaded under this path.

Returns Checkpoint path were the model was saved.

set_normalized_actions(*actions*)

set_normalized_states(*states*)

should_stop()

class tensorforce.agents.**DQFDAgent**(*states*, *actions*, *network*, *batched_observe=True*, *batching_capacity=1000*, *scope='dqfd'*, *device=None*, *saver=None*, *summarizer=None*, *execution=None*, *variable_noise=None*, *states_preprocessing=None*, *actions_exploration=None*, *reward_preprocessing=None*, *update_mode=None*, *memory=None*, *optimizer=None*, *discount=0.99*, *distributions=None*, *entropy_regularization=None*, *target_sync_frequency=10000*, *target_update_weight=1.0*, *huber_loss=None*, *expert_margin=0.5*, *supervised_weight=0.1*, *demo_memory_capacity=10000*, *demo_sampling_ratio=0.2*)

Bases: *tensorforce.agents.learning_agent.LearningAgent*

Deep Q-learning from demonstration agent (Hester et al., 2017).

__init__(*states*, *actions*, *network*, *batched_observe=True*, *batching_capacity=1000*, *scope='dqfd'*, *device=None*, *saver=None*, *summarizer=None*, *execution=None*, *variable_noise=None*, *states_preprocessing=None*, *actions_exploration=None*, *reward_preprocessing=None*, *update_mode=None*, *memory=None*, *optimizer=None*, *discount=0.99*, *distributions=None*, *entropy_regularization=None*, *target_sync_frequency=10000*, *target_update_weight=1.0*, *huber_loss=None*, *expert_margin=0.5*, *supervised_weight=0.1*, *demo_memory_capacity=10000*, *demo_sampling_ratio=0.2*)

Initializes the DQFD agent.

Parameters **update_mode** – Update mode specification, with the following attributes:

Parameters

- **memory** (*spec*) – Memory specification, see core.memories module for more information (default: {type='replay', include_next_states=true, capacity=1000*batch_size}).
- **optimizer** (*spec*) – Optimizer specification, see core.optimizers module for more information (default: {type='adam', learning_rate=1e-3}).
- **target_sync_frequency** (*int*) – Target network sync frequency (default: 10000).
- **target_update_weight** (*float*) – Target network update weight (default: 1.0).
- **huber_loss** (*float*) – Huber loss clipping (default: none).
- **expert_margin** (*float*) – Enforced supervised margin between expert action Q-value and other Q-values (default: 0.5).
- **supervised_weight** (*float*) – Weight of supervised loss term (default: 0.1).
- **demo_memory_capacity** (*int*) – Capacity of expert demonstration memory (default: 10000).
- **demo_sampling_ratio** (*float*) – Runtime sampling ratio of expert data (default: 0.2).

act(*states*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)

Return action(s) for given state(s). States preprocessing and exploration are applied if configured accordingly.

Parameters

- **states** (*any*) – One state (usually a value tuple) or dict of states if multiple states are expected.
- **deterministic** (*bool*) – If true, no exploration and sampling is applied.

- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).
- **fetch_tensors** (*list*) – Optional String of named tensors to fetch

Returns Scalar value of the action or dict of multiple actions the agent wants to execute. (fetched_tensors) Optional dict() with named tensors fetched

close()

from_spec(*spec*, *kwargs*)
Creates an agent from a specification dict.

import_demonstrations(*demonstrations*)
Imports demonstrations, i.e. expert observations. Note that for large numbers of observations, set_demonstrations is more appropriate, which directly sets memory contents to an array an expects a different layout.

Parameters **demonstrations** – List of observation dicts

import_experience(*experiences*)
Imports experiences.

Parameters **experiences** –

initialize_model()

last_observation()

observe(*terminal*, *reward*)
Observe experience from the environment to learn from. Optionally pre-processes rewards Child classes should call super to get the processed reward EX: terminal, reward = super()...

Parameters

- **terminal** (*bool*) – boolean indicating if the episode terminated after the observation.
- **reward** (*float*) – scalar reward that resulted from executing the action.

pretrain(*steps*)
Computes pre-train updates.

Parameters **steps** – Number of updates to execute.

reset()
Reset the agent to its initial state (e.g. on experiment start). Updates the Model's internal episode and time step counter, internal states, and resets preprocessors.

restore_model(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters

- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

save_model(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters

- **directory** (*str*) – Optional checkpoint directory.

- **append_timestep** (*bool*) – Appends the current timestep to the checkpoint file if true. If this is set to True, the load path must include the checkpoint timestep suffix. For example, if stored to models/ and set to true, the exported file will be of the form models/model.ckpt-X where X is the last timestep saved. The load path must precisely match this file name. If this option is turned off, the checkpoint will always overwrite the file specified in path and the model can always be loaded under this path.

Returns Checkpoint path were the model was saved.

set_normalized_actions(*actions*)

set_normalized_states(*states*)

should_stop()

class tensorforce.agents.**DQNAgent**(*states*, *actions*, *network*, *batched_observe=True*, *batching_capacity=1000*, *scope='dqn'*, *device=None*, *saver=None*, *summarizer=None*, *execution=None*, *variable_noise=None*, *states_preprocessing=None*, *actions_exploration=None*, *reward_preprocessing=None*, *update_mode=None*, *memory=None*, *optimizer=None*, *discount=0.99*, *distributions=None*, *entropy_regularization=None*, *target_sync_frequency=10000*, *target_update_weight=1.0*, *double_q_model=False*, *huber_loss=None*)

Bases: *tensorforce.agents.learning_agent.LearningAgent*

Deep Q-Network agent (Mnih et al., 2015).

__init__(*states*, *actions*, *network*, *batched_observe=True*, *batching_capacity=1000*, *scope='dqn'*, *device=None*, *saver=None*, *summarizer=None*, *execution=None*, *variable_noise=None*, *states_preprocessing=None*, *actions_exploration=None*, *reward_preprocessing=None*, *update_mode=None*, *memory=None*, *optimizer=None*, *discount=0.99*, *distributions=None*, *entropy_regularization=None*, *target_sync_frequency=10000*, *target_update_weight=1.0*, *double_q_model=False*, *huber_loss=None*)

Initializes the DQN agent.

Parameters **update_mode** – Update mode specification, with the following attributes:

Parameters

- **memory** (*spec*) – Memory specification, see core.memories module for more information (default: {type='replay', include_next_states=true, capacity=1000*batch_size}).
- **optimizer** (*spec*) – Optimizer specification, see core.optimizers module for more information (default: {type='adam', learning_rate=1e-3}).
- **target_sync_frequency** (*int*) – Target network sync frequency (default: 10000).
- **target_update_weight** (*float*) – Target network update weight (default: 1.0).
- **double_q_model** (*bool*) – Specifies whether double DQN mode is used (default: false).
- **huber_loss** (*float*) – Huber loss clipping (default: none).

act(*states*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)

Return action(s) for given state(s). States preprocessing and exploration are applied if configured accordingly.

Parameters

- **states** (*any*) – One state (usually a value tuple) or dict of states if multiple states are expected.
- **deterministic** (*bool*) – If true, no exploration and sampling is applied.
- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).
- **fetch_tensors** (*list*) – Optional String of named tensors to fetch

Returns Scalar value of the action or dict of multiple actions the agent wants to execute. (fetched_tensors) Optional dict() with named tensors fetched

close()

from_spec(*spec*, *kwargs*)
Creates an agent from a specification dict.

import_experience(*experiences*)
Imports experiences.

Parameters **experiences** –

initialize_model()

last_observation()

observe(*terminal*, *reward*)
Observe experience from the environment to learn from. Optionally pre-processes rewards Child classes should call super to get the processed reward EX: terminal, reward = super()...

Parameters

- **terminal** (*bool*) – boolean indicating if the episode terminated after the observation.
- **reward** (*float*) – scalar reward that resulted from executing the action.

reset()
Reset the agent to its initial state (e.g. on experiment start). Updates the Model's internal episode and time step counter, internal states, and resets preprocessors.

restore_model(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters

- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

save_model(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters

- **directory** (*str*) – Optional checkpoint directory.
- **append_timestep** (*bool*) – Appends the current timestep to the checkpoint file if true. If this is set to True, the load path must include the checkpoint timestep suffix. For example, if stored to models/ and set to true, the exported file will be of the form models/model.ckpt-X where X is the last timestep saved. The load path must precisely

match this file name. If this option is turned off, the checkpoint will always overwrite the file specified in path and the model can always be loaded under this path.

Returns Checkpoint path were the model was saved.

set_normalized_actions(*actions*)

set_normalized_states(*states*)

should_stop()

class tensorforce.agents.**DQNNstepAgent**(*states*, *actions*, *network*, *batched_observe=True*, *batching_capacity=1000*, *scope='dqn-nstep'*, *device=None*, *saver=None*, *summarizer=None*, *execution=None*, *variable_noise=None*, *states_preprocessing=None*, *actions_exploration=None*, *reward_preprocessing=None*, *update_mode=None*, *memory=None*, *optimizer=None*, *discount=0.99*, *distributions=None*, *entropy_regularization=None*, *target_sync_frequency=10000*, *target_update_weight=1.0*, *double_q_model=False*, *huber_loss=None*)

Bases: *tensorforce.agents.learning_agent.LearningAgent*

DQN n-step agent.

__init__(*states*, *actions*, *network*, *batched_observe=True*, *batching_capacity=1000*, *scope='dqn-nstep'*, *device=None*, *saver=None*, *summarizer=None*, *execution=None*, *variable_noise=None*, *states_preprocessing=None*, *actions_exploration=None*, *reward_preprocessing=None*, *update_mode=None*, *memory=None*, *optimizer=None*, *discount=0.99*, *distributions=None*, *entropy_regularization=None*, *target_sync_frequency=10000*, *target_update_weight=1.0*, *double_q_model=False*, *huber_loss=None*)

Initializes the DQN n-step agent.

Parameters **update_mode** – Update mode specification, with the following attributes:

Parameters

- **memory** (*spec*) – Memory specification, see core.memories module for more information (default: {type='latest', include_next_states=true, capacity=1000*batch_size}).
- **optimizer** (*spec*) – Optimizer specification, see core.optimizers module for more information (default: {type='adam', learning_rate=1e-3}).
- **target_sync_frequency** (*int*) – Target network sync frequency (default: 10000).
- **target_update_weight** (*float*) – Target network update weight (default: 1.0).
- **double_q_model** (*bool*) – Specifies whether double DQN mode is used (default: false).
- **huber_loss** (*float*) – Huber loss clipping (default: none).

act(*states*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)

Return action(s) for given state(s). States preprocessing and exploration are applied if configured accordingly.

Parameters

- **states** (*any*) – One state (usually a value tuple) or dict of states if multiple states are expected.
- **deterministic** (*bool*) – If true, no exploration and sampling is applied.
- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).
- **fetch_tensors** (*list*) – Optional String of named tensors to fetch

Returns Scalar value of the action or dict of multiple actions the agent wants to execute. (fetched_tensors) Optional dict() with named tensors fetched

close()

from_spec(*spec*, *kwargs*)
Creates an agent from a specification dict.

import_experience(*experiences*)
Imports experiences.

Parameters **experiences** –

initialize_model()

last_observation()

observe(*terminal*, *reward*)
Observe experience from the environment to learn from. Optionally pre-processes rewards Child classes should call super to get the processed reward EX: terminal, reward = super()...

Parameters

- **terminal** (*bool*) – boolean indicating if the episode terminated after the observation.
- **reward** (*float*) – scalar reward that resulted from executing the action.

reset()
Reset the agent to its initial state (e.g. on experiment start). Updates the Model's internal episode and time step counter, internal states, and resets preprocessors.

restore_model(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters

- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

save_model(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters

- **directory** (*str*) – Optional checkpoint directory.
- **append_timestep** (*bool*) – Appends the current timestep to the checkpoint file if true. If this is set to True, the load path must include the checkpoint timestep suffix. For example, if stored to models/ and set to true, the exported file will be of the form models/model.ckpt-X where X is the last timestep saved. The load path must precisely

match this file name. If this option is turned off, the checkpoint will always overwrite the file specified in path and the model can always be loaded under this path.

Returns Checkpoint path were the model was saved.

set_normalized_actions(*actions*)

set_normalized_states(*states*)

should_stop()

class tensorforce.agents.**NAFAgent**(*states*, *actions*, *network*, *batched_observe=True*, *batching_capacity=1000*, *scope='naf'*, *device=None*, *saver=None*, *summarizer=None*, *execution=None*, *variable_noise=None*, *states_preprocessing=None*, *actions_exploration=None*, *reward_preprocessing=None*, *update_mode=None*, *memory=None*, *optimizer=None*, *discount=0.99*, *distributions=None*, *entropy_regularization=None*, *target_sync_frequency=10000*, *target_update_weight=1.0*, *double_q_model=False*, *huber_loss=None*)

Bases: *tensorforce.agents.learning_agent.LearningAgent*

Normalized Advantage Function agent (Gu et al., 2016).

__init__(*states*, *actions*, *network*, *batched_observe=True*, *batching_capacity=1000*, *scope='naf'*, *device=None*, *saver=None*, *summarizer=None*, *execution=None*, *variable_noise=None*, *states_preprocessing=None*, *actions_exploration=None*, *reward_preprocessing=None*, *update_mode=None*, *memory=None*, *optimizer=None*, *discount=0.99*, *distributions=None*, *entropy_regularization=None*, *target_sync_frequency=10000*, *target_update_weight=1.0*, *double_q_model=False*, *huber_loss=None*)

Initializes the NAF agent.

Parameters **update_mode** – Update mode specification, with the following attributes:

Parameters

- **memory** (*spec*) – Memory specification, see core.memories module for more information (default: {type='replay', include_next_states=true, capacity=1000*batch_size}).
- **optimizer** (*spec*) – Optimizer specification, see core.optimizers module for more information (default: {type='adam', learning_rate=1e-3}).
- **target_sync_frequency** (*int*) – Target network sync frequency (default: 10000).
- **target_update_weight** (*float*) – Target network update weight (default: 1.0).
- **double_q_model** (*bool*) – Specifies whether double DQN mode is used (default: false).
- **huber_loss** (*float*) – Huber loss clipping (default: none).

act(*states*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)

Return action(s) for given state(s). States preprocessing and exploration are applied if configured accordingly.

Parameters

- **states** (*any*) – One state (usually a value tuple) or dict of states if multiple states are expected.
- **deterministic** (*bool*) – If true, no exploration and sampling is applied.

- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).
- **fetch_tensors** (*list*) – Optional String of named tensors to fetch

Returns Scalar value of the action or dict of multiple actions the agent wants to execute. (fetched_tensors) Optional dict() with named tensors fetched

close()

from_spec(*spec*, *kwargs*)
Creates an agent from a specification dict.

import_experience(*experiences*)
Imports experiences.

Parameters **experiences** –

initialize_model()

last_observation()

observe(*terminal*, *reward*)
Observe experience from the environment to learn from. Optionally pre-processes rewards Child classes should call super to get the processed reward EX: terminal, reward = super()...

Parameters

- **terminal** (*bool*) – boolean indicating if the episode terminated after the observation.
- **reward** (*float*) – scalar reward that resulted from executing the action.

reset()
Reset the agent to its initial state (e.g. on experiment start). Updates the Model's internal episode and time step counter, internal states, and resets preprocessors.

restore_model(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters

- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

save_model(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters

- **directory** (*str*) – Optional checkpoint directory.
- **append_timestep** (*bool*) – Appends the current timestep to the checkpoint file if true. If this is set to True, the load path must include the checkpoint timestep suffix. For example, if stored to models/ and set to true, the exported file will be of the form models/model.ckpt-X where X is the last timestep saved. The load path must precisely match this file name. If this option is turned off, the checkpoint will always overwrite the file specified in path and the model can always be loaded under this path.

Returns Checkpoint path were the model was saved.

set_normalized_actions(*actions*)

set_normalized_states(*states*)

should_stop()

class tensorforce.agents.**PPOAgent**(*states*, *actions*, *network*, *batched_observe=True*, *batching_capacity=1000*, *scope='ppo'*, *device=None*, *saver=None*, *summarizer=None*, *execution=None*, *variable_noise=None*, *states_preprocessing=None*, *actions_exploration=None*, *reward_preprocessing=None*, *update_mode=None*, *memory=None*, *discount=0.99*, *distributions=None*, *entropy_regularization=None*, *baseline_mode=None*, *baseline=None*, *baseline_optimizer=None*, *gae_lambda=None*, *likelihood_ratio_clipping=0.2*, *step_optimizer=None*, *subsampling_fraction=0.1*, *optimization_steps=50*)

Bases: *tensorforce.agents.learning_agent.LearningAgent*

Proximal Policy Optimization agent (Schulman et al., 2017).

__init__(*states*, *actions*, *network*, *batched_observe=True*, *batching_capacity=1000*, *scope='ppo'*, *device=None*, *saver=None*, *summarizer=None*, *execution=None*, *variable_noise=None*, *states_preprocessing=None*, *actions_exploration=None*, *reward_preprocessing=None*, *update_mode=None*, *memory=None*, *discount=0.99*, *distributions=None*, *entropy_regularization=None*, *baseline_mode=None*, *baseline=None*, *baseline_optimizer=None*, *gae_lambda=None*, *likelihood_ratio_clipping=0.2*, *step_optimizer=None*, *subsampling_fraction=0.1*, *optimization_steps=50*)

Initializes the PPO agent.

Parameters **update_mode** – Update mode specification, with the following attributes:

Parameters

- **memory** (*spec*) – Memory specification, see core.memories module for more information (default: {type='latest', include_next_states=false, capacity=1000*batch_size}).
- **optimizer** (*spec*) – PPO agent implicitly defines a multi-step subsampling optimizer.
- **baseline_mode** (*str*) – One of 'states', 'network' (default: none).
- **baseline** (*spec*) – Baseline specification, see core.baselines module for more information (default: none).
- **baseline_optimizer** (*spec*) – Baseline optimizer specification, see core.optimizers module for more information (default: none).
- **gae_lambda** (*float*) – Lambda factor for generalized advantage estimation (default: none).
- **likelihood_ratio_clipping** (*float*) – Likelihood ratio clipping for policy gradient (default: 0.2).
- **step_optimizer** (*spec*) – Step optimizer specification of implicit multi-step subsampling optimizer, see core.optimizers module for more information (default: {type='adam', learning_rate=1e-3}).
- **subsampling_fraction** (*float*) – Subsampling fraction of implicit subsampling optimizer (default: 0.1).
- **optimization_steps** (*int*) – Number of optimization steps for implicit multi-step optimizer (default: 50).

act(*states*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)
Return action(s) for given state(s). States preprocessing and exploration are applied if configured accordingly.

Parameters

- **states** (*any*) – One state (usually a value tuple) or dict of states if multiple states are expected.
- **deterministic** (*bool*) – If true, no exploration and sampling is applied.
- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).
- **fetch_tensors** (*list*) – Optional String of named tensors to fetch

Returns Scalar value of the action or dict of multiple actions the agent wants to execute. (fetched_tensors) Optional dict() with named tensors fetched

close()

from_spec(*spec*, *kwargs*)
Creates an agent from a specification dict.

import_experience(*experiences*)
Imports experiences.

Parameters **experiences** –

initialize_model()

last_observation()

observe(*terminal*, *reward*)
Observe experience from the environment to learn from. Optionally pre-processes rewards Child classes should call super to get the processed reward EX: terminal, reward = super()...

Parameters

- **terminal** (*bool*) – boolean indicating if the episode terminated after the observation.
- **reward** (*float*) – scalar reward that resulted from executing the action.

reset()
Reset the agent to its initial state (e.g. on experiment start). Updates the Model's internal episode and time step counter, internal states, and resets preprocessors.

restore_model(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters

- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

save_model(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters

- **directory** (*str*) – Optional checkpoint directory.

- **append_timestep** (*bool*) – Appends the current timestep to the checkpoint file if true. If this is set to True, the load path must include the checkpoint timestep suffix. For example, if stored to models/ and set to true, the exported file will be of the form models/model.ckpt-X where X is the last timestep saved. The load path must precisely match this file name. If this option is turned off, the checkpoint will always overwrite the file specified in path and the model can always be loaded under this path.

Returns Checkpoint path were the model was saved.

set_normalized_actions(*actions*)

set_normalized_states(*states*)

should_stop()

class tensorforce.agents.**TRPOAgent**(*states*, *actions*, *network*, *batched_observe=True*, *batching_capacity=1000*, *scope='trpo'*, *device=None*, *saver=None*, *summarizer=None*, *execution=None*, *variable_noise=None*, *states_preprocessing=None*, *actions_exploration=None*, *reward_preprocessing=None*, *update_mode=None*, *memory=None*, *discount=0.99*, *distributions=None*, *entropy_regularization=None*, *baseline_mode=None*, *baseline=None*, *baseline_optimizer=None*, *gae_lambda=None*, *likelihood_ratio_clipping=None*, *learning_rate=0.001*, *cg_max_iterations=20*, *cg_damping=0.001*, *cg_unroll_loop=False*, *ls_max_iterations=10*, *ls_accept_ratio=0.9*, *ls_unroll_loop=False*)

Bases: *tensorforce.agents.learning_agent.LearningAgent*

Trust Region Policy Optimization agent (Schulman et al., 2015).

__init__(*states*, *actions*, *network*, *batched_observe=True*, *batching_capacity=1000*, *scope='trpo'*, *device=None*, *saver=None*, *summarizer=None*, *execution=None*, *variable_noise=None*, *states_preprocessing=None*, *actions_exploration=None*, *reward_preprocessing=None*, *update_mode=None*, *memory=None*, *discount=0.99*, *distributions=None*, *entropy_regularization=None*, *baseline_mode=None*, *baseline=None*, *baseline_optimizer=None*, *gae_lambda=None*, *likelihood_ratio_clipping=None*, *learning_rate=0.001*, *cg_max_iterations=20*, *cg_damping=0.001*, *cg_unroll_loop=False*, *ls_max_iterations=10*, *ls_accept_ratio=0.9*, *ls_unroll_loop=False*)

Initializes the TRPO agent.

Parameters **update_mode** – Update mode specification, with the following attributes:

Parameters

- **memory** (*spec*) – Memory specification, see core.memories module for more information (default: {type='latest', include_next_states=false, capacity=1000*batch_size}).
- **optimizer** (*spec*) – TRPO agent implicitly defines a optimized-step natural-gradient optimizer.
- **baseline_mode** (*str*) – One of 'states', 'network' (default: none).
- **baseline** (*spec*) – Baseline specification, see core.baselines module for more information (default: none).
- **baseline_optimizer** (*spec*) – Baseline optimizer specification, see core.optimizers module for more information (default: none).

- **gae_lambda** (*float*) – Lambda factor for generalized advantage estimation (default: none).
- **likelihood_ratio_clipping** (*float*) – Likelihood ratio clipping for policy gradient (default: none).
- **learning_rate** (*float*) – Learning rate of natural-gradient optimizer (default: 1e-3).
- **cg_max_iterations** (*int*) – Conjugate-gradient max iterations (default: 20).
- **cg_damping** (*float*) – Conjugate-gradient damping (default: 1e-3).
- **cg_unroll_loop** (*bool*) – Conjugate-gradient unroll loop (default: false).
- **ls_max_iterations** (*int*) – Line-search max iterations (default: 10).
- **ls_accept_ratio** (*float*) – Line-search accept ratio (default: 0.9).
- **ls_unroll_loop** (*bool*) – Line-search unroll loop (default: false).

act(*states*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)
Return action(s) for given state(s). States preprocessing and exploration are applied if configured accordingly.

Parameters

- **states** (*any*) – One state (usually a value tuple) or dict of states if multiple states are expected.
- **deterministic** (*bool*) – If true, no exploration and sampling is applied.
- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).
- **fetch_tensors** (*list*) – Optional String of named tensors to fetch

Returns Scalar value of the action or dict of multiple actions the agent wants to execute. (fetched_tensors) Optional dict() with named tensors fetched

close()

from_spec(*spec*, *kwargs*)
Creates an agent from a specification dict.

import_experience(*experiences*)
Imports experiences.

Parameters **experiences** –

initialize_model()

last_observation()

observe(*terminal*, *reward*)
Observe experience from the environment to learn from. Optionally pre-processes rewards Child classes should call super to get the processed reward EX: terminal, reward = super()...

Parameters

- **terminal** (*bool*) – boolean indicating if the episode terminated after the observation.
- **reward** (*float*) – scalar reward that resulted from executing the action.

reset()
Reset the agent to its initial state (e.g. on experiment start). Updates the Model's internal episode and time step counter, internal states, and resets preprocessors.

restore_model(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters

- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

save_model(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters

- **directory** (*str*) – Optional checkpoint directory.
- **append_timestep** (*bool*) – Appends the current timestep to the checkpoint file if true. If this is set to True, the load path must include the checkpoint timestep suffix. For example, if stored to models/ and set to true, the exported file will be of the form models/model.ckpt-X where X is the last timestep saved. The load path must precisely match this file name. If this option is turned off, the checkpoint will always overwrite the file specified in path and the model can always be loaded under this path.

Returns Checkpoint path were the model was saved.

set_normalized_actions(*actions*)

set_normalized_states(*states*)

should_stop()

class tensorforce.agents.**VPGAgent**(*states*, *actions*, *network*, *batched_observe=True*, *batching_capacity=1000*, *scope='vpg'*, *device=None*, *saver=None*, *summarizer=None*, *execution=None*, *variable_noise=None*, *states_preprocessing=None*, *actions_exploration=None*, *reward_preprocessing=None*, *update_mode=None*, *memory=None*, *optimizer=None*, *discount=0.99*, *distributions=None*, *entropy_regularization=None*, *baseline_mode=None*, *baseline=None*, *baseline_optimizer=None*, *gae_lambda=None*)
Bases: *tensorforce.agents.learning_agent.LearningAgent*

Vanilla policy gradient agent (Williams, 1992)).

__init__(*states*, *actions*, *network*, *batched_observe=True*, *batching_capacity=1000*, *scope='vpg'*, *device=None*, *saver=None*, *summarizer=None*, *execution=None*, *variable_noise=None*, *states_preprocessing=None*, *actions_exploration=None*, *reward_preprocessing=None*, *update_mode=None*, *memory=None*, *optimizer=None*, *discount=0.99*, *distributions=None*, *entropy_regularization=None*, *baseline_mode=None*, *baseline=None*, *baseline_optimizer=None*, *gae_lambda=None*)
Initializes the VPG agent.

Parameters **update_mode** – Update mode specification, with the following attributes:

Parameters

- **memory** (*spec*) – Memory specification, see core.memories module for more information (default: {type='latest', include_next_states=false, capacity=1000*batch_size}).

- **optimizer** (*spec*) – Optimizer specification, see core.optimizers module for more information (default: {type='adam', learning_rate=1e-3}).
- **baseline_mode** (*str*) – One of 'states', 'network' (default: none).
- **baseline** (*spec*) – Baseline specification, see core.baselines module for more information (default: none).
- **baseline_optimizer** (*spec*) – Baseline optimizer specification, see core.optimizers module for more information (default: none).
- **gae_lambda** (*float*) – Lambda factor for generalized advantage estimation (default: none).

act (*states*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)
Return action(s) for given state(s). States preprocessing and exploration are applied if configured accordingly.

Parameters

- **states** (*any*) – One state (usually a value tuple) or dict of states if multiple states are expected.
- **deterministic** (*bool*) – If true, no exploration and sampling is applied.
- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).
- **fetch_tensors** (*list*) – Optional String of named tensors to fetch

Returns Scalar value of the action or dict of multiple actions the agent wants to execute. (fetched_tensors) Optional dict() with named tensors fetched

close ()

from_spec (*spec*, *kwargs*)
Creates an agent from a specification dict.

import_experience (*experiences*)
Imports experiences.

Parameters **experiences** –

initialize_model ()

last_observation ()

observe (*terminal*, *reward*)
Observe experience from the environment to learn from. Optionally pre-processes rewards Child classes should call super to get the processed reward EX: terminal, reward = super()...

Parameters

- **terminal** (*bool*) – boolean indicating if the episode terminated after the observation.
- **reward** (*float*) – scalar reward that resulted from executing the action.

reset ()
Reset the agent to its initial state (e.g. on experiment start). Updates the Model's internal episode and time step counter, internal states, and resets preprocessors.

restore_model (*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters

- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

save_model(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters

- **directory** (*str*) – Optional checkpoint directory.
- **append_timestep** (*bool*) – Appends the current timestep to the checkpoint file if true. If this is set to True, the load path must include the checkpoint timestep suffix. For example, if stored to models/ and set to true, the exported file will be of the form models/model.ckpt-X where X is the last timestep saved. The load path must precisely match this file name. If this option is turned off, the checkpoint will always overwrite the file specified in path and the model can always be loaded under this path.

Returns Checkpoint path were the model was saved.

set_normalized_actions(*actions*)

set_normalized_states(*states*)

should_stop()

class tensorforce.agents.**DDPGAgent**(*states*, *actions*, *network*, *batched_observe=True*, *batching_capacity=1000*, *scope='ddpg'*, *device=None*, *saver=None*, *summarizer=None*, *execution=None*, *variable_noise=None*, *states_preprocessing=None*, *actions_exploration=None*, *reward_preprocessing=None*, *update_mode=None*, *memory=None*, *optimizer=None*, *discount=0.99*, *distributions=None*, *entropy_regularization=None*, *critic_network=None*, *critic_optimizer=None*, *target_sync_frequency=10000*, *target_update_weight=1.0*)
Bases: *tensorforce.agents.learning_agent.LearningAgent*

Deep Deterministic Policy Gradient agent (Lillicrap et al., 2015).

__init__(*states*, *actions*, *network*, *batched_observe=True*, *batching_capacity=1000*, *scope='ddpg'*, *device=None*, *saver=None*, *summarizer=None*, *execution=None*, *variable_noise=None*, *states_preprocessing=None*, *actions_exploration=None*, *reward_preprocessing=None*, *update_mode=None*, *memory=None*, *optimizer=None*, *discount=0.99*, *distributions=None*, *entropy_regularization=None*, *critic_network=None*, *critic_optimizer=None*, *target_sync_frequency=10000*, *target_update_weight=1.0*)
Initializes the DDPG agent.

Parameters **update_mode** – Update mode specification, with the following attributes:

Parameters

- **memory** (*spec*) – Memory specification, see core.memories module for more information (default: {type='replay', include_next_states=true, capacity=1000*batch_size}).
- **optimizer** (*spec*) – Optimizer specification, see core.optimizers module for more information (default: {type='adam', learning_rate=1e-3}).

- **critic_network** (*spec*) – Critic network specification, usually a list of layer specifications, see core.networks module for more information (default: network).
- **critic_optimizer** (*spec*) – Critic optimizer specification, see core.optimizers module for more information (default: {type='adam', learning_rate=1e-3}).
- **target_sync_frequency** (*int*) – Target network sync frequency (default: 10000).
- **target_update_weight** (*float*) – Target network update weight (default: 1.0).

act(*states*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)
Return action(s) for given state(s). States preprocessing and exploration are applied if configured accordingly.

Parameters

- **states** (*any*) – One state (usually a value tuple) or dict of states if multiple states are expected.
- **deterministic** (*bool*) – If true, no exploration and sampling is applied.
- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).
- **fetch_tensors** (*list*) – Optional String of named tensors to fetch

Returns Scalar value of the action or dict of multiple actions the agent wants to execute. (fetched_tensors) Optional dict() with named tensors fetched

close()

from_spec(*spec*, *kwargs*)
Creates an agent from a specification dict.

import_experience(*experiences*)
Imports experiences.

Parameters **experiences** –

initialize_model()

last_observation()

observe(*terminal*, *reward*)
Observe experience from the environment to learn from. Optionally pre-processes rewards Child classes should call super to get the processed reward EX: terminal, reward = super()...

Parameters

- **terminal** (*bool*) – boolean indicating if the episode terminated after the observation.
- **reward** (*float*) – scalar reward that resulted from executing the action.

reset()
Reset the agent to its initial state (e.g. on experiment start). Updates the Model's internal episode and time step counter, internal states, and resets preprocessors.

restore_model(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters

- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

save_model(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters

- **directory** (*str*) – Optional checkpoint directory.
- **append_timestep** (*bool*) – Appends the current timestep to the checkpoint file if true. If this is set to True, the load path must include the checkpoint timestep suffix. For example, if stored to models/ and set to true, the exported file will be of the form models/model.ckpt-X where X is the last timestep saved. The load path must precisely match this file name. If this option is turned off, the checkpoint will always overwrite the file specified in path and the model can always be loaded under this path.

Returns Checkpoint path were the model was saved.

set_normalized_actions(*actions*)

set_normalized_states(*states*)

should_stop()

tensorforce.contrib package

Submodules

tensorforce.contrib.ale module

class tensorforce.contrib.ale.**ALE**(*rom*, *frame_skip=1*, *repeat_action_probability=0.0*, *loss_of_life_termination=False*, *loss_of_life_reward=0*, *display_screen=False*, *seed=<mtrand.RandomState object>*)
Bases: *tensorforce.environments.environment.Environment*

Arcade Learning Environment (ALE). https://github.com/mgbellemare/Arcade-Learning-Environment

__init__(*rom*, *frame_skip=1*, *repeat_action_probability=0.0*, *loss_of_life_termination=False*, *loss_of_life_reward=0*, *display_screen=False*, *seed=<mtrand.RandomState object>*)
Initialize ALE.

Parameters

- **rom** – Rom filename and directory.
- **frame_skip** – Repeat action for n frames. Default 1.
- **repeat_action_probability** – Repeats last action with given probability. Default 0.
- **loss_of_life_termination** – Signals a terminal state on loss of life. Default False.
- **loss_of_life_reward** – Reward/Penalty on loss of life (negative values are a penalty). Default 0.
- **display_screen** – Displays the emulator screen. Default False.
- **seed** – Random seed

action_names

actions

close()

current_state

execute(*actions*)

from_spec(*spec*, *kwargs*)
Creates an environment from a specification dict.

is_terminal

reset()

seed(*seed*)
Sets the random seed of the environment to the given value (current time, if seed=None). Naturally deterministic Environments (e.g. ALE or some gym Envs) don't have to implement this method.

Parameters **seed** (*int*) – The seed to use for initializing the pseudo-random number generator (default=epoch time in sec).

Returns: The actual seed (int) used OR None if Environment did not override this method (no seeding supported).

states

tensorforce.contrib.deepmind_lab module

class tensorforce.contrib.deepmind_lab.**DeepMindLab**(*level_id*, *repeat_action=1*, *state_attribute='RGB_INTERLACED'*, *settings={'width': '320', 'appendCommand': '', 'fps': '60', 'height': '240'}*)

Bases: *tensorforce.environments.environment.Environment*

DeepMind Lab Integration: https://arxiv.org/abs/1612.03801 https://github.com/deepmind/lab

Since DeepMind lab is only available as source code, a manual install via bazel is required. Further, due to the way bazel handles external dependencies, cloning TensorForce into lab is the most convenient way to run it using the bazel BUILD file we provide. To use lab, first download and install it according to instructions https://github.com/deepmind/lab/blob/master/docs/build.md:

```
git clone https://github.com/deepmind/lab.git
```

Add to the lab main BUILD file:

Clone TensorForce into the lab directory, then run the TensorForce bazel runner.

Note that using any specific configuration file currently requires changing the Tensorforce BUILD file to adjust environment parameters.

```
bazel run //tensorforce:lab_runner
```

Please note that we have not tried to reproduce any lab results yet, and these instructions just explain connectivity in case someone wants to get started there.

__init__(*level_id*, *repeat_action=1*, *state_attribute='RGB_INTERLACED'*, *settings={'width': '320', 'appendCommand': '', 'fps': '60', 'height': '240'}*)
Initialize DeepMind Lab environment.

Parameters

- **level_id** – string with id/descriptor of the level, e.g. 'seekavoid_arena_01'.
- **repeat_action** – number of frames the environment is advanced, executing the given action during every frame.
- **state_attribute** – Attributes which represents the state for this environment, should adhere to the specification given in DeepMindLabEnvironment.state_spec(level_id).
- **settings** – dict specifying additional settings as key-value string pairs. The following options are recognized: 'width' (horizontal resolution of the observation frames), 'height' (vertical resolution of the observation frames), 'fps' (frames per second) and 'appendCommand' (commands for the internal Quake console).

actions

close()
Closes the environment and releases the underlying Quake III Arena instance. No other method calls possible afterwards.

execute(*actions*)
Pass action to universe environment, return reward, next step, terminal state and additional info.

Parameters **action** – action to execute as numpy array, should have dtype np.intc and should adhere to the specification given in DeepMindLabEnvironment.action_spec(level_id)

Returns dict containing the next state, the reward, and a boolean indicating if the next state is a terminal state

fps
An advisory metric that correlates discrete environment steps ("frames") with real (wallclock) time: the number of frames per (real) second.

from_spec(*spec*, *kwargs*)
Creates an environment from a specification dict.

num_steps
Number of frames since the last reset() call.

reset()
Resets the environment to its initialization state. This method needs to be called to start a new episode after the last episode ended.

Returns initial state

seed(*seed*)
Sets the random seed of the environment to the given value (current time, if seed=None). Naturally deterministic Environments (e.g. ALE or some gym Envs) don't have to implement this method.

Parameters **seed** (*int*) – The seed to use for initializing the pseudo-random number generator (default=epoch time in sec).

Returns: The actual seed (int) used OR None if Environment did not override this method (no seeding supported).

states

tensorforce.contrib.maze_explorer module

class tensorforce.contrib.maze_explorer.**MazeExplorer**(*mode_id=0*, *visible=True*)
Bases: *tensorforce.environments.environment.Environment*

MazeExplorer Integration: https://github.com/mryellow/maze_explorer.

__init__(*mode_id=0*, *visible=True*)
Initialize MazeExplorer.

Parameters

- **mode_id** – Game mode ID. See https://github.com/mryellow/maze_explorer
- **visible** – Show output window

actions

close()

execute(*actions*)

from_spec(*spec*, *kwargs*)
Creates an environment from a specification dict.

reset()

seed(*seed*)
Sets the random seed of the environment to the given value (current time, if seed=None). Naturally deterministic Environments (e.g. ALE or some gym Envs) don't have to implement this method.

Parameters **seed** (*int*) – The seed to use for initializing the pseudo-random number generator (default=epoch time in sec).

Returns: The actual seed (int) used OR None if Environment did not override this method (no seeding supported).

states

tensorforce.contrib.openai_gym module

OpenAI Gym Integration: https://gym.openai.com/.

class tensorforce.contrib.openai_gym.**OpenAIGym**(*gym_id*, *monitor=None*, *monitor_safe=False*, *monitor_video=0*, *visualize=False*)
Bases: *tensorforce.environments.environment.Environment*

__init__(*gym_id*, *monitor=None*, *monitor_safe=False*, *monitor_video=0*, *visualize=False*)
Initialize OpenAI Gym.

Parameters

- **gym_id** – OpenAI Gym environment ID. See https://gym.openai.com/envs
- **monitor** – Output directory. Setting this to None disables monitoring.
- **monitor_safe** – Setting this to True prevents existing log files to be overwritten. Default False.
- **monitor_video** – Save a video every monitor_video steps. Setting this to 0 disables recording of videos.
- **visualize** – If set True, the program will visualize the trainings of gym's environment. Note that such visualization is probabily going to slow down the training.

static action_from_space(*space*)

actions

close()

execute(*actions*)

from_spec(*spec*, *kwargs*)
Creates an environment from a specification dict.

reset()

seed(*seed*)
Sets the random seed of the environment to the given value (current time, if seed=None). Naturally deterministic Environments (e.g. ALE or some gym Envs) don't have to implement this method.

Parameters **seed** (*int*) – The seed to use for initializing the pseudo-random number generator (default=epoch time in sec).

Returns: The actual seed (int) used OR None if Environment did not override this method (no seeding supported).

static state_from_space(*space*)

states

tensorforce.contrib.openai_universe module

class tensorforce.contrib.openai_universe.**OpenAIUniverse**(*env_id*)
Bases: *tensorforce.environments.environment.Environment*

OpenAI Universe Integration: https://universe.openai.com/. Contains OpenAI Gym: https://gym.openai.com/.

__init__(*env_id*)
Initialize OpenAI universe environment.

Parameters **env_id** – string with id/descriptor of the universe environment, e.g. 'HarvestDay-v0'.

actions

close()

configure(**args*, ***kwargs*)

execute(*actions*)

from_spec(*spec*, *kwargs*)
Creates an environment from a specification dict.

render(**args*, ***kwargs*)

reset()

seed(*seed*)
Sets the random seed of the environment to the given value (current time, if seed=None). Naturally deterministic Environments (e.g. ALE or some gym Envs) don't have to implement this method.

Parameters **seed** (*int*) – The seed to use for initializing the pseudo-random number generator (default=epoch time in sec).

Returns: The actual seed (int) used OR None if Environment did not override this method (no seeding supported).

states

tensorforce.contrib.remote_environment module

class tensorforce.contrib.remote_environment.**MsgPackNumpyProtocol**(*max_msg_len=8192*)
Bases: object

A simple protocol to communicate over tcp sockets, which can be used by RemoteEnvironment implementations. The protocol is based on msgpack-numpy encoding and decoding.

Each message has a simple 8-byte header, which encodes the length of the subsequent msgpack-numpy encoded byte-string. All messages received need to have the 'status' field set to 'ok'. If 'status' is set to 'error', the field 'message' should be populated with some error information.

Examples: client sends: "[8-byte header]msgpack-encoded({"cmd": "seed", "value": 200})" server responds: "[8-byte header]msgpack-encoded({"status": "ok", "value": 200})"

client sends: "[8-byte header]msgpack-encoded({"cmd": "reset"})" server responds: "[8-byte header]msgpack-encoded({"status": "ok"})"

client sends: "[8-byte header]msgpack-encoded({"cmd": "step", "action": 5})" server responds: "[8-byte header]msgpack-encoded({"status": "ok", "obs_dict": {... some observations}, "reward": -10.0, "is_terminal": False})"

__init__(*max_msg_len=8192*)

Parameters **max_msg_len** (*int*) – The maximum number of bytes to read from the socket.

recv(*socket_*)
Receives a message as msgpack-numpy encoded byte-string from the given socket object. Blocks until something was received.

Parameters **socket** – The python socket object to use.

Returns: The decoded (as dict) message received.

send(*message*, *socket_*)

Sends a message (dict) to the socket. Message consists of a 8-byte len header followed by a msgpack-numpy encoded dict.

Parameters

- **message** – The message dict (e.g. {"cmd": "reset"})
- **socket** – The python socket object to use.

class tensorforce.contrib.remote_environment.**RemoteEnvironment**(*host='localhost'*, *port=6025*)
Bases: *tensorforce.environments.environment.Environment*

__init__(*host='localhost'*, *port=6025*)
A remote Environment that one can connect to through tcp. Implements a simple msgpack protocol to get the step/reset/etc.. commands to the remote server and simply waits (blocks) for a response.

Parameters

- **host** (*str*) – The hostname to connect to.
- **port** (*int*) – The port to connect to.

actions
Return the action space. Might include subdicts if multiple actions are available simultaneously.

Returns: dict of action properties (continuous, number of actions)

close()
Same as disconnect method.

connect(*timeout=600*)
Starts the server tcp connection on the given host:port.

Parameters **timeout** (*int*) – The time (in seconds) for which we will attempt a connection to the remote (every 5sec). After that (or if timeout is None or 0), an error is raised.

current_state

disconnect()
Ends our server tcp connection.

execute(*actions*)
Executes action, observes next state(s) and reward.

Parameters **actions** – Actions to execute.

Returns (Dict of) next state(s), boolean indicating terminal, and reward signal.

from_spec(*spec*, *kwargs*)
Creates an environment from a specification dict.

reset()
Reset environment and setup for new episode.

Returns initial state of reset environment.

seed(*seed*)
Sets the random seed of the environment to the given value (current time, if seed=None). Naturally deterministic Environments (e.g. ALE or some gym Envs) don't have to implement this method.

Parameters **seed** (*int*) – The seed to use for initializing the pseudo-random number generator (default=epoch time in sec).

Returns: The actual seed (int) used OR None if Environment did not override this method (no seeding supported).

states
Return the state space. Might include subdicts if multiple states are available simultaneously.

Returns: dict of state properties (shape and type).

tensorforce.contrib.state_settable_environment module

class tensorforce.contrib.state_settable_environment.**StateSettableEnvironment**
Bases: *tensorforce.environments.environment.Environment*

An Environment that implements the set_state method to set the current state to some new state using setter instructions.

__init__
x.**init**(...) initializes x; see help(type(x)) for signature

actions
Return the action space. Might include subdicts if multiple actions are available simultaneously.

Returns: dict of action properties (continuous, number of actions)

close()
Close environment. No other method calls possible afterwards.

execute(*actions*)
Executes action, observes next state(s) and reward.

Parameters **actions** – Actions to execute.

Returns (Dict of) next state(s), boolean indicating terminal, and reward signal.

from_spec(*spec*, *kwargs*)
Creates an environment from a specification dict.

reset()
Reset environment and setup for new episode.

Returns initial state of reset environment.

seed(*seed*)
Sets the random seed of the environment to the given value (current time, if seed=None). Naturally deterministic Environments (e.g. ALE or some gym Envs) don't have to implement this method.

Parameters **seed** (*int*) – The seed to use for initializing the pseudo-random number generator (default=epoch time in sec).

Returns: The actual seed (int) used OR None if Environment did not override this method (no seeding supported).

set_state(***kwargs*)
Sets the current state of the environment manually to some other state and returns a new observation.

Parameters ****kwargs** –

The set instruction(s) to be executed by the environment. A single set instruction usually set a single property of the

state/observation vector to some new value.

Returns: The observation dictionary of the Environment after(!) setting it to the new state.

states
Return the state space. Might include subdicts if multiple states are available simultaneously.

Returns: dict of state properties (shape and type).

tensorforce.contrib.unreal_engine module

class tensorforce.contrib.unreal_engine.**UE4Environment**(*host='localhost'*, *port=6025*, *connect=True*, *discretize_actions=False*, *delta_time=0*, *num_ticks=4*)
Bases: *tensorforce.contrib.remote_environment.RemoteEnvironment*, *tensorforce.contrib.state_settable_environment.StateSettableEnvironment*

A special RemoteEnvironment for UE4 game connections. Communicates with the remote to receive information on the definitions of action- and observation spaces. Sends UE4 Action- and Axis-mappings as RL-actions and receives observations back defined by MLObserver objects placed in the Game (these could be camera pixels or other observations, e.g. a x/y/z position of some game actor).

__init__(*host='localhost'*, *port=6025*, *connect=True*, *discretize_actions=False*, *delta_time=0*, *num_ticks=4*)

Parameters

- **host** (*str*) – The hostname to connect to.

- **port** (*int*) – The port to connect to.
- **connect** (*bool*) – Whether to connect already in this c'tor.
- **discretize_actions** (*bool*) – Whether to treat axis-mappings defined in UE4 game as discrete actions. This would be necessary e.g. for agents that use q-networks where the output are q-values per discrete state-action pair.
- **delta_time** (*float*) – The fake delta time to use for each single game tick.
- **num_ticks** (*int*) – The number of ticks to be executed in a single act call (each tick will repeat the same given actions).

actions()

close()
Same as disconnect method.

connect(*timeout=600*)

current_state

disconnect()
Ends our server tcp connection.

discretize_action_space_desc()
Creates a list of discrete action(-combinations) in case we want to learn with a discrete set of actions, but only have action-combinations (maybe even continuous) available from the env. E.g. the UE4 game has the following action/axis-mappings:

```
{
'Fire':
    {'type': 'action', 'keys': ('SpaceBar',)},
'MoveRight':
    {'type': 'axis', 'keys': (('Right', 1.0), ('Left', -1.0), ('A', -1.0), ('D
↪', 1.0))},
}
```

-> this method will discretize them into the following 6 discrete actions:

```
[
[(Right, 0.0),(SpaceBar, False)],
[(Right, 0.0),(SpaceBar, True)]
[(Right, -1.0),(SpaceBar, False)],
[(Right, -1.0),(SpaceBar, True)],
[(Right, 1.0),(SpaceBar, False)],
[(Right, 1.0),(SpaceBar, True)],
]
```

execute(*actions*)
Executes a single step in the UE4 game. This step may be comprised of one or more actual game ticks for all of which the same given action- and axis-inputs (or action number in case of discretized actions) are repeated. UE4 distinguishes between action-mappings, which are boolean actions (e.g. jump or dont-jump) and axis-mappings, which are continuous actions like MoveForward with values between -1.0 (run backwards) and 1.0 (run forwards), 0.0 would mean: stop.

static extract_observation(*message*)

from_spec(*spec*, *kwargs*)
Creates an environment from a specification dict.

reset()
same as step (no kwargs to pass), but needs to block and return observation_dict

- stores the received observation in self.last_observation

seed(*seed=None*)

set_state(*setters*, ***kwargs*)

states()

translate_abstract_actions_to_keys(*abstract*)
Translates a list of tuples ([pretty mapping], [value]) to a list of tuples ([some key], [translated value]) each single item in abstract will undergo the following translation:

Example1: we want: "MoveRight": 5.0 possible keys for the action are: ("Right", 1.0), ("Left", -1.0) result: "Right": 5.0 * 1.0 = 5.0

Example2: we want: "MoveRight": -0.5 possible keys for the action are: ("Left", -1.0), ("Right", 1.0) result: "Left": -0.5 * -1.0 = 0.5 (same as "Right": -0.5)

Module contents

tensorforce.core package

Subpackages

tensorforce.core.baselines package

Submodules

tensorforce.core.baselines.aggregated_baseline module

class tensorforce.core.baselines.aggregated_baseline.**AggregatedBaseline**(*baselines*, *scope='aggregated-baseline'*, *summary_labels=()*)

Bases: *tensorforce.core.baselines.baseline.Baseline*

Baseline which aggregates per-state baselines.

__init__(*baselines*, *scope='aggregated-baseline'*, *summary_labels=()*)
Aggregated baseline.

Parameters **baselines** – Dict of per-state baseline specification dicts

from_spec(*spec*, *kwargs=None*)
Creates a baseline from a specification dict.

get_summaries()

get_variables(*include_nontrainable=False*)

tf_loss(*states*, *internals*, *reward*, *update*, *reference=None*)
Creates the TensorFlow operations for calculating the L2 loss between predicted state values and actual rewards.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **reward** – Reward tensor.
- **update** – Boolean tensor indicating whether this call happens during an update.
- **reference** – Optional reference tensor(s), in case of a comparative loss.

Returns Loss tensor

tf_predict(*states*, *internals*, *update*)

tf_reference(*states*, *internals*, *reward*, *update*)
Creates the TensorFlow operations for obtaining the reference tensor(s), in case of a comparative loss.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **reward** – Reward tensor.
- **update** – Boolean tensor indicating whether this call happens during an update.

Returns Reference tensor(s).

tf_regularization_loss()

tensorforce.core.baselines.baseline module

class tensorforce.core.baselines.baseline.**Baseline**(*scope='baseline'*, *summary_labels=None*)

Bases: object

Base class for baseline value functions.

__init__(*scope='baseline'*, *summary_labels=None*)
Baseline.

static from_spec(*spec*, *kwargs=None*)
Creates a baseline from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the baseline

Returns List of summaries

get_variables(*include_nontrainable=False*)
Returns the TensorFlow variables used by the baseline.

Returns List of variables

tf_loss(*states*, *internals*, *reward*, *update*, *reference=None*)
Creates the TensorFlow operations for calculating the L2 loss between predicted state values and actual rewards.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **reward** – Reward tensor.

- **update** – Boolean tensor indicating whether this call happens during an update.
- **reference** – Optional reference tensor(s), in case of a comparative loss.

Returns Loss tensor

tf_predict(*states*, *internals*, *update*)
Creates the TensorFlow operations for predicting the value function of given states. :param states: Dict of state tensors. :param internals: List of prior internal state tensors. :param update: Boolean tensor indicating whether this call happens during an update.

Returns State value tensor

tf_reference(*states*, *internals*, *reward*, *update*)
Creates the TensorFlow operations for obtaining the reference tensor(s), in case of a comparative loss.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **reward** – Reward tensor.
- **update** – Boolean tensor indicating whether this call happens during an update.

Returns Reference tensor(s).

tf_regularization_loss()
Creates the TensorFlow operations for the baseline regularization loss/

Returns Regularization loss tensor

tensorforce.core.baselines.cnn_baseline module

class tensorforce.core.baselines.cnn_baseline.**CNNBaseline**(*conv_sizes*, *dense_sizes*, *scope='cnn-baseline'*, *summary_labels=()*)
Bases: *tensorforce.core.baselines.network_baseline.NetworkBaseline*

CNN baseline (single-state) consisting of convolutional layers followed by dense layers.

__init__(*conv_sizes*, *dense_sizes*, *scope='cnn-baseline'*, *summary_labels=()*)
CNN baseline.

Parameters

- **conv_sizes** – List of convolutional layer sizes
- **dense_sizes** – List of dense layer sizes

from_spec(*spec*, *kwargs=None*)
Creates a baseline from a specification dict.

get_summaries()

get_variables(*include_nontrainable=False*)

tf_loss(*states*, *internals*, *reward*, *update*, *reference=None*)
Creates the TensorFlow operations for calculating the L2 loss between predicted state values and actual rewards.

Parameters

- **states** – Dict of state tensors.

- **internals** – List of prior internal state tensors.
- **reward** – Reward tensor.
- **update** – Boolean tensor indicating whether this call happens during an update.
- **reference** – Optional reference tensor(s), in case of a comparative loss.

Returns Loss tensor

tf_predict(*states*, *internals*, *update*)

tf_reference(*states*, *internals*, *reward*, *update*)
Creates the TensorFlow operations for obtaining the reference tensor(s), in case of a comparative loss.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **reward** – Reward tensor.
- **update** – Boolean tensor indicating whether this call happens during an update.

Returns Reference tensor(s).

tf_regularization_loss()

tensorforce.core.baselines.mlp_baseline module

class tensorforce.core.baselines.mlp_baseline.**MLPBaseline**(*sizes*, *scope='mlp-baseline'*, *summary_labels=()*)
Bases: *tensorforce.core.baselines.network_baseline.NetworkBaseline*

Multi-layer perceptron baseline (single-state) consisting of dense layers.

__init__(*sizes*, *scope='mlp-baseline'*, *summary_labels=()*)
Multi-layer perceptron baseline.

Parameters **sizes** – List of dense layer sizes

from_spec(*spec*, *kwargs=None*)
Creates a baseline from a specification dict.

get_summaries()

get_variables(*include_nontrainable=False*)

tf_loss(*states*, *internals*, *reward*, *update*, *reference=None*)
Creates the TensorFlow operations for calculating the L2 loss between predicted state values and actual rewards.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **reward** – Reward tensor.
- **update** – Boolean tensor indicating whether this call happens during an update.
- **reference** – Optional reference tensor(s), in case of a comparative loss.

Returns Loss tensor

tf_predict(*states*, *internals*, *update*)

tf_reference(*states*, *internals*, *reward*, *update*)
Creates the TensorFlow operations for obtaining the reference tensor(s), in case of a comparative loss.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **reward** – Reward tensor.
- **update** – Boolean tensor indicating whether this call happens during an update.

Returns Reference tensor(s).

tf_regularization_loss()

tensorforce.core.baselines.network_baseline module

class tensorforce.core.baselines.network_baseline.**NetworkBaseline**(*network*, *scope='network-baseline'*, *summary_labels=()*)

Bases: *tensorforce.core.baselines.baseline.Baseline*

Baseline based on a TensorForce network, used when parameters are shared between the value function and the baseline.

__init__(*network*, *scope='network-baseline'*, *summary_labels=()*)
Network baseline.

Parameters **network_spec** – Network specification dict

from_spec(*spec*, *kwargs=None*)
Creates a baseline from a specification dict.

get_summaries()

get_variables(*include_nontrainable=False*)

tf_loss(*states*, *internals*, *reward*, *update*, *reference=None*)
Creates the TensorFlow operations for calculating the L2 loss between predicted state values and actual rewards.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **reward** – Reward tensor.
- **update** – Boolean tensor indicating whether this call happens during an update.
- **reference** – Optional reference tensor(s), in case of a comparative loss.

Returns Loss tensor

tf_predict(*states*, *internals*, *update*)

tf_reference(*states*, *internals*, *reward*, *update*)
Creates the TensorFlow operations for obtaining the reference tensor(s), in case of a comparative loss.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **reward** – Reward tensor.
- **update** – Boolean tensor indicating whether this call happens during an update.

Returns Reference tensor(s).

tf_regularization_loss()

Module contents

class tensorforce.core.baselines.**Baseline**(*scope='baseline'*, *summary_labels=None*)

Bases: object

Base class for baseline value functions.

__init__(*scope='baseline'*, *summary_labels=None*)

Baseline.

static from_spec(*spec*, *kwargs=None*)

Creates a baseline from a specification dict.

get_summaries()

Returns the TensorFlow summaries reported by the baseline

Returns List of summaries

get_variables(*include_nontrainable=False*)

Returns the TensorFlow variables used by the baseline.

Returns List of variables

tf_loss(*states*, *internals*, *reward*, *update*, *reference=None*)

Creates the TensorFlow operations for calculating the L2 loss between predicted state values and actual rewards.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **reward** – Reward tensor.
- **update** – Boolean tensor indicating whether this call happens during an update.
- **reference** – Optional reference tensor(s), in case of a comparative loss.

Returns Loss tensor

tf_predict(*states*, *internals*, *update*)

Creates the TensorFlow operations for predicting the value function of given states. :param states: Dict of state tensors. :param internals: List of prior internal state tensors. :param update: Boolean tensor indicating whether this call happens during an update.

Returns State value tensor

tf_reference(*states*, *internals*, *reward*, *update*)

Creates the TensorFlow operations for obtaining the reference tensor(s), in case of a comparative loss.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **reward** – Reward tensor.
- **update** – Boolean tensor indicating whether this call happens during an update.

Returns Reference tensor(s).

tf_regularization_loss()
Creates the TensorFlow operations for the baseline regularization loss/

Returns Regularization loss tensor

class tensorforce.core.baselines.**AggregatedBaseline**(*baselines*, *scope='aggregated-baseline'*, *summary_labels=()*)
Bases: *tensorforce.core.baselines.baseline.Baseline*

Baseline which aggregates per-state baselines.

__init__(*baselines*, *scope='aggregated-baseline'*, *summary_labels=()*)
Aggregated baseline.

Parameters **baselines** – Dict of per-state baseline specification dicts

from_spec(*spec*, *kwargs=None*)
Creates a baseline from a specification dict.

get_summaries()

get_variables(*include_nontrainable=False*)

tf_loss(*states*, *internals*, *reward*, *update*, *reference=None*)
Creates the TensorFlow operations for calculating the L2 loss between predicted state values and actual rewards.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **reward** – Reward tensor.
- **update** – Boolean tensor indicating whether this call happens during an update.
- **reference** – Optional reference tensor(s), in case of a comparative loss.

Returns Loss tensor

tf_predict(*states*, *internals*, *update*)

tf_reference(*states*, *internals*, *reward*, *update*)
Creates the TensorFlow operations for obtaining the reference tensor(s), in case of a comparative loss.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **reward** – Reward tensor.
- **update** – Boolean tensor indicating whether this call happens during an update.

Returns Reference tensor(s).

tf_regularization_loss()

class tensorforce.core.baselines.**NetworkBaseline**(*network*, *scope='network-baseline'*, *summary_labels=()*)

Bases: *tensorforce.core.baselines.baseline.Baseline*

Baseline based on a TensorForce network, used when parameters are shared between the value function and the baseline.

__init__(*network*, *scope='network-baseline'*, *summary_labels=()*)

Network baseline.

Parameters **network_spec** – Network specification dict

from_spec(*spec*, *kwargs=None*)

Creates a baseline from a specification dict.

get_summaries()

get_variables(*include_nontrainable=False*)

tf_loss(*states*, *internals*, *reward*, *update*, *reference=None*)

Creates the TensorFlow operations for calculating the L2 loss between predicted state values and actual rewards.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **reward** – Reward tensor.
- **update** – Boolean tensor indicating whether this call happens during an update.
- **reference** – Optional reference tensor(s), in case of a comparative loss.

Returns Loss tensor

tf_predict(*states*, *internals*, *update*)

tf_reference(*states*, *internals*, *reward*, *update*)

Creates the TensorFlow operations for obtaining the reference tensor(s), in case of a comparative loss.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **reward** – Reward tensor.
- **update** – Boolean tensor indicating whether this call happens during an update.

Returns Reference tensor(s).

tf_regularization_loss()

class tensorforce.core.baselines.**MLPBaseline**(*sizes*, *scope='mlp-baseline'*, *summary_labels=()*)

Bases: *tensorforce.core.baselines.network_baseline.NetworkBaseline*

Multi-layer perceptron baseline (single-state) consisting of dense layers.

__init__(*sizes*, *scope='mlp-baseline'*, *summary_labels=()*)

Multi-layer perceptron baseline.

Parameters **sizes** – List of dense layer sizes

from_spec(*spec*, *kwargs=None*)
Creates a baseline from a specification dict.

get_summaries()

get_variables(*include_nontrainable=False*)

tf_loss(*states*, *internals*, *reward*, *update*, *reference=None*)
Creates the TensorFlow operations for calculating the L2 loss between predicted state values and actual rewards.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **reward** – Reward tensor.
- **update** – Boolean tensor indicating whether this call happens during an update.
- **reference** – Optional reference tensor(s), in case of a comparative loss.

Returns Loss tensor

tf_predict(*states*, *internals*, *update*)

tf_reference(*states*, *internals*, *reward*, *update*)
Creates the TensorFlow operations for obtaining the reference tensor(s), in case of a comparative loss.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **reward** – Reward tensor.
- **update** – Boolean tensor indicating whether this call happens during an update.

Returns Reference tensor(s).

tf_regularization_loss()

class tensorforce.core.baselines.**CNNBaseline**(*conv_sizes*, *dense_sizes*, *scope='cnn-baseline'*, *summary_labels=()*)
Bases: *tensorforce.core.baselines.network_baseline.NetworkBaseline*

CNN baseline (single-state) consisting of convolutional layers followed by dense layers.

__init__(*conv_sizes*, *dense_sizes*, *scope='cnn-baseline'*, *summary_labels=()*)
CNN baseline.

Parameters

- **conv_sizes** – List of convolutional layer sizes
- **dense_sizes** – List of dense layer sizes

from_spec(*spec*, *kwargs=None*)
Creates a baseline from a specification dict.

get_summaries()

get_variables(*include_nontrainable=False*)

tf_loss(*states*, *internals*, *reward*, *update*, *reference=None*)
Creates the TensorFlow operations for calculating the L2 loss between predicted state values and actual rewards.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **reward** – Reward tensor.
- **update** – Boolean tensor indicating whether this call happens during an update.
- **reference** – Optional reference tensor(s), in case of a comparative loss.

Returns Loss tensor

tf_predict(*states*, *internals*, *update*)

tf_reference(*states*, *internals*, *reward*, *update*)
Creates the TensorFlow operations for obtaining the reference tensor(s), in case of a comparative loss.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **reward** – Reward tensor.
- **update** – Boolean tensor indicating whether this call happens during an update.

Returns Reference tensor(s).

tf_regularization_loss()

tensorforce.core.distributions package

Submodules

tensorforce.core.distributions.bernoulli module

class tensorforce.core.distributions.bernoulli.**Bernoulli**(*shape*, *probability=0.5*, *scope='bernoulli'*, *summary_labels=()*)

Bases: *tensorforce.core.distributions.distribution.Distribution*

Bernoulli distribution, for binary boolean actions.

__init__(*shape*, *probability=0.5*, *scope='bernoulli'*, *summary_labels=()*)
Bernoulli distribution.

Parameters

- **shape** – Action shape.
- **probability** – Optional distribution bias.

from_spec(*spec*, *kwargs=None*)
Creates a distribution from a specification dict.

get_summaries()

get_variables(*include_nontrainable=False*)

state_action_value(*distr_params*, *action=None*)

state_value(*distr_params*)

tf_entropy(*distr_params*)

tf_kl_divergence(*distr_params1*, *distr_params2*)

tf_log_probability(*distr_params*, *action*)

tf_parameterize(*x*)

tf_regularization_loss()

tf_sample(*distr_params*, *deterministic*)

tensorforce.core.distributions.beta module

class tensorforce.core.distributions.beta.**Beta**(*shape*, *min_value*, *max_value*, *alpha=0.0*, *beta=0.0*, *scope='beta'*, *summary_labels=()*)

Bases: *tensorforce.core.distributions.distribution.Distribution*

Beta distribution, for bounded continuous actions.

__init__(*shape*, *min_value*, *max_value*, *alpha=0.0*, *beta=0.0*, *scope='beta'*, *summary_labels=()*)

Beta distribution.

Parameters

- **shape** – Action shape.
- **min_value** – Minimum value of continuous actions.
- **max_value** – Maximum value of continuous actions.
- **alpha** – Optional distribution bias for the alpha value.
- **beta** – Optional distribution bias for the beta value.

from_spec(*spec*, *kwargs=None*)

Creates a distribution from a specification dict.

get_summaries()

get_variables(*include_nontrainable=False*)

tf_entropy(*distr_params*)

tf_kl_divergence(*distr_params1*, *distr_params2*)

tf_log_probability(*distr_params*, *action*)

tf_parameterize(*x*)

tf_regularization_loss()

tf_sample(*distr_params*, *deterministic*)

tensorforce.core.distributions.categorical module

class tensorforce.core.distributions.categorical.**Categorical**(*shape*, *num_actions*, *probabilities=None*, *scope='categorical'*, *summary_labels=()*)

Bases: *tensorforce.core.distributions.distribution.Distribution*

Categorical distribution, for discrete actions.

__init__(*shape*, *num_actions*, *probabilities=None*, *scope='categorical'*, *summary_labels=()*)

Categorical distribution.

Parameters

- **shape** – Action shape.
- **num_actions** – Number of discrete action alternatives.
- **probabilities** – Optional distribution bias.

from_spec(*spec*, *kwargs=None*)

Creates a distribution from a specification dict.

get_summaries()

get_variables(*include_nontrainable=False*)

state_action_value(*distr_params*, *action=None*)

state_value(*distr_params*)

tf_entropy(*distr_params*)

tf_kl_divergence(*distr_params1*, *distr_params2*)

tf_log_probability(*distr_params*, *action*)

tf_parameterize(*x*)

tf_regularization_loss()

tf_sample(*distr_params*, *deterministic*)

tensorforce.core.distributions.distribution module

class tensorforce.core.distributions.distribution.**Distribution**(*shape*, *scope='distribution'*, *summary_labels=None*)

Bases: object

Base class for policy distributions.

__init__(*shape*, *scope='distribution'*, *summary_labels=None*)

Distribution.

Parameters **shape** – Action shape.

static from_spec(*spec*, *kwargs=None*)

Creates a distribution from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the distribution.

Returns List of summaries.

get_variables(*include_nontrainable=False*)
Returns the TensorFlow variables used by the distribution.

Returns List of variables.

tf_entropy(*distr_params*)
Creates the TensorFlow operations for calculating the entropy of a distribution.

Parameters **distr_params** – Tuple of distribution parameter tensors.

Returns Entropy tensor.

tf_kl_divergence(*distr_params1*, *distr_params2*)
Creates the TensorFlow operations for calculating the KL divergence between two distributions.

Parameters

- **distr_params1** – Tuple of parameter tensors for first distribution.
- **distr_params2** – Tuple of parameter tensors for second distribution.

Returns KL divergence tensor.

tf_log_probability(*distr_params*, *action*)
Creates the TensorFlow operations for calculating the log probability of an action for a distribution.

Parameters

- **distr_params** – Tuple of distribution parameter tensors.
- **action** – Action tensor.

Returns KL divergence tensor.

tf_parameterize(*x*)
Creates the TensorFlow operations for parameterizing a distribution conditioned on the given input.

Parameters **x** – Input tensor which the distribution is conditioned on.

Returns Tuple of distribution parameter tensors.

tf_regularization_loss()
Creates the TensorFlow operations for the distribution regularization loss.

Returns Regularization loss tensor.

tf_sample(*distr_params*, *deterministic*)
Creates the TensorFlow operations for sampling an action based on a distribution.

Parameters

- **distr_params** – Tuple of distribution parameter tensors.
- **deterministic** – Boolean input tensor indicating whether the maximum likelihood action should be returned.

Returns Sampled action tensor.

tensorforce.core.distributions.gaussian module

class tensorforce.core.distributions.gaussian.**Gaussian**(*shape*, *mean=0.0*, *log_stddev=0.0*, *scope='gaussian'*, *summary_labels=()*)

Bases: *tensorforce.core.distributions.distribution.Distribution*

Gaussian distribution, for unbounded continuous actions.

__init__(*shape*, *mean=0.0*, *log_stddev=0.0*, *scope='gaussian'*, *summary_labels=()*)

Categorical distribution.

Parameters

- **shape** – Action shape.
- **mean** – Optional distribution bias for the mean.
- **log_stddev** – Optional distribution bias for the standard deviation.

from_spec(*spec*, *kwargs=None*)

Creates a distribution from a specification dict.

get_summaries()

get_variables(*include_nontrainable=False*)

state_action_value(*distr_params*, *action*)

state_value(*distr_params*)

tf_entropy(*distr_params*)

tf_kl_divergence(*distr_params1*, *distr_params2*)

tf_log_probability(*distr_params*, *action*)

tf_parameterize(*x*)

tf_regularization_loss()

tf_sample(*distr_params*, *deterministic*)

Module contents

class tensorforce.core.distributions.**Distribution**(*shape*, *scope='distribution'*, *summary_labels=None*)

Bases: `object`

Base class for policy distributions.

__init__(*shape*, *scope='distribution'*, *summary_labels=None*)

Distribution.

Parameters **shape** – Action shape.

static from_spec(*spec*, *kwargs=None*)

Creates a distribution from a specification dict.

get_summaries()

Returns the TensorFlow summaries reported by the distribution.

Returns List of summaries.

get_variables(*include_nontrainable=False*)
Returns the TensorFlow variables used by the distribution.

Returns List of variables.

tf_entropy(*distr_params*)
Creates the TensorFlow operations for calculating the entropy of a distribution.

Parameters **distr_params** – Tuple of distribution parameter tensors.

Returns Entropy tensor.

tf_kl_divergence(*distr_params1*, *distr_params2*)
Creates the TensorFlow operations for calculating the KL divergence between two distributions.

Parameters

- **distr_params1** – Tuple of parameter tensors for first distribution.
- **distr_params2** – Tuple of parameter tensors for second distribution.

Returns KL divergence tensor.

tf_log_probability(*distr_params*, *action*)
Creates the TensorFlow operations for calculating the log probability of an action for a distribution.

Parameters

- **distr_params** – Tuple of distribution parameter tensors.
- **action** – Action tensor.

Returns KL divergence tensor.

tf_parameterize(*x*)
Creates the TensorFlow operations for parameterizing a distribution conditioned on the given input.

Parameters **x** – Input tensor which the distribution is conditioned on.

Returns Tuple of distribution parameter tensors.

tf_regularization_loss()
Creates the TensorFlow operations for the distribution regularization loss.

Returns Regularization loss tensor.

tf_sample(*distr_params*, *deterministic*)
Creates the TensorFlow operations for sampling an action based on a distribution.

Parameters

- **distr_params** – Tuple of distribution parameter tensors.
- **deterministic** – Boolean input tensor indicating whether the maximum likelihood action should be returned.

Returns Sampled action tensor.

class tensorforce.core.distributions.**Bernoulli**(*shape*, *probability=0.5*, *scope='bernoulli'*, *summary_labels=()*)
Bases: *tensorforce.core.distributions.distribution.Distribution*

Bernoulli distribution, for binary boolean actions.

__init__(*shape*, *probability=0.5*, *scope='bernoulli'*, *summary_labels=()*)
Bernoulli distribution.

Parameters

- **shape** – Action shape.
- **probability** – Optional distribution bias.

from_spec(*spec*, *kwargs=None*)
Creates a distribution from a specification dict.

get_summaries()

get_variables(*include_nontrainable=False*)

state_action_value(*distr_params*, *action=None*)

state_value(*distr_params*)

tf_entropy(*distr_params*)

tf_kl_divergence(*distr_params1*, *distr_params2*)

tf_log_probability(*distr_params*, *action*)

tf_parameterize(*x*)

tf_regularization_loss()

tf_sample(*distr_params*, *deterministic*)

class tensorforce.core.distributions.**Categorical**(*shape*, *num_actions*, *probabilities=None*, *scope='categorical'*, *summary_labels=()*)

Bases: *tensorforce.core.distributions.distribution.Distribution*

Categorical distribution, for discrete actions.

__init__(*shape*, *num_actions*, *probabilities=None*, *scope='categorical'*, *summary_labels=()*)
Categorical distribution.

Parameters

- **shape** – Action shape.
- **num_actions** – Number of discrete action alternatives.
- **probabilities** – Optional distribution bias.

from_spec(*spec*, *kwargs=None*)
Creates a distribution from a specification dict.

get_summaries()

get_variables(*include_nontrainable=False*)

state_action_value(*distr_params*, *action=None*)

state_value(*distr_params*)

tf_entropy(*distr_params*)

tf_kl_divergence(*distr_params1*, *distr_params2*)

tf_log_probability(*distr_params*, *action*)

tf_parameterize(*x*)

tf_regularization_loss()

tf_sample(*distr_params*, *deterministic*)

class tensorforce.core.distributions.**Gaussian**(*shape*, *mean=0.0*, *log_stddev=0.0*, *scope='gaussian'*, *summary_labels=()*)

Bases: *tensorforce.core.distributions.distribution.Distribution*

Gaussian distribution, for unbounded continuous actions.

__init__(*shape*, *mean=0.0*, *log_stddev=0.0*, *scope='gaussian'*, *summary_labels=()*)

Categorical distribution.

Parameters

- **shape** – Action shape.
- **mean** – Optional distribution bias for the mean.
- **log_stddev** – Optional distribution bias for the standard deviation.

from_spec(*spec*, *kwargs=None*)

Creates a distribution from a specification dict.

get_summaries()

get_variables(*include_nontrainable=False*)

state_action_value(*distr_params*, *action*)

state_value(*distr_params*)

tf_entropy(*distr_params*)

tf_kl_divergence(*distr_params1*, *distr_params2*)

tf_log_probability(*distr_params*, *action*)

tf_parameterize(*x*)

tf_regularization_loss()

tf_sample(*distr_params*, *deterministic*)

class tensorforce.core.distributions.**Beta**(*shape*, *min_value*, *max_value*, *alpha=0.0*, *beta=0.0*, *scope='beta'*, *summary_labels=()*)

Bases: *tensorforce.core.distributions.distribution.Distribution*

Beta distribution, for bounded continuous actions.

__init__(*shape*, *min_value*, *max_value*, *alpha=0.0*, *beta=0.0*, *scope='beta'*, *summary_labels=()*)

Beta distribution.

Parameters

- **shape** – Action shape.
- **min_value** – Minimum value of continuous actions.
- **max_value** – Maximum value of continuous actions.
- **alpha** – Optional distribution bias for the alpha value.
- **beta** – Optional distribution bias for the beta value.

from_spec(*spec*, *kwargs=None*)

Creates a distribution from a specification dict.

get_summaries()

get_variables(*include_nontrainable=False*)

tf_entropy(*distr_params*)

tf_kl_divergence(*distr_params1*, *distr_params2*)

tf_log_probability(*distr_params*, *action*)

tf_parameterize(*x*)

tf_regularization_loss()

tf_sample(*distr_params*, *deterministic*)

tensorforce.core.explorations package

Submodules

tensorforce.core.explorations.constant module

class tensorforce.core.explorations.constant.**Constant**(*constant=0.0*, *scope='constant'*, *summary_labels=()*)

Bases: *tensorforce.core.explorations.exploration.Exploration*

Explore via adding a constant term.

__init__(*constant=0.0*, *scope='constant'*, *summary_labels=()*)

from_spec(*spec*)
Creates an exploration object from a specification dict.

get_variables()
Returns exploration variables.

Returns List of variables.

tf_explore(*episode*, *timestep*, *action_spec=None*)

tensorforce.core.explorations.epsilon_anneal module

class tensorforce.core.explorations.epsilon_anneal.**EpsilonAnneal**(*initial_epsilon=1.0*, *final_epsilon=0.1*, *timesteps=10000*, *start_timestep=0*, *scope='epsilon_anneal'*, *summary_labels=()*)

Bases: *tensorforce.core.explorations.exploration.Exploration*

Annealing epsilon parameter based on ratio of current timestep to total timesteps.

__init__(*initial_epsilon=1.0*, *final_epsilon=0.1*, *timesteps=10000*, *start_timestep=0*, *scope='epsilon_anneal'*, *summary_labels=()*)

from_spec(*spec*)
Creates an exploration object from a specification dict.

get_variables()
Returns exploration variables.

Returns List of variables.

tf_explore(*episode*, *timestep*, *action_spec=None*)

tensorforce.core.explorations.epsilon_decay module

class tensorforce.core.explorations.epsilon_decay.**EpsilonDecay**(*initial_epsilon=1.0*, *final_epsilon=0.1*, *timesteps=10000*, *start_timestep=0*, *half_lives=10*, *scope='epsilon_anneal'*, *summary_labels=()*)

Bases: *tensorforce.core.explorations.exploration.Exploration*

Exponentially decaying epsilon parameter based on ratio of difference between current and final epsilon to total timesteps.

__init__(*initial_epsilon=1.0*, *final_epsilon=0.1*, *timesteps=10000*, *start_timestep=0*, *half_lives=10*, *scope='epsilon_anneal'*, *summary_labels=()*)

from_spec(*spec*)
Creates an exploration object from a specification dict.

get_variables()
Returns exploration variables.

Returns List of variables.

tf_explore(*episode=0*, *timestep=0*, *action_spec=None*)

tensorforce.core.explorations.exploration module

class tensorforce.core.explorations.exploration.**Exploration**(*scope='exploration'*, *summary_labels=None*)

Bases: object

Abstract exploration object.

__init__(*scope='exploration'*, *summary_labels=None*)

static from_spec(*spec*)
Creates an exploration object from a specification dict.

get_variables()
Returns exploration variables.

Returns List of variables.

tf_explore(*episode*, *timestep*, *action_spec*)
Creates exploration value, e.g. compute an epsilon for epsilon-greedy or sample normal noise.

tensorforce.core.explorations.linear_decay module

tensorforce.core.explorations.ornstein_uhlenbeck_process module

class tensorforce.core.explorations.ornstein_uhlenbeck_process.**OrnsteinUhlenbeckProcess**(*sig*
mu
the
sco
sum
ma

Bases: *tensorforce.core.explorations.exploration.Exploration*

Explores via an Ornstein-Uhlenbeck process.

__init__(*sigma=0.3*, *mu=0.0*, *theta=0.15*, *scope='ornstein_uhlenbeck'*, *summary_labels=()*)
Initializes an Ornstein-Uhlenbeck process which is a mean reverting stochastic process introducing time-correlated noise.

from_spec(*spec*)
Creates an exploration object from a specification dict.

get_variables()
Returns exploration variables.

Returns List of variables.

tf_explore(*episode*, *timestep*, *action_spec*)

Module contents

class tensorforce.core.explorations.**Exploration**(*scope='exploration'*, *summary_labels=None*)

Bases: object

Abstract exploration object.

__init__(*scope='exploration'*, *summary_labels=None*)

static from_spec(*spec*)
Creates an exploration object from a specification dict.

get_variables()
Returns exploration variables.

Returns List of variables.

tf_explore(*episode*, *timestep*, *action_spec*)
Creates exploration value, e.g. compute an epsilon for epsilon-greedy or sample normal noise.

class tensorforce.core.explorations.**Constant**(*constant=0.0*, *scope='constant'*, *summary_labels=()*)

Bases: *tensorforce.core.explorations.exploration.Exploration*

Explore via adding a constant term.

__init__(*constant=0.0*, *scope='constant'*, *summary_labels=()*)

from_spec(*spec*)
Creates an exploration object from a specification dict.

get_variables()
Returns exploration variables.

Returns List of variables.

tf_explore(*episode*, *timestep*, *action_spec=None*)

class tensorforce.core.explorations.**EpsilonAnneal**(*initial_epsilon=1.0*, *final_epsilon=0.1*, *timesteps=10000*, *start_timestep=0*, *scope='epsilon_anneal'*, *summary_labels=()*)
Bases: *tensorforce.core.explorations.exploration.Exploration*

Annealing epsilon parameter based on ratio of current timestep to total timesteps.

__init__(*initial_epsilon=1.0*, *final_epsilon=0.1*, *timesteps=10000*, *start_timestep=0*, *scope='epsilon_anneal'*, *summary_labels=()*)

from_spec(*spec*)
Creates an exploration object from a specification dict.

get_variables()
Returns exploration variables.

Returns List of variables.

tf_explore(*episode*, *timestep*, *action_spec=None*)

class tensorforce.core.explorations.**EpsilonDecay**(*initial_epsilon=1.0*, *final_epsilon=0.1*, *timesteps=10000*, *start_timestep=0*, *half_lives=10*, *scope='epsilon_anneal'*, *summary_labels=()*)
Bases: *tensorforce.core.explorations.exploration.Exploration*

Exponentially decaying epsilon parameter based on ratio of difference between current and final epsilon to total timesteps.

__init__(*initial_epsilon=1.0*, *final_epsilon=0.1*, *timesteps=10000*, *start_timestep=0*, *half_lives=10*, *scope='epsilon_anneal'*, *summary_labels=()*)

from_spec(*spec*)
Creates an exploration object from a specification dict.

get_variables()
Returns exploration variables.

Returns List of variables.

tf_explore(*episode=0*, *timestep=0*, *action_spec=None*)

class tensorforce.core.explorations.**GaussianNoise**(*sigma=0.3*, *mu=0.0*, *scope='gaussian_noise'*, *summary_labels=()*)
Bases: *tensorforce.core.explorations.exploration.Exploration*

Explores via gaussian noise.

__init__(*sigma=0.3*, *mu=0.0*, *scope='gaussian_noise'*, *summary_labels=()*)
Initializes distribution values for gaussian noise

from_spec(*spec*)
Creates an exploration object from a specification dict.

get_variables()
Returns exploration variables.

Returns List of variables.

tf_explore(*episode*, *timestep*, *action_spec*)

class tensorforce.core.explorations.**OrnsteinUhlenbeckProcess**(*sigma=0.3*, *mu=0.0*, *theta=0.15*, *scope='ornstein_uhlenbeck'*, *summary_labels=()*)
Bases: *tensorforce.core.explorations.exploration.Exploration*

Explores via an Ornstein-Uhlenbeck process.

__init__(*sigma=0.3*, *mu=0.0*, *theta=0.15*, *scope='ornstein_uhlenbeck'*, *summary_labels=()*)
Initializes an Ornstein-Uhlenbeck process which is a mean reverting stochastic process introducing time-correlated noise.

from_spec(*spec*)
Creates an exploration object from a specification dict.

get_variables()
Returns exploration variables.

Returns List of variables.

tf_explore(*episode*, *timestep*, *action_spec*)

tensorforce.core.memories package

Submodules

tensorforce.core.memories.memory module

class tensorforce.core.memories.memory.**Memory**(*states*, *internals*, *actions*, *include_next_states*, *scope='memory'*, *summary_labels=None*)
Bases: object

Base class for memories.

__init__(*states*, *internals*, *actions*, *include_next_states*, *scope='memory'*, *summary_labels=None*)
Memory.

Parameters

- **states** – States specifiction.
- **internals** – Internal states specification.
- **actions** – Actions specification.
- **include_next_states** – Include subsequent state if true.

static from_spec(*spec*, *kwargs=None*)
Creates a memory from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the memory.

Returns List of summaries.

get_variables()
Returns the TensorFlow variables used by the memory.

Returns List of variables.

tf_initialize()
Initializes memory.

tf_retrieve_episodes(*n*)
Retrieves a given number of episodes from the stored experiences.

Parameters **n** – Number of episodes to retrieve.

Returns Dicts containing the retrieved experiences.

tf_retrieve_sequences(*n*, *sequence_length*)
Retrieves a given number of temporally consistent timestep sequences from the stored experiences.

Parameters

- **n** – Number of sequences to retrieve.
- **sequence_length** – Length of timestep sequences.

Returns Dicts containing the retrieved experiences.

tf_retrieve_timesteps(*n*)
Retrieves a given number of timesteps from the stored experiences.

Parameters **n** – Number of timesteps to retrieve.

Returns Dicts containing the retrieved experiences.

tf_store(*states*, *internals*, *actions*, *terminal*, *reward*)
" Stores experiences, i.e. a batch of timesteps.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **actions** – Dict of action tensors.
- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.

tf_update_batch(*loss_per_instance*)
Updates the internal information of the latest batch instances based on their loss.

Parameters **loss_per_instance** – Loss per instance tensor.

tensorforce.core.memories.naive_prioritized_replay module

tensorforce.core.memories.prioritized_replay module

class tensorforce.core.memories.prioritized_replay.**PrioritizedReplay**(*states*, *internals*, *actions*, *include_next_states*, *capacity*, *prioritization_weight=1.0*, *buffer_size=100*, *scope='queue'*, *summary_labels=None*)

Bases: *tensorforce.core.memories.memory.Memory*

Memory organized as a priority queue, which randomly retrieves experiences sampled according their priority values.

__init__(*states*, *internals*, *actions*, *include_next_states*, *capacity*, *prioritization_weight=1.0*, *buffer_size=100*, *scope='queue'*, *summary_labels=None*)

Prioritized experience replay.

Parameters

- **states** – States specifiction.
- **internals** – Internal states specification.
- **actions** – Actions specification.
- **include_next_states** – Include subsequent state if true.
- **capacity** – Memory capacity.
- **prioritization_weight** – Prioritization weight.
- **buffer_size** – Buffer size. The buffer is used to insert experiences before experiences have been computed via updates.

from_spec(*spec*, *kwargs=None*)

Creates a memory from a specification dict.

get_summaries()

Returns the TensorFlow summaries reported by the memory.

Returns List of summaries.

get_variables()

Returns the TensorFlow variables used by the memory.

Returns List of variables.

tf_initialize()

tf_retrieve_episodes(*n*)

tf_retrieve_indices(*buffer_elements*, *priority_indices*)
Fetches experiences for given indices by combining entries from buffer which have no priorities, and entries from priority memory.

Parameters

- **buffer_elements** – Number of buffer elements to retrieve
- **priority_indices** – Index tensor for priority memory

Returns: Batch of experiences

tf_retrieve_sequences(*n*, *sequence_length*)

tf_retrieve_timesteps(*n*)

tf_store(*states*, *internals*, *actions*, *terminal*, *reward*)

tf_update_batch(*loss_per_instance*)
Updates priority memory by performing the following steps:

1. Use saved indices from prior retrieval to reconstruct the batch elements which will have their priorities updated.
2. Compute priorities for these elements.
3. Insert buffer elements to memory, potentially overwriting existing elements.
4. Update priorities of existing memory elements
5. Resort memory.
6. Update buffer insertion index.

Note that this implementation could be made more efficient by maintaining a sorted version via sum trees.

Parameters **loss_per_instance** – Losses from recent batch to perform priority update

tensorforce.core.memories.replay module

class tensorforce.core.memories.replay.**Replay**(*states*, *internals*, *actions*, *include_next_states*, *capacity*, *scope='replay'*, *summary_labels=None*)
Bases: tensorforce.core.memories.queue.Queue

Memory which randomly retrieves experiences.

__init__(*states*, *internals*, *actions*, *include_next_states*, *capacity*, *scope='replay'*, *summary_labels=None*)
Replay memory.

Parameters

- **states** – States specification.
- **internals** – Internal states specification.
- **actions** – Actions specification.
- **include_next_states** – Include subsequent state if true.
- **capacity** – Memory capacity.

from_spec(*spec*, *kwargs=None*)
Creates a memory from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the memory.

Returns List of summaries.

get_variables()
Returns the TensorFlow variables used by the memory.

Returns List of variables.

tf_initialize()

tf_retrieve_episodes(*n*)

tf_retrieve_indices(*indices*)
Fetches experiences for given indices.

Parameters **indices** – Index tensor

Returns: Batch of experiences

tf_retrieve_sequences(*n*, *sequence_length*)

tf_retrieve_timesteps(*n*)

tf_store(*states*, *internals*, *actions*, *terminal*, *reward*)

tf_update_batch(*loss_per_instance*)
Updates the internal information of the latest batch instances based on their loss.

Parameters **loss_per_instance** – Loss per instance tensor.

Module contents

class tensorforce.core.memories.**Memory**(*states*, *internals*, *actions*, *include_next_states*, *scope='memory'*, *summary_labels=None*)

Bases: object

Base class for memories.

__init__(*states*, *internals*, *actions*, *include_next_states*, *scope='memory'*, *summary_labels=None*)
Memory.

Parameters

- **states** – States specifiction.
- **internals** – Internal states specification.
- **actions** – Actions specification.
- **include_next_states** – Include subsequent state if true.

static from_spec(*spec*, *kwargs=None*)
Creates a memory from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the memory.

Returns List of summaries.

get_variables()
Returns the TensorFlow variables used by the memory.

Returns List of variables.

tf_initialize()
Initializes memory.

tf_retrieve_episodes(*n*)
Retrieves a given number of episodes from the stored experiences.

Parameters **n** – Number of episodes to retrieve.

Returns Dicts containing the retrieved experiences.

tf_retrieve_sequences(*n*, *sequence_length*)
Retrieves a given number of temporally consistent timestep sequences from the stored experiences.

Parameters

- **n** – Number of sequences to retrieve.
- **sequence_length** – Length of timestep sequences.

Returns Dicts containing the retrieved experiences.

tf_retrieve_timesteps(*n*)
Retrieves a given number of timesteps from the stored experiences.

Parameters **n** – Number of timesteps to retrieve.

Returns Dicts containing the retrieved experiences.

tf_store(*states*, *internals*, *actions*, *terminal*, *reward*)
" Stores experiences, i.e. a batch of timesteps.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **actions** – Dict of action tensors.
- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.

tf_update_batch(*loss_per_instance*)
Updates the internal information of the latest batch instances based on their loss.

Parameters **loss_per_instance** – Loss per instance tensor.

class tensorforce.core.memories.**Queue**(*states*, *internals*, *actions*, *include_next_states*, *capacity*, *scope='queue'*, *summary_labels=None*)
Bases: *tensorforce.core.memories.memory.Memory*

Base class for memories organized as a queue (FIFO).

__init__(*states*, *internals*, *actions*, *include_next_states*, *capacity*, *scope='queue'*, *summary_labels=None*)
Queue memory.

Parameters

- **states** – States specifiction.
- **internals** – Internal states specification.
- **actions** – Actions specification.
- **include_next_states** – Include subsequent state if true.
- **capacity** – Memory capacity.

from_spec(*spec*, *kwargs=None*)
Creates a memory from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the memory.

Returns List of summaries.

get_variables()
Returns the TensorFlow variables used by the memory.

Returns List of variables.

tf_initialize()

tf_retrieve_episodes(*n*)
Retrieves a given number of episodes from the stored experiences.

Parameters **n** – Number of episodes to retrieve.

Returns Dicts containing the retrieved experiences.

tf_retrieve_indices(*indices*)
Fetches experiences for given indices.

Parameters **indices** – Index tensor

Returns: Batch of experiences

tf_retrieve_sequences(*n*, *sequence_length*)
Retrieves a given number of temporally consistent timestep sequences from the stored experiences.

Parameters

- **n** – Number of sequences to retrieve.
- **sequence_length** – Length of timestep sequences.

Returns Dicts containing the retrieved experiences.

tf_retrieve_timesteps(*n*)
Retrieves a given number of timesteps from the stored experiences.

Parameters **n** – Number of timesteps to retrieve.

Returns Dicts containing the retrieved experiences.

tf_store(*states*, *internals*, *actions*, *terminal*, *reward*)

tf_update_batch(*loss_per_instance*)
Updates the internal information of the latest batch instances based on their loss.

Parameters **loss_per_instance** – Loss per instance tensor.

class tensorforce.core.memories.**Latest**(*states*, *internals*, *actions*, *include_next_states*, *capacity*, *scope='latest'*, *summary_labels=None*)
Bases: tensorforce.core.memories.queue.Queue

Memory which always retrieves most recent experiences.

__init__(*states*, *internals*, *actions*, *include_next_states*, *capacity*, *scope='latest'*, *summary_labels=None*)
Latest memory.

Parameters

- **states** – States specifiction.

- **internals** – Internal states specification.
- **actions** – Actions specification.
- **include_next_states** – Include subsequent state if true.
- **capacity** – Memory capacity.

from_spec(*spec*, *kwargs=None*)
Creates a memory from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the memory.

Returns List of summaries.

get_variables()
Returns the TensorFlow variables used by the memory.

Returns List of variables.

tf_initialize()

tf_retrieve_episodes(*n*)

tf_retrieve_indices(*indices*)
Fetches experiences for given indices.

Parameters **indices** – Index tensor

Returns: Batch of experiences

tf_retrieve_sequences(*n*, *sequence_length*)

tf_retrieve_timesteps(*n*)

tf_store(*states*, *internals*, *actions*, *terminal*, *reward*)

tf_update_batch(*loss_per_instance*)
Updates the internal information of the latest batch instances based on their loss.

Parameters **loss_per_instance** – Loss per instance tensor.

class tensorforce.core.memories.**Replay**(*states*, *internals*, *actions*, *include_next_states*, *capacity*, *scope='replay'*, *summary_labels=None*)
Bases: tensorforce.core.memories.queue.Queue

Memory which randomly retrieves experiences.

__init__(*states*, *internals*, *actions*, *include_next_states*, *capacity*, *scope='replay'*, *summary_labels=None*)
Replay memory.

Parameters

- **states** – States specification.
- **internals** – Internal states specification.
- **actions** – Actions specification.
- **include_next_states** – Include subsequent state if true.
- **capacity** – Memory capacity.

from_spec(*spec*, *kwargs=None*)
Creates a memory from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the memory.

Returns List of summaries.

get_variables()
Returns the TensorFlow variables used by the memory.

Returns List of variables.

tf_initialize()

tf_retrieve_episodes(*n*)

tf_retrieve_indices(*indices*)
Fetches experiences for given indices.

Parameters **indices** – Index tensor

Returns: Batch of experiences

tf_retrieve_sequences(*n*, *sequence_length*)

tf_retrieve_timesteps(*n*)

tf_store(*states*, *internals*, *actions*, *terminal*, *reward*)

tf_update_batch(*loss_per_instance*)
Updates the internal information of the latest batch instances based on their loss.

Parameters **loss_per_instance** – Loss per instance tensor.

class tensorforce.core.memories.**PrioritizedReplay**(*states*, *internals*, *actions*, *include_next_states*, *capacity*, *prioritization_weight=1.0*, *buffer_size=100*, *scope='queue'*, *summary_labels=None*)
Bases: *tensorforce.core.memories.memory.Memory*

Memory organized as a priority queue, which randomly retrieves experiences sampled according their priority values.

__init__(*states*, *internals*, *actions*, *include_next_states*, *capacity*, *prioritization_weight=1.0*, *buffer_size=100*, *scope='queue'*, *summary_labels=None*)
Prioritized experience replay.

Parameters

- **states** – States specifiction.
- **internals** – Internal states specification.
- **actions** – Actions specification.
- **include_next_states** – Include subsequent state if true.
- **capacity** – Memory capacity.
- **prioritization_weight** – Prioritization weight.
- **buffer_size** – Buffer size. The buffer is used to insert experiences before experiences have been computed via updates.

from_spec(*spec*, *kwargs=None*)
Creates a memory from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the memory.

Returns List of summaries.

get_variables()
Returns the TensorFlow variables used by the memory.

Returns List of variables.

tf_initialize()

tf_retrieve_episodes(*n*)

tf_retrieve_indices(*buffer_elements*, *priority_indices*)
Fetches experiences for given indices by combining entries from buffer which have no priorities, and entries from priority memory.

Parameters

- **buffer_elements** – Number of buffer elements to retrieve
- **priority_indices** – Index tensor for priority memory

Returns: Batch of experiences

tf_retrieve_sequences(*n*, *sequence_length*)

tf_retrieve_timesteps(*n*)

tf_store(*states*, *internals*, *actions*, *terminal*, *reward*)

tf_update_batch(*loss_per_instance*)
Updates priority memory by performing the following steps:

1. Use saved indices from prior retrieval to reconstruct the batch elements which will have their priorities updated.
2. Compute priorities for these elements.
3. Insert buffer elements to memory, potentially overwriting existing elements.
4. Update priorities of existing memory elements
5. Resort memory.
6. Update buffer insertion index.

Note that this implementation could be made more efficient by maintaining a sorted version via sum trees.

Parameters **loss_per_instance** – Losses from recent batch to perform priority update

tensorforce.core.networks package

Submodules

tensorforce.core.networks.complex_network module

class tensorforce.core.networks.complex_network.**ComplexLayeredNetwork**(*complex_layers_spec*, *scope='layered-network'*, *summary_labels=()*)

Bases: *tensorforce.core.networks.network.LayerBasedNetwork*

Complex Network consisting of a sequence of layers, which can be created from a specification dict.

__init__(*complex_layers_spec*, *scope='layered-network'*, *summary_labels=()*)

Complex Layered network.

Parameters **complex_layers_spec** – List of layer specification dicts

add_layer(*layer*)

static from_json(*filename*)

Creates a complex_layered_network_builder from a JSON.

Parameters **filename** – Path to configuration

Returns: A ComplexLayeredNetwork class with layers generated from the JSON

from_spec(*spec*, *kwargs=None*)

Creates a network from a specification dict.

get_list_of_named_tensor()

Returns a list of the names of tensors available.

Returns List of the names of tensors available.

get_named_tensor(*name*)

Returns a named tensor if available.

Returns True if named tensor found, False otherwise tensor: If valid, will be a tensor, otherwise None

Return type valid

get_summaries()

get_variables(*include_nontrainable=False*)

internals_spec()

set_named_tensor(*name*, *tensor*)

Returns the TensorFlow summaries reported by the network.

Returns None

tf_apply(*x*, *internals*, *update*, *return_internals=False*)

tf_regularization_loss()

class tensorforce.core.networks.complex_network.**Input**(*inputs*, *axis=1*, *scope='merge_inputs'*, *summary_labels=()*)

Bases: *tensorforce.core.networks.layer.Layer*

Input layer. Used for ComplexLayerNetwork's to collect data together as a form of output to the next layer. Allows for multiple inputs to merge into a single import for next layer.

__init__(*inputs*, *axis=1*, *scope='merge_inputs'*, *summary_labels=()*)
Input layer.

Parameters

- **inputs** – A list of strings that name the inputs to merge
- **axis** – Axis to merge the inputs

from_spec(*spec*, *kwargs=None*)
Creates a layer from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the layer.

Returns List of summaries.

get_variables(*include_nontrainable=False*)
Returns the TensorFlow variables used by the layer.

Returns List of variables.

internals_spec()
Returns the internal states specification.

Returns Internal states specification

tf_apply(*x*, *update*)

tf_regularization_loss()
Creates the TensorFlow operations for the layer regularization loss.

Returns Regularization loss tensor.

tf_tensors(*named_tensors*)
Attaches the named_tensors dictionary to the layer for examination and update.

Parameters **named_tensors** – Dictionary of named tensors to be used as Input's or recorded from outputs

Returns NA

class tensorforce.core.networks.complex_network.**Output**(*output*, *scope='output'*, *summary_labels=()*)
Bases: *tensorforce.core.networks.layer.Layer*

Output layer. Used for ComplexLayerNetwork's to capture the tensor under and name for use with Input layers. Acts as a input to output passthrough.

__init__(*output*, *scope='output'*, *summary_labels=()*)
Output layer.

Parameters **output** – A string that names the tensor, will be added to available inputs

from_spec(*spec*, *kwargs=None*)
Creates a layer from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the layer.

Returns List of summaries.

get_variables(*include_nontrainable=False*)
Returns the TensorFlow variables used by the layer.

Returns List of variables.

internals_spec()
Returns the internal states specification.

Returns Internal states specification

tf_apply(*x*, *update*)

tf_regularization_loss()
Creates the TensorFlow operations for the layer regularization loss.

Returns Regularization loss tensor.

tf_tensors(*named_tensors*)
Attaches the named_tensors dictionary to the layer for examination and update.

Parameters **named_tensors** – Dictionary of named tensors to be used as Input's or recorded from outputs

Returns NA

tensorforce.core.networks.layer module

Collection of custom layer implementations.

class tensorforce.core.networks.layer.**Conv1d**(*size*, *window=3*, *stride=1*, *padding='SAME'*, *bias=True*, *activation='relu'*, *l2_regularization=0.0*, *l1_regularization=0.0*, *scope='conv1d'*, *summary_labels=()*)
Bases: *tensorforce.core.networks.layer.Layer*

1-dimensional convolutional layer.

__init__(*size*, *window=3*, *stride=1*, *padding='SAME'*, *bias=True*, *activation='relu'*, *l2_regularization=0.0*, *l1_regularization=0.0*, *scope='conv1d'*, *summary_labels=()*)
1D convolutional layer.

Parameters

- **size** – Number of filters
- **window** – Convolution window size
- **stride** – Convolution stride
- **padding** – Convolution padding, one of 'VALID' or 'SAME'
- **bias** – If true, a bias is added
- **activation** – Type of nonlinearity, or dict with name & arguments
- **l2_regularization** – L2 regularization weight
- **l1_regularization** – L1 regularization weight

from_spec(*spec*, *kwargs=None*)
Creates a layer from a specification dict.

get_summaries()

get_variables(*include_nontrainable=False*)

internals_spec()
Returns the internal states specification.

Returns Internal states specification

tf_apply(*x*, *update*)

tf_regularization_loss()

tf_tensors(*named_tensors*)
Attaches the named_tensors dictionary to the layer for examination and update.

Parameters **named_tensors** – Dictionary of named tensors to be used as Input's or recorded from outputs

Returns NA

class tensorforce.core.networks.layer.**Conv2d**(*size*, *window=3*, *stride=1*, *padding='SAME'*, *bias=True*, *activation='relu'*, *l2_regularization=0.0*, *l1_regularization=0.0*, *scope='conv2d'*, *summary_labels=()*)
Bases: *tensorforce.core.networks.layer.Layer*

2-dimensional convolutional layer.

__init__(*size*, *window=3*, *stride=1*, *padding='SAME'*, *bias=True*, *activation='relu'*, *l2_regularization=0.0*, *l1_regularization=0.0*, *scope='conv2d'*, *summary_labels=()*)
2D convolutional layer.

Parameters

- **size** – Number of filters
- **window** – Convolution window size, either an integer or pair of integers.
- **stride** – Convolution stride, either an integer or pair of integers.
- **padding** – Convolution padding, one of 'VALID' or 'SAME'
- **bias** – If true, a bias is added
- **activation** – Type of nonlinearity, or dict with name & arguments
- **l2_regularization** – L2 regularization weight
- **l1_regularization** – L1 regularization weight

from_spec(*spec*, *kwargs=None*)
Creates a layer from a specification dict.

get_summaries()

get_variables(*include_nontrainable=False*)

internals_spec()
Returns the internal states specification.

Returns Internal states specification

tf_apply(*x*, *update*)

tf_regularization_loss()

tf_tensors(*named_tensors*)
Attaches the named_tensors dictionary to the layer for examination and update.

Parameters **named_tensors** – Dictionary of named tensors to be used as Input's or recorded from outputs

Returns NA

class tensorforce.core.networks.layer.**Dense**(*size=None*, *weights=None*, *bias=True*, *activation='relu'*, *l2_regularization=0.0*, *l1_regularization=0.0*, *skip=False*, *scope='dense'*, *summary_labels=()*)

Bases: *tensorforce.core.networks.layer.Layer*

Dense layer, i.e. linear fully connected layer with subsequent non-linearity.

__init__(*size=None*, *weights=None*, *bias=True*, *activation='relu'*, *l2_regularization=0.0*, *l1_regularization=0.0*, *skip=False*, *scope='dense'*, *summary_labels=()*)

Dense layer.

Parameters

- **size** – Layer size, if None than input size matches the output size of the layer
- **weights** – Weight initialization, random if None.
- **bias** – If true, bias is added.
- **activation** – Type of nonlinearity, or dict with name & arguments
- **l2_regularization** – L2 regularization weight.
- **l1_regularization** – L1 regularization weight.
- **skip** – Add skip connection like ResNet (https://arxiv.org/pdf/1512.03385.pdf), doubles layers and ShortCut from Input to output

from_spec(*spec*, *kwargs=None*)

Creates a layer from a specification dict.

get_summaries()

get_variables(*include_nontrainable=False*)

internals_spec()

Returns the internal states specification.

Returns Internal states specification

tf_apply(*x*, *update*)

tf_regularization_loss()

tf_tensors(*named_tensors*)

Attaches the named_tensors dictionary to the layer for examination and update.

Parameters **named_tensors** – Dictionary of named tensors to be used as Input's or recorded from outputs

Returns NA

class tensorforce.core.networks.layer.**Dropout**(*rate=0.0*, *scope='dropout'*, *summary_labels=()*)

Bases: *tensorforce.core.networks.layer.Layer*

Dropout layer. If using dropout, add this layer after inputs and after dense layers. For LSTM, dropout is handled independently as an argument. Not available for Conv2d yet.

__init__(*rate=0.0*, *scope='dropout'*, *summary_labels=()*)

from_spec(*spec*, *kwargs=None*)
Creates a layer from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the layer.

Returns List of summaries.

get_variables(*include_nontrainable=False*)
Returns the TensorFlow variables used by the layer.

Returns List of variables.

internals_spec()
Returns the internal states specification.

Returns Internal states specification

tf_apply(*x*, *update*)

tf_regularization_loss()
Creates the TensorFlow operations for the layer regularization loss.

Returns Regularization loss tensor.

tf_tensors(*named_tensors*)
Attaches the named_tensors dictionary to the layer for examination and update.

Parameters **named_tensors** – Dictionary of named tensors to be used as Input's or recorded from outputs

Returns NA

class tensorforce.core.networks.layer.**Dueling**(*size*, *bias=False*, *activation='none'*, *l2_regularization=0.0*, *l1_regularization=0.0*, *output=None*, *scope='dueling'*, *summary_labels=()*)
Bases: *tensorforce.core.networks.layer.Layer*

Dueling layer, i.e. Duel pipelines for Exp & Adv to help with stability

__init__(*size*, *bias=False*, *activation='none'*, *l2_regularization=0.0*, *l1_regularization=0.0*, *output=None*, *scope='dueling'*, *summary_labels=()*)
Dueling layer.

[Dueling Networks] (https://arxiv.org/pdf/1511.06581.pdf) Implement Y = Expectation[x] + (Advantage[x] - Mean(Advantage[x]))

Parameters

- **size** – Layer size.
- **bias** – If true, bias is added.
- **activation** – Type of nonlinearity, or dict with name & arguments
- **l2_regularization** – L2 regularization weight.
- **l1_regularization** – L1 regularization weight.
- **output** – None or tuple of output names for ('expectation','advantage','mean_advantage')

from_spec(*spec*, *kwargs=None*)
Creates a layer from a specification dict.

get_summaries()

get_variables(*include_nontrainable=False*)

internals_spec()
Returns the internal states specification.

Returns Internal states specification

tf_apply(*x*, *update*)

tf_regularization_loss()

tf_tensors(*named_tensors*)
Attaches the named_tensors dictionary to the layer for examination and update.

Parameters **named_tensors** – Dictionary of named tensors to be used as Input's or recorded from outputs

Returns NA

class tensorforce.core.networks.layer.**Embedding**(*indices*, *size*, *l2_regularization=0.0*, *l1_regularization=0.0*, *scope='embedding'*, *summary_labels=()*)
Bases: *tensorforce.core.networks.layer.Layer*

Embedding layer.

__init__(*indices*, *size*, *l2_regularization=0.0*, *l1_regularization=0.0*, *scope='embedding'*, *summary_labels=()*)
Embedding layer.

Parameters

- **indices** – Number of embedding indices.
- **size** – Embedding size.
- **l2_regularization** – L2 regularization weight.
- **l1_regularization** – L1 regularization weight.

from_spec(*spec*, *kwargs=None*)
Creates a layer from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the layer.

Returns List of summaries.

get_variables(*include_nontrainable=False*)
Returns the TensorFlow variables used by the layer.

Returns List of variables.

internals_spec()
Returns the internal states specification.

Returns Internal states specification

tf_apply(*x*, *update*)

tf_regularization_loss()

tf_tensors(*named_tensors*)
Attaches the named_tensors dictionary to the layer for examination and update.

Parameters **named_tensors** – Dictionary of named tensors to be used as Input's or recorded from outputs

Returns NA

class tensorforce.core.networks.layer.**Flatten**(*scope='flatten'*, *summary_labels=()*)

Bases: *tensorforce.core.networks.layer.Layer*

Flatten layer reshaping the input.

__init__(*scope='flatten'*, *summary_labels=()*)

from_spec(*spec*, *kwargs=None*)
Creates a layer from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the layer.

Returns List of summaries.

get_variables(*include_nontrainable=False*)
Returns the TensorFlow variables used by the layer.

Returns List of variables.

internals_spec()
Returns the internal states specification.

Returns Internal states specification

tf_apply(*x*, *update*)

tf_regularization_loss()
Creates the TensorFlow operations for the layer regularization loss.

Returns Regularization loss tensor.

tf_tensors(*named_tensors*)
Attaches the named_tensors dictionary to the layer for examination and update.

Parameters **named_tensors** – Dictionary of named tensors to be used as Input's or recorded from outputs

Returns NA

class tensorforce.core.networks.layer.**InternalLstm**(*size*, *dropout=None*, *lstmcell_args={}*, *scope='internal_lstm'*, *summary_labels=()*)

Bases: *tensorforce.core.networks.layer.Layer*

Long short-term memory layer for internal state management.

__init__(*size*, *dropout=None*, *lstmcell_args={}*, *scope='internal_lstm'*, *summary_labels=()*)
LSTM layer.

Parameters

- **size** – LSTM size.
- **dropout** – Dropout rate.

from_spec(*spec*, *kwargs=None*)
Creates a layer from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the layer.

Returns List of summaries.

get_variables(*include_nontrainable=False*)
Returns the TensorFlow variables used by the layer.

Returns List of variables.

internals_spec()

tf_apply(*x*, *update*, *state*)

tf_regularization_loss()
Creates the TensorFlow operations for the layer regularization loss.

Returns Regularization loss tensor.

tf_tensors(*named_tensors*)
Attaches the named_tensors dictionary to the layer for examination and update.

Parameters **named_tensors** – Dictionary of named tensors to be used as Input's or recorded from outputs

Returns NA

class tensorforce.core.networks.layer.**Layer**(*scope='layer'*, *summary_labels=None*)
Bases: object

Base class for network layers.

__init__(*scope='layer'*, *summary_labels=None*)
Layer.

static from_spec(*spec*, *kwargs=None*)
Creates a layer from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the layer.

Returns List of summaries.

get_variables(*include_nontrainable=False*)
Returns the TensorFlow variables used by the layer.

Returns List of variables.

internals_spec()
Returns the internal states specification.

Returns Internal states specification

tf_apply(*x*, *update*)
Creates the TensorFlow operations for applying the layer to the given input.

Parameters

- **x** – Layer input tensor.
- **update** – Boolean tensor indicating whether this call happens during an update.

Returns Layer output tensor.

tf_regularization_loss()
Creates the TensorFlow operations for the layer regularization loss.

Returns Regularization loss tensor.

tf_tensors(*named_tensors*)
Attaches the named_tensors dictionary to the layer for examination and update.

Parameters **named_tensors** – Dictionary of named tensors to be used as Input's or recorded from outputs

Returns NA

class tensorforce.core.networks.layer.**Linear**(*size*, *weights=None*, *bias=True*, *l2_regularization=0.0*, *l1_regularization=0.0*, *scope='linear'*, *summary_labels=()*)
Bases: *tensorforce.core.networks.layer.Layer*

Linear fully-connected layer.

__init__(*size*, *weights=None*, *bias=True*, *l2_regularization=0.0*, *l1_regularization=0.0*, *scope='linear'*, *summary_labels=()*)
Linear layer.

Parameters

- **size** – Layer size.
- **weights** – Weight initialization, random if None.
- **bias** – Bias initialization, random if True, no bias added if False.
- **l2_regularization** – L2 regularization weight.
- **l1_regularization** – L1 regularization weight.

from_spec(*spec*, *kwargs=None*)
Creates a layer from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the layer.

Returns List of summaries.

get_variables(*include_nontrainable=False*)
Returns the TensorFlow variables used by the layer.

Returns List of variables.

internals_spec()
Returns the internal states specification.

Returns Internal states specification

tf_apply(*x*, *update=False*)

tf_regularization_loss()

tf_tensors(*named_tensors*)
Attaches the named_tensors dictionary to the layer for examination and update.

Parameters **named_tensors** – Dictionary of named tensors to be used as Input's or recorded from outputs

Returns NA

class tensorforce.core.networks.layer.**Lstm**(*size*, *dropout=None*, *scope='lstm'*, *summary_labels=()*, *return_final_state=True*)
Bases: *tensorforce.core.networks.layer.Layer*

__init__(*size*, *dropout=None*, *scope='lstm'*, *summary_labels=()*, *return_final_state=True*)
LSTM layer.

Parameters

- **size** – LSTM size.
- **dropout** – Dropout rate.

from_spec(*spec*, *kwargs=None*)
Creates a layer from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the layer.

Returns List of summaries.

get_variables(*include_nontrainable=False*)
Returns the TensorFlow variables used by the layer.

Returns List of variables.

internals_spec()
Returns the internal states specification.

Returns Internal states specification

tf_apply(*x*, *update*, *sequence_length=None*)

tf_regularization_loss()
Creates the TensorFlow operations for the layer regularization loss.

Returns Regularization loss tensor.

tf_tensors(*named_tensors*)
Attaches the named_tensors dictionary to the layer for examination and update.

Parameters **named_tensors** – Dictionary of named tensors to be used as Input's or recorded from outputs

Returns NA

class tensorforce.core.networks.layer.**Nonlinearity**(*name='relu'*, *alpha=None*, *beta=1.0*, *max=None*, *min=None*, *scope='nonlinearity'*, *summary_labels=()*)
Bases: *tensorforce.core.networks.layer.Layer*

Non-linearity layer applying a non-linear transformation.

__init__(*name='relu'*, *alpha=None*, *beta=1.0*, *max=None*, *min=None*, *scope='nonlinearity'*, *summary_labels=()*)
Non-linearity activation layer.

Parameters

- **name** – Non-linearity name, one of 'elu', 'relu', 'selu', 'sigmoid', 'swish', 'softmax', 'leaky_relu' (or 'lrelu'), 'crelu', 'softmax', 'softplus', 'softsign', 'tanh' or 'none'.
- **alpha** – (floatlint) Alpha value for leaky Relu
- **beta** – (floatlintl'learn') Beta value or 'learn' to train value (default 1.0)
- **max** – (floatlint) maximum (beta * input) value passed to non-linearity function
- **min** – (floatlint) minimum (beta * input) value passed to non-linearity function

- **summary_labels** – Requested summary labels for tensorboard export, add 'beta' to watch beta learning

from_spec(*spec*, *kwargs=None*)
Creates a layer from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the layer.

Returns List of summaries.

get_variables(*include_nontrainable=False*)
Returns the TensorFlow variables used by the layer.

Returns List of variables.

internals_spec()
Returns the internal states specification.

Returns Internal states specification

tf_apply(*x*, *update*)

tf_regularization_loss()
Creates the TensorFlow operations for the layer regularization loss.

Returns Regularization loss tensor.

tf_tensors(*named_tensors*)
Attaches the named_tensors dictionary to the layer for examination and update.

Parameters **named_tensors** – Dictionary of named tensors to be used as Input's or recorded from outputs

Returns NA

class tensorforce.core.networks.layer.**Pool2d**(*pooling_type='max'*, *window=2*, *stride=2*, *padding='SAME'*, *scope='pool2d'*, *summary_labels=()*)
Bases: *tensorforce.core.networks.layer.Layer*

2-dimensional pooling layer.

__init__(*pooling_type='max'*, *window=2*, *stride=2*, *padding='SAME'*, *scope='pool2d'*, *summary_labels=()*)
2-dimensional pooling layer.

Parameters

- **pooling_type** – Either 'max' or 'average'.
- **window** – Pooling window size, either an integer or pair of integers.
- **stride** – Pooling stride, either an integer or pair of integers.
- **padding** – Pooling padding, one of 'VALID' or 'SAME'.

from_spec(*spec*, *kwargs=None*)
Creates a layer from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the layer.

Returns List of summaries.

get_variables(*include_nontrainable=False*)
Returns the TensorFlow variables used by the layer.

Returns List of variables.

internals_spec()
Returns the internal states specification.

Returns Internal states specification

tf_apply(*x*, *update*)

tf_regularization_loss()
Creates the TensorFlow operations for the layer regularization loss.

Returns Regularization loss tensor.

tf_tensors(*named_tensors*)
Attaches the named_tensors dictionary to the layer for examination and update.

Parameters **named_tensors** – Dictionary of named tensors to be used as Input's or recorded from outputs

Returns NA

class tensorforce.core.networks.layer.**TFLayer**(*layer*, *scope='tf-layer'*, *summary_labels=()*, ***kwargs*)
Bases: *tensorforce.core.networks.layer.Layer*

Wrapper class for TensorFlow layers.

__init__(*layer*, *scope='tf-layer'*, *summary_labels=()*, ***kwargs*)
Creates a new layer instance of a TensorFlow layer.

Parameters

- **name** – The name of the layer, one of 'dense'.
- ****kwargs** – Additional arguments passed on to the TensorFlow layer constructor.

from_spec(*spec*, *kwargs=None*)
Creates a layer from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the layer.

Returns List of summaries.

get_variables(*include_nontrainable=False*)
Returns the TensorFlow variables used by the layer.

Returns List of variables.

internals_spec()
Returns the internal states specification.

Returns Internal states specification

tf_apply(*x*, *update*)

tf_layers = {'conv1d': <sphinx.ext.autodoc._MockObject object>, 'dropout': <sphinx.e

tf_regularization_loss()

tf_tensors(*named_tensors*)
Attaches the named_tensors dictionary to the layer for examination and update.

Parameters **named_tensors** – Dictionary of named tensors to be used as Input's or recorded from outputs

Returns NA

tensorforce.core.networks.network module

class tensorforce.core.networks.network.**LayerBasedNetwork**(*scope='layerbased-network'*, *summary_labels=()*)

Bases: *tensorforce.core.networks.network.Network*

Base class for networks using TensorForce layers.

__init__(*scope='layerbased-network'*, *summary_labels=()*)
Layer-based network.

add_layer(*layer*)

from_spec(*spec*, *kwargs=None*)
Creates a network from a specification dict.

get_list_of_named_tensor()
Returns a list of the names of tensors available.

Returns List of the names of tensors available.

get_named_tensor(*name*)
Returns a named tensor if available.

Returns True if named tensor found, False otherwise tensor: If valid, will be a tensor, otherwise None

Return type valid

get_summaries()

get_variables(*include_nontrainable=False*)

internals_spec()

set_named_tensor(*name*, *tensor*)
Returns the TensorFlow summaries reported by the network.

Returns None

tf_apply(*x*, *internals*, *update*, *return_internals=False*)
Creates the TensorFlow operations for applying the network to the given input.

Parameters

- **x** – Network input tensor or dict of input tensors.
- **internals** – List of prior internal state tensors
- **update** – Boolean tensor indicating whether this call happens during an update.
- **return_internals** – If true, also returns posterior internal state tensors

Returns Network output tensor, plus optionally list of posterior internal state tensors

tf_regularization_loss()

class tensorforce.core.networks.network.**LayeredNetwork**(*layers*, *scope='layered-network'*, *summary_labels=()*)

Bases: *tensorforce.core.networks.network.LayerBasedNetwork*

Network consisting of a sequence of layers, which can be created from a specification dict.

__init__(*layers*, *scope='layered-network'*, *summary_labels=()*)
Single-stack layered network.

Parameters **layers** – List of layer specification dicts.

add_layer(*layer*)

static from_json(*filename*)
Creates a layer_networkd_builder from a JSON.

Parameters **filename** – Path to configuration

Returns: A layered_network_builder function with layers generated from the JSON

from_spec(*spec*, *kwargs=None*)
Creates a network from a specification dict.

get_list_of_named_tensor()
Returns a list of the names of tensors available.

Returns List of the names of tensors available.

get_named_tensor(*name*)
Returns a named tensor if available.

Returns True if named tensor found, False otherwise tensor: If valid, will be a tensor, otherwise None

Return type valid

get_summaries()

get_variables(*include_nontrainable=False*)

internals_spec()

set_named_tensor(*name*, *tensor*)
Returns the TensorFlow summaries reported by the network.

Returns None

tf_apply(*x*, *internals*, *update*, *return_internals=False*)

tf_regularization_loss()

class tensorforce.core.networks.network.**Network**(*scope='network'*, *summary_labels=None*)

Bases: object

Base class for neural networks.

__init__(*scope='network'*, *summary_labels=None*)
Neural network.

static from_spec(*spec*, *kwargs=None*)
Creates a network from a specification dict.

get_list_of_named_tensor()
Returns a list of the names of tensors available.

Returns List of the names of tensors available.

get_named_tensor(*name*)
Returns a named tensor if available.

Returns True if named tensor found, False otherwise tensor: If valid, will be a tensor, otherwise None

Return type valid

get_summaries()
Returns the TensorFlow summaries reported by the network.

Returns List of summaries

get_variables(*include_nontrainable=False*)
Returns the TensorFlow variables used by the network.

Returns List of variables

internals_spec()
Returns the internal states specification.

Returns Internal states specification

set_named_tensor(*name*, *tensor*)
Returns the TensorFlow summaries reported by the network.

Returns None

tf_apply(*x*, *internals*, *update*, *return_internals=False*)
Creates the TensorFlow operations for applying the network to the given input.

Parameters

- **x** – Network input tensor or dict of input tensors.
- **internals** – List of prior internal state tensors
- **update** – Boolean tensor indicating whether this call happens during an update.
- **return_internals** – If true, also returns posterior internal state tensors

Returns Network output tensor, plus optionally list of posterior internal state tensors

tf_regularization_loss()
Creates the TensorFlow operations for the network regularization loss.

Returns Regularization loss tensor

Module contents

class tensorforce.core.networks.**Layer**(*scope='layer'*, *summary_labels=None*)
Bases: object

Base class for network layers.

__init__(*scope='layer'*, *summary_labels=None*)
Layer.

static from_spec(*spec*, *kwargs=None*)
Creates a layer from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the layer.

Returns List of summaries.

get_variables(*include_nontrainable=False*)
Returns the TensorFlow variables used by the layer.

Returns List of variables.

internals_spec()
Returns the internal states specification.

Returns Internal states specification

tf_apply(*x*, *update*)
Creates the TensorFlow operations for applying the layer to the given input.

Parameters

- **x** – Layer input tensor.
- **update** – Boolean tensor indicating whether this call happens during an update.

Returns Layer output tensor.

tf_regularization_loss()
Creates the TensorFlow operations for the layer regularization loss.

Returns Regularization loss tensor.

tf_tensors(*named_tensors*)
Attaches the named_tensors dictionary to the layer for examination and update.

Parameters **named_tensors** – Dictionary of named tensors to be used as Input's or recorded from outputs

Returns NA

class tensorforce.core.networks.**TFLayer**(*layer*, *scope='tf-layer'*, *summary_labels=()*, ***kwargs*)
Bases: *tensorforce.core.networks.layer.Layer*

Wrapper class for TensorFlow layers.

__init__(*layer*, *scope='tf-layer'*, *summary_labels=()*, ***kwargs*)
Creates a new layer instance of a TensorFlow layer.

Parameters

- **name** – The name of the layer, one of 'dense'.
- ****kwargs** – Additional arguments passed on to the TensorFlow layer constructor.

from_spec(*spec*, *kwargs=None*)
Creates a layer from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the layer.

Returns List of summaries.

get_variables(*include_nontrainable=False*)
Returns the TensorFlow variables used by the layer.

Returns List of variables.

internals_spec()
Returns the internal states specification.

Returns Internal states specification

tf_apply(*x*, *update*)

tf_layers = {'conv1d': <sphinx.ext.autodoc._MockObject object>, 'dropout': <sphinx.e

tf_regularization_loss()

tf_tensors(*named_tensors*)
Attaches the named_tensors dictionary to the layer for examination and update.

Parameters **named_tensors** – Dictionary of named tensors to be used as Input's or recorded from outputs

Returns NA

class tensorforce.core.networks.**Nonlinearity**(*name='relu'*, *alpha=None*, *beta=1.0*, *max=None*, *min=None*, *scope='nonlinearity'*, *summary_labels=()*)
Bases: *tensorforce.core.networks.layer.Layer*

Non-linearity layer applying a non-linear transformation.

__init__(*name='relu'*, *alpha=None*, *beta=1.0*, *max=None*, *min=None*, *scope='nonlinearity'*, *summary_labels=()*)
Non-linearity activation layer.

Parameters

- **name** – Non-linearity name, one of 'elu', 'relu', 'selu', 'sigmoid', 'swish', 'softmax', 'leaky_relu' (or 'lrelu'), 'crelu', 'softmax', 'softplus', 'softsign', 'tanh' or 'none'.
- **alpha** – (float|int) Alpha value for leaky Relu
- **beta** – (float|int|'learn') Beta value or 'learn' to train value (default 1.0)
- **max** – (float|int) maximum (beta * input) value passed to non-linearity function
- **min** – (float|int) minimum (beta * input) value passed to non-linearity function
- **summary_labels** – Requested summary labels for tensorboard export, add 'beta' to watch beta learning

from_spec(*spec*, *kwargs=None*)
Creates a layer from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the layer.

Returns List of summaries.

get_variables(*include_nontrainable=False*)
Returns the TensorFlow variables used by the layer.

Returns List of variables.

internals_spec()
Returns the internal states specification.

Returns Internal states specification

tf_apply(*x*, *update*)

tf_regularization_loss()
Creates the TensorFlow operations for the layer regularization loss.

Returns Regularization loss tensor.

tf_tensors(*named_tensors*)
Attaches the named_tensors dictionary to the layer for examination and update.

Parameters **named_tensors** – Dictionary of named tensors to be used as Input's or recorded from outputs

Returns NA

class tensorforce.core.networks.**Dropout**(*rate=0.0*, *scope='dropout'*, *summary_labels=()*)
Bases: *tensorforce.core.networks.layer.Layer*

Dropout layer. If using dropout, add this layer after inputs and after dense layers. For LSTM, dropout is handled independently as an argument. Not available for Conv2d yet.

__init__(*rate=0.0*, *scope='dropout'*, *summary_labels=()*)

from_spec(*spec*, *kwargs=None*)
Creates a layer from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the layer.

Returns List of summaries.

get_variables(*include_nontrainable=False*)
Returns the TensorFlow variables used by the layer.

Returns List of variables.

internals_spec()
Returns the internal states specification.

Returns Internal states specification

tf_apply(*x*, *update*)

tf_regularization_loss()
Creates the TensorFlow operations for the layer regularization loss.

Returns Regularization loss tensor.

tf_tensors(*named_tensors*)
Attaches the named_tensors dictionary to the layer for examination and update.

Parameters **named_tensors** – Dictionary of named tensors to be used as Input's or recorded from outputs

Returns NA

class tensorforce.core.networks.**Flatten**(*scope='flatten'*, *summary_labels=()*)
Bases: *tensorforce.core.networks.layer.Layer*

Flatten layer reshaping the input.

__init__(*scope='flatten'*, *summary_labels=()*)

from_spec(*spec*, *kwargs=None*)
Creates a layer from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the layer.

Returns List of summaries.

get_variables(*include_nontrainable=False*)
Returns the TensorFlow variables used by the layer.

Returns List of variables.

internals_spec()
Returns the internal states specification.

Returns Internal states specification

tf_apply(*x*, *update*)

tf_regularization_loss()
Creates the TensorFlow operations for the layer regularization loss.

Returns Regularization loss tensor.

tf_tensors(*named_tensors*)
Attaches the named_tensors dictionary to the layer for examination and update.

Parameters **named_tensors** – Dictionary of named tensors to be used as Input's or recorded from outputs

Returns NA

class tensorforce.core.networks.**Pool2d**(*pooling_type='max'*, *window=2*, *stride=2*, *padding='SAME'*, *scope='pool2d'*, *summary_labels=()*)
Bases: *tensorforce.core.networks.layer.Layer*

2-dimensional pooling layer.

__init__(*pooling_type='max'*, *window=2*, *stride=2*, *padding='SAME'*, *scope='pool2d'*, *summary_labels=()*)
2-dimensional pooling layer.

Parameters

- **pooling_type** – Either 'max' or 'average'.
- **window** – Pooling window size, either an integer or pair of integers.
- **stride** – Pooling stride, either an integer or pair of integers.
- **padding** – Pooling padding, one of 'VALID' or 'SAME'.

from_spec(*spec*, *kwargs=None*)
Creates a layer from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the layer.

Returns List of summaries.

get_variables(*include_nontrainable=False*)
Returns the TensorFlow variables used by the layer.

Returns List of variables.

internals_spec()
Returns the internal states specification.

Returns Internal states specification

tf_apply(*x*, *update*)

tf_regularization_loss()
Creates the TensorFlow operations for the layer regularization loss.

Returns Regularization loss tensor.

tf_tensors(*named_tensors*)
Attaches the named_tensors dictionary to the layer for examination and update.

Parameters **named_tensors** – Dictionary of named tensors to be used as Input's or recorded from outputs

Returns NA

class tensorforce.core.networks.**Embedding**(*indices*, *size*, *l2_regularization=0.0*, *l1_regularization=0.0*, *scope='embedding'*, *summary_labels=()*)

Bases: *tensorforce.core.networks.layer.Layer*

Embedding layer.

__init__(*indices*, *size*, *l2_regularization=0.0*, *l1_regularization=0.0*, *scope='embedding'*, *summary_labels=()*)

Embedding layer.

Parameters

- **indices** – Number of embedding indices.
- **size** – Embedding size.
- **l2_regularization** – L2 regularization weight.
- **l1_regularization** – L1 regularization weight.

from_spec(*spec*, *kwargs=None*)

Creates a layer from a specification dict.

get_summaries()

Returns the TensorFlow summaries reported by the layer.

Returns List of summaries.

get_variables(*include_nontrainable=False*)

Returns the TensorFlow variables used by the layer.

Returns List of variables.

internals_spec()

Returns the internal states specification.

Returns Internal states specification

tf_apply(*x*, *update*)

tf_regularization_loss()

tf_tensors(*named_tensors*)

Attaches the named_tensors dictionary to the layer for examination and update.

Parameters **named_tensors** – Dictionary of named tensors to be used as Input's or recorded from outputs

Returns NA

class tensorforce.core.networks.**Linear**(*size*, *weights=None*, *bias=True*, *l2_regularization=0.0*, *l1_regularization=0.0*, *scope='linear'*, *summary_labels=()*)

Bases: *tensorforce.core.networks.layer.Layer*

Linear fully-connected layer.

__init__(*size*, *weights=None*, *bias=True*, *l2_regularization=0.0*, *l1_regularization=0.0*, *scope='linear'*, *summary_labels=()*)

Linear layer.

Parameters

- **size** – Layer size.
- **weights** – Weight initialization, random if None.

- **bias** – Bias initialization, random if True, no bias added if False.
- **l2_regularization** – L2 regularization weight.
- **l1_regularization** – L1 regularization weight.

from_spec(*spec*, *kwargs=None*)
Creates a layer from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the layer.

Returns List of summaries.

get_variables(*include_nontrainable=False*)
Returns the TensorFlow variables used by the layer.

Returns List of variables.

internals_spec()
Returns the internal states specification.

Returns Internal states specification

tf_apply(*x*, *update=False*)

tf_regularization_loss()

tf_tensors(*named_tensors*)
Attaches the named_tensors dictionary to the layer for examination and update.

Parameters **named_tensors** – Dictionary of named tensors to be used as Input's or recorded from outputs

Returns NA

class tensorforce.core.networks.**Dense**(*size=None*, *weights=None*, *bias=True*, *activation='relu'*, *l2_regularization=0.0*, *l1_regularization=0.0*, *skip=False*, *scope='dense'*, *summary_labels=()*)
Bases: *tensorforce.core.networks.layer.Layer*

Dense layer, i.e. linear fully connected layer with subsequent non-linearity.

__init__(*size=None*, *weights=None*, *bias=True*, *activation='relu'*, *l2_regularization=0.0*, *l1_regularization=0.0*, *skip=False*, *scope='dense'*, *summary_labels=()*)
Dense layer.

Parameters

- **size** – Layer size, if None than input size matches the output size of the layer
- **weights** – Weight initialization, random if None.
- **bias** – If true, bias is added.
- **activation** – Type of nonlinearity, or dict with name & arguments
- **l2_regularization** – L2 regularization weight.
- **l1_regularization** – L1 regularization weight.
- **skip** – Add skip connection like ResNet (https://arxiv.org/pdf/1512.03385.pdf), doubles layers and ShortCut from Input to output

from_spec(*spec*, *kwargs=None*)
Creates a layer from a specification dict.

get_summaries()

get_variables(*include_nontrainable=False*)

internals_spec()
Returns the internal states specification.

Returns Internal states specification

tf_apply(*x*, *update*)

tf_regularization_loss()

tf_tensors(*named_tensors*)
Attaches the named_tensors dictionary to the layer for examination and update.

Parameters **named_tensors** – Dictionary of named tensors to be used as Input's or recorded from outputs

Returns NA

class tensorforce.core.networks.**Dueling**(*size*, *bias=False*, *activation='none'*, *l2_regularization=0.0*, *l1_regularization=0.0*, *output=None*, *scope='dueling'*, *summary_labels=()*)
Bases: *tensorforce.core.networks.layer.Layer*

Dueling layer, i.e. Duel pipelines for Exp & Adv to help with stability

__init__(*size*, *bias=False*, *activation='none'*, *l2_regularization=0.0*, *l1_regularization=0.0*, *output=None*, *scope='dueling'*, *summary_labels=()*)
Dueling layer.

[Dueling Networks] (https://arxiv.org/pdf/1511.06581.pdf) Implement Y = Expectation[x] + (Advantage[x] - Mean(Advantage[x]))

Parameters

- **size** – Layer size.
- **bias** – If true, bias is added.
- **activation** – Type of nonlinearity, or dict with name & arguments
- **l2_regularization** – L2 regularization weight.
- **l1_regularization** – L1 regularization weight.
- **output** – None or tuple of output names for ('expectation','advantage','mean_advantage')

from_spec(*spec*, *kwargs=None*)
Creates a layer from a specification dict.

get_summaries()

get_variables(*include_nontrainable=False*)

internals_spec()
Returns the internal states specification.

Returns Internal states specification

tf_apply(*x*, *update*)

tf_regularization_loss()

tf_tensors(*named_tensors*)
Attaches the named_tensors dictionary to the layer for examination and update.

Parameters **named_tensors** – Dictionary of named tensors to be used as Input's or recorded from outputs

Returns NA

class tensorforce.core.networks.**Conv1d**(*size*, *window=3*, *stride=1*, *padding='SAME'*, *bias=True*, *activation='relu'*, *l2_regularization=0.0*, *l1_regularization=0.0*, *scope='conv1d'*, *summary_labels=()*)

Bases: *tensorforce.core.networks.layer.Layer*

1-dimensional convolutional layer.

__init__(*size*, *window=3*, *stride=1*, *padding='SAME'*, *bias=True*, *activation='relu'*, *l2_regularization=0.0*, *l1_regularization=0.0*, *scope='conv1d'*, *summary_labels=()*)

1D convolutional layer.

Parameters

- **size** – Number of filters
- **window** – Convolution window size
- **stride** – Convolution stride
- **padding** – Convolution padding, one of 'VALID' or 'SAME'
- **bias** – If true, a bias is added
- **activation** – Type of nonlinearity, or dict with name & arguments
- **l2_regularization** – L2 regularization weight
- **l1_regularization** – L1 regularization weight

from_spec(*spec*, *kwargs=None*)

Creates a layer from a specification dict.

get_summaries()

get_variables(*include_nontrainable=False*)

internals_spec()

Returns the internal states specification.

Returns Internal states specification

tf_apply(*x*, *update*)

tf_regularization_loss()

tf_tensors(*named_tensors*)

Attaches the named_tensors dictionary to the layer for examination and update.

Parameters **named_tensors** – Dictionary of named tensors to be used as Input's or recorded from outputs

Returns NA

class tensorforce.core.networks.**Conv2d**(*size*, *window=3*, *stride=1*, *padding='SAME'*, *bias=True*, *activation='relu'*, *l2_regularization=0.0*, *l1_regularization=0.0*, *scope='conv2d'*, *summary_labels=()*)

Bases: *tensorforce.core.networks.layer.Layer*

2-dimensional convolutional layer.

__init__(*size*, *window=3*, *stride=1*, *padding='SAME'*, *bias=True*, *activation='relu'*, *l2_regularization=0.0*, *l1_regularization=0.0*, *scope='conv2d'*, *summary_labels=()*)
2D convolutional layer.

Parameters

- **size** – Number of filters
- **window** – Convolution window size, either an integer or pair of integers.
- **stride** – Convolution stride, either an integer or pair of integers.
- **padding** – Convolution padding, one of 'VALID' or 'SAME'
- **bias** – If true, a bias is added
- **activation** – Type of nonlinearity, or dict with name & arguments
- **l2_regularization** – L2 regularization weight
- **l1_regularization** – L1 regularization weight

from_spec(*spec*, *kwargs=None*)
Creates a layer from a specification dict.

get_summaries()

get_variables(*include_nontrainable=False*)

internals_spec()
Returns the internal states specification.

Returns Internal states specification

tf_apply(*x*, *update*)

tf_regularization_loss()

tf_tensors(*named_tensors*)
Attaches the named_tensors dictionary to the layer for examination and update.

Parameters **named_tensors** – Dictionary of named tensors to be used as Input's or recorded from outputs

Returns NA

class tensorforce.core.networks.**InternalLstm**(*size*, *dropout=None*, *lstmcell_args={}*, *scope='internal_lstm'*, *summary_labels=()*)
Bases: *tensorforce.core.networks.layer.Layer*

Long short-term memory layer for internal state management.

__init__(*size*, *dropout=None*, *lstmcell_args={}*, *scope='internal_lstm'*, *summary_labels=()*)
LSTM layer.

Parameters

- **size** – LSTM size.
- **dropout** – Dropout rate.

from_spec(*spec*, *kwargs=None*)
Creates a layer from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the layer.

Returns List of summaries.

get_variables(*include_nontrainable=False*)
Returns the TensorFlow variables used by the layer.

Returns List of variables.

internals_spec()

tf_apply(*x*, *update*, *state*)

tf_regularization_loss()
Creates the TensorFlow operations for the layer regularization loss.

Returns Regularization loss tensor.

tf_tensors(*named_tensors*)
Attaches the named_tensors dictionary to the layer for examination and update.

Parameters **named_tensors** – Dictionary of named tensors to be used as Input's or recorded from outputs

Returns NA

class tensorforce.core.networks.**Lstm**(*size*, *dropout=None*, *scope='lstm'*, *summary_labels=()*, *return_final_state=True*)
Bases: *tensorforce.core.networks.layer.Layer*

__init__(*size*, *dropout=None*, *scope='lstm'*, *summary_labels=()*, *return_final_state=True*)
LSTM layer.

Parameters

- **size** – LSTM size.
- **dropout** – Dropout rate.

from_spec(*spec*, *kwargs=None*)
Creates a layer from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the layer.

Returns List of summaries.

get_variables(*include_nontrainable=False*)
Returns the TensorFlow variables used by the layer.

Returns List of variables.

internals_spec()
Returns the internal states specification.

Returns Internal states specification

tf_apply(*x*, *update*, *sequence_length=None*)

tf_regularization_loss()
Creates the TensorFlow operations for the layer regularization loss.

Returns Regularization loss tensor.

tf_tensors(*named_tensors*)
Attaches the named_tensors dictionary to the layer for examination and update.

Parameters **named_tensors** – Dictionary of named tensors to be used as Input's or recorded from outputs

Returns NA

class tensorforce.core.networks.**Network**(*scope='network'*, *summary_labels=None*)
Bases: object

Base class for neural networks.

__init__(*scope='network'*, *summary_labels=None*)
Neural network.

static from_spec(*spec*, *kwargs=None*)
Creates a network from a specification dict.

get_list_of_named_tensor()
Returns a list of the names of tensors available.

Returns List of the names of tensors available.

get_named_tensor(*name*)
Returns a named tensor if available.

Returns True if named tensor found, False otherwise tensor: If valid, will be a tensor, otherwise None

Return type valid

get_summaries()
Returns the TensorFlow summaries reported by the network.

Returns List of summaries

get_variables(*include_nontrainable=False*)
Returns the TensorFlow variables used by the network.

Returns List of variables

internals_spec()
Returns the internal states specification.

Returns Internal states specification

set_named_tensor(*name*, *tensor*)
Returns the TensorFlow summaries reported by the network.

Returns None

tf_apply(*x*, *internals*, *update*, *return_internals=False*)
Creates the TensorFlow operations for applying the network to the given input.

Parameters

- **x** – Network input tensor or dict of input tensors.
- **internals** – List of prior internal state tensors
- **update** – Boolean tensor indicating whether this call happens during an update.
- **return_internals** – If true, also returns posterior internal state tensors

Returns Network output tensor, plus optionally list of posterior internal state tensors

tf_regularization_loss()
Creates the TensorFlow operations for the network regularization loss.

Returns Regularization loss tensor

class tensorforce.core.networks.**LayerBasedNetwork**(*scope='layerbased-network'*, *summary_labels=()*)

Bases: *tensorforce.core.networks.network.Network*

Base class for networks using TensorForce layers.

__init__(*scope='layerbased-network'*, *summary_labels=()*)
Layer-based network.

add_layer(*layer*)

from_spec(*spec*, *kwargs=None*)
Creates a network from a specification dict.

get_list_of_named_tensor()
Returns a list of the names of tensors available.

Returns List of the names of tensors available.

get_named_tensor(*name*)
Returns a named tensor if available.

Returns True if named tensor found, False otherwise tensor: If valid, will be a tensor, otherwise None

Return type valid

get_summaries()

get_variables(*include_nontrainable=False*)

internals_spec()

set_named_tensor(*name*, *tensor*)
Returns the TensorFlow summaries reported by the network.

Returns None

tf_apply(*x*, *internals*, *update*, *return_internals=False*)
Creates the TensorFlow operations for applying the network to the given input.

Parameters

- **x** – Network input tensor or dict of input tensors.
- **internals** – List of prior internal state tensors
- **update** – Boolean tensor indicating whether this call happens during an update.
- **return_internals** – If true, also returns posterior internal state tensors

Returns Network output tensor, plus optionally list of posterior internal state tensors

tf_regularization_loss()

class tensorforce.core.networks.**LayeredNetwork**(*layers*, *scope='layered-network'*, *summary_labels=()*)

Bases: *tensorforce.core.networks.network.LayerBasedNetwork*

Network consisting of a sequence of layers, which can be created from a specification dict.

__init__(*layers*, *scope='layered-network'*, *summary_labels=()*)
Single-stack layered network.

Parameters **layers** – List of layer specification dicts.

add_layer(*layer*)

static from_json(*filename*)
Creates a layer_networkd_builder from a JSON.

Parameters **filename** – Path to configuration

Returns: A layered_network_builder function with layers generated from the JSON

from_spec(*spec*, *kwargs=None*)
Creates a network from a specification dict.

get_list_of_named_tensor()
Returns a list of the names of tensors available.

Returns List of the names of tensors available.

get_named_tensor(*name*)
Returns a named tensor if available.

Returns True if named tensor found, False otherwise tensor: If valid, will be a tensor, otherwise None

Return type valid

get_summaries()

get_variables(*include_nontrainable=False*)

internals_spec()

set_named_tensor(*name*, *tensor*)
Returns the TensorFlow summaries reported by the network.

Returns None

tf_apply(*x*, *internals*, *update*, *return_internals=False*)

tf_regularization_loss()

tensorforce.core.optimizers package

Subpackages

tensorforce.core.optimizers.solvers package

Submodules

tensorforce.core.optimizers.solvers.conjugate_gradient module

class tensorforce.core.optimizers.solvers.conjugate_gradient.**ConjugateGradient**(*max_iterations*, *damping*, *unroll_loop=False*)

Bases: *tensorforce.core.optimizers.solvers.iterative.Iterative*

Conjugate gradient algorithm which iteratively finds a solution x for a system of linear equations of the form $A x = b$, where $A x$ could be, for instance, a locally linear approximation of a high-dimensional function.

See below pseudo-code taken from Wikipedia:

```
def conjgrad(A, b, x_0):
    r_0 := b - A * x_0
    c_0 := r_0
    r_0^2 := r^T * r

    for t in 0, ..., max_iterations - 1:
        Ac := A * c_t
        cAc := c_t^T * Ac
        lpha := r_t^2 / cAc
        x_{t+1} := x_t + lpha * c_t
        r_{t+1} := r_t - lpha * Ac
        r_{t+1}^2 := r_{t+1}^T * r_{t+1}
        if r_{t+1} < \epsilon:
            break
        eta = r_{t+1}^2 / r_t^2
        c_{t+1} := r_{t+1} + eta * c_t

    return x_{t+1}
```

__init__ (*max_iterations*, *damping*, *unroll_loop=False*)
Creates a new conjugate gradient solver instance.

Parameters

- **max_iterations** – Maximum number of iterations before termination.
- **damping** – Damping factor.
- **unroll_loop** – Unrolls the TensorFlow while loop if true.

from_config (*config*, *kwargs=None*)
Creates a solver from a specification dict.

tf_initialize (*x_init*, *b*)
Initialization step preparing the arguments for the first iteration of the loop body: $x_0, 0, p_0, r_0, r_0^2$.

Parameters

- **x_init** – Initial solution guess x_0, zero vector if None.
- **b** – The right-hand side b of the system of linear equations.

Returns Initial arguments for tf_step.

tf_next_step (*x*, *iteration*, *conjugate*, *residual*, *squared_residual*)
Termination condition: max number of iterations, or residual sufficiently small.

Parameters

- **x** – Current solution estimate x_t.
- **iteration** – Current iteration counter t.
- **conjugate** – Current conjugate c_t.
- **residual** – Current residual r_t.
- **squared_residual** – Current squared residual r_t^2.

Returns True if another iteration should be performed.

tf_solve (*fn_x*, *x_init*, *b*)
Iteratively solves the system of linear equations $A x = b$.

Parameters

- **fn_x** – A callable returning the left-hand side $A x$ of the system of linear equations.
- **x_init** – Initial solution guess x_0, zero vector if None.
- **b** – The right-hand side b of the system of linear equations.

Returns A solution x to the problem as given by the solver.

tf_step(*x*, *iteration*, *conjugate*, *residual*, *squared_residual*)
Iteration loop body of the conjugate gradient algorithm.

Parameters

- **x** – Current solution estimate x_t.
- **iteration** – Current iteration counter t.
- **conjugate** – Current conjugate c_t.
- **residual** – Current residual r_t.
- **squared_residual** – Current squared residual r_t^2.

Returns Updated arguments for next iteration.

tensorforce.core.optimizers.solvers.iterative module

class tensorforce.core.optimizers.solvers.iterative.**Iterative**(*max_iterations*, *unroll_loop=False*)
Bases: *tensorforce.core.optimizers.solvers.solver.Solver*

Generic solver which *iteratively* solves an equation/optimization problem. Involves an initialization step, the iteration loop body and the termination condition.

__init__(*max_iterations*, *unroll_loop=False*)
Creates a new iterative solver instance.

Parameters

- **max_iterations** – Maximum number of iterations before termination.
- **unroll_loop** – Unrolls the TensorFlow while loop if true.

from_config(*config*, *kwargs=None*)
Creates a solver from a specification dict.

tf_initialize(*x_init*, **args*)
Initialization step preparing the arguments for the first iteration of the loop body (default: initial solution guess and iteration counter).

Parameters

- **x_init** – Initial solution guess x_0.
- ***args** – Additional solver-specific arguments.

Returns Initial arguments for tf_step.

tf_next_step(*x*, *iteration*, **args*)
Termination condition (default: max number of iterations).

Parameters

- **x** – Current solution estimate.

- **iteration** – Current iteration counter.
- ***args** – Additional solver-specific arguments.

Returns True if another iteration should be performed.

tf_solve(*fn_x*, *x_init*, **args*)
Iteratively solves an equation/optimization for x involving an expression $f(x)$.

Parameters

- **fn_x** – A callable returning an expression $f(x)$ given x.
- **x_init** – Initial solution guess x_0.
- ***args** – Additional solver-specific arguments.

Returns A solution x to the problem as given by the solver.

tf_step(*x*, *iteration*, **args*)
Iteration loop body of the iterative solver (default: increment iteration step). The first two loop arguments have to be the current solution estimate and the iteration step.

Parameters

- **x** – Current solution estimate.
- **iteration** – Current iteration counter.
- ***args** – Additional solver-specific arguments.

Returns Updated arguments for next iteration.

tensorforce.core.optimizers.solvers.line_search module

class tensorforce.core.optimizers.solvers.line_search.**LineSearch**(*max_iterations*, *accept_ratio*, *mode*, *parameter*, *unroll_loop=False*)
Bases: *tensorforce.core.optimizers.solvers.iterative.Iterative*

Line search algorithm which iteratively optimizes the value $f(x)$ for x on the line between x' and x_0 by optimistically taking the first acceptable x starting from x_0 and moving towards x'.

__init__(*max_iterations*, *accept_ratio*, *mode*, *parameter*, *unroll_loop=False*)
Creates a new line search solver instance.

Parameters

- **max_iterations** – Maximum number of iterations before termination.
- **accept_ratio** – Lower limit of what improvement ratio over $x = x'$ is acceptable (based either on a given estimated improvement or with respect to the value at $x = x'$).
- **mode** – Mode of movement between x_0 and x', either 'linear' or 'exponential'.
- **parameter** – Movement mode parameter, additive or multiplicative, respectively.
- **unroll_loop** – Unrolls the TensorFlow while loop if true.

from_config(*config*, *kwargs=None*)
Creates a solver from a specification dict.

tf_initialize(*x_init*, *base_value*, *target_value*, *estimated_improvement*)
Initialization step preparing the arguments for the first iteration of the loop body.

Parameters

- **x_init** – Initial solution guess x_0.
- **base_value** – Value $f(x')$ at $x = x'$.
- **target_value** – Value $f(x_0)$ at $x = x_0$.
- **estimated_improvement** – Estimated value at $x = x_0$, $f(x')$ if None.

Returns Initial arguments for tf_step.

tf_next_step(*x*, *iteration*, *deltas*, *improvement*, *last_improvement*, *estimated_improvement*)
Termination condition: max number of iterations, or no improvement for last step, or improvement less than acceptable ratio, or estimated value not positive.

Parameters

- **x** – Current solution estimate x_t.
- **iteration** – Current iteration counter t.
- **deltas** – Current difference $x_t - x'$.
- **improvement** – Current improvement $(f(x_t) - f(x')) / v'$.
- **last*improvement** – Last improvement $(f(x*{t-1}) - f(x')) / v'$.
- **estimated_improvement** – Current estimated value v'.

Returns True if another iteration should be performed.

tf_solve(*fn_x*, *x_init*, *base_value*, *target_value*, *estimated_improvement=None*)
Iteratively optimizes $f(x)$ for x on the line between x' and x_0.

Parameters

- **fn_x** – A callable returning the value $f(x)$ at x.
- **x_init** – Initial solution guess x_0.
- **base_value** – Value $f(x')$ at $x = x'$.
- **target_value** – Value $f(x_0)$ at $x = x_0$.
- **estimated_improvement** – Estimated improvement for $x = x_0$, $f(x')$ if None.

Returns A solution x to the problem as given by the solver.

tf_step(*x*, *iteration*, *deltas*, *improvement*, *last_improvement*, *estimated_improvement*)
Iteration loop body of the line search algorithm.

Parameters

- **x** – Current solution estimate x_t.
- **iteration** – Current iteration counter t.
- **deltas** – Current difference $x_t - x'$.
- **improvement** – Current improvement $(f(x_t) - f(x')) / v'$.
- **last*improvement** – Last improvement $(f(x*{t-1}) - f(x')) / v'$.
- **estimated_improvement** – Current estimated value v'.

Returns Updated arguments for next iteration.

tensorforce.core.optimizers.solvers.solver module

class tensorforce.core.optimizers.solvers.solver.**Solver**
Bases: object

Generic TensorFlow-based solver which solves a not yet further specified equation/optimization problem.

__init__()
Creates a new solver instance.

static from_config(*config*, *kwargs=None*)
Creates a solver from a specification dict.

tf_solve(*fn_x*, **args*)
Solves an equation/optimization for x involving an expression $f(x)$.

Parameters

- **fn_x** – A callable returning an expression $f(x)$ given x.
- ***args** – Additional solver-specific arguments.

Returns A solution x to the problem as given by the solver.

Module contents

class tensorforce.core.optimizers.solvers.**Solver**
Bases: object

Generic TensorFlow-based solver which solves a not yet further specified equation/optimization problem.

__init__()
Creates a new solver instance.

static from_config(*config*, *kwargs=None*)
Creates a solver from a specification dict.

tf_solve(*fn_x*, **args*)
Solves an equation/optimization for x involving an expression $f(x)$.

Parameters

- **fn_x** – A callable returning an expression $f(x)$ given x.
- ***args** – Additional solver-specific arguments.

Returns A solution x to the problem as given by the solver.

class tensorforce.core.optimizers.solvers.**Iterative**(*max_iterations*, *unroll_loop=False*)
Bases: *tensorforce.core.optimizers.solvers.solver.Solver*

Generic solver which *iteratively* solves an equation/optimization problem. Involves an initialization step, the iteration loop body and the termination condition.

__init__(*max_iterations*, *unroll_loop=False*)
Creates a new iterative solver instance.

Parameters

- **max_iterations** – Maximum number of iterations before termination.
- **unroll_loop** – Unrolls the TensorFlow while loop if true.

from_config(*config*, *kwargs=None*)
Creates a solver from a specification dict.

tf_initialize(*x_init*, **args*)
Initialization step preparing the arguments for the first iteration of the loop body (default: initial solution guess and iteration counter).

Parameters

- **x_init** – Initial solution guess x_0.
- ***args** – Additional solver-specific arguments.

Returns Initial arguments for tf_step.

tf_next_step(*x*, *iteration*, **args*)
Termination condition (default: max number of iterations).

Parameters

- **x** – Current solution estimate.
- **iteration** – Current iteration counter.
- ***args** – Additional solver-specific arguments.

Returns True if another iteration should be performed.

tf_solve(*fn_x*, *x_init*, **args*)
Iteratively solves an equation/optimization for x involving an expression $f(x)$.

Parameters

- **fn_x** – A callable returning an expression $f(x)$ given x.
- **x_init** – Initial solution guess x_0.
- ***args** – Additional solver-specific arguments.

Returns A solution x to the problem as given by the solver.

tf_step(*x*, *iteration*, **args*)
Iteration loop body of the iterative solver (default: increment iteration step). The first two loop arguments have to be the current solution estimate and the iteration step.

Parameters

- **x** – Current solution estimate.
- **iteration** – Current iteration counter.
- ***args** – Additional solver-specific arguments.

Returns Updated arguments for next iteration.

class tensorforce.core.optimizers.solvers.**ConjugateGradient**(*max_iterations*, *damping*, *unroll_loop=False*)
Bases: *tensorforce.core.optimizers.solvers.iterative.Iterative*

Conjugate gradient algorithm which iteratively finds a solution x for a system of linear equations of the form $A x = b$, where $A x$ could be, for instance, a locally linear approximation of a high-dimensional function.

See below pseudo-code taken from Wikipedia:

```
def conjgrad(A, b, x_0):
    r_0 := b - A * x_0
    c_0 := r_0
    r_0^2 := r^T * r

    for t in 0, ..., max_iterations - 1:
        Ac := A * c_t
        cAc := c_t^T * Ac
        lpha := r_t^2 / cAc
        x_{t+1} := x_t + lpha * c_t
        r_{t+1} := r_t - lpha * Ac
        r_{t+1}^2 := r_{t+1}^T * r_{t+1}
        if r_{t+1} < \epsilon:
            break
        eta = r_{t+1}^2 / r_t^2
        c_{t+1} := r_{t+1} + eta * c_t

    return x_{t+1}
```

__init__ (*max_iterations*, *damping*, *unroll_loop=False*)
Creates a new conjugate gradient solver instance.

Parameters

- **max_iterations** – Maximum number of iterations before termination.
- **damping** – Damping factor.
- **unroll_loop** – Unrolls the TensorFlow while loop if true.

from_config (*config*, *kwargs=None*)
Creates a solver from a specification dict.

tf_initialize (*x_init*, *b*)
Initialization step preparing the arguments for the first iteration of the loop body: $x_0, 0, p_0, r_0, r_0^2$.

Parameters

- **x_init** – Initial solution guess x_0, zero vector if None.
- **b** – The right-hand side b of the system of linear equations.

Returns Initial arguments for tf_step.

tf_next_step (*x*, *iteration*, *conjugate*, *residual*, *squared_residual*)
Termination condition: max number of iterations, or residual sufficiently small.

Parameters

- **x** – Current solution estimate x_t.
- **iteration** – Current iteration counter t.
- **conjugate** – Current conjugate c_t.
- **residual** – Current residual r_t.
- **squared_residual** – Current squared residual r_t^2.

Returns True if another iteration should be performed.

tf_solve (*fn_x*, *x_init*, *b*)
Iteratively solves the system of linear equations $A x = b$.

Parameters

- **fn_x** – A callable returning the left-hand side $A x$ of the system of linear equations.
- **x_init** – Initial solution guess x_0, zero vector if None.
- **b** – The right-hand side b of the system of linear equations.

Returns A solution x to the problem as given by the solver.

tf_step(*x*, *iteration*, *conjugate*, *residual*, *squared_residual*)
Iteration loop body of the conjugate gradient algorithm.

Parameters

- **x** – Current solution estimate x_t.
- **iteration** – Current iteration counter t.
- **conjugate** – Current conjugate c_t.
- **residual** – Current residual r_t.
- **squared_residual** – Current squared residual r_t^2.

Returns Updated arguments for next iteration.

class tensorforce.core.optimizers.solvers.**LineSearch**(*max_iterations*, *accept_ratio*, *mode*, *parameter*, *unroll_loop=False*)
Bases: *tensorforce.core.optimizers.solvers.iterative.Iterative*

Line search algorithm which iteratively optimizes the value $f(x)$ for x on the line between x' and x_0 by optimistically taking the first acceptable x starting from x_0 and moving towards x'.

__init__(*max_iterations*, *accept_ratio*, *mode*, *parameter*, *unroll_loop=False*)
Creates a new line search solver instance.

Parameters

- **max_iterations** – Maximum number of iterations before termination.
- **accept_ratio** – Lower limit of what improvement ratio over $x = x'$ is acceptable (based either on a given estimated improvement or with respect to the value at $x = x'$).
- **mode** – Mode of movement between x_0 and x', either 'linear' or 'exponential'.
- **parameter** – Movement mode parameter, additive or multiplicative, respectively.
- **unroll_loop** – Unrolls the TensorFlow while loop if true.

from_config(*config*, *kwargs=None*)
Creates a solver from a specification dict.

tf_initialize(*x_init*, *base_value*, *target_value*, *estimated_improvement*)
Initialization step preparing the arguments for the first iteration of the loop body.

Parameters

- **x_init** – Initial solution guess x_0.
- **base_value** – Value $f(x')$ at $x = x'$.
- **target_value** – Value $f(x_0)$ at $x = x_0$.
- **estimated_improvement** – Estimated value at $x = x_0$, $f(x')$ if None.

Returns Initial arguments for tf_step.

tf_next_step(*x*, *iteration*, *deltas*, *improvement*, *last_improvement*, *estimated_improvement*)
Termination condition: max number of iterations, or no improvement for last step, or improvement less than acceptable ratio, or estimated value not positive.

Parameters

- **x** – Current solution estimate x_t.
- **iteration** – Current iteration counter t.
- **deltas** – Current difference $x_t - x'$.
- **improvement** – Current improvement $(f(x_t) - f(x')) / v'$.
- **last*improvement** – Last improvement $(f(x*{t-1}) - f(x')) / v'$.
- **estimated_improvement** – Current estimated value v'.

Returns True if another iteration should be performed.

tf_solve(*fn_x*, *x_init*, *base_value*, *target_value*, *estimated_improvement=None*)
Iteratively optimizes $f(x)$ for x on the line between x' and x_0.

Parameters

- **fn_x** – A callable returning the value $f(x)$ at x.
- **x_init** – Initial solution guess x_0.
- **base_value** – Value $f(x')$ at $x = x'$.
- **target_value** – Value $f(x_0)$ at $x = x_0$.
- **estimated_improvement** – Estimated improvement for $x = x_0$, $f(x')$ if None.

Returns A solution x to the problem as given by the solver.

tf_step(*x*, *iteration*, *deltas*, *improvement*, *last_improvement*, *estimated_improvement*)
Iteration loop body of the line search algorithm.

Parameters

- **x** – Current solution estimate x_t.
- **iteration** – Current iteration counter t.
- **deltas** – Current difference $x_t - x'$.
- **improvement** – Current improvement $(f(x_t) - f(x')) / v'$.
- **last*improvement** – Last improvement $(f(x*{t-1}) - f(x')) / v'$.
- **estimated_improvement** – Current estimated value v'.

Returns Updated arguments for next iteration.

Submodules

tensorforce.core.optimizers.clipped_step module

class tensorforce.core.optimizers.clipped_step.**ClippedStep**(*optimizer*, *clipping_value*, *scope='clipped-step'*, *summary_labels=()*)
Bases: *tensorforce.core.optimizers.meta_optimizer.MetaOptimizer*

The clipped-shep meta optimizer clips the values of the optimization step proposed by another optimizer.

__init__ (*optimizer*, *clipping_value*, *scope='clipped-step'*, *summary_labels=()*)
Creates a new multi-step meta optimizer instance.

Parameters

- **optimizer** – The optimizer which is modified by this meta optimizer.
- **clipping_value** – Clip deltas at this value.

apply_step (*variables*, *deltas*)
Applies step deltas to variable values.

Parameters

- **variables** – List of variables.
- **deltas** – List of deltas of same length.

Returns The step-applied operation.

from_spec (*spec*, *kwargs=None*)
Creates an optimizer from a specification dict.

get_summaries ()
Returns the TensorFlow summaries reported by the optimizer.

Returns List of summaries.

get_variables ()

minimize (*time*, *variables*, ***kwargs*)
Performs an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.
- ****kwargs** – Additional optimizer-specific arguments. The following arguments are used by some optimizers:
- **arguments** (–) – Dict of arguments for callables, like fn_loss.
- **fn_loss** (–) – A callable returning the loss of the current model.
- **fn_reference** (–) – A callable returning the reference values, in case of a comparative loss.
- **fn_kl_divergence** (–) – A callable returning the KL-divergence relative to the current model.
- **return_estimated_improvement** (–) – Returns the estimated improvement resulting from the natural gradient calculation if true.
- **source_variables** (–) – List of source variables to synchronize with.
- **global_variables** (–) – List of global variables to apply the proposed optimization step to.

Returns The optimization operation.

tf_step (*time*, *variables*, ***kwargs*)
Creates the TensorFlow operations for performing an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.
- ****kwargs** – Additional arguments passed on to the internal optimizer.

Returns List of delta tensors corresponding to the updates for each optimized variable.

tensorforce.core.optimizers.evolutionary module

class tensorforce.core.optimizers.evolutionary.**Evolutionary**(*learning_rate*, *num_samples=1*, *unroll_loop=False*, *scope='evolutionary'*, *summary_labels=()*)

Bases: *tensorforce.core.optimizers.optimizer.Optimizer*

Evolutionary optimizer which samples random perturbations and applies them either positively or negatively, depending on their improvement of the loss.

__init__(*learning_rate*, *num_samples=1*, *unroll_loop=False*, *scope='evolutionary'*, *summary_labels=()*)

Creates a new evolutionary optimizer instance.

Parameters

- **learning_rate** – Learning rate.
- **num_samples** – Number of sampled perturbations.

apply_step(*variables*, *deltas*)

Applies step deltas to variable values.

Parameters

- **variables** – List of variables.
- **deltas** – List of deltas of same length.

Returns The step-applied operation.

from_spec(*spec*, *kwargs=None*)

Creates an optimizer from a specification dict.

get_summaries()

Returns the TensorFlow summaries reported by the optimizer.

Returns List of summaries.

get_variables()

Returns the TensorFlow variables used by the optimizer.

Returns List of variables.

minimize(*time*, *variables*, ***kwargs*)

Performs an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.

- ****kwargs** – Additional optimizer-specific arguments. The following arguments are used by some optimizers:
- **arguments** (–) – Dict of arguments for callables, like fn_loss.
- **fn_loss** (–) – A callable returning the loss of the current model.
- **fn_reference** (–) – A callable returning the reference values, in case of a comparative loss.
- **fn_kl_divergence** (–) – A callable returning the KL-divergence relative to the current model.
- **return_estimated_improvement** (–) – Returns the estimated improvement resulting from the natural gradient calculation if true.
- **source_variables** (–) – List of source variables to synchronize with.
- **global_variables** (–) – List of global variables to apply the proposed optimization step to.

Returns The optimization operation.

tf_step(*time*, *variables*, *arguments*, *fn_loss*, ***kwargs*)
Creates the TensorFlow operations for performing an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.
- **arguments** – Dict of arguments for callables, like fn_loss.
- **fn_loss** – A callable returning the loss of the current model.
- ****kwargs** – Additional arguments, not used.

Returns List of delta tensors corresponding to the updates for each optimized variable.

tensorforce.core.optimizers.global_optimizer module

class tensorforce.core.optimizers.global_optimizer.**GlobalOptimizer**(*optimizer*, *scope='global-optimizer'*, *summary_labels=()*)
Bases: *tensorforce.core.optimizers.meta_optimizer.MetaOptimizer*

The global optimizer applies an optimizer to the local variables. In addition, it also applies the update to a corresponding set of global variables and subsequently updates the local variables to the value of these global variables. Note: This is used for the current distributed mode, and will likely change with the next major version update.

__init__(*optimizer*, *scope='global-optimizer'*, *summary_labels=()*)
Creates a new global optimizer instance.

Parameters **optimizer** – The optimizer which is modified by this meta optimizer.

apply_step(*variables*, *deltas*)
Applies step deltas to variable values.

Parameters

- **variables** – List of variables.
- **deltas** – List of deltas of same length.

Returns The step-applied operation.

from_spec(*spec*, *kwargs=None*)
Creates an optimizer from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the optimizer.

Returns List of summaries.

get_variables()

minimize(*time*, *variables*, ***kwargs*)
Performs an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.
- ****kwargs** – Additional optimizer-specific arguments. The following arguments are used by some optimizers:
- **arguments** (–) – Dict of arguments for callables, like fn_loss.
- **fn_loss** (–) – A callable returning the loss of the current model.
- **fn_reference** (–) – A callable returning the reference values, in case of a comparative loss.
- **fn_kl_divergence** (–) – A callable returning the KL-divergence relative to the current model.
- **return_estimated_improvement** (–) – Returns the estimated improvement resulting from the natural gradient calculation if true.
- **source_variables** (–) – List of source variables to synchronize with.
- **global_variables** (–) – List of global variables to apply the proposed optimization step to.

Returns The optimization operation.

tf_step(*time*, *variables*, *global_variables*, ***kwargs*)
Creates the TensorFlow operations for performing an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.
- **global_variables** – List of global variables to apply the proposed optimization step to.
- ****kwargs** – ??? coming soon

Returns List of delta tensors corresponding to the updates for each optimized variable.

tensorforce.core.optimizers.meta_optimizer module

class tensorforce.core.optimizers.meta_optimizer.**MetaOptimizer**(*optimizer*, *scope='meta-optimizer'*, *summary_labels=()*, ***kwargs*)

Bases: *tensorforce.core.optimizers.optimizer.Optimizer*

A meta optimizer takes the optimization implemented by another optimizer and modifies/optimizes its proposed result. For example, line search might be applied to find a more optimal step size.

__init__(*optimizer*, *scope='meta-optimizer'*, *summary_labels=()*, ***kwargs*)
Creates a new meta optimizer instance.

Parameters **optimizer** – The optimizer which is modified by this meta optimizer.

apply_step(*variables*, *deltas*)
Applies step deltas to variable values.

Parameters

- **variables** – List of variables.
- **deltas** – List of deltas of same length.

Returns The step-applied operation.

from_spec(*spec*, *kwargs=None*)
Creates an optimizer from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the optimizer.

Returns List of summaries.

get_variables()

minimize(*time*, *variables*, ***kwargs*)
Performs an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.
- ****kwargs** – Additional optimizer-specific arguments. The following arguments are used by some optimizers:
- **arguments** (–) – Dict of arguments for callables, like fn_loss.
- **fn_loss** (–) – A callable returning the loss of the current model.
- **fn_reference** (–) – A callable returning the reference values, in case of a comparative loss.
- **fn_kl_divergence** (–) – A callable returning the KL-divergence relative to the current model.
- **return_estimated_improvement** (–) – Returns the estimated improvement resulting from the natural gradient calculation if true.
- **source_variables** (–) – List of source variables to synchronize with.

- **global_variables** (–) – List of global variables to apply the proposed optimization step to.

Returns The optimization operation.

tf_step(*time*, *variables*, ***kwargs*)
Creates the TensorFlow operations for performing an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.
- ****kwargs** – Additional arguments depending on the specific optimizer implementation. For instance, often includes `fn_loss` if a loss function is optimized.

Returns List of delta tensors corresponding to the updates for each optimized variable.

tensorforce.core.optimizers.multi_step module

class tensorforce.core.optimizers.multi_step.**MultiStep**(*optimizer*, *num_steps=10*, *unroll_loop=False*, *scope='multi-step'*, *summary_labels=()*)
Bases: *tensorforce.core.optimizers.meta_optimizer.MetaOptimizer*

The multi-step meta optimizer repeatedly applies the optimization step proposed by another optimizer a number of times.

__init__(*optimizer*, *num_steps=10*, *unroll_loop=False*, *scope='multi-step'*, *summary_labels=()*)
Creates a new multi-step meta optimizer instance.

Parameters

- **optimizer** – The optimizer which is modified by this meta optimizer.
- **num_steps** – Number of optimization steps to perform.

apply_step(*variables*, *deltas*)
Applies step deltas to variable values.

Parameters

- **variables** – List of variables.
- **deltas** – List of deltas of same length.

Returns The step-applied operation.

from_spec(*spec*, *kwargs=None*)
Creates an optimizer from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the optimizer.

Returns List of summaries.

get_variables()

minimize(*time*, *variables*, ***kwargs*)
Performs an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.
- ****kwargs** – Additional optimizer-specific arguments. The following arguments are used by some optimizers:
- **arguments** (–) – Dict of arguments for callables, like fn_loss.
- **fn_loss** (–) – A callable returning the loss of the current model.
- **fn_reference** (–) – A callable returning the reference values, in case of a comparative loss.
- **fn_kl_divergence** (–) – A callable returning the KL-divergence relative to the current model.
- **return_estimated_improvement** (–) – Returns the estimated improvement resulting from the natural gradient calculation if true.
- **source_variables** (–) – List of source variables to synchronize with.
- **global_variables** (–) – List of global variables to apply the proposed optimization step to.

Returns The optimization operation.

tf_step(*time*, *variables*, *arguments*, *fn_reference=None*, ***kwargs*)

Creates the TensorFlow operations for performing an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.
- **arguments** – Dict of arguments for callables, like fn_loss.
- **fn_reference** – A callable returning the reference values, in case of a comparative loss.
- ****kwargs** – Additional arguments passed on to the internal optimizer.

Returns List of delta tensors corresponding to the updates for each optimized variable.

tensorforce.core.optimizers.natural_gradient module

class tensorforce.core.optimizers.natural_gradient.**NaturalGradient**(*learning_rate*, *cg_max_iterations=20*, *cg_damping=0.001*, *cg_unroll_loop=False*, *scope='natural-gradient'*, *summary_labels=()*)

Bases: *tensorforce.core.optimizers.optimizer.Optimizer*

Natural gradient optimizer.

__init__(*learning_rate*, *cg_max_iterations=20*, *cg_damping=0.001*, *cg_unroll_loop=False*, *scope='natural-gradient'*, *summary_labels=()*)

Creates a new natural gradient optimizer instance.

Parameters

- **learning_rate** – Learning rate, i.e. KL-divergence of distributions between optimization steps.
- **cg_max_iterations** – Conjugate gradient solver max iterations.
- **cg_damping** – Conjugate gradient solver damping factor.
- **cg_unroll_loop** – Unroll conjugate gradient loop if true.

apply_step(*variables*, *deltas*)
Applies step deltas to variable values.

Parameters

- **variables** – List of variables.
- **deltas** – List of deltas of same length.

Returns The step-applied operation.

from_spec(*spec*, *kwargs=None*)
Creates an optimizer from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the optimizer.

Returns List of summaries.

get_variables()
Returns the TensorFlow variables used by the optimizer.

Returns List of variables.

minimize(*time*, *variables*, ***kwargs*)
Performs an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.
- ****kwargs** – Additional optimizer-specific arguments. The following arguments are used by some optimizers:
- **arguments** (–) – Dict of arguments for callables, like fn_loss.
- **fn_loss** (–) – A callable returning the loss of the current model.
- **fn_reference** (–) – A callable returning the reference values, in case of a comparative loss.
- **fn_kl_divergence** (–) – A callable returning the KL-divergence relative to the current model.
- **return_estimated_improvement** (–) – Returns the estimated improvement resulting from the natural gradient calculation if true.
- **source_variables** (–) – List of source variables to synchronize with.
- **global_variables** (–) – List of global variables to apply the proposed optimization step to.

Returns The optimization operation.

tf_step(*time*, *variables*, *arguments*, *fn_loss*, *fn_kl_divergence*, *return_estimated_improvement=False*, ***kwargs*)
Creates the TensorFlow operations for performing an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.
- **arguments** – Dict of arguments for callables, like fn_loss.
- **fn_loss** – A callable returning the loss of the current model.
- **fn_kl_divergence** – A callable returning the KL-divergence relative to the current model.
- **return_estimated_improvement** – Returns the estimated improvement resulting from the natural gradient calculation if true.
- ****kwargs** – Additional arguments, not used.

Returns List of delta tensors corresponding to the updates for each optimized variable.

tensorforce.core.optimizers.optimized_step module

class tensorforce.core.optimizers.optimized_step.**OptimizedStep**(*optimizer*, *ls_max_iterations=10*, *ls_accept_ratio=0.9*, *ls_mode='exponential'*, *ls_parameter=0.5*, *ls_unroll_loop=False*, *scope='optimized-step'*, *summary_labels=()*)

Bases: *tensorforce.core.optimizers.meta_optimizer.MetaOptimizer*

The optimized-step meta optimizer applies line search to the proposed optimization step of another optimizer to find a more optimal step size.

__init__(*optimizer*, *ls_max_iterations=10*, *ls_accept_ratio=0.9*, *ls_mode='exponential'*, *ls_parameter=0.5*, *ls_unroll_loop=False*, *scope='optimized-step'*, *summary_labels=()*)

Creates a new optimized step meta optimizer instance.

Parameters

- **optimizer** – The optimizer which is modified by this meta optimizer.
- **ls_max_iterations** – Maximum number of line search iterations.
- **ls_accept_ratio** – Line search acceptance ratio.
- **ls_mode** – Line search mode, see LineSearch solver.
- **ls_parameter** – Line search parameter, see LineSearch solver.
- **ls_unroll_loop** – Unroll line search loop if true.

apply_step(*variables*, *deltas*)

Applies step deltas to variable values.

Parameters

- **variables** – List of variables.
- **deltas** – List of deltas of same length.

Returns The step-applied operation.

from_spec(*spec*, *kwargs=None*)
Creates an optimizer from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the optimizer.

Returns List of summaries.

get_variables()

minimize(*time*, *variables*, ***kwargs*)
Performs an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.
- ****kwargs** – Additional optimizer-specific arguments. The following arguments are used by some optimizers:
- **arguments** (–) – Dict of arguments for callables, like fn_loss.
- **fn_loss** (–) – A callable returning the loss of the current model.
- **fn_reference** (–) – A callable returning the reference values, in case of a comparative loss.
- **fn_kl_divergence** (–) – A callable returning the KL-divergence relative to the current model.
- **return_estimated_improvement** (–) – Returns the estimated improvement resulting from the natural gradient calculation if true.
- **source_variables** (–) – List of source variables to synchronize with.
- **global_variables** (–) – List of global variables to apply the proposed optimization step to.

Returns The optimization operation.

tf_step(*time*, *variables*, *arguments*, *fn_loss*, *fn_reference*, ***kwargs*)
Creates the TensorFlow operations for performing an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.
- **arguments** – Dict of arguments for callables, like fn_loss.
- **fn_loss** – A callable returning the loss of the current model.
- **fn_reference** – A callable returning the reference values, in case of a comparative loss.
- ****kwargs** – Additional arguments passed on to the internal optimizer.

Returns List of delta tensors corresponding to the updates for each optimized variable.

tensorforce.core.optimizers.optimizer module

class tensorforce.core.optimizers.optimizer.**Optimizer**(*scope='optimizer'*, *summary_labels=None*)

Bases: object

Base class for optimizers which minimize a not yet further specified expression, usually some kind of loss function. More generally, an optimizer can be considered as some method of updating a set of variables.

__init__(*scope='optimizer'*, *summary_labels=None*)
Creates a new optimizer instance.

apply_step(*variables*, *deltas*)
Applies step deltas to variable values.

Parameters

- **variables** – List of variables.
- **deltas** – List of deltas of same length.

Returns The step-applied operation.

static from_spec(*spec*, *kwargs=None*)
Creates an optimizer from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the optimizer.

Returns List of summaries.

get_variables()
Returns the TensorFlow variables used by the optimizer.

Returns List of variables.

minimize(*time*, *variables*, ***kwargs*)
Performs an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.
- ****kwargs** – Additional optimizer-specific arguments. The following arguments are used by some optimizers:
- **arguments** (–) – Dict of arguments for callables, like fn_loss.
- **fn_loss** (–) – A callable returning the loss of the current model.
- **fn_reference** (–) – A callable returning the reference values, in case of a comparative loss.
- **fn_kl_divergence** (–) – A callable returning the KL-divergence relative to the current model.
- **return_estimated_improvement** (–) – Returns the estimated improvement resulting from the natural gradient calculation if true.
- **source_variables** (–) – List of source variables to synchronize with.
- **global_variables** (–) – List of global variables to apply the proposed optimization step to.

Returns The optimization operation.

tf_step(*time*, *variables*, ***kwargs*)
Creates the TensorFlow operations for performing an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.
- ****kwargs** – Additional arguments depending on the specific optimizer implementation. For instance, often includes `fn_loss` if a loss function is optimized.

Returns List of delta tensors corresponding to the updates for each optimized variable.

tensorforce.core.optimizers.synchronization module

class tensorforce.core.optimizers.synchronization.**Synchronization**(*sync_frequency=1*, *update_weight=1.0*, *scope='synchronization'*, *summary_labels=()*)

Bases: *tensorforce.core.optimizers.optimizer.Optimizer*

The synchronization optimizer updates variables periodically to the value of a corresponding set of source variables.

__init__(*sync_frequency=1*, *update_weight=1.0*, *scope='synchronization'*, *summary_labels=()*)
Creates a new synchronization optimizer instance.

Parameters

- **sync_frequency** – The interval between optimization calls actually performing a
- **step.** (synchronization) –
- **update_weight** – The update weight, 1.0 meaning a full assignment of the source
- **values.** (*variables*) –

apply_step(*variables*, *deltas*)
Applies step deltas to variable values.

Parameters

- **variables** – List of variables.
- **deltas** – List of deltas of same length.

Returns The step-applied operation.

from_spec(*spec*, *kwargs=None*)
Creates an optimizer from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the optimizer.

Returns List of summaries.

get_variables()
Returns the TensorFlow variables used by the optimizer.

Returns List of variables.

minimize(*time*, *variables*, ***kwargs*)
Performs an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.
- ****kwargs** – Additional optimizer-specific arguments. The following arguments are used by some optimizers:
- **arguments** (–) – Dict of arguments for callables, like fn_loss.
- **fn_loss** (–) – A callable returning the loss of the current model.
- **fn_reference** (–) – A callable returning the reference values, in case of a comparative loss.
- **fn_kl_divergence** (–) – A callable returning the KL-divergence relative to the current model.
- **return_estimated_improvement** (–) – Returns the estimated improvement resulting from the natural gradient calculation if true.
- **source_variables** (–) – List of source variables to synchronize with.
- **global_variables** (–) – List of global variables to apply the proposed optimization step to.

Returns The optimization operation.

tf_step(*time*, *variables*, *source_variables*, ***kwargs*)
Creates the TensorFlow operations for performing an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.
- **source_variables** – List of source variables to synchronize with.
- ****kwargs** – Additional arguments, not used.

Returns List of delta tensors corresponding to the updates for each optimized variable.

tensorforce.core.optimizers.tf_optimizer module

class tensorforce.core.optimizers.tf_optimizer.**TFOptimizer**(*optimizer*, *scope=None*, *summary_labels=()*, ***kwargs*)

Bases: *tensorforce.core.optimizers.optimizer.Optimizer*

Wrapper class for TensorFlow optimizers.

__init__(*optimizer*, *scope=None*, *summary_labels=()*, ***kwargs*)
Creates a new optimizer instance of a TensorFlow optimizer.

Parameters

- **optimizer** – The name of the optimizer, one of 'adadelta', 'adagrad', 'adam', 'nadam',
- **'momentum', 'rmsprop'.** (*'gradient_descent',*) –

- ****kwargs** – Additional arguments passed on to the TensorFlow optimizer constructor.

apply_step(*variables*, *deltas*)
Applies step deltas to variable values.

Parameters

- **variables** – List of variables.
- **deltas** – List of deltas of same length.

Returns The step-applied operation.

from_spec(*spec*, *kwargs=None*)
Creates an optimizer from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the optimizer.

Returns List of summaries.

get_variables()

static get_wrapper(*optimizer*)
Returns a TFOptimizer constructor callable for the given optimizer name.

Parameters

- **optimizer** – The name of the optimizer, one of 'adadelta', 'adagrad', 'adam', 'nadam',
- **'momentum', 'rmsprop'.** (*'gradient_descent',*) –

Returns The TFOptimizer constructor callable.

minimize(*time*, *variables*, ***kwargs*)
Performs an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.
- ****kwargs** – Additional optimizer-specific arguments. The following arguments are used by some optimizers:
- **arguments** (–) – Dict of arguments for callables, like fn_loss.
- **fn_loss** (–) – A callable returning the loss of the current model.
- **fn_reference** (–) – A callable returning the reference values, in case of a comparative loss.
- **fn_kl_divergence** (–) – A callable returning the KL-divergence relative to the current model.
- **return_estimated_improvement** (–) – Returns the estimated improvement resulting from the natural gradient calculation if true.
- **source_variables** (–) – List of source variables to synchronize with.
- **global_variables** (–) – List of global variables to apply the proposed optimization step to.

Returns The optimization operation.

tf_optimizers = {'nadam': <sphinx.ext.autodoc._MockObject object>, 'adam': <sphinx.e

tf_step(*time*, *variables*, *arguments*, *fn_loss*, ***kwargs*)
Creates the TensorFlow operations for performing an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.
- **arguments** – Dict of arguments for callables, like fn_loss.
- **fn_loss** – A callable returning the loss of the current model.
- ****kwargs** – Additional arguments, not used.

Returns List of delta tensors corresponding to the updates for each optimized variable.

Module contents

class tensorforce.core.optimizers.**Optimizer**(*scope='optimizer'*, *summary_labels=None*)
Bases: object

Base class for optimizers which minimize a not yet further specified expression, usually some kind of loss function. More generally, an optimizer can be considered as some method of updating a set of variables.

__init__(*scope='optimizer'*, *summary_labels=None*)
Creates a new optimizer instance.

apply_step(*variables*, *deltas*)
Applies step deltas to variable values.

Parameters

- **variables** – List of variables.
- **deltas** – List of deltas of same length.

Returns The step-applied operation.

static from_spec(*spec*, *kwargs=None*)
Creates an optimizer from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the optimizer.

Returns List of summaries.

get_variables()
Returns the TensorFlow variables used by the optimizer.

Returns List of variables.

minimize(*time*, *variables*, ***kwargs*)
Performs an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.
- ****kwargs** – Additional optimizer-specific arguments. The following arguments are used by some optimizers:
- **arguments** (–) – Dict of arguments for callables, like fn_loss.

- **fn_loss** (–) – A callable returning the loss of the current model.
- **fn_reference** (–) – A callable returning the reference values, in case of a comparative loss.
- **fn_kl_divergence** (–) – A callable returning the KL-divergence relative to the current model.
- **return_estimated_improvement** (–) – Returns the estimated improvement resulting from the natural gradient calculation if true.
- **source_variables** (–) – List of source variables to synchronize with.
- **global_variables** (–) – List of global variables to apply the proposed optimization step to.

Returns The optimization operation.

tf_step(*time*, *variables*, ***kwargs*)
Creates the TensorFlow operations for performing an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.
- ****kwargs** – Additional arguments depending on the specific optimizer implementation. For instance, often includes `fn_loss` if a loss function is optimized.

Returns List of delta tensors corresponding to the updates for each optimized variable.

class tensorforce.core.optimizers.**MetaOptimizer**(*optimizer*, *scope='meta-optimizer'*, *summary_labels=()*, ***kwargs*)
Bases: *tensorforce.core.optimizers.optimizer.Optimizer*

A meta optimizer takes the optimization implemented by another optimizer and modifies/optimizes its proposed result. For example, line search might be applied to find a more optimal step size.

__init__(*optimizer*, *scope='meta-optimizer'*, *summary_labels=()*, ***kwargs*)
Creates a new meta optimizer instance.

Parameters **optimizer** – The optimizer which is modified by this meta optimizer.

apply_step(*variables*, *deltas*)
Applies step deltas to variable values.

Parameters

- **variables** – List of variables.
- **deltas** – List of deltas of same length.

Returns The step-applied operation.

from_spec(*spec*, *kwargs=None*)
Creates an optimizer from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the optimizer.

Returns List of summaries.

get_variables()

minimize(*time*, *variables*, ***kwargs*)
Performs an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.
- ****kwargs** – Additional optimizer-specific arguments. The following arguments are used by some optimizers:
- **arguments** (–) – Dict of arguments for callables, like fn_loss.
- **fn_loss** (–) – A callable returning the loss of the current model.
- **fn_reference** (–) – A callable returning the reference values, in case of a comparative loss.
- **fn_kl_divergence** (–) – A callable returning the KL-divergence relative to the current model.
- **return_estimated_improvement** (–) – Returns the estimated improvement resulting from the natural gradient calculation if true.
- **source_variables** (–) – List of source variables to synchronize with.
- **global_variables** (–) – List of global variables to apply the proposed optimization step to.

Returns The optimization operation.

tf_step(*time*, *variables*, ***kwargs*)
Creates the TensorFlow operations for performing an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.
- ****kwargs** – Additional arguments depending on the specific optimizer implementation. For instance, often includes `fn_loss` if a loss function is optimized.

Returns List of delta tensors corresponding to the updates for each optimized variable.

class tensorforce.core.optimizers.**GlobalOptimizer**(*optimizer*, *scope='global-optimizer'*, *summary_labels=()*)
Bases: *tensorforce.core.optimizers.meta_optimizer.MetaOptimizer*

The global optimizer applies an optimizer to the local variables. In addition, it also applies the update to a corresponding set of global variables and subsequently updates the local variables to the value of these global variables. Note: This is used for the current distributed mode, and will likely change with the next major version update.

__init__(*optimizer*, *scope='global-optimizer'*, *summary_labels=()*)
Creates a new global optimizer instance.

Parameters **optimizer** – The optimizer which is modified by this meta optimizer.

apply_step(*variables*, *deltas*)
Applies step deltas to variable values.

Parameters

- **variables** – List of variables.
- **deltas** – List of deltas of same length.

Returns The step-applied operation.

from_spec(*spec*, *kwargs=None*)
Creates an optimizer from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the optimizer.

Returns List of summaries.

get_variables()

minimize(*time*, *variables*, ***kwargs*)
Performs an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.
- ****kwargs** – Additional optimizer-specific arguments. The following arguments are used by some optimizers:
- **arguments** (–) – Dict of arguments for callables, like fn_loss.
- **fn_loss** (–) – A callable returning the loss of the current model.
- **fn_reference** (–) – A callable returning the reference values, in case of a comparative loss.
- **fn_kl_divergence** (–) – A callable returning the KL-divergence relative to the current model.
- **return_estimated_improvement** (–) – Returns the estimated improvement resulting from the natural gradient calculation if true.
- **source_variables** (–) – List of source variables to synchronize with.
- **global_variables** (–) – List of global variables to apply the proposed optimization step to.

Returns The optimization operation.

tf_step(*time*, *variables*, *global_variables*, ***kwargs*)
Creates the TensorFlow operations for performing an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.
- **global_variables** – List of global variables to apply the proposed optimization step to.
- ****kwargs** – ??? coming soon

Returns List of delta tensors corresponding to the updates for each optimized variable.

class tensorforce.core.optimizers.**TFOptimizer**(*optimizer*, *scope=None*, *summary_labels=()*, ***kwargs*)
Bases: *tensorforce.core.optimizers.optimizer.Optimizer*

Wrapper class for TensorFlow optimizers.

__init__(*optimizer*, *scope=None*, *summary_labels=()*, ***kwargs*)
Creates a new optimizer instance of a TensorFlow optimizer.

Parameters

- **optimizer** – The name of the optimizer, one of 'adadelta', 'adagrad', 'adam', 'nadam',
- **'momentum', 'rmsprop'.** (*'gradient_descent',*) –
- ****kwargs** – Additional arguments passed on to the TensorFlow optimizer constructor.

apply_step(*variables*, *deltas*)
Applies step deltas to variable values.

Parameters

- **variables** – List of variables.
- **deltas** – List of deltas of same length.

Returns The step-applied operation.

from_spec(*spec*, *kwargs=None*)
Creates an optimizer from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the optimizer.

Returns List of summaries.

get_variables()

static get_wrapper(*optimizer*)
Returns a TFOptimizer constructor callable for the given optimizer name.

Parameters

- **optimizer** – The name of the optimizer, one of 'adadelta', 'adagrad', 'adam', 'nadam',
- **'momentum', 'rmsprop'.** (*'gradient_descent',*) –

Returns The TFOptimizer constructor callable.

minimize(*time*, *variables*, ***kwargs*)
Performs an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.
- ****kwargs** – Additional optimizer-specific arguments. The following arguments are used by some optimizers:
- **arguments** (–) – Dict of arguments for callables, like fn_loss.
- **fn_loss** (–) – A callable returning the loss of the current model.
- **fn_reference** (–) – A callable returning the reference values, in case of a comparative loss.
- **fn_kl_divergence** (–) – A callable returning the KL-divergence relative to the current model.
- **return_estimated_improvement** (–) – Returns the estimated improvement resulting from the natural gradient calculation if true.
- **source_variables** (–) – List of source variables to synchronize with.
- **global_variables** (–) – List of global variables to apply the proposed optimization step to.

Returns The optimization operation.

tf_optimizers = {'nadam': <sphinx.ext.autodoc._MockObject object>, 'adam': <sphinx.e

tf_step(*time*, *variables*, *arguments*, *fn_loss*, ***kwargs*)
Creates the TensorFlow operations for performing an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.
- **arguments** – Dict of arguments for callables, like fn_loss.
- **fn_loss** – A callable returning the loss of the current model.
- ****kwargs** – Additional arguments, not used.

Returns List of delta tensors corresponding to the updates for each optimized variable.

class tensorforce.core.optimizers.**Evolutionary**(*learning_rate*, *num_samples=1*, *unroll_loop=False*, *scope='evolutionary'*, *summary_labels=()*)
Bases: *tensorforce.core.optimizers.optimizer.Optimizer*

Evolutionary optimizer which samples random perturbations and applies them either positively or negatively, depending on their improvement of the loss.

__init__(*learning_rate*, *num_samples=1*, *unroll_loop=False*, *scope='evolutionary'*, *summary_labels=()*)
Creates a new evolutionary optimizer instance.

Parameters

- **learning_rate** – Learning rate.
- **num_samples** – Number of sampled perturbations.

apply_step(*variables*, *deltas*)
Applies step deltas to variable values.

Parameters

- **variables** – List of variables.
- **deltas** – List of deltas of same length.

Returns The step-applied operation.

from_spec(*spec*, *kwargs=None*)
Creates an optimizer from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the optimizer.

Returns List of summaries.

get_variables()
Returns the TensorFlow variables used by the optimizer.

Returns List of variables.

minimize(*time*, *variables*, ***kwargs*)
Performs an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.
- ****kwargs** – Additional optimizer-specific arguments. The following arguments are used by some optimizers:
- **arguments** (–) – Dict of arguments for callables, like fn_loss.
- **fn_loss** (–) – A callable returning the loss of the current model.
- **fn_reference** (–) – A callable returning the reference values, in case of a comparative loss.
- **fn_kl_divergence** (–) – A callable returning the KL-divergence relative to the current model.
- **return_estimated_improvement** (–) – Returns the estimated improvement resulting from the natural gradient calculation if true.
- **source_variables** (–) – List of source variables to synchronize with.
- **global_variables** (–) – List of global variables to apply the proposed optimization step to.

Returns The optimization operation.

tf_step(*time*, *variables*, *arguments*, *fn_loss*, ***kwargs*)
Creates the TensorFlow operations for performing an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.
- **arguments** – Dict of arguments for callables, like fn_loss.
- **fn_loss** – A callable returning the loss of the current model.
- ****kwargs** – Additional arguments, not used.

Returns List of delta tensors corresponding to the updates for each optimized variable.

class tensorforce.core.optimizers.**NaturalGradient**(*learning_rate*, *cg_max_iterations=20*, *cg_damping=0.001*, *cg_unroll_loop=False*, *scope='natural-gradient'*, *summary_labels=()*)
Bases: *tensorforce.core.optimizers.optimizer.Optimizer*

Natural gradient optimizer.

__init__(*learning_rate*, *cg_max_iterations=20*, *cg_damping=0.001*, *cg_unroll_loop=False*, *scope='natural-gradient'*, *summary_labels=()*)
Creates a new natural gradient optimizer instance.

Parameters

- **learning_rate** – Learning rate, i.e. KL-divergence of distributions between optimization steps.
- **cg_max_iterations** – Conjugate gradient solver max iterations.
- **cg_damping** – Conjugate gradient solver damping factor.

- **cg_unroll_loop** – Unroll conjugate gradient loop if true.

apply_step(*variables*, *deltas*)
Applies step deltas to variable values.

Parameters

- **variables** – List of variables.
- **deltas** – List of deltas of same length.

Returns The step-applied operation.

from_spec(*spec*, *kwargs=None*)
Creates an optimizer from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the optimizer.

Returns List of summaries.

get_variables()
Returns the TensorFlow variables used by the optimizer.

Returns List of variables.

minimize(*time*, *variables*, ***kwargs*)
Performs an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.
- ****kwargs** – Additional optimizer-specific arguments. The following arguments are used by some optimizers:
- **arguments** (–) – Dict of arguments for callables, like fn_loss.
- **fn_loss** (–) – A callable returning the loss of the current model.
- **fn_reference** (–) – A callable returning the reference values, in case of a comparative loss.
- **fn_kl_divergence** (–) – A callable returning the KL-divergence relative to the current model.
- **return_estimated_improvement** (–) – Returns the estimated improvement resulting from the natural gradient calculation if true.
- **source_variables** (–) – List of source variables to synchronize with.
- **global_variables** (–) – List of global variables to apply the proposed optimization step to.

Returns The optimization operation.

tf_step(*time*, *variables*, *arguments*, *fn_loss*, *fn_kl_divergence*, *return_estimated_improvement=False*, ***kwargs*)
Creates the TensorFlow operations for performing an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.

- **arguments** – Dict of arguments for callables, like fn_loss.
- **fn_loss** – A callable returning the loss of the current model.
- **fn_kl_divergence** – A callable returning the KL-divergence relative to the current model.
- **return_estimated_improvement** – Returns the estimated improvement resulting from the natural gradient calculation if true.
- ****kwargs** – Additional arguments, not used.

Returns List of delta tensors corresponding to the updates for each optimized variable.

class tensorforce.core.optimizers.**ClippedStep**(*optimizer*, *clipping_value*, *scope='clipped-step'*, *summary_labels=()*)

Bases: *tensorforce.core.optimizers.meta_optimizer.MetaOptimizer*

The clipped-shep meta optimizer clips the values of the optimization step proposed by another optimizer.

__init__(*optimizer*, *clipping_value*, *scope='clipped-step'*, *summary_labels=()*)

Creates a new multi-step meta optimizer instance.

Parameters

- **optimizer** – The optimizer which is modified by this meta optimizer.
- **clipping_value** – Clip deltas at this value.

apply_step(*variables*, *deltas*)

Applies step deltas to variable values.

Parameters

- **variables** – List of variables.
- **deltas** – List of deltas of same length.

Returns The step-applied operation.

from_spec(*spec*, *kwargs=None*)

Creates an optimizer from a specification dict.

get_summaries()

Returns the TensorFlow summaries reported by the optimizer.

Returns List of summaries.

get_variables()

minimize(*time*, *variables*, ***kwargs*)

Performs an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.
- ****kwargs** – Additional optimizer-specific arguments. The following arguments are used by some optimizers:
- **arguments** (–) – Dict of arguments for callables, like fn_loss.
- **fn_loss** (–) – A callable returning the loss of the current model.

- **fn_reference** (–) – A callable returning the reference values, in case of a comparative loss.
- **fn_kl_divergence** (–) – A callable returning the KL-divergence relative to the current model.
- **return_estimated_improvement** (–) – Returns the estimated improvement resulting from the natural gradient calculation if true.
- **source_variables** (–) – List of source variables to synchronize with.
- **global_variables** (–) – List of global variables to apply the proposed optimization step to.

Returns The optimization operation.

tf_step(*time*, *variables*, ***kwargs*)
Creates the TensorFlow operations for performing an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.
- ****kwargs** – Additional arguments passed on to the internal optimizer.

Returns List of delta tensors corresponding to the updates for each optimized variable.

class tensorforce.core.optimizers.**MultiStep**(*optimizer*, *num_steps=10*, *unroll_loop=False*, *scope='multi-step'*, *summary_labels=()*)
Bases: *tensorforce.core.optimizers.meta_optimizer.MetaOptimizer*

The multi-step meta optimizer repeatedly applies the optimization step proposed by another optimizer a number of times.

__init__(*optimizer*, *num_steps=10*, *unroll_loop=False*, *scope='multi-step'*, *summary_labels=()*)
Creates a new multi-step meta optimizer instance.

Parameters

- **optimizer** – The optimizer which is modified by this meta optimizer.
- **num_steps** – Number of optimization steps to perform.

apply_step(*variables*, *deltas*)
Applies step deltas to variable values.

Parameters

- **variables** – List of variables.
- **deltas** – List of deltas of same length.

Returns The step-applied operation.

from_spec(*spec*, *kwargs=None*)
Creates an optimizer from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the optimizer.

Returns List of summaries.

get_variables()

minimize(*time*, *variables*, ***kwargs*)
Performs an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.
- ****kwargs** – Additional optimizer-specific arguments. The following arguments are used by some optimizers:
- **arguments** (–) – Dict of arguments for callables, like fn_loss.
- **fn_loss** (–) – A callable returning the loss of the current model.
- **fn_reference** (–) – A callable returning the reference values, in case of a comparative loss.
- **fn_kl_divergence** (–) – A callable returning the KL-divergence relative to the current model.
- **return_estimated_improvement** (–) – Returns the estimated improvement resulting from the natural gradient calculation if true.
- **source_variables** (–) – List of source variables to synchronize with.
- **global_variables** (–) – List of global variables to apply the proposed optimization step to.

Returns The optimization operation.

tf_step(*time*, *variables*, *arguments*, *fn_reference=None*, ***kwargs*)
Creates the TensorFlow operations for performing an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.
- **arguments** – Dict of arguments for callables, like fn_loss.
- **fn_reference** – A callable returning the reference values, in case of a comparative loss.
- ****kwargs** – Additional arguments passed on to the internal optimizer.

Returns List of delta tensors corresponding to the updates for each optimized variable.

class tensorforce.core.optimizers.**OptimizedStep**(*optimizer*, *ls_max_iterations=10*, *ls_accept_ratio=0.9*, *ls_mode='exponential'*, *ls_parameter=0.5*, *ls_unroll_loop=False*, *scope='optimized-step'*, *summary_labels=()*)
Bases: *tensorforce.core.optimizers.meta_optimizer.MetaOptimizer*

The optimized-step meta optimizer applies line search to the proposed optimization step of another optimizer to find a more optimal step size.

__init__(*optimizer*, *ls_max_iterations=10*, *ls_accept_ratio=0.9*, *ls_mode='exponential'*, *ls_parameter=0.5*, *ls_unroll_loop=False*, *scope='optimized-step'*, *summary_labels=()*)
Creates a new optimized step meta optimizer instance.

Parameters

- **optimizer** – The optimizer which is modified by this meta optimizer.
- **ls_max_iterations** – Maximum number of line search iterations.
- **ls_accept_ratio** – Line search acceptance ratio.
- **ls_mode** – Line search mode, see LineSearch solver.
- **ls_parameter** – Line search parameter, see LineSearch solver.
- **ls_unroll_loop** – Unroll line search loop if true.

apply_step(*variables*, *deltas*)
Applies step deltas to variable values.

Parameters

- **variables** – List of variables.
- **deltas** – List of deltas of same length.

Returns The step-applied operation.

from_spec(*spec*, *kwargs=None*)
Creates an optimizer from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the optimizer.

Returns List of summaries.

get_variables()

minimize(*time*, *variables*, ***kwargs*)
Performs an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.
- ****kwargs** – Additional optimizer-specific arguments. The following arguments are used by some optimizers:
- **arguments** (–) – Dict of arguments for callables, like fn_loss.
- **fn_loss** (–) – A callable returning the loss of the current model.
- **fn_reference** (–) – A callable returning the reference values, in case of a comparative loss.
- **fn_kl_divergence** (–) – A callable returning the KL-divergence relative to the current model.
- **return_estimated_improvement** (–) – Returns the estimated improvement resulting from the natural gradient calculation if true.
- **source_variables** (–) – List of source variables to synchronize with.
- **global_variables** (–) – List of global variables to apply the proposed optimization step to.

Returns The optimization operation.

tf_step(*time*, *variables*, *arguments*, *fn_loss*, *fn_reference*, ***kwargs*)
Creates the TensorFlow operations for performing an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.
- **arguments** – Dict of arguments for callables, like fn_loss.
- **fn_loss** – A callable returning the loss of the current model.
- **fn_reference** – A callable returning the reference values, in case of a comparative loss.
- ****kwargs** – Additional arguments passed on to the internal optimizer.

Returns List of delta tensors corresponding to the updates for each optimized variable.

class tensorforce.core.optimizers.**SubsamplingStep**(*optimizer*, *fraction=0.1*, *scope='subsampling-step'*, *summary_labels=()*)
Bases: *tensorforce.core.optimizers.meta_optimizer.MetaOptimizer*

The subsampling-step meta optimizer randomly samples a subset of batch instances to calculate the optimization step of another optimizer.

__init__(*optimizer*, *fraction=0.1*, *scope='subsampling-step'*, *summary_labels=()*)
Creates a new subsampling-step meta optimizer instance.

Parameters

- **optimizer** – The optimizer which is modified by this meta optimizer.
- **fraction** – The fraction of instances of the batch to subsample.

apply_step(*variables*, *deltas*)
Applies step deltas to variable values.

Parameters

- **variables** – List of variables.
- **deltas** – List of deltas of same length.

Returns The step-applied operation.

from_spec(*spec*, *kwargs=None*)
Creates an optimizer from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the optimizer.

Returns List of summaries.

get_variables()

minimize(*time*, *variables*, ***kwargs*)
Performs an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.

- ****kwargs** – Additional optimizer-specific arguments. The following arguments are used by some optimizers:
- **arguments** (–) – Dict of arguments for callables, like fn_loss.
- **fn_loss** (–) – A callable returning the loss of the current model.
- **fn_reference** (–) – A callable returning the reference values, in case of a comparative loss.
- **fn_kl_divergence** (–) – A callable returning the KL-divergence relative to the current model.
- **return_estimated_improvement** (–) – Returns the estimated improvement resulting from the natural gradient calculation if true.
- **source_variables** (–) – List of source variables to synchronize with.
- **global_variables** (–) – List of global variables to apply the proposed optimization step to.

Returns The optimization operation.

tf_step(*time*, *variables*, *arguments*, ***kwargs*)
Creates the TensorFlow operations for performing an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.
- **arguments** – Dict of arguments for callables, like fn_loss.
- ****kwargs** – Additional arguments passed on to the internal optimizer.

Returns List of delta tensors corresponding to the updates for each optimized variable.

class tensorforce.core.optimizers.**Synchronization**(*sync_frequency=1*, *update_weight=1.0*, *scope='synchronization'*, *summary_labels=()*)
Bases: *tensorforce.core.optimizers.optimizer.Optimizer*

The synchronization optimizer updates variables periodically to the value of a corresponding set of source variables.

__init__(*sync_frequency=1*, *update_weight=1.0*, *scope='synchronization'*, *summary_labels=()*)
Creates a new synchronization optimizer instance.

Parameters

- **sync_frequency** – The interval between optimization calls actually performing a
- **step.** (synchronization) –
- **update_weight** – The update weight, 1.0 meaning a full assignment of the source
- **values.** (*variables*) –

apply_step(*variables*, *deltas*)
Applies step deltas to variable values.

Parameters

- **variables** – List of variables.
- **deltas** – List of deltas of same length.

Returns The step-applied operation.

from_spec(*spec*, *kwargs=None*)
Creates an optimizer from a specification dict.

get_summaries()
Returns the TensorFlow summaries reported by the optimizer.

Returns List of summaries.

get_variables()
Returns the TensorFlow variables used by the optimizer.

Returns List of variables.

minimize(*time*, *variables*, ***kwargs*)
Performs an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.
- ****kwargs** – Additional optimizer-specific arguments. The following arguments are used by some optimizers:
- **arguments** (–) – Dict of arguments for callables, like fn_loss.
- **fn_loss** (–) – A callable returning the loss of the current model.
- **fn_reference** (–) – A callable returning the reference values, in case of a comparative loss.
- **fn_kl_divergence** (–) – A callable returning the KL-divergence relative to the current model.
- **return_estimated_improvement** (–) – Returns the estimated improvement resulting from the natural gradient calculation if true.
- **source_variables** (–) – List of source variables to synchronize with.
- **global_variables** (–) – List of global variables to apply the proposed optimization step to.

Returns The optimization operation.

tf_step(*time*, *variables*, *source_variables*, ***kwargs*)
Creates the TensorFlow operations for performing an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.
- **source_variables** – List of source variables to synchronize with.
- ****kwargs** – Additional arguments, not used.

Returns List of delta tensors corresponding to the updates for each optimized variable.

tensorforce.core.preprocessing package

Submodules

tensorforce.core.preprocessing.clip module

tensorforce.core.preprocessing.divide module

tensorforce.core.preprocessing.grayscale module

tensorforce.core.preprocessing.image_resize module

tensorforce.core.preprocessing.normalize module

tensorforce.core.preprocessing.preprocessor module

tensorforce.core.preprocessing.preprocessor_stack module

tensorforce.core.preprocessing.running_standardize module

tensorforce.core.preprocessing.sequence module

tensorforce.core.preprocessing.standardize module

Module contents

Module contents

tensorforce.environments package

Submodules

tensorforce.environments.environment module

class tensorforce.environments.environment.**Environment**

Bases: object

Base environment class.

__init__

x.**init**(...) initializes x; see help(type(x)) for signature

actions

Return the action space. Might include subdicts if multiple actions are available simultaneously.

Returns: dict of action properties (continuous, number of actions)

close()

Close environment. No other method calls possible afterwards.

execute(*actions*)

Executes action, observes next state(s) and reward.

Parameters **actions** – Actions to execute.

Returns (Dict of) next state(s), boolean indicating terminal, and reward signal.

static from_spec(*spec*, *kwargs*)
Creates an environment from a specification dict.

reset()
Reset environment and setup for new episode.

Returns initial state of reset environment.

seed(*seed*)
Sets the random seed of the environment to the given value (current time, if seed=None). Naturally deterministic Environments (e.g. ALE or some gym Envs) don't have to implement this method.

Parameters **seed** (*int*) – The seed to use for initializing the pseudo-random number generator (default=epoch time in sec).

Returns: The actual seed (int) used OR None if Environment did not override this method (no seeding supported).

states
Return the state space. Might include subdicts if multiple states are available simultaneously.

Returns: dict of state properties (shape and type).

tensorforce.environments.minimal_test module

Module contents

class tensorforce.environments.**Environment**
Bases: object

Base environment class.

__init__
x.**init**(...) initializes x; see help(type(x)) for signature

actions
Return the action space. Might include subdicts if multiple actions are available simultaneously.

Returns: dict of action properties (continuous, number of actions)

close()
Close environment. No other method calls possible afterwards.

execute(*actions*)
Executes action, observes next state(s) and reward.

Parameters **actions** – Actions to execute.

Returns (Dict of) next state(s), boolean indicating terminal, and reward signal.

static from_spec(*spec*, *kwargs*)
Creates an environment from a specification dict.

reset()
Reset environment and setup for new episode.

Returns initial state of reset environment.

seed(*seed*)
Sets the random seed of the environment to the given value (current time, if seed=None). Naturally deterministic Environments (e.g. ALE or some gym Envs) don't have to implement this method.

Parameters **seed** (*int*) – The seed to use for initializing the pseudo-random number generator (default=epoch time in sec).

Returns: The actual seed (int) used OR None if Environment did not override this method (no seeding supported).

states
Return the state space. Might include subdicts if multiple states are available simultaneously.

Returns: dict of state properties (shape and type).

class tensorforce.environments.**MinimalTest**(*specification*)
Bases: *tensorforce.environments.environment.Environment*

__init__(*specification*)
Initializes a minimal test environment, which is used for the unit tests. Given a specification of actions types and shapes, the environment states consist of the same number of pairs (x, y). The (mean of) an action a gives the next state via (1-a, a), and the 'correct' state is always (0, 1).

Parameters **specification** – Takes a dict type (keys)-> shape (values specifying the action structure of the environment. Use shape () for single scalar actions.

actions

close()

execute(*actions*)

from_spec(*spec*, *kwargs*)
Creates an environment from a specification dict.

reset()

seed(*seed*)
Sets the random seed of the environment to the given value (current time, if seed=None). Naturally deterministic Environments (e.g. ALE or some gym Envs) don't have to implement this method.

Parameters **seed** (*int*) – The seed to use for initializing the pseudo-random number generator (default=epoch time in sec).

Returns: The actual seed (int) used OR None if Environment did not override this method (no seeding supported).

states

tensorforce.execution package

Submodules

tensorforce.execution.runner module

tensorforce.execution.runner.**DistributedTFRunner**
alias of *Runner*

class tensorforce.execution.runner.**Runner**(*agent*, *environment*, *repeat_actions=1*, *history=None*, *id_=0*)
Bases: tensorforce.execution.base_runner.BaseRunner

Simple runner for non-realtime single-process execution.

__init__(*agent*, *environment*, *repeat_actions=1*, *history=None*, *id_=0*)
Initialize a single Runner object (one Agent/one Environment).

Parameters **id*** – The ID of this Runner (for distributed TF runs).

:type id*: int

close()

episode
Deprecated property `episode` -> global_episode.

episode_timestep

reset(*history=None*)
Resets the Runner's internal stats counters. If history is empty, use default values in history.get().

Parameters **history** (*dict*) – A dictionary containing an already run experiment's results. Keys should be: episode_rewards (list of rewards), episode_timesteps (lengths of episodes), episode_times (run-times)

run(*num_timesteps=None*, *num_episodes=None*, *max_episode_timesteps=None*, *deterministic=False*, *episode_finished=None*, *summary_report=None*, *summary_interval=None*, *timesteps=None*, *episodes=None*)

Parameters

- **timesteps** (*int*) – Deprecated; see num_timesteps.
- **episodes** (*int*) – Deprecated; see num_episodes.

timestep
Deprecated property `timestep` -> global_timestep.

tensorforce.execution.runner.**SingleRunner**
alias of *Runner*

tensorforce.execution.threaded_runner module

class tensorforce.execution.threaded_runner.**ThreadedRunner**(*agent*, *environment*, *repeat_actions=1*, *save_path=None*, *save_episodes=None*, *save_frequency=None*, *save_frequency_unit=None*, *agents=None*, *environments=None*)

Bases: `tensorforce.execution.base_runner.BaseRunner`

Runner for non-realtime threaded execution of multiple agents.

__init__(*agent*, *environment*, *repeat_actions=1*, *save_path=None*, *save_episodes=None*, *save_frequency=None*, *save_frequency_unit=None*, *agents=None*, *environments=None*)
Initialize a ThreadedRunner object.

Parameters

- **save_path** (*str*) – Path where to save the shared model.
- **save_episodes** (*int*) – Deprecated: Every how many (global) episodes do we save the shared model?

- **save_frequency** (*int*) – The frequency with which to save the model (could be sec, steps, or episodes).
- **save_frequency_unit** (*str*) – "s" (sec), "t" (timesteps), "e" (episodes)
- **agents** (*List[Agent]*) – Deprecated: List of Agent objects. Use `agent`, instead.
- **environments** (*List[Environment]*) – Deprecated: List of Environment objects. Use `environment`, instead.

agents

close()

environments

episode
Deprecated property `episode` -> global_episode.

episode_lengths

global_step

reset(*history=None*)
Resets the Runner's internal stats counters. If history is empty, use default values in history.get().

Parameters **history** (*dict*) – A dictionary containing an already run experiment's results. Keys should be: episode_rewards (list of rewards), episode_timesteps (lengths of episodes), episode_times (run-times)

run(*num_episodes=-1*, *max_episode_timesteps=-1*, *episode_finished=None*, *summary_report=None*, *summary_interval=0*, *num_timesteps=None*, *deterministic=False*, *episodes=None*, *max_timesteps=None*)
Executes this runner by starting all Agents in parallel (each one in one thread).

Parameters

- **episodes** (*int*) – Deprecated; see num_episodes.
- **max_timesteps** (*int*) – Deprecated; see max_episode_timesteps.

timestep
Deprecated property `timestep` -> global_timestep.

`tensorforce.execution.threaded_runner.`**WorkerAgentGenerator**(*agent_class*)
Worker Agent generator, receives an Agent class and creates a Worker Agent class that inherits from that Agent.

`tensorforce.execution.threaded_runner.`**clone_worker_agent**(*agent*, *factor*, *environment*, *network*, *agent_config*)
Clones a given Agent (`factor` times) and returns a list of the cloned Agents with the original Agent in the first slot.

Parameters

- **agent** (`Agent`) – The Agent object to clone.
- **factor** (*int*) – The length of the final list.
- **environment** (`Environment`) – The Environment to use for all cloned agents.
- **network** (`LayeredNetwork`) – The Network to use (or None) for an Agent's Model.
- **agent_config** (*dict*) – A dict of Agent specifications passed into the Agent's c'tor as kwargs.

Returns The list with `factor` cloned agents (including the original one).

Module contents

class tensorforce.execution.**BaseRunner**(*agent*, *environment*, *repeat_actions=1*, *history=None*)

Bases: object

Base class for all runner classes. Implements the run method.

__init__(*agent*, *environment*, *repeat_actions=1*, *history=None*)

Parameters

- **agent** (Agent) – Agent object (or list of Agent objects) to use for the run.
- **environment** (Environment) – Environment object (or list of Environment objects) to use for the run.
- **repeat_actions** (*int*) – How many times the same given action will be repeated in subsequent calls to Environment's execute method. Rewards collected in these calls are accumulated and reported as a sum in the following call to Agent's observe method.
- **history** (*dict*) – A dictionary containing an already run experiment's results. Keys should be: episode_rewards (list of rewards), episode_timesteps (lengths of episodes), episode_times (run-times)

close()

Should perform clean up operations on Runner's Agent(s) and Environment(s).

episode

Deprecated property episode -> global_episode.

reset(*history=None*)

Resets the Runner's internal stats counters. If history is empty, use default values in history.get().

Parameters **history** (*dict*) – A dictionary containing an already run experiment's results. Keys should be: episode_rewards (list of rewards), episode_timesteps (lengths of episodes), episode_times (run-times)

run(*num_episodes*, *num_timesteps*, *max_episode_timesteps*, *deterministic*, *episode_finished*, *summary_report*, *summary_interval*)

Executes this runner by starting to act (via Agent(s)) in the given Environment(s). Stops execution according to certain conditions (e.g. max. number of episodes, etc..). Calls callback functions after each episode and/or after some summary criteria are met.

Parameters

- **num_episodes** (*int*) – Max. number of episodes to run globally in total (across all threads/workers).
- **num_timesteps** (*int*) – Max. number of time steps to run globally in total (across all threads/workers)
- **max_episode_timesteps** (*int*) – Max. number of timesteps per episode.
- **deterministic** (*bool*) – Whether to use exploration when selecting actions.
- **episode_finished** (*callable*) – A function to be called once an episodes has finished. Should take a BaseRunner object and some worker ID (e.g. thread-ID or task-ID). Can decide for itself every how many episodes it should report something and what to report.
- **summary_report** (*callable*) – Deprecated; Function that could produce a summary over the training progress so far.

- **summary_interval** (*int*) – Deprecated; The number of time steps to execute (globally) before summary_report is called.

timestep
Deprecated property `timestep` -> global_timestep.

`tensorforce.execution.`**SingleRunner**
alias of *Runner*

`tensorforce.execution.`**DistributedTFRunner**
alias of *Runner*

class `tensorforce.execution.`**Runner**(*agent*, *environment*, *repeat_actions=1*, *history=None*, *id_=0*)
Bases: `tensorforce.execution.base_runner.BaseRunner`

Simple runner for non-realtime single-process execution.

__init__(*agent*, *environment*, *repeat_actions=1*, *history=None*, *id_=0*)
Initialize a single Runner object (one Agent/one Environment).

Parameters **id*** – The ID of this Runner (for distributed TF runs).

:type id*: int

close()

episode
Deprecated property `episode` -> global_episode.

episode_timestep

reset(*history=None*)
Resets the Runner's internal stats counters. If history is empty, use default values in history.get().

Parameters **history** (*dict*) – A dictionary containing an already run experiment's results. Keys should be: episode_rewards (list of rewards), episode_timesteps (lengths of episodes), episode_times (run-times)

run(*num_timesteps=None*, *num_episodes=None*, *max_episode_timesteps=None*, *deterministic=False*, *episode_finished=None*, *summary_report=None*, *summary_interval=None*, *timesteps=None*, *episodes=None*)

Parameters

- **timesteps** (*int*) – Deprecated; see num_timesteps.
- **episodes** (*int*) – Deprecated; see num_episodes.

timestep
Deprecated property `timestep` -> global_timestep.

class `tensorforce.execution.`**ThreadedRunner**(*agent*, *environment*, *repeat_actions=1*, *save_path=None*, *save_episodes=None*, *save_frequency=None*, *save_frequency_unit=None*, *agents=None*, *environments=None*)
Bases: `tensorforce.execution.base_runner.BaseRunner`

Runner for non-realtime threaded execution of multiple agents.

__init__(*agent*, *environment*, *repeat_actions=1*, *save_path=None*, *save_episodes=None*, *save_frequency=None*, *save_frequency_unit=None*, *agents=None*, *environments=None*)
Initialize a ThreadedRunner object.

Parameters

- **save_path** (*str*) – Path where to save the shared model.
- **save_episodes** (*int*) – Deprecated: Every how many (global) episodes do we save the shared model?
- **save_frequency** (*int*) – The frequency with which to save the model (could be sec, steps, or episodes).
- **save_frequency_unit** (*str*) – "s" (sec), "t" (timesteps), "e" (episodes)
- **agents** (*List[Agent]*) – Deprecated: List of Agent objects. Use `agent`, instead.
- **environments** (*List[Environment]*) – Deprecated: List of Environment objects. Use `environment`, instead.

agents

close()

environments

episode
Deprecated property `episode` -> global_episode.

episode_lengths

global_step

reset(*history=None*)
Resets the Runner's internal stats counters. If history is empty, use default values in history.get().

Parameters **history** (*dict*) – A dictionary containing an already run experiment's results. Keys should be: episode_rewards (list of rewards), episode_timesteps (lengths of episodes), episode_times (run-times)

run(*num_episodes=-1*, *max_episode_timesteps=-1*, *episode_finished=None*, *summary_report=None*, *summary_interval=0*, *num_timesteps=None*, *deterministic=False*, *episodes=None*, *max_timesteps=None*)
Executes this runner by starting all Agents in parallel (each one in one thread).

Parameters

- **episodes** (*int*) – Deprecated; see num_episodes.
- **max_timesteps** (*int*) – Deprecated; see max_episode_timesteps.

timestep
Deprecated property `timestep` -> global_timestep.

tensorforce.execution.**WorkerAgentGenerator**(*agent_class*)
Worker Agent generator, receives an Agent class and creates a Worker Agent class that inherits from that Agent.

tensorforce.models package

Submodules

tensorforce.models.constant_model module

class tensorforce.models.constant_model.**ConstantModel**(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*, *action_values*)
Bases: *tensorforce.models.model.Model*

Utility class to return constant actions of a desired shape and with given bounds.

__init__(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*, *action_values*)

act(*states*, *internals*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)
Does a forward pass through the model to retrieve action (outputs) given inputs for state (and internal state, if applicable (e.g. RNNs))

Parameters

- **states** (*dict*) – Dict of state values (each key represents one state space component).
- **internals** (*dict*) – Dict of internal state values (each key represents one internal state component).
- **deterministic** (*bool*) – If True, will not apply exploration after actions are calculated.
- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).

Returns

- Actual action-outputs (batched if state input is a batch).

Return type tuple

as_local_model()

close()

create_act_operations(*states*, *internals*, *deterministic*, *independent*)

create_observe_operations(*terminal*, *reward*)

create_operations(*states*, *internals*, *actions*, *terminal*, *reward*, *deterministic*, *independent*)
Creates output operations for acting, observing and interacting with the memory.

get_component(*component_name*)
Looks up a component by its name.

Parameters **component_name** – The name of the component to look up.

Returns The component for the provided name or None if there is no such component.

get_components()
Returns a dictionary of component name to component of all the components within this model.

Returns (dict) The mapping of name to component.

get_feed_dict(*states=None*, *internals=None*, *actions=None*, *terminal=None*, *reward=None*, *deterministic=None*, *independent=None*)

get_savable_components()
Returns the list of all of the components this model consists of that can be individually saved and restored. For instance the network or distribution.

Returns List of util.SavableComponent

get_summaries()
Returns the TensorFlow summaries reported by the model

Returns List of summaries

get_variables(*include_submodules=False*, *include_nontrainable=False*)
Returns the TensorFlow variables used by the model.

Parameters

- **include_submodules** – Includes variables of submodules (e.g. baseline, target network) if true.
- **include_nontrainable** – Includes non-trainable variables if true.

Returns List of variables.

initialize(*custom_getter*)
Creates the TensorFlow placeholders and functions for this model. Moreover adds the internal state placeholders and initialization values to the model.

Parameters **custom_getter** – The `custom_getter_` object to use for `tf.make_template` when creating TensorFlow functions.

observe(*terminal*, *reward*)
Adds an observation (reward and is-terminal) to the model without updating its trainable variables.

Parameters

- **terminal** (*bool*) – Whether the episode has terminated.
- **reward** (*float*) – The observed reward value.

Returns The value of the model-internal episode counter.

reset()
Resets the model to its initial state on episode start. This should also reset all preprocessor(s).

Returns Current episode, timestep counter and the shallow-copied list of internal state initialization Tensors.

Return type tuple

restore(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters

- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

restore_component(*component_name*, *save_path*)
Restores a component's parameters from a save location.

Parameters

- **component_name** – The component to restore.
- **save_path** – The save location.

save(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters

- **directory** – Optional checkpoint directory.
- **append_timestep** – Appends the current timestep to the checkpoint file if true.

Returns Checkpoint path where the model was saved.

save_component(*component_name*, *save_path*)
Saves a component of this model to the designated location.

Parameters

- **component_name** – The component to save.
- **save_path** – The location to save to.

Returns Checkpoint path where the component was saved.

setup()
Sets up the TensorFlow model graph and initializes (and enters) the TensorFlow session.

tf_action_exploration(*action*, *exploration*, *action_spec*)
Applies optional exploration to the action (post-processor for action outputs).

Parameters

- **action** (*tf.Tensor*) – The original output action tensor (to be post-processed).
- **exploration** (Exploration) – The Exploration object to use.
- **action_spec** (*dict*) – Dict specifying the action space.

Returns The post-processed action output tensor.

tf_actions_and_internals(*states*, *internals*, *deterministic*)

tf_initialize()

tf_observe_timestep(*states*, *internals*, *actions*, *terminal*, *reward*)

tf_preprocess(*states*, *actions*, *reward*)

tensorforce.models.distribution_model module

class tensorforce.models.distribution_model.**DistributionModel**(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*, *variable_noise*, *states_preprocessing*, *actions_exploration*, *reward_preprocessing*, *update_mode*, *memory*, *optimizer*, *discount*, *network*, *distributions*, *entropy_regularization*, *requires_deterministic*)

Bases: tensorforce.models.memory_model.MemoryModel

Base class for models using distributions parametrized by a neural network.

COMPONENT_DISTRIBUTION = 'distribution'

COMPONENT_NETWORK = 'network'

__init__(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*, *variable_noise*, *states_preprocessing*, *actions_exploration*, *reward_preprocessing*, *update_mode*, *memory*, *optimizer*, *discount*, *network*, *distributions*, *entropy_regularization*, *requires_deterministic*)

act(*states*, *internals*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)
Does a forward pass through the model to retrieve action (outputs) given inputs for state (and internal state, if applicable (e.g. RNNs))

Parameters

- **states** (*dict*) – Dict of state values (each key represents one state space component).
- **internals** (*dict*) – Dict of internal state values (each key represents one internal state component).
- **deterministic** (*bool*) – If True, will not apply exploration after actions are calculated.
- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).

Returns

- Actual action-outputs (batched if state input is a batch).

Return type tuple

as_local_model()

close()

create_act_operations(*states*, *internals*, *deterministic*, *independent*)

create_distributions()

create_observe_operations(*terminal*, *reward*)

create_operations(*states*, *internals*, *actions*, *terminal*, *reward*, *deterministic*, *independent*)

get_component(*component_name*)
Looks up a component by its name.

Parameters **component_name** – The name of the component to look up.

Returns The component for the provided name or None if there is no such component.

get_components()

get_feed_dict(*states=None*, *internals=None*, *actions=None*, *terminal=None*, *reward=None*, *deterministic=None*, *independent=None*)

get_savable_components()
Returns the list of all of the components this model consists of that can be individually saved and restored. For instance the network or distribution.

Returns List of util.SavableComponent

get_summaries()

get_variables(*include_submodules=False*, *include_nontrainable=False*)

import_experience(*states*, *internals*, *actions*, *terminal*, *reward*)
Stores experiences.

initialize(*custom_getter*)

observe(*terminal*, *reward*)
Adds an observation (reward and is-terminal) to the model without updating its trainable variables.

Parameters

- **terminal** (*bool*) – Whether the episode has terminated.
- **reward** (*float*) – The observed reward value.

Returns The value of the model-internal episode counter.

optimizer_arguments(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*)

reset()
Resets the model to its initial state on episode start. This should also reset all preprocessor(s).

Returns Current episode, timestep counter and the shallow-copied list of internal state initialization Tensors.

Return type tuple

restore(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters

- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

restore_component(*component_name*, *save_path*)
Restores a component's parameters from a save location.

Parameters

- **component_name** – The component to restore.
- **save_path** – The save location.

save(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters

- **directory** – Optional checkpoint directory.
- **append_timestep** – Appends the current timestep to the checkpoint file if true.

Returns Checkpoint path where the model was saved.

save_component(*component_name*, *save_path*)
Saves a component of this model to the designated location.

Parameters

- **component_name** – The component to save.
- **save_path** – The location to save to.

Returns Checkpoint path where the component was saved.

setup()
Sets up the TensorFlow model graph and initializes (and enters) the TensorFlow session.

tf_action_exploration(*action*, *exploration*, *action_spec*)
Applies optional exploration to the action (post-processor for action outputs).

Parameters

- **action** (*tf.Tensor*) – The original output action tensor (to be post-processed).
- **exploration** (Exploration) – The Exploration object to use.
- **action_spec** (*dict*) – Dict specifying the action space.

Returns The post-processed action output tensor.

tf_actions_and_internals(*states*, *internals*, *deterministic*)

tf_discounted_cumulative_reward(*terminal*, *reward*, *discount*, *final_reward=0.0*)
Creates the TensorFlow operations for calculating the discounted cumulative rewards for a given sequence of rewards.

Parameters

- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **discount** – Discount factor.
- **final_reward** – Last reward value in the sequence.

Returns Discounted cumulative reward tensor.

tf_import_experience(*states*, *internals*, *actions*, *terminal*, *reward*)
Imports experiences into the TensorFlow memory structure. Can be used to import off-policy data.

Parameters

- **states** – Dict of state values to import with keys as state names and values as values to set.
- **internals** – Internal values to set, can be fetched from agent via agent.current_internals if no values available.
- **actions** – Dict of action values to import with keys as action names and values as values to set.
- **terminal** – Terminal value(s)
- **reward** – Reward value(s)

tf_initialize()

tf_kl_divergence(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)

tf_loss(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)
Creates the TensorFlow operations for calculating the full loss of a batch.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **actions** – Dict of action tensors.

- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **next_states** – Dict of successor state tensors.
- **next_internals** – List of posterior internal state tensors.
- **update** – Boolean tensor indicating whether this call happens during an update.
- **reference** – Optional reference tensor(s), in case of a comparative loss.

Returns Loss tensor.

tf_loss_per_instance(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)

Creates the TensorFlow operations for calculating the loss per batch instance.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **actions** – Dict of action tensors.
- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **next_states** – Dict of successor state tensors.
- **next_internals** – List of posterior internal state tensors.
- **update** – Boolean tensor indicating whether this call happens during an update.
- **reference** – Optional reference tensor(s), in case of a comparative loss.

Returns Loss per instance tensor.

tf_observe_timestep(*states*, *internals*, *actions*, *terminal*, *reward*)

tf_optimization(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states=None*, *next_internals=None*)

Creates the TensorFlow operations for performing an optimization update step based on the given input states and actions batch.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **actions** – Dict of action tensors.
- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **next_states** – Dict of successor state tensors.
- **next_internals** – List of posterior internal state tensors.

Returns The optimization operation.

tf_preprocess(*states*, *actions*, *reward*)

tf_reference(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*)

Creates the TensorFlow operations for obtaining the reference tensor(s), in case of a comparative loss.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **actions** – Dict of action tensors.
- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **next_states** – Dict of successor state tensors.
- **next_internals** – List of posterior internal state tensors.
- **update** – Boolean tensor indicating whether this call happens during an update.

Returns Reference tensor(s).

tf_regularization_losses(*states*, *internals*, *update*)

tensorforce.models.model module

The `Model` class coordinates the creation and execution of all TensorFlow operations within a model. It implements the `reset`, `act` and `update` functions, which form the interface the `Agent` class communicates with, and which should not need to be overwritten. Instead, the following TensorFlow functions need to be implemented:

- **tf_actions_and_internals(states, internals, deterministic) returning the batch of** actions and successor internal states.
- **tf_loss_per_instance(states, internals, actions, terminal, reward) returning the loss** per instance for a batch.

Further, the following TensorFlow functions should be extended accordingly:

- `initialize(custom_getter)` defining TensorFlow placeholders/functions and adding internal states.
- `get_variables()` returning the list of TensorFlow variables (to be optimized) of this model.
- `tf_regularization_losses(states, internals)` returning a dict of regularization losses.
- **get_optimizer_kwargs(states, internals, actions, terminal, reward) returning a dict of potential** arguments (argument-free functions) to the optimizer.

Finally, the following TensorFlow functions can be useful in some cases:

- `preprocess_states(states)` for state preprocessing, returning the processed batch of states.
- **tf_action_exploration(action, exploration, action_spec) for action postprocessing (e.g. exploration),** returning the processed batch of actions.
- **tf_preprocess_reward(states, internals, terminal, reward) for reward preprocessing (e.g. reward no** returning the processed batch of rewards.
- **create_output_operations(states, internals, actions, terminal, reward, deterministic) fo** similar to the two above for `Model.act` and `Model.update`.
- **tf_optimization(states, internals, actions, terminal, reward) for further optimization operations** (e.g. the baseline update in a `PGModel` or the target network update in a `QModel`), returning a single grouped optimization operation.

class tensorforce.models.model.**Model**(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*, *variable_noise*, *states_preprocessing*, *actions_exploration*, *reward_preprocessing*)

Bases: object

Base class for all (TensorFlow-based) models.

__init__(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*, *variable_noise*, *states_preprocessing*, *actions_exploration*, *reward_preprocessing*)

Model.

Parameters

- **states** (*spec*) – The state-space description dictionary.
- **actions** (*spec*) – The action-space description dictionary.
- **scope** (*str*) – The root scope str to use for tf variable scoping.
- **device** (*str*) – The name of the device to run the graph of this model on.
- **saver** (*spec*) – Dict specifying whether and how to save the model's parameters.
- **summarizer** (*spec*) – Dict specifying which tensorboard summaries should be created and added to the graph.
- **execution** (*spec*) – Dict specifying whether and how to do distributed training on the model's graph.
- **batching_capacity** (*int*) – Batching capacity.
- **variable_noise** (*float*) – The stddev value of a Normal distribution used for adding random noise to the model's output (for each batch, noise can be toggled and - if active - will be resampled). Use None for not adding any noise.
- **states_preprocessing** (*spec / dict of specs*) – Dict specifying whether and how to preprocess state signals (e.g. normalization, greyscale, etc..).
- **actions_exploration** (*spec / dict of specs*) – Dict specifying whether and how to add exploration to the model's "action outputs" (e.g. epsilon-greedy).
- **reward_preprocessing** (*spec*) – Dict specifying whether and how to preprocess rewards coming from the Environment (e.g. reward normalization).

act(*states*, *internals*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)

Does a forward pass through the model to retrieve action (outputs) given inputs for state (and internal state, if applicable (e.g. RNNs))

Parameters

- **states** (*dict*) – Dict of state values (each key represents one state space component).
- **internals** (*dict*) – Dict of internal state values (each key represents one internal state component).
- **deterministic** (*bool*) – If True, will not apply exploration after actions are calculated.
- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).

Returns

- Actual action-outputs (batched if state input is a batch).

Return type tuple

as_local_model()

close()

create_act_operations(*states*, *internals*, *deterministic*, *independent*)

create_observe_operations(*terminal*, *reward*)

create_operations(*states*, *internals*, *actions*, *terminal*, *reward*, *deterministic*, *independent*)
Creates output operations for acting, observing and interacting with the memory.

get_component(*component_name*)
Looks up a component by its name.

Parameters **component_name** – The name of the component to look up.

Returns The component for the provided name or None if there is no such component.

get_components()
Returns a dictionary of component name to component of all the components within this model.

Returns (dict) The mapping of name to component.

get_feed_dict(*states=None*, *internals=None*, *actions=None*, *terminal=None*, *reward=None*, *deterministic=None*, *independent=None*)

get_savable_components()
Returns the list of all of the components this model consists of that can be individually saved and restored. For instance the network or distribution.

Returns List of util.SavableComponent

get_summaries()
Returns the TensorFlow summaries reported by the model

Returns List of summaries

get_variables(*include_submodules=False*, *include_nontrainable=False*)
Returns the TensorFlow variables used by the model.

Parameters

- **include_submodules** – Includes variables of submodules (e.g. baseline, target network) if true.
- **include_nontrainable** – Includes non-trainable variables if true.

Returns List of variables.

initialize(*custom_getter*)
Creates the TensorFlow placeholders and functions for this model. Moreover adds the internal state placeholders and initialization values to the model.

Parameters **custom_getter** – The `custom_getter_` object to use for `tf.make_template` when creating TensorFlow functions.

observe(*terminal*, *reward*)
Adds an observation (reward and is-terminal) to the model without updating its trainable variables.

Parameters

- **terminal** (*bool*) – Whether the episode has terminated.
- **reward** (*float*) – The observed reward value.

Returns The value of the model-internal episode counter.

reset()
Resets the model to its initial state on episode start. This should also reset all preprocessor(s).

Returns Current episode, timestep counter and the shallow-copied list of internal state initialization Tensors.

Return type tuple

restore(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters

- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

restore_component(*component_name*, *save_path*)
Restores a component's parameters from a save location.

Parameters

- **component_name** – The component to restore.
- **save_path** – The save location.

save(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters

- **directory** – Optional checkpoint directory.
- **append_timestep** – Appends the current timestep to the checkpoint file if true.

Returns Checkpoint path where the model was saved.

save_component(*component_name*, *save_path*)
Saves a component of this model to the designated location.

Parameters

- **component_name** – The component to save.
- **save_path** – The location to save to.

Returns Checkpoint path where the component was saved.

setup()
Sets up the TensorFlow model graph and initializes (and enters) the TensorFlow session.

tf_action_exploration(*action*, *exploration*, *action_spec*)
Applies optional exploration to the action (post-processor for action outputs).

Parameters

- **action** (*tf.Tensor*) – The original output action tensor (to be post-processed).
- **exploration** (Exploration) – The Exploration object to use.
- **action_spec** (*dict*) – Dict specifying the action space.

Returns The post-processed action output tensor.

tf_actions_and_internals(*states*, *internals*, *deterministic*)
Creates and returns the TensorFlow operations for retrieving the actions and - if applicable - the posterior internal state Tensors in reaction to the given input states (and prior internal states).

Parameters

- **states** (*dict*) – Dict of state tensors (each key represents one state space component).
- **internals** – List of prior internal state tensors.
- **deterministic** – Boolean tensor indicating whether action should be chosen deterministically.

Returns

1. dict of output actions (with or without exploration applied (see `deterministic`))
2. list of posterior internal state Tensors (empty for non-internal state models)

Return type tuple

tf_initialize()

tf_observe_timestep(*states*, *internals*, *actions*, *terminal*, *reward*)
Creates the TensorFlow operations for performing the observation of a full time step's information.

Parameters

- **states** (*dict*) – Dict of state tensors (each key represents one state space component).
- **internals** – List of prior internal state tensors.
- **actions** – Dict of action tensors.
- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.

Returns The observation operation.

tf_preprocess(*states*, *actions*, *reward*)

tensorforce.models.pg_log_prob_model module

class tensorforce.models.pg_log_prob_model.**PGLogProbModel**(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*, *variable_noise*, *states_preprocessing*, *actions_exploration*, *reward_preprocessing*, *update_mode*, *memory*, *optimizer*, *discount*, *network*, *distributions*, *entropy_regularization*, *baseline_mode*, *baseline*, *baseline_optimizer*, *gae_lambda*)

Bases: *tensorforce.models.pg_model.PGModel*

Policy gradient model based on computing log likelihoods, e.g. VPG.

COMPONENT_BASELINE = 'baseline'

COMPONENT_DISTRIBUTION = 'distribution'

COMPONENT_NETWORK = 'network'

__init__(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*, *variable_noise*, *states_preprocessing*, *actions_exploration*, *reward_preprocessing*, *update_mode*, *memory*, *optimizer*, *discount*, *network*, *distributions*, *entropy_regularization*, *baseline_mode*, *baseline*, *baseline_optimizer*, *gae_lambda*)

act(*states*, *internals*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)
Does a forward pass through the model to retrieve action (outputs) given inputs for state (and internal state, if applicable (e.g. RNNs))

Parameters

- **states** (*dict*) – Dict of state values (each key represents one state space component).
- **internals** (*dict*) – Dict of internal state values (each key represents one internal state component).
- **deterministic** (*bool*) – If True, will not apply exploration after actions are calculated.
- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).

Returns

- Actual action-outputs (batched if state input is a batch).

Return type tuple

as_local_model()

baseline_optimizer_arguments(*states*, *internals*, *reward*)
Returns the baseline optimizer arguments including the time, the list of variables to optimize, and various functions which the optimizer might require to perform an update step.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **reward** – Reward tensor.

Returns Baseline optimizer arguments as dict.

close()

create_act_operations(*states*, *internals*, *deterministic*, *independent*)

create_distributions()

create_observe_operations(*terminal*, *reward*)

create_operations(*states*, *internals*, *actions*, *terminal*, *reward*, *deterministic*, *independent*)

get_component(*component_name*)
Looks up a component by its name.

Parameters **component_name** – The name of the component to look up.

Returns The component for the provided name or None if there is no such component.

get_components()

get_feed_dict(*states=None*, *internals=None*, *actions=None*, *terminal=None*, *reward=None*, *deterministic=None*, *independent=None*)

get_savable_components()
Returns the list of all of the components this model consists of that can be individually saved and restored. For instance the network or distribution.

Returns List of util.SavableComponent

get_summaries()

get_variables(*include_submodules=False*, *include_nontrainable=False*)

import_experience(*states*, *internals*, *actions*, *terminal*, *reward*)
Stores experiences.

initialize(*custom_getter*)

observe(*terminal*, *reward*)
Adds an observation (reward and is-terminal) to the model without updating its trainable variables.

Parameters

- **terminal** (*bool*) – Whether the episode has terminated.
- **reward** (*float*) – The observed reward value.

Returns The value of the model-internal episode counter.

optimizer_arguments(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*)

reset()
Resets the model to its initial state on episode start. This should also reset all preprocessor(s).

Returns Current episode, timestep counter and the shallow-copied list of internal state initialization Tensors.

Return type tuple

restore(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters

- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

restore_component(*component_name*, *save_path*)
Restores a component's parameters from a save location.

Parameters

- **component_name** – The component to restore.
- **save_path** – The save location.

save(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters

- **directory** – Optional checkpoint directory.

- **append_timestep** – Appends the current timestep to the checkpoint file if true.

Returns Checkpoint path where the model was saved.

save_component(*component_name*, *save_path*)
Saves a component of this model to the designated location.

Parameters

- **component_name** – The component to save.
- **save_path** – The location to save to.

Returns Checkpoint path where the component was saved.

setup()
Sets up the TensorFlow model graph and initializes (and enters) the TensorFlow session.

tf_action_exploration(*action*, *exploration*, *action_spec*)
Applies optional exploration to the action (post-processor for action outputs).

Parameters

- **action** (*tf.Tensor*) – The original output action tensor (to be post-processed).
- **exploration** (Exploration) – The Exploration object to use.
- **action_spec** (*dict*) – Dict specifying the action space.

Returns The post-processed action output tensor.

tf_actions_and_internals(*states*, *internals*, *deterministic*)

tf_baseline_loss(*states*, *internals*, *reward*, *update*, *reference=None*)
Creates the TensorFlow operations for calculating the baseline loss of a batch.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **reward** – Reward tensor.
- **update** – Boolean tensor indicating whether this call happens during an update.
- **reference** – Optional reference tensor(s), in case of a comparative loss.

Returns Loss tensor.

tf_discounted_cumulative_reward(*terminal*, *reward*, *discount*, *final_reward=0.0*)
Creates the TensorFlow operations for calculating the discounted cumulative rewards for a given sequence of rewards.

Parameters

- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **discount** – Discount factor.
- **final_reward** – Last reward value in the sequence.

Returns Discounted cumulative reward tensor.

tf_import_experience(*states*, *internals*, *actions*, *terminal*, *reward*)
Imports experiences into the TensorFlow memory structure. Can be used to import off-policy data.

Parameters

- **states** – Dict of state values to import with keys as state names and values as values to set.
- **internals** – Internal values to set, can be fetched from agent via agent.current_internals if no values available.
- **actions** – Dict of action values to import with keys as action names and values as values to set.
- **terminal** – Terminal value(s)
- **reward** – Reward value(s)

tf_initialize()

tf_kl_divergence(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)

tf_loss(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)

Creates the TensorFlow operations for calculating the full loss of a batch.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **actions** – Dict of action tensors.
- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **next_states** – Dict of successor state tensors.
- **next_internals** – List of posterior internal state tensors.
- **update** – Boolean tensor indicating whether this call happens during an update.
- **reference** – Optional reference tensor(s), in case of a comparative loss.

Returns Loss tensor.

tf_loss_per_instance(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)

tf_observe_timestep(*states*, *internals*, *actions*, *terminal*, *reward*)

tf_optimization(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states=None*, *next_internals=None*)

tf_preprocess(*states*, *actions*, *reward*)

tf_reference(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*)

Creates the TensorFlow operations for obtaining the reference tensor(s), in case of a comparative loss.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **actions** – Dict of action tensors.
- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.

- **next_states** – Dict of successor state tensors.
- **next_internals** – List of posterior internal state tensors.
- **update** – Boolean tensor indicating whether this call happens during an update.

Returns Reference tensor(s).

tf_regularization_losses(*states*, *internals*, *update*)

tf_reward_estimation(*states*, *internals*, *terminal*, *reward*, *update*)

tensorforce.models.pg_model module

class tensorforce.models.pg_model.**PGModel**(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*, *variable_noise*, *states_preprocessing*, *actions_exploration*, *reward_preprocessing*, *update_mode*, *memory*, *optimizer*, *discount*, *network*, *distributions*, *entropy_regularization*, *baseline_mode*, *baseline*, *baseline_optimizer*, *gae_lambda*)

Bases: *tensorforce.models.distribution_model.DistributionModel*

Base class for policy gradient models. It optionally defines a baseline and handles its optimization. It implements the `tf_loss_per_instance` function, but requires subclasses to implement `tf_pg_loss_per_instance`.

COMPONENT_BASELINE = 'baseline'

COMPONENT_DISTRIBUTION = 'distribution'

COMPONENT_NETWORK = 'network'

__init__(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*, *variable_noise*, *states_preprocessing*, *actions_exploration*, *reward_preprocessing*, *update_mode*, *memory*, *optimizer*, *discount*, *network*, *distributions*, *entropy_regularization*, *baseline_mode*, *baseline*, *baseline_optimizer*, *gae_lambda*)

act(*states*, *internals*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)

Does a forward pass through the model to retrieve action (outputs) given inputs for state (and internal state, if applicable (e.g. RNNs))

Parameters

- **states** (*dict*) – Dict of state values (each key represents one state space component).
- **internals** (*dict*) – Dict of internal state values (each key represents one internal state component).
- **deterministic** (*bool*) – If True, will not apply exploration after actions are calculated.
- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).

Returns

- Actual action-outputs (batched if state input is a batch).

Return type tuple

as_local_model()

baseline_optimizer_arguments(*states*, *internals*, *reward*)
Returns the baseline optimizer arguments including the time, the list of variables to optimize, and various functions which the optimizer might require to perform an update step.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **reward** – Reward tensor.

Returns Baseline optimizer arguments as dict.

close()

create_act_operations(*states*, *internals*, *deterministic*, *independent*)

create_distributions()

create_observe_operations(*terminal*, *reward*)

create_operations(*states*, *internals*, *actions*, *terminal*, *reward*, *deterministic*, *independent*)

get_component(*component_name*)
Looks up a component by its name.

Parameters **component_name** – The name of the component to look up.

Returns The component for the provided name or None if there is no such component.

get_components()

get_feed_dict(*states=None*, *internals=None*, *actions=None*, *terminal=None*, *reward=None*, *deterministic=None*, *independent=None*)

get_savable_components()
Returns the list of all of the components this model consists of that can be individually saved and restored. For instance the network or distribution.

Returns List of util.SavableComponent

get_summaries()

get_variables(*include_submodules=False*, *include_nontrainable=False*)

import_experience(*states*, *internals*, *actions*, *terminal*, *reward*)
Stores experiences.

initialize(*custom_getter*)

observe(*terminal*, *reward*)
Adds an observation (reward and is-terminal) to the model without updating its trainable variables.

Parameters

- **terminal** (*bool*) – Whether the episode has terminated.
- **reward** (*float*) – The observed reward value.

Returns The value of the model-internal episode counter.

optimizer_arguments(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*)

reset()
Resets the model to its initial state on episode start. This should also reset all preprocessor(s).

Returns Current episode, timestep counter and the shallow-copied list of internal state initialization Tensors.

Return type tuple

restore(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters

- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

restore_component(*component_name*, *save_path*)
Restores a component's parameters from a save location.

Parameters

- **component_name** – The component to restore.
- **save_path** – The save location.

save(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters

- **directory** – Optional checkpoint directory.
- **append_timestep** – Appends the current timestep to the checkpoint file if true.

Returns Checkpoint path where the model was saved.

save_component(*component_name*, *save_path*)
Saves a component of this model to the designated location.

Parameters

- **component_name** – The component to save.
- **save_path** – The location to save to.

Returns Checkpoint path where the component was saved.

setup()
Sets up the TensorFlow model graph and initializes (and enters) the TensorFlow session.

tf_action_exploration(*action*, *exploration*, *action_spec*)
Applies optional exploration to the action (post-processor for action outputs).

Parameters

- **action** (*tf.Tensor*) – The original output action tensor (to be post-processed).
- **exploration** (Exploration) – The Exploration object to use.
- **action_spec** (*dict*) – Dict specifying the action space.

Returns The post-processed action output tensor.

tf_actions_and_internals(*states*, *internals*, *deterministic*)

tf_baseline_loss(*states*, *internals*, *reward*, *update*, *reference=None*)
Creates the TensorFlow operations for calculating the baseline loss of a batch.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **reward** – Reward tensor.
- **update** – Boolean tensor indicating whether this call happens during an update.
- **reference** – Optional reference tensor(s), in case of a comparative loss.

Returns Loss tensor.

tf_discounted_cumulative_reward(*terminal*, *reward*, *discount*, *final_reward=0.0*)

Creates the TensorFlow operations for calculating the discounted cumulative rewards for a given sequence of rewards.

Parameters

- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **discount** – Discount factor.
- **final_reward** – Last reward value in the sequence.

Returns Discounted cumulative reward tensor.

tf_import_experience(*states*, *internals*, *actions*, *terminal*, *reward*)

Imports experiences into the TensorFlow memory structure. Can be used to import off-policy data.

Parameters

- **states** – Dict of state values to import with keys as state names and values as values to set.
- **internals** – Internal values to set, can be fetched from agent via agent.current_internals if no values available.
- **actions** – Dict of action values to import with keys as action names and values as values to set.
- **terminal** – Terminal value(s)
- **reward** – Reward value(s)

tf_initialize()

tf_kl_divergence(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)

tf_loss(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)

Creates the TensorFlow operations for calculating the full loss of a batch.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **actions** – Dict of action tensors.
- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.

- **next_states** – Dict of successor state tensors.
- **next_internals** – List of posterior internal state tensors.
- **update** – Boolean tensor indicating whether this call happens during an update.
- **reference** – Optional reference tensor(s), in case of a comparative loss.

Returns Loss tensor.

tf_loss_per_instance(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)

Creates the TensorFlow operations for calculating the loss per batch instance.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **actions** – Dict of action tensors.
- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **next_states** – Dict of successor state tensors.
- **next_internals** – List of posterior internal state tensors.
- **update** – Boolean tensor indicating whether this call happens during an update.
- **reference** – Optional reference tensor(s), in case of a comparative loss.

Returns Loss per instance tensor.

tf_observe_timestep(*states*, *internals*, *actions*, *terminal*, *reward*)

tf_optimization(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states=None*, *next_internals=None*)

tf_preprocess(*states*, *actions*, *reward*)

tf_reference(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*)

Creates the TensorFlow operations for obtaining the reference tensor(s), in case of a comparative loss.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **actions** – Dict of action tensors.
- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **next_states** – Dict of successor state tensors.
- **next_internals** – List of posterior internal state tensors.
- **update** – Boolean tensor indicating whether this call happens during an update.

Returns Reference tensor(s).

tf_regularization_losses(*states*, *internals*, *update*)

tf_reward_estimation(*states*, *internals*, *terminal*, *reward*, *update*)

tensorforce.models.pg_prob_ratio_model module

class tensorforce.models.pg_prob_ratio_model.**PGProbRatioModel**(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*, *variable_noise*, *states_preprocessing*, *actions_exploration*, *reward_preprocessing*, *update_mode*, *memory*, *optimizer*, *discount*, *network*, *distributions*, *entropy_regularization*, *baseline_mode*, *baseline*, *baseline_optimizer*, *gae_lambda*, *likelihood_ratio_clipping*)

Bases: *tensorforce.models.pg_model.PGModel*

Policy gradient model based on computing likelihood ratios, e.g. TRPO and PPO.

COMPONENT_BASELINE = 'baseline'

COMPONENT_DISTRIBUTION = 'distribution'

COMPONENT_NETWORK = 'network'

__init__(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*, *variable_noise*, *states_preprocessing*, *actions_exploration*, *reward_preprocessing*, *update_mode*, *memory*, *optimizer*, *discount*, *network*, *distributions*, *entropy_regularization*, *baseline_mode*, *baseline*, *baseline_optimizer*, *gae_lambda*, *likelihood_ratio_clipping*)

act(*states*, *internals*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)

Does a forward pass through the model to retrieve action (outputs) given inputs for state (and internal state, if applicable (e.g. RNNs))

Parameters

- **states** (*dict*) – Dict of state values (each key represents one state space component).
- **internals** (*dict*) – Dict of internal state values (each key represents one internal state component).
- **deterministic** (*bool*) – If True, will not apply exploration after actions are calculated.
- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).

Returns

- Actual action-outputs (batched if state input is a batch).

Return type tuple

as_local_model()

baseline_optimizer_arguments(*states*, *internals*, *reward*)
Returns the baseline optimizer arguments including the time, the list of variables to optimize, and various functions which the optimizer might require to perform an update step.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **reward** – Reward tensor.

Returns Baseline optimizer arguments as dict.

close()

create_act_operations(*states*, *internals*, *deterministic*, *independent*)

create_distributions()

create_observe_operations(*terminal*, *reward*)

create_operations(*states*, *internals*, *actions*, *terminal*, *reward*, *deterministic*, *independent*)

get_component(*component_name*)
Looks up a component by its name.

Parameters **component_name** – The name of the component to look up.

Returns The component for the provided name or None if there is no such component.

get_components()

get_feed_dict(*states=None*, *internals=None*, *actions=None*, *terminal=None*, *reward=None*, *deterministic=None*, *independent=None*)

get_savable_components()
Returns the list of all of the components this model consists of that can be individually saved and restored. For instance the network or distribution.

Returns List of util.SavableComponent

get_summaries()

get_variables(*include_submodules=False*, *include_nontrainable=False*)

import_experience(*states*, *internals*, *actions*, *terminal*, *reward*)
Stores experiences.

initialize(*custom_getter*)

observe(*terminal*, *reward*)
Adds an observation (reward and is-terminal) to the model without updating its trainable variables.

Parameters

- **terminal** (*bool*) – Whether the episode has terminated.
- **reward** (*float*) – The observed reward value.

Returns The value of the model-internal episode counter.

optimizer_arguments(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*)

reset()
Resets the model to its initial state on episode start. This should also reset all preprocessor(s).

Returns Current episode, timestep counter and the shallow-copied list of internal state initialization Tensors.

Return type tuple

restore(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters

- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

restore_component(*component_name*, *save_path*)
Restores a component's parameters from a save location.

Parameters

- **component_name** – The component to restore.
- **save_path** – The save location.

save(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters

- **directory** – Optional checkpoint directory.
- **append_timestep** – Appends the current timestep to the checkpoint file if true.

Returns Checkpoint path where the model was saved.

save_component(*component_name*, *save_path*)
Saves a component of this model to the designated location.

Parameters

- **component_name** – The component to save.
- **save_path** – The location to save to.

Returns Checkpoint path where the component was saved.

setup()
Sets up the TensorFlow model graph and initializes (and enters) the TensorFlow session.

tf_action_exploration(*action*, *exploration*, *action_spec*)
Applies optional exploration to the action (post-processor for action outputs).

Parameters

- **action** (*tf.Tensor*) – The original output action tensor (to be post-processed).
- **exploration** (Exploration) – The Exploration object to use.
- **action_spec** (*dict*) – Dict specifying the action space.

Returns The post-processed action output tensor.

tf_actions_and_internals(*states*, *internals*, *deterministic*)

tf_baseline_loss(*states*, *internals*, *reward*, *update*, *reference=None*)
Creates the TensorFlow operations for calculating the baseline loss of a batch.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **reward** – Reward tensor.
- **update** – Boolean tensor indicating whether this call happens during an update.
- **reference** – Optional reference tensor(s), in case of a comparative loss.

Returns Loss tensor.

tf_discounted_cumulative_reward(*terminal*, *reward*, *discount*, *final_reward=0.0*)
Creates the TensorFlow operations for calculating the discounted cumulative rewards for a given sequence of rewards.

Parameters

- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **discount** – Discount factor.
- **final_reward** – Last reward value in the sequence.

Returns Discounted cumulative reward tensor.

tf_import_experience(*states*, *internals*, *actions*, *terminal*, *reward*)
Imports experiences into the TensorFlow memory structure. Can be used to import off-policy data.

Parameters

- **states** – Dict of state values to import with keys as state names and values as values to set.
- **internals** – Internal values to set, can be fetched from agent via agent.current_internals if no values available.
- **actions** – Dict of action values to import with keys as action names and values as values to set.
- **terminal** – Terminal value(s)
- **reward** – Reward value(s)

tf_initialize()

tf_kl_divergence(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)

tf_loss(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)
Creates the TensorFlow operations for calculating the full loss of a batch.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **actions** – Dict of action tensors.

- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **next_states** – Dict of successor state tensors.
- **next_internals** – List of posterior internal state tensors.
- **update** – Boolean tensor indicating whether this call happens during an update.
- **reference** – Optional reference tensor(s), in case of a comparative loss.

Returns Loss tensor.

tf_loss_per_instance(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)

tf_observe_timestep(*states*, *internals*, *actions*, *terminal*, *reward*)

tf_optimization(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states=None*, *next_internals=None*)

tf_preprocess(*states*, *actions*, *reward*)

tf_reference(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*)

tf_regularization_losses(*states*, *internals*, *update*)

tf_reward_estimation(*states*, *internals*, *terminal*, *reward*, *update*)

tensorforce.models.q_demo_model module

class tensorforce.models.q_demo_model.**QDemoModel**(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*, *variable_noise*, *states_preprocessing*, *actions_exploration*, *reward_preprocessing*, *update_mode*, *memory*, *optimizer*, *discount*, *network*, *distributions*, *entropy_regularization*, *target_sync_frequency*, *target_update_weight*, *double_q_model*, *huber_loss*, *expert_margin*, *supervised_weight*, *demo_memory_capacity*, *demo_batch_size*)

Bases: *tensorforce.models.q_model.QModel*

Model for deep Q-learning from demonstration. Principal structure similar to double deep Q-networks but uses additional loss terms for demo data.

COMPONENT_DISTRIBUTION = 'distribution'

COMPONENT_NETWORK = 'network'

COMPONENT_TARGET_DISTRIBUTION = 'target_distribution'

COMPONENT_TARGET_NETWORK = 'target_network'

__init__(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*, *variable_noise*, *states_preprocessing*, *actions_exploration*, *reward_preprocessing*, *update_mode*, *memory*, *optimizer*, *discount*, *network*, *distributions*, *entropy_regularization*, *target_sync_frequency*, *target_update_weight*, *double_q_model*, *huber_loss*, *expert_margin*, *supervised_weight*, *demo_memory_capacity*, *demo_batch_size*)

act(*states*, *internals*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)
Does a forward pass through the model to retrieve action (outputs) given inputs for state (and internal state, if applicable (e.g. RNNs))

Parameters

- **states** (*dict*) – Dict of state values (each key represents one state space component).
- **internals** (*dict*) – Dict of internal state values (each key represents one internal state component).
- **deterministic** (*bool*) – If True, will not apply exploration after actions are calculated.
- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).

Returns

- Actual action-outputs (batched if state input is a batch).

Return type tuple

as_local_model()

close()

create_act_operations(*states*, *internals*, *deterministic*, *independent*)

create_distributions()

create_observe_operations(*terminal*, *reward*)

create_operations(*states*, *internals*, *actions*, *terminal*, *reward*, *deterministic*, *independent*)

demo_update()
Performs a demonstration update by calling the demo optimization operation. Note that the batch data does not have to be fetched from the demo memory as this is now part of the TensorFlow operation of the demo update.

get_component(*component_name*)
Looks up a component by its name.

Parameters **component_name** – The name of the component to look up.

Returns The component for the provided name or None if there is no such component.

get_components()

get_feed_dict(*states=None*, *internals=None*, *actions=None*, *terminal=None*, *reward=None*, *deterministic=None*, *independent=None*)

get_savable_components()
Returns the list of all of the components this model consists of that can be individually saved and restored. For instance the network or distribution.

Returns List of util.SavableComponent

get_summaries()

get_variables(*include_submodules=False*, *include_nontrainable=False*)
Returns the TensorFlow variables used by the model.

Returns List of variables.

import_demo_experience(*states*, *internals*, *actions*, *terminal*, *reward*)
Stores demonstrations in the demo memory.

import_experience(*states*, *internals*, *actions*, *terminal*, *reward*)
Stores experiences.

initialize(*custom_getter*)

observe(*terminal*, *reward*)
Adds an observation (reward and is-terminal) to the model without updating its trainable variables.

Parameters

- **terminal** (*bool*) – Whether the episode has terminated.
- **reward** (*float*) – The observed reward value.

Returns The value of the model-internal episode counter.

optimizer_arguments(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*)

reset()
Resets the model to its initial state on episode start. This should also reset all preprocessor(s).

Returns Current episode, timestep counter and the shallow-copied list of internal state initialization Tensors.

Return type tuple

restore(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters

- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

restore_component(*component_name*, *save_path*)
Restores a component's parameters from a save location.

Parameters

- **component_name** – The component to restore.
- **save_path** – The save location.

save(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters

- **directory** – Optional checkpoint directory.
- **append_timestep** – Appends the current timestep to the checkpoint file if true.

Returns Checkpoint path where the model was saved.

save_component(*component_name*, *save_path*)
Saves a component of this model to the designated location.

Parameters

- **component_name** – The component to save.
- **save_path** – The location to save to.

Returns Checkpoint path where the component was saved.

setup()
Sets up the TensorFlow model graph and initializes (and enters) the TensorFlow session.

target_optimizer_arguments()
Returns the target optimizer arguments including the time, the list of variables to optimize, and various functions which the optimizer might require to perform an update step.

Returns Target optimizer arguments as dict.

tf_action_exploration(*action*, *exploration*, *action_spec*)
Applies optional exploration to the action (post-processor for action outputs).

Parameters

- **action** (*tf.Tensor*) – The original output action tensor (to be post-processed).
- **exploration** (Exploration) – The Exploration object to use.
- **action_spec** (*dict*) – Dict specifying the action space.

Returns The post-processed action output tensor.

tf_actions_and_internals(*states*, *internals*, *deterministic*)

tf_combined_loss(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)
Combines Q-loss and demo loss.

tf_demo_loss(*states*, *actions*, *terminal*, *reward*, *internals*, *update*, *reference=None*)
Extends the q-model loss via the dqfd large-margin loss.

tf_demo_optimization(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*)

tf_discounted_cumulative_reward(*terminal*, *reward*, *discount*, *final_reward=0.0*)
Creates the TensorFlow operations for calculating the discounted cumulative rewards for a given sequence of rewards.

Parameters

- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **discount** – Discount factor.
- **final_reward** – Last reward value in the sequence.

Returns Discounted cumulative reward tensor.

tf_import_demo_experience(*states*, *internals*, *actions*, *terminal*, *reward*)
Imports a single experience to memory.

tf_import_experience(*states*, *internals*, *actions*, *terminal*, *reward*)
Imports experiences into the TensorFlow memory structure. Can be used to import off-policy data.

Parameters

- **states** – Dict of state values to import with keys as state names and values as values to set.
- **internals** – Internal values to set, can be fetched from agent via agent.current_internals if no values available.
- **actions** – Dict of action values to import with keys as action names and values as values to set.
- **terminal** – Terminal value(s)
- **reward** – Reward value(s)

tf_initialize()

tf_kl_divergence(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)

tf_loss(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)

Creates the TensorFlow operations for calculating the full loss of a batch.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **actions** – Dict of action tensors.
- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **next_states** – Dict of successor state tensors.
- **next_internals** – List of posterior internal state tensors.
- **update** – Boolean tensor indicating whether this call happens during an update.
- **reference** – Optional reference tensor(s), in case of a comparative loss.

Returns Loss tensor.

tf_loss_per_instance(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)

tf_observe_timestep(*states*, *internals*, *actions*, *terminal*, *reward*)

tf_optimization(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states=None*, *next_internals=None*)

tf_preprocess(*states*, *actions*, *reward*)

tf_q_delta(*q_value*, *next_q_value*, *terminal*, *reward*)

Creates the deltas (or advantage) of the Q values.

Returns A list of deltas per action

tf_q_value(*embedding*, *distr_params*, *action*, *name*)

tf_reference(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*)

Creates the TensorFlow operations for obtaining the reference tensor(s), in case of a comparative loss.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.

- **actions** – Dict of action tensors.
- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **next_states** – Dict of successor state tensors.
- **next_internals** – List of posterior internal state tensors.
- **update** – Boolean tensor indicating whether this call happens during an update.

Returns Reference tensor(s).

tf_regularization_losses(*states*, *internals*, *update*)

tensorforce.models.q_model module

class tensorforce.models.q_model.**QModel**(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*, *variable_noise*, *states_preprocessing*, *actions_exploration*, *reward_preprocessing*, *update_mode*, *memory*, *optimizer*, *discount*, *network*, *distributions*, *entropy_regularization*, *target_sync_frequency*, *target_update_weight*, *double_q_model*, *huber_loss*)

Bases: *tensorforce.models.distribution_model.DistributionModel*

Q-value model.

COMPONENT_DISTRIBUTION = 'distribution'

COMPONENT_NETWORK = 'network'

COMPONENT_TARGET_DISTRIBUTION = 'target_distribution'

COMPONENT_TARGET_NETWORK = 'target_network'

__init__(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*, *variable_noise*, *states_preprocessing*, *actions_exploration*, *reward_preprocessing*, *update_mode*, *memory*, *optimizer*, *discount*, *network*, *distributions*, *entropy_regularization*, *target_sync_frequency*, *target_update_weight*, *double_q_model*, *huber_loss*)

act(*states*, *internals*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)

Does a forward pass through the model to retrieve action (outputs) given inputs for state (and internal state, if applicable (e.g. RNNs))

Parameters

- **states** (*dict*) – Dict of state values (each key represents one state space component).
- **internals** (*dict*) – Dict of internal state values (each key represents one internal state component).
- **deterministic** (*bool*) – If True, will not apply exploration after actions are calculated.
- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).

Returns

- Actual action-outputs (batched if state input is a batch).

Return type tuple

as_local_model()

close()

create_act_operations(*states*, *internals*, *deterministic*, *independent*)

create_distributions()

create_observe_operations(*terminal*, *reward*)

create_operations(*states*, *internals*, *actions*, *terminal*, *reward*, *deterministic*, *independent*)

get_component(*component_name*)
Looks up a component by its name.

Parameters **component_name** – The name of the component to look up.

Returns The component for the provided name or None if there is no such component.

get_components()

get_feed_dict(*states=None*, *internals=None*, *actions=None*, *terminal=None*, *reward=None*, *deterministic=None*, *independent=None*)

get_savable_components()
Returns the list of all of the components this model consists of that can be individually saved and restored. For instance the network or distribution.

Returns List of util.SavableComponent

get_summaries()

get_variables(*include_submodules=False*, *include_nontrainable=False*)

import_experience(*states*, *internals*, *actions*, *terminal*, *reward*)
Stores experiences.

initialize(*custom_getter*)

observe(*terminal*, *reward*)
Adds an observation (reward and is-terminal) to the model without updating its trainable variables.

Parameters

- **terminal** (*bool*) – Whether the episode has terminated.
- **reward** (*float*) – The observed reward value.

Returns The value of the model-internal episode counter.

optimizer_arguments(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*)

reset()
Resets the model to its initial state on episode start. This should also reset all preprocessor(s).

Returns Current episode, timestep counter and the shallow-copied list of internal state initialization Tensors.

Return type tuple

restore(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters

- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

restore_component(*component_name*, *save_path*)
Restores a component's parameters from a save location.

Parameters

- **component_name** – The component to restore.
- **save_path** – The save location.

save(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters

- **directory** – Optional checkpoint directory.
- **append_timestep** – Appends the current timestep to the checkpoint file if true.

Returns Checkpoint path where the model was saved.

save_component(*component_name*, *save_path*)
Saves a component of this model to the designated location.

Parameters

- **component_name** – The component to save.
- **save_path** – The location to save to.

Returns Checkpoint path where the component was saved.

setup()
Sets up the TensorFlow model graph and initializes (and enters) the TensorFlow session.

target_optimizer_arguments()
Returns the target optimizer arguments including the time, the list of variables to optimize, and various functions which the optimizer might require to perform an update step.

Returns Target optimizer arguments as dict.

tf_action_exploration(*action*, *exploration*, *action_spec*)
Applies optional exploration to the action (post-processor for action outputs).

Parameters

- **action** (*tf.Tensor*) – The original output action tensor (to be post-processed).
- **exploration** (Exploration) – The Exploration object to use.
- **action_spec** (*dict*) – Dict specifying the action space.

Returns The post-processed action output tensor.

tf_actions_and_internals(*states*, *internals*, *deterministic*)

tf_discounted_cumulative_reward(*terminal*, *reward*, *discount*, *final_reward=0.0*)
Creates the TensorFlow operations for calculating the discounted cumulative rewards for a given sequence of rewards.

Parameters

- **terminal** – Terminal boolean tensor.

- **reward** – Reward tensor.
- **discount** – Discount factor.
- **final_reward** – Last reward value in the sequence.

Returns Discounted cumulative reward tensor.

tf_import_experience(*states*, *internals*, *actions*, *terminal*, *reward*)
Imports experiences into the TensorFlow memory structure. Can be used to import off-policy data.

Parameters

- **states** – Dict of state values to import with keys as state names and values as values to set.
- **internals** – Internal values to set, can be fetched from agent via agent.current_internals if no values available.
- **actions** – Dict of action values to import with keys as action names and values as values to set.
- **terminal** – Terminal value(s)
- **reward** – Reward value(s)

tf_initialize()

tf_kl_divergence(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)

tf_loss(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)
Creates the TensorFlow operations for calculating the full loss of a batch.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **actions** – Dict of action tensors.
- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **next_states** – Dict of successor state tensors.
- **next_internals** – List of posterior internal state tensors.
- **update** – Boolean tensor indicating whether this call happens during an update.
- **reference** – Optional reference tensor(s), in case of a comparative loss.

Returns Loss tensor.

tf_loss_per_instance(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)

tf_observe_timestep(*states*, *internals*, *actions*, *terminal*, *reward*)

tf_optimization(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states=None*, *next_internals=None*)

tf_preprocess(*states*, *actions*, *reward*)

tf_q_delta(*q_value*, *next_q_value*, *terminal*, *reward*)
Creates the deltas (or advantage) of the Q values.

Returns A list of deltas per action

tf_q_value(*embedding*, *distr_params*, *action*, *name*)

tf_reference(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*)
Creates the TensorFlow operations for obtaining the reference tensor(s), in case of a comparative loss.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **actions** – Dict of action tensors.
- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **next_states** – Dict of successor state tensors.
- **next_internals** – List of posterior internal state tensors.
- **update** – Boolean tensor indicating whether this call happens during an update.

Returns Reference tensor(s).

tf_regularization_losses(*states*, *internals*, *update*)

tensorforce.models.q_naf_model module

class tensorforce.models.q_naf_model.**QNAFModel**(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*, *variable_noise*, *states_preprocessing*, *actions_exploration*, *reward_preprocessing*, *update_mode*, *memory*, *optimizer*, *discount*, *network*, *distributions*, *entropy_regularization*, *target_sync_frequency*, *target_update_weight*, *double_q_model*, *huber_loss*)

Bases: *tensorforce.models.q_model.QModel*

COMPONENT_DISTRIBUTION = 'distribution'

COMPONENT_NETWORK = 'network'

COMPONENT_TARGET_DISTRIBUTION = 'target_distribution'

COMPONENT_TARGET_NETWORK = 'target_network'

__init__(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*, *variable_noise*, *states_preprocessing*, *actions_exploration*, *reward_preprocessing*, *update_mode*, *memory*, *optimizer*, *discount*, *network*, *distributions*, *entropy_regularization*, *target_sync_frequency*, *target_update_weight*, *double_q_model*, *huber_loss*)

act(*states*, *internals*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)
Does a forward pass through the model to retrieve action (outputs) given inputs for state (and internal state, if applicable (e.g. RNNs))

Parameters

- **states** (*dict*) – Dict of state values (each key represents one state space component).

- **internals** (*dict*) – Dict of internal state values (each key represents one internal state component).
- **deterministic** (*bool*) – If True, will not apply exploration after actions are calculated.
- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).

Returns

- Actual action-outputs (batched if state input is a batch).

Return type tuple

as_local_model()

close()

create_act_operations(*states*, *internals*, *deterministic*, *independent*)

create_distributions()

create_observe_operations(*terminal*, *reward*)

create_operations(*states*, *internals*, *actions*, *terminal*, *reward*, *deterministic*, *independent*)

get_component(*component_name*)
Looks up a component by its name.

Parameters **component_name** – The name of the component to look up.

Returns The component for the provided name or None if there is no such component.

get_components()

get_feed_dict(*states=None*, *internals=None*, *actions=None*, *terminal=None*, *reward=None*, *deterministic=None*, *independent=None*)

get_savable_components()
Returns the list of all of the components this model consists of that can be individually saved and restored. For instance the network or distribution.

Returns List of util.SavableComponent

get_summaries()

get_variables(*include_submodules=False*, *include_nontrainable=False*)

import_experience(*states*, *internals*, *actions*, *terminal*, *reward*)
Stores experiences.

initialize(*custom_getter*)

observe(*terminal*, *reward*)
Adds an observation (reward and is-terminal) to the model without updating its trainable variables.

Parameters

- **terminal** (*bool*) – Whether the episode has terminated.
- **reward** (*float*) – The observed reward value.

Returns The value of the model-internal episode counter.

optimizer_arguments(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*)

reset()
Resets the model to its initial state on episode start. This should also reset all preprocessor(s).

Returns Current episode, timestep counter and the shallow-copied list of internal state initialization Tensors.

Return type tuple

restore(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters

- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

restore_component(*component_name*, *save_path*)
Restores a component's parameters from a save location.

Parameters

- **component_name** – The component to restore.
- **save_path** – The save location.

save(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters

- **directory** – Optional checkpoint directory.
- **append_timestep** – Appends the current timestep to the checkpoint file if true.

Returns Checkpoint path where the model was saved.

save_component(*component_name*, *save_path*)
Saves a component of this model to the designated location.

Parameters

- **component_name** – The component to save.
- **save_path** – The location to save to.

Returns Checkpoint path where the component was saved.

setup()
Sets up the TensorFlow model graph and initializes (and enters) the TensorFlow session.

target_optimizer_arguments()
Returns the target optimizer arguments including the time, the list of variables to optimize, and various functions which the optimizer might require to perform an update step.

Returns Target optimizer arguments as dict.

tf_action_exploration(*action*, *exploration*, *action_spec*)
Applies optional exploration to the action (post-processor for action outputs).

Parameters

- **action** (*tf.Tensor*) – The original output action tensor (to be post-processed).

- **exploration** (Exploration) – The Exploration object to use.
- **action_spec** (*dict*) – Dict specifying the action space.

Returns The post-processed action output tensor.

tf_actions_and_internals(*states*, *internals*, *deterministic*)

tf_discounted_cumulative_reward(*terminal*, *reward*, *discount*, *final_reward=0.0*)
Creates the TensorFlow operations for calculating the discounted cumulative rewards for a given sequence of rewards.

Parameters

- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **discount** – Discount factor.
- **final_reward** – Last reward value in the sequence.

Returns Discounted cumulative reward tensor.

tf_import_experience(*states*, *internals*, *actions*, *terminal*, *reward*)
Imports experiences into the TensorFlow memory structure. Can be used to import off-policy data.

Parameters

- **states** – Dict of state values to import with keys as state names and values as values to set.
- **internals** – Internal values to set, can be fetched from agent via agent.current_internals if no values available.
- **actions** – Dict of action values to import with keys as action names and values as values to set.
- **terminal** – Terminal value(s)
- **reward** – Reward value(s)

tf_initialize()

tf_kl_divergence(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)

tf_loss(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)
Creates the TensorFlow operations for calculating the full loss of a batch.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **actions** – Dict of action tensors.
- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **next_states** – Dict of successor state tensors.
- **next_internals** – List of posterior internal state tensors.
- **update** – Boolean tensor indicating whether this call happens during an update.

- **reference** – Optional reference tensor(s), in case of a comparative loss.

Returns Loss tensor.

tf_loss_per_instance(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)

tf_observe_timestep(*states*, *internals*, *actions*, *terminal*, *reward*)

tf_optimization(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states=None*, *next_internals=None*)

tf_preprocess(*states*, *actions*, *reward*)

tf_q_delta(*q_value*, *next_q_value*, *terminal*, *reward*)
Creates the deltas (or advantage) of the Q values.

Returns A list of deltas per action

tf_q_value(*embedding*, *distr_params*, *action*, *name*)

tf_reference(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*)
Creates the TensorFlow operations for obtaining the reference tensor(s), in case of a comparative loss.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **actions** – Dict of action tensors.
- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **next_states** – Dict of successor state tensors.
- **next_internals** – List of posterior internal state tensors.
- **update** – Boolean tensor indicating whether this call happens during an update.

Returns Reference tensor(s).

tf_regularization_losses(*states*, *internals*, *update*)

tensorforce.models.q_nstep_model module

class tensorforce.models.q_nstep_model.**QNstepModel**(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*, *variable_noise*, *states_preprocessing*, *actions_exploration*, *reward_preprocessing*, *update_mode*, *memory*, *optimizer*, *discount*, *network*, *distributions*, *entropy_regularization*, *target_sync_frequency*, *target_update_weight*, *double_q_model*, *huber_loss*)

Bases: *tensorforce.models.q_model.QModel*

Deep Q network using n-step rewards as described in Asynchronous Methods for Reinforcement Learning.

COMPONENT_DISTRIBUTION = 'distribution'

COMPONENT_NETWORK = 'network'

COMPONENT_TARGET_DISTRIBUTION = 'target_distribution'

COMPONENT_TARGET_NETWORK = 'target_network'

__init__(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*, *variable_noise*, *states_preprocessing*, *actions_exploration*, *reward_preprocessing*, *update_mode*, *memory*, *optimizer*, *discount*, *network*, *distributions*, *entropy_regularization*, *target_sync_frequency*, *target_update_weight*, *double_q_model*, *huber_loss*)

act(*states*, *internals*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)
Does a forward pass through the model to retrieve action (outputs) given inputs for state (and internal state, if applicable (e.g. RNNs))

Parameters

- **states** (*dict*) – Dict of state values (each key represents one state space component).
- **internals** (*dict*) – Dict of internal state values (each key represents one internal state component).
- **deterministic** (*bool*) – If True, will not apply exploration after actions are calculated.
- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).

Returns

- Actual action-outputs (batched if state input is a batch).

Return type tuple

as_local_model()

close()

create_act_operations(*states*, *internals*, *deterministic*, *independent*)

create_distributions()

create_observe_operations(*terminal*, *reward*)

create_operations(*states*, *internals*, *actions*, *terminal*, *reward*, *deterministic*, *independent*)

get_component(*component_name*)
Looks up a component by its name.

Parameters **component_name** – The name of the component to look up.

Returns The component for the provided name or None if there is no such component.

get_components()

get_feed_dict(*states=None*, *internals=None*, *actions=None*, *terminal=None*, *reward=None*, *deterministic=None*, *independent=None*)

get_savable_components()
Returns the list of all of the components this model consists of that can be individually saved and restored. For instance the network or distribution.

Returns List of util.SavableComponent

get_summaries()

get_variables(*include_submodules=False*, *include_nontrainable=False*)

import_experience(*states*, *internals*, *actions*, *terminal*, *reward*)
Stores experiences.

initialize(*custom_getter*)

observe(*terminal*, *reward*)
Adds an observation (reward and is-terminal) to the model without updating its trainable variables.

Parameters

- **terminal** (*bool*) – Whether the episode has terminated.
- **reward** (*float*) – The observed reward value.

Returns The value of the model-internal episode counter.

optimizer_arguments(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*)

reset()
Resets the model to its initial state on episode start. This should also reset all preprocessor(s).

Returns Current episode, timestep counter and the shallow-copied list of internal state initialization Tensors.

Return type tuple

restore(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters

- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

restore_component(*component_name*, *save_path*)
Restores a component's parameters from a save location.

Parameters

- **component_name** – The component to restore.
- **save_path** – The save location.

save(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters

- **directory** – Optional checkpoint directory.
- **append_timestep** – Appends the current timestep to the checkpoint file if true.

Returns Checkpoint path where the model was saved.

save_component(*component_name*, *save_path*)
Saves a component of this model to the designated location.

Parameters

- **component_name** – The component to save.
- **save_path** – The location to save to.

Returns Checkpoint path where the component was saved.

setup()
Sets up the TensorFlow model graph and initializes (and enters) the TensorFlow session.

target_optimizer_arguments()
Returns the target optimizer arguments including the time, the list of variables to optimize, and various functions which the optimizer might require to perform an update step.

Returns Target optimizer arguments as dict.

tf_action_exploration(*action*, *exploration*, *action_spec*)
Applies optional exploration to the action (post-processor for action outputs).

Parameters

- **action** (*tf.Tensor*) – The original output action tensor (to be post-processed).
- **exploration** (Exploration) – The Exploration object to use.
- **action_spec** (*dict*) – Dict specifying the action space.

Returns The post-processed action output tensor.

tf_actions_and_internals(*states*, *internals*, *deterministic*)

tf_discounted_cumulative_reward(*terminal*, *reward*, *discount*, *final_reward=0.0*)
Creates the TensorFlow operations for calculating the discounted cumulative rewards for a given sequence of rewards.

Parameters

- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **discount** – Discount factor.
- **final_reward** – Last reward value in the sequence.

Returns Discounted cumulative reward tensor.

tf_import_experience(*states*, *internals*, *actions*, *terminal*, *reward*)
Imports experiences into the TensorFlow memory structure. Can be used to import off-policy data.

Parameters

- **states** – Dict of state values to import with keys as state names and values as values to set.
- **internals** – Internal values to set, can be fetched from agent via agent.current_internals if no values available.
- **actions** – Dict of action values to import with keys as action names and values as values to set.
- **terminal** – Terminal value(s)
- **reward** – Reward value(s)

tf_initialize()

tf_kl_divergence(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)

tf_loss(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)
Creates the TensorFlow operations for calculating the full loss of a batch.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **actions** – Dict of action tensors.
- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **next_states** – Dict of successor state tensors.
- **next_internals** – List of posterior internal state tensors.
- **update** – Boolean tensor indicating whether this call happens during an update.
- **reference** – Optional reference tensor(s), in case of a comparative loss.

Returns Loss tensor.

tf_loss_per_instance(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)

tf_observe_timestep(*states*, *internals*, *actions*, *terminal*, *reward*)

tf_optimization(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states=None*, *next_internals=None*)

tf_preprocess(*states*, *actions*, *reward*)

tf_q_delta(*q_value*, *next_q_value*, *terminal*, *reward*)

tf_q_value(*embedding*, *distr_params*, *action*, *name*)

tf_reference(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*)
Creates the TensorFlow operations for obtaining the reference tensor(s), in case of a comparative loss.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **actions** – Dict of action tensors.
- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **next_states** – Dict of successor state tensors.
- **next_internals** – List of posterior internal state tensors.
- **update** – Boolean tensor indicating whether this call happens during an update.

Returns Reference tensor(s).

tf_regularization_losses(*states*, *internals*, *update*)

tensorforce.models.random_model module

class tensorforce.models.random_model.**RandomModel**(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*)

Bases: *tensorforce.models.model.Model*

Utility class to return random actions of a desired shape and with given bounds.

__init__(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*)

act(*states*, *internals*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)
Does a forward pass through the model to retrieve action (outputs) given inputs for state (and internal state, if applicable (e.g. RNNs))

Parameters

- **states** (*dict*) – Dict of state values (each key represents one state space component).
- **internals** (*dict*) – Dict of internal state values (each key represents one internal state component).
- **deterministic** (*bool*) – If True, will not apply exploration after actions are calculated.
- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).

Returns

- Actual action-outputs (batched if state input is a batch).

Return type tuple

as_local_model()

close()

create_act_operations(*states*, *internals*, *deterministic*, *independent*)

create_observe_operations(*terminal*, *reward*)

create_operations(*states*, *internals*, *actions*, *terminal*, *reward*, *deterministic*, *independent*)
Creates output operations for acting, observing and interacting with the memory.

get_component(*component_name*)
Looks up a component by its name.

Parameters **component_name** – The name of the component to look up.

Returns The component for the provided name or None if there is no such component.

get_components()
Returns a dictionary of component name to component of all the components within this model.

Returns (dict) The mapping of name to component.

get_feed_dict(*states=None*, *internals=None*, *actions=None*, *terminal=None*, *reward=None*, *deterministic=None*, *independent=None*)

get_savable_components()
Returns the list of all of the components this model consists of that can be individually saved and restored. For instance the network or distribution.

Returns List of util.SavableComponent

get_summaries()
Returns the TensorFlow summaries reported by the model

Returns List of summaries

get_variables(*include_submodules=False*, *include_nontrainable=False*)
Returns the TensorFlow variables used by the model.

Parameters

- **include_submodules** – Includes variables of submodules (e.g. baseline, target network) if true.
- **include_nontrainable** – Includes non-trainable variables if true.

Returns List of variables.

initialize(*custom_getter*)
Creates the TensorFlow placeholders and functions for this model. Moreover adds the internal state placeholders and initialization values to the model.

Parameters **custom_getter** – The `custom_getter_` object to use for `tf.make_template` when creating TensorFlow functions.

observe(*terminal*, *reward*)
Adds an observation (reward and is-terminal) to the model without updating its trainable variables.

Parameters

- **terminal** (*bool*) – Whether the episode has terminated.
- **reward** (*float*) – The observed reward value.

Returns The value of the model-internal episode counter.

reset()
Resets the model to its initial state on episode start. This should also reset all preprocessor(s).

Returns Current episode, timestep counter and the shallow-copied list of internal state initialization Tensors.

Return type tuple

restore(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters

- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

restore_component(*component_name*, *save_path*)
Restores a component's parameters from a save location.

Parameters

- **component_name** – The component to restore.
- **save_path** – The save location.

save(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters

- **directory** – Optional checkpoint directory.
- **append_timestep** – Appends the current timestep to the checkpoint file if true.

Returns Checkpoint path where the model was saved.

save_component(*component_name*, *save_path*)
Saves a component of this model to the designated location.

Parameters

- **component_name** – The component to save.
- **save_path** – The location to save to.

Returns Checkpoint path where the component was saved.

setup()
Sets up the TensorFlow model graph and initializes (and enters) the TensorFlow session.

tf_action_exploration(*action*, *exploration*, *action_spec*)
Applies optional exploration to the action (post-processor for action outputs).

Parameters

- **action** (*tf.Tensor*) – The original output action tensor (to be post-processed).
- **exploration** (Exploration) – The Exploration object to use.
- **action_spec** (*dict*) – Dict specifying the action space.

Returns The post-processed action output tensor.

tf_actions_and_internals(*states*, *internals*, *deterministic*)

tf_initialize()

tf_observe_timestep(*states*, *internals*, *actions*, *terminal*, *reward*)

tf_preprocess(*states*, *actions*, *reward*)

Module contents

class tensorforce.models.**Model**(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*, *variable_noise*, *states_preprocessing*, *actions_exploration*, *reward_preprocessing*)

Bases: object

Base class for all (TensorFlow-based) models.

__init__(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*, *variable_noise*, *states_preprocessing*, *actions_exploration*, *reward_preprocessing*)
Model.

Parameters

- **states** (*spec*) – The state-space description dictionary.
- **actions** (*spec*) – The action-space description dictionary.
- **scope** (*str*) – The root scope str to use for tf variable scoping.
- **device** (*str*) – The name of the device to run the graph of this model on.
- **saver** (*spec*) – Dict specifying whether and how to save the model's parameters.
- **summarizer** (*spec*) – Dict specifying which tensorboard summaries should be created and added to the graph.
- **execution** (*spec*) – Dict specifying whether and how to do distributed training on the model's graph.
- **batching_capacity** (*int*) – Batching capacity.

- **variable_noise** (*float*) – The stddev value of a Normal distribution used for adding random noise to the model's output (for each batch, noise can be toggled and - if active - will be resampled). Use None for not adding any noise.
- **states_preprocessing** (*spec / dict of specs*) – Dict specifying whether and how to preprocess state signals (e.g. normalization, greyscale, etc..).
- **actions_exploration** (*spec / dict of specs*) – Dict specifying whether and how to add exploration to the model's "action outputs" (e.g. epsilon-greedy).
- **reward_preprocessing** (*spec*) – Dict specifying whether and how to preprocess rewards coming from the Environment (e.g. reward normalization).

act(*states*, *internals*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)
Does a forward pass through the model to retrieve action (outputs) given inputs for state (and internal state, if applicable (e.g. RNNs))

Parameters

- **states** (*dict*) – Dict of state values (each key represents one state space component).
- **internals** (*dict*) – Dict of internal state values (each key represents one internal state component).
- **deterministic** (*bool*) – If True, will not apply exploration after actions are calculated.
- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).

Returns

- Actual action-outputs (batched if state input is a batch).

Return type tuple

as_local_model()

close()

create_act_operations(*states*, *internals*, *deterministic*, *independent*)

create_observe_operations(*terminal*, *reward*)

create_operations(*states*, *internals*, *actions*, *terminal*, *reward*, *deterministic*, *independent*)
Creates output operations for acting, observing and interacting with the memory.

get_component(*component_name*)
Looks up a component by its name.

Parameters **component_name** – The name of the component to look up.

Returns The component for the provided name or None if there is no such component.

get_components()
Returns a dictionary of component name to component of all the components within this model.

Returns (dict) The mapping of name to component.

get_feed_dict(*states=None*, *internals=None*, *actions=None*, *terminal=None*, *reward=None*, *deterministic=None*, *independent=None*)

get_savable_components()
Returns the list of all of the components this model consists of that can be individually saved and restored. For instance the network or distribution.

Returns List of util.SavableComponent

get_summaries()
Returns the TensorFlow summaries reported by the model

Returns List of summaries

get_variables(*include_submodules=False*, *include_nontrainable=False*)
Returns the TensorFlow variables used by the model.

Parameters

- **include_submodules** – Includes variables of submodules (e.g. baseline, target network) if true.
- **include_nontrainable** – Includes non-trainable variables if true.

Returns List of variables.

initialize(*custom_getter*)
Creates the TensorFlow placeholders and functions for this model. Moreover adds the internal state placeholders and initialization values to the model.

Parameters **custom_getter** – The `custom_getter_` object to use for `tf.make_template` when creating TensorFlow functions.

observe(*terminal*, *reward*)
Adds an observation (reward and is-terminal) to the model without updating its trainable variables.

Parameters

- **terminal** (*bool*) – Whether the episode has terminated.
- **reward** (*float*) – The observed reward value.

Returns The value of the model-internal episode counter.

reset()
Resets the model to its initial state on episode start. This should also reset all preprocessor(s).

Returns Current episode, timestep counter and the shallow-copied list of internal state initialization Tensors.

Return type tuple

restore(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters

- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

restore_component(*component_name*, *save_path*)
Restores a component's parameters from a save location.

Parameters

- **component_name** – The component to restore.
- **save_path** – The save location.

save(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used.

Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters

- **directory** – Optional checkpoint directory.
- **append_timestep** – Appends the current timestep to the checkpoint file if true.

Returns Checkpoint path where the model was saved.

save_component(*component_name*, *save_path*)
Saves a component of this model to the designated location.

Parameters

- **component_name** – The component to save.
- **save_path** – The location to save to.

Returns Checkpoint path where the component was saved.

setup()
Sets up the TensorFlow model graph and initializes (and enters) the TensorFlow session.

tf_action_exploration(*action*, *exploration*, *action_spec*)
Applies optional exploration to the action (post-processor for action outputs).

Parameters

- **action** (*tf.Tensor*) – The original output action tensor (to be post-processed).
- **exploration** (Exploration) – The Exploration object to use.
- **action_spec** (*dict*) – Dict specifying the action space.

Returns The post-processed action output tensor.

tf_actions_and_internals(*states*, *internals*, *deterministic*)
Creates and returns the TensorFlow operations for retrieving the actions and - if applicable - the posterior internal state Tensors in reaction to the given input states (and prior internal states).

Parameters

- **states** (*dict*) – Dict of state tensors (each key represents one state space component).
- **internals** – List of prior internal state tensors.
- **deterministic** – Boolean tensor indicating whether action should be chosen deterministically.

Returns

1. dict of output actions (with or without exploration applied (see `deterministic`))
2. list of posterior internal state Tensors (empty for non-internal state models)

Return type tuple

tf_initialize()

tf_observe_timestep(*states*, *internals*, *actions*, *terminal*, *reward*)
Creates the TensorFlow operations for performing the observation of a full time step's information.

Parameters

- **states** (*dict*) – Dict of state tensors (each key represents one state space component).
- **internals** – List of prior internal state tensors.

- **actions** – Dict of action tensors.
- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.

Returns The observation operation.

tf_preprocess(*states*, *actions*, *reward*)

class tensorforce.models.**MemoryModel**(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*, *variable_noise*, *states_preprocessing*, *actions_exploration*, *reward_preprocessing*, *update_mode*, *memory*, *optimizer*, *discount*)

Bases: *tensorforce.models.model.Model*

A memory model is a generical model to accumulate and sample data.

__init__(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*, *variable_noise*, *states_preprocessing*, *actions_exploration*, *reward_preprocessing*, *update_mode*, *memory*, *optimizer*, *discount*)

Memory model.

Parameters

- **states** (*spec*) – The state-space description dictionary.
- **actions** (*spec*) – The action-space description dictionary.
- **scope** (*str*) – The root scope str to use for tf variable scoping.
- **device** (*str*) – The name of the device to run the graph of this model on.
- **saver** (*spec*) – Dict specifying whether and how to save the model's parameters.
- **summarizer** (*spec*) – Dict specifying which tensorboard summaries should be created and added to the graph.
- **execution** (*spec*) – Dict specifying whether and how to do distributed training on the model's graph.
- **batching_capacity** (*int*) – Batching capacity.
- **variable_noise** (*float*) – The stddev value of a Normal distribution used for adding random noise to the model's output (for each batch, noise can be toggled and - if active - will be resampled). Use None for not adding any noise.
- **states_preprocessing** (*spec / dict of specs*) – Dict specifying whether and how to preprocess state signals (e.g. normalization, greyscale, etc..).
- **actions_exploration** (*spec / dict of specs*) – Dict specifying whether and how to add exploration to the model's "action outputs" (e.g. epsilon-greedy).
- **reward_preprocessing** (*spec*) – Dict specifying whether and how to preprocess rewards coming from the Environment (e.g. reward normalization).
- **update_mode** (*spec*) – Update mode.
- **memory** (*spec*) – Memory.
- **optimizer** (*spec*) – Dict specifying the tf optimizer to use for tuning the model's trainable parameters.
- **discount** (*float*) – The RL reward discount factor (gamma).

act(*states*, *internals*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)
Does a forward pass through the model to retrieve action (outputs) given inputs for state (and internal state, if applicable (e.g. RNNs))

Parameters

- **states** (*dict*) – Dict of state values (each key represents one state space component).
- **internals** (*dict*) – Dict of internal state values (each key represents one internal state component).
- **deterministic** (*bool*) – If True, will not apply exploration after actions are calculated.
- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).

Returns

- Actual action-outputs (batched if state input is a batch).

Return type tuple

as_local_model()

close()

create_act_operations(*states*, *internals*, *deterministic*, *independent*)

create_observe_operations(*terminal*, *reward*)

create_operations(*states*, *internals*, *actions*, *terminal*, *reward*, *deterministic*, *independent*)

get_component(*component_name*)
Looks up a component by its name.

Parameters **component_name** – The name of the component to look up.

Returns The component for the provided name or None if there is no such component.

get_components()
Returns a dictionary of component name to component of all the components within this model.

Returns (dict) The mapping of name to component.

get_feed_dict(*states=None*, *internals=None*, *actions=None*, *terminal=None*, *reward=None*, *deterministic=None*, *independent=None*)

get_savable_components()
Returns the list of all of the components this model consists of that can be individually saved and restored. For instance the network or distribution.

Returns List of util.SavableComponent

get_summaries()

get_variables(*include_submodules=False*, *include_nontrainable=False*)

import_experience(*states*, *internals*, *actions*, *terminal*, *reward*)
Stores experiences.

initialize(*custom_getter*)

observe(*terminal*, *reward*)
Adds an observation (reward and is-terminal) to the model without updating its trainable variables.

Parameters

- **terminal** (*bool*) – Whether the episode has terminated.
- **reward** (*float*) – The observed reward value.

Returns The value of the model-internal episode counter.

optimizer_arguments(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*)
Returns the optimizer arguments including the time, the list of variables to optimize, and various functions which the optimizer might require to perform an update step.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **actions** – Dict of action tensors.
- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **next_states** – Dict of successor state tensors.
- **next_internals** – List of posterior internal state tensors.

Returns Optimizer arguments as dict.

reset()
Resets the model to its initial state on episode start. This should also reset all preprocessor(s).

Returns Current episode, timestep counter and the shallow-copied list of internal state initialization Tensors.

Return type tuple

restore(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters

- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

restore_component(*component_name*, *save_path*)
Restores a component's parameters from a save location.

Parameters

- **component_name** – The component to restore.
- **save_path** – The save location.

save(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters

- **directory** – Optional checkpoint directory.
- **append_timestep** – Appends the current timestep to the checkpoint file if true.

Returns Checkpoint path where the model was saved.

save_component(*component_name*, *save_path*)
Saves a component of this model to the designated location.

Parameters

- **component_name** – The component to save.
- **save_path** – The location to save to.

Returns Checkpoint path where the component was saved.

setup()
Sets up the TensorFlow model graph and initializes (and enters) the TensorFlow session.

tf_action_exploration(*action*, *exploration*, *action_spec*)
Applies optional exploration to the action (post-processor for action outputs).

Parameters

- **action** (*tf.Tensor*) – The original output action tensor (to be post-processed).
- **exploration** (Exploration) – The Exploration object to use.
- **action_spec** (*dict*) – Dict specifying the action space.

Returns The post-processed action output tensor.

tf_actions_and_internals(*states*, *internals*, *deterministic*)
Creates and returns the TensorFlow operations for retrieving the actions and - if applicable - the posterior internal state Tensors in reaction to the given input states (and prior internal states).

Parameters

- **states** (*dict*) – Dict of state tensors (each key represents one state space component).
- **internals** – List of prior internal state tensors.
- **deterministic** – Boolean tensor indicating whether action should be chosen deterministically.

Returns

1. dict of output actions (with or without exploration applied (see `deterministic`))
2. list of posterior internal state Tensors (empty for non-internal state models)

Return type tuple

tf_discounted_cumulative_reward(*terminal*, *reward*, *discount*, *final_reward=0.0*)
Creates the TensorFlow operations for calculating the discounted cumulative rewards for a given sequence of rewards.

Parameters

- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **discount** – Discount factor.
- **final_reward** – Last reward value in the sequence.

Returns Discounted cumulative reward tensor.

tf_import_experience(*states*, *internals*, *actions*, *terminal*, *reward*)
Imports experiences into the TensorFlow memory structure. Can be used to import off-policy data.

Parameters

- **states** – Dict of state values to import with keys as state names and values as values to set.
- **internals** – Internal values to set, can be fetched from agent via agent.current_internals if no values available.
- **actions** – Dict of action values to import with keys as action names and values as values to set.
- **terminal** – Terminal value(s)
- **reward** – Reward value(s)

tf_initialize()

tf_loss(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)

Creates the TensorFlow operations for calculating the full loss of a batch.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **actions** – Dict of action tensors.
- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **next_states** – Dict of successor state tensors.
- **next_internals** – List of posterior internal state tensors.
- **update** – Boolean tensor indicating whether this call happens during an update.
- **reference** – Optional reference tensor(s), in case of a comparative loss.

Returns Loss tensor.

tf_loss_per_instance(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)

Creates the TensorFlow operations for calculating the loss per batch instance.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **actions** – Dict of action tensors.
- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **next_states** – Dict of successor state tensors.
- **next_internals** – List of posterior internal state tensors.
- **update** – Boolean tensor indicating whether this call happens during an update.
- **reference** – Optional reference tensor(s), in case of a comparative loss.

Returns Loss per instance tensor.

tf_observe_timestep(*states*, *internals*, *actions*, *terminal*, *reward*)

tf_optimization(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states=None*, *next_internals=None*)

Creates the TensorFlow operations for performing an optimization update step based on the given input states and actions batch.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **actions** – Dict of action tensors.
- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **next_states** – Dict of successor state tensors.
- **next_internals** – List of posterior internal state tensors.

Returns The optimization operation.

tf_preprocess(*states*, *actions*, *reward*)

tf_reference(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*)

Creates the TensorFlow operations for obtaining the reference tensor(s), in case of a comparative loss.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **actions** – Dict of action tensors.
- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **next_states** – Dict of successor state tensors.
- **next_internals** – List of posterior internal state tensors.
- **update** – Boolean tensor indicating whether this call happens during an update.

Returns Reference tensor(s).

tf_regularization_losses(*states*, *internals*, *update*)

Creates the TensorFlow operations for calculating the regularization losses for the given input states.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **update** – Boolean tensor indicating whether this call happens during an update.

Returns Dict of regularization loss tensors.

class tensorforce.models.**DistributionModel**(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*, *variable_noise*, *states_preprocessing*, *actions_exploration*, *reward_preprocessing*, *update_mode*, *memory*, *optimizer*, *discount*, *network*, *distributions*, *entropy_regularization*, *requires_deterministic*)

Bases: tensorforce.models.memory_model.MemoryModel

Base class for models using distributions parametrized by a neural network.

COMPONENT_DISTRIBUTION = 'distribution'

COMPONENT_NETWORK = 'network'

__init__(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*, *variable_noise*, *states_preprocessing*, *actions_exploration*, *reward_preprocessing*, *update_mode*, *memory*, *optimizer*, *discount*, *network*, *distributions*, *entropy_regularization*, *requires_deterministic*)

act(*states*, *internals*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)
Does a forward pass through the model to retrieve action (outputs) given inputs for state (and internal state, if applicable (e.g. RNNs))

Parameters

- **states** (*dict*) – Dict of state values (each key represents one state space component).
- **internals** (*dict*) – Dict of internal state values (each key represents one internal state component).
- **deterministic** (*bool*) – If True, will not apply exploration after actions are calculated.
- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).

Returns

- Actual action-outputs (batched if state input is a batch).

Return type tuple

as_local_model()

close()

create_act_operations(*states*, *internals*, *deterministic*, *independent*)

create_distributions()

create_observe_operations(*terminal*, *reward*)

create_operations(*states*, *internals*, *actions*, *terminal*, *reward*, *deterministic*, *independent*)

get_component(*component_name*)
Looks up a component by its name.

Parameters **component_name** – The name of the component to look up.

Returns The component for the provided name or None if there is no such component.

get_components()

get_feed_dict(*states=None*, *internals=None*, *actions=None*, *terminal=None*, *reward=None*, *deterministic=None*, *independent=None*)

get_savable_components()
Returns the list of all of the components this model consists of that can be individually saved and restored. For instance the network or distribution.

Returns List of util.SavableComponent

get_summaries()

get_variables(*include_submodules=False*, *include_nontrainable=False*)

import_experience(*states*, *internals*, *actions*, *terminal*, *reward*)
Stores experiences.

initialize(*custom_getter*)

observe(*terminal*, *reward*)
Adds an observation (reward and is-terminal) to the model without updating its trainable variables.

Parameters

- **terminal** (*bool*) – Whether the episode has terminated.
- **reward** (*float*) – The observed reward value.

Returns The value of the model-internal episode counter.

optimizer_arguments(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*)

reset()
Resets the model to its initial state on episode start. This should also reset all preprocessor(s).

Returns Current episode, timestep counter and the shallow-copied list of internal state initialization Tensors.

Return type tuple

restore(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters

- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

restore_component(*component_name*, *save_path*)
Restores a component's parameters from a save location.

Parameters

- **component_name** – The component to restore.
- **save_path** – The save location.

save(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters

- **directory** – Optional checkpoint directory.
- **append_timestep** – Appends the current timestep to the checkpoint file if true.

Returns Checkpoint path where the model was saved.

save_component(*component_name*, *save_path*)
Saves a component of this model to the designated location.

Parameters

- **component_name** – The component to save.
- **save_path** – The location to save to.

Returns Checkpoint path where the component was saved.

setup()
Sets up the TensorFlow model graph and initializes (and enters) the TensorFlow session.

tf_action_exploration(*action*, *exploration*, *action_spec*)
Applies optional exploration to the action (post-processor for action outputs).

Parameters

- **action** (*tf.Tensor*) – The original output action tensor (to be post-processed).
- **exploration** (Exploration) – The Exploration object to use.
- **action_spec** (*dict*) – Dict specifying the action space.

Returns The post-processed action output tensor.

tf_actions_and_internals(*states*, *internals*, *deterministic*)

tf_discounted_cumulative_reward(*terminal*, *reward*, *discount*, *final_reward=0.0*)
Creates the TensorFlow operations for calculating the discounted cumulative rewards for a given sequence of rewards.

Parameters

- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **discount** – Discount factor.
- **final_reward** – Last reward value in the sequence.

Returns Discounted cumulative reward tensor.

tf_import_experience(*states*, *internals*, *actions*, *terminal*, *reward*)
Imports experiences into the TensorFlow memory structure. Can be used to import off-policy data.

Parameters

- **states** – Dict of state values to import with keys as state names and values as values to set.
- **internals** – Internal values to set, can be fetched from agent via agent.current_internals if no values available.
- **actions** – Dict of action values to import with keys as action names and values as values to set.
- **terminal** – Terminal value(s)
- **reward** – Reward value(s)

tf_initialize()

tf_kl_divergence(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)

tf_loss(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)
Creates the TensorFlow operations for calculating the full loss of a batch.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.

- **actions** – Dict of action tensors.
- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **next_states** – Dict of successor state tensors.
- **next_internals** – List of posterior internal state tensors.
- **update** – Boolean tensor indicating whether this call happens during an update.
- **reference** – Optional reference tensor(s), in case of a comparative loss.

Returns Loss tensor.

tf_loss_per_instance(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)

Creates the TensorFlow operations for calculating the loss per batch instance.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **actions** – Dict of action tensors.
- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **next_states** – Dict of successor state tensors.
- **next_internals** – List of posterior internal state tensors.
- **update** – Boolean tensor indicating whether this call happens during an update.
- **reference** – Optional reference tensor(s), in case of a comparative loss.

Returns Loss per instance tensor.

tf_observe_timestep(*states*, *internals*, *actions*, *terminal*, *reward*)

tf_optimization(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states=None*, *next_internals=None*)

Creates the TensorFlow operations for performing an optimization update step based on the given input states and actions batch.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **actions** – Dict of action tensors.
- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **next_states** – Dict of successor state tensors.
- **next_internals** – List of posterior internal state tensors.

Returns The optimization operation.

tf_preprocess(*states*, *actions*, *reward*)

tf_reference(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*)
Creates the TensorFlow operations for obtaining the reference tensor(s), in case of a comparative loss.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **actions** – Dict of action tensors.
- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **next_states** – Dict of successor state tensors.
- **next_internals** – List of posterior internal state tensors.
- **update** – Boolean tensor indicating whether this call happens during an update.

Returns Reference tensor(s).

tf_regularization_losses(*states*, *internals*, *update*)

class tensorforce.models.**PGModel**(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*, *variable_noise*, *states_preprocessing*, *actions_exploration*, *reward_preprocessing*, *update_mode*, *memory*, *optimizer*, *discount*, *network*, *distributions*, *entropy_regularization*, *baseline_mode*, *baseline*, *baseline_optimizer*, *gae_lambda*)
Bases: *tensorforce.models.distribution_model.DistributionModel*

Base class for policy gradient models. It optionally defines a baseline and handles its optimization. It implements the `tf_loss_per_instance` function, but requires subclasses to implement `tf_pg_loss_per_instance`.

COMPONENT_BASELINE = 'baseline'

COMPONENT_DISTRIBUTION = 'distribution'

COMPONENT_NETWORK = 'network'

__init__(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*, *variable_noise*, *states_preprocessing*, *actions_exploration*, *reward_preprocessing*, *update_mode*, *memory*, *optimizer*, *discount*, *network*, *distributions*, *entropy_regularization*, *baseline_mode*, *baseline*, *baseline_optimizer*, *gae_lambda*)

act(*states*, *internals*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)
Does a forward pass through the model to retrieve action (outputs) given inputs for state (and internal state, if applicable (e.g. RNNs))

Parameters

- **states** (*dict*) – Dict of state values (each key represents one state space component).
- **internals** (*dict*) – Dict of internal state values (each key represents one internal state component).
- **deterministic** (*bool*) – If True, will not apply exploration after actions are calculated.
- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).

Returns

- Actual action-outputs (batched if state input is a batch).

Return type tuple

as_local_model()

baseline_optimizer_arguments(*states*, *internals*, *reward*)
Returns the baseline optimizer arguments including the time, the list of variables to optimize, and various functions which the optimizer might require to perform an update step.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **reward** – Reward tensor.

Returns Baseline optimizer arguments as dict.

close()

create_act_operations(*states*, *internals*, *deterministic*, *independent*)

create_distributions()

create_observe_operations(*terminal*, *reward*)

create_operations(*states*, *internals*, *actions*, *terminal*, *reward*, *deterministic*, *independent*)

get_component(*component_name*)
Looks up a component by its name.

Parameters **component_name** – The name of the component to look up.

Returns The component for the provided name or None if there is no such component.

get_components()

get_feed_dict(*states=None*, *internals=None*, *actions=None*, *terminal=None*, *reward=None*, *deterministic=None*, *independent=None*)

get_savable_components()
Returns the list of all of the components this model consists of that can be individually saved and restored. For instance the network or distribution.

Returns List of util.SavableComponent

get_summaries()

get_variables(*include_submodules=False*, *include_nontrainable=False*)

import_experience(*states*, *internals*, *actions*, *terminal*, *reward*)
Stores experiences.

initialize(*custom_getter*)

observe(*terminal*, *reward*)
Adds an observation (reward and is-terminal) to the model without updating its trainable variables.

Parameters

- **terminal** (*bool*) – Whether the episode has terminated.
- **reward** (*float*) – The observed reward value.

Returns The value of the model-internal episode counter.

optimizer_arguments(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*)

reset()
Resets the model to its initial state on episode start. This should also reset all preprocessor(s).

Returns Current episode, timestep counter and the shallow-copied list of internal state initialization Tensors.

Return type tuple

restore(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters

- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

restore_component(*component_name*, *save_path*)
Restores a component's parameters from a save location.

Parameters

- **component_name** – The component to restore.
- **save_path** – The save location.

save(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters

- **directory** – Optional checkpoint directory.
- **append_timestep** – Appends the current timestep to the checkpoint file if true.

Returns Checkpoint path where the model was saved.

save_component(*component_name*, *save_path*)
Saves a component of this model to the designated location.

Parameters

- **component_name** – The component to save.
- **save_path** – The location to save to.

Returns Checkpoint path where the component was saved.

setup()
Sets up the TensorFlow model graph and initializes (and enters) the TensorFlow session.

tf_action_exploration(*action*, *exploration*, *action_spec*)
Applies optional exploration to the action (post-processor for action outputs).

Parameters

- **action** (*tf.Tensor*) – The original output action tensor (to be post-processed).
- **exploration** (Exploration) – The Exploration object to use.
- **action_spec** (*dict*) – Dict specifying the action space.

Returns The post-processed action output tensor.

tf_actions_and_internals(*states*, *internals*, *deterministic*)

tf_baseline_loss(*states*, *internals*, *reward*, *update*, *reference=None*)
Creates the TensorFlow operations for calculating the baseline loss of a batch.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **reward** – Reward tensor.
- **update** – Boolean tensor indicating whether this call happens during an update.
- **reference** – Optional reference tensor(s), in case of a comparative loss.

Returns Loss tensor.

tf_discounted_cumulative_reward(*terminal*, *reward*, *discount*, *final_reward=0.0*)
Creates the TensorFlow operations for calculating the discounted cumulative rewards for a given sequence of rewards.

Parameters

- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **discount** – Discount factor.
- **final_reward** – Last reward value in the sequence.

Returns Discounted cumulative reward tensor.

tf_import_experience(*states*, *internals*, *actions*, *terminal*, *reward*)
Imports experiences into the TensorFlow memory structure. Can be used to import off-policy data.

Parameters

- **states** – Dict of state values to import with keys as state names and values as values to set.
- **internals** – Internal values to set, can be fetched from agent via agent.current_internals if no values available.
- **actions** – Dict of action values to import with keys as action names and values as values to set.
- **terminal** – Terminal value(s)
- **reward** – Reward value(s)

tf_initialize()

tf_kl_divergence(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)

tf_loss(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)
Creates the TensorFlow operations for calculating the full loss of a batch.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **actions** – Dict of action tensors.

- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **next_states** – Dict of successor state tensors.
- **next_internals** – List of posterior internal state tensors.
- **update** – Boolean tensor indicating whether this call happens during an update.
- **reference** – Optional reference tensor(s), in case of a comparative loss.

Returns Loss tensor.

tf_loss_per_instance(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)

Creates the TensorFlow operations for calculating the loss per batch instance.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **actions** – Dict of action tensors.
- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **next_states** – Dict of successor state tensors.
- **next_internals** – List of posterior internal state tensors.
- **update** – Boolean tensor indicating whether this call happens during an update.
- **reference** – Optional reference tensor(s), in case of a comparative loss.

Returns Loss per instance tensor.

tf_observe_timestep(*states*, *internals*, *actions*, *terminal*, *reward*)

tf_optimization(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states=None*, *next_internals=None*)

tf_preprocess(*states*, *actions*, *reward*)

tf_reference(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*)

Creates the TensorFlow operations for obtaining the reference tensor(s), in case of a comparative loss.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **actions** – Dict of action tensors.
- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **next_states** – Dict of successor state tensors.
- **next_internals** – List of posterior internal state tensors.
- **update** – Boolean tensor indicating whether this call happens during an update.

Returns Reference tensor(s).

tf_regularization_losses(*states*, *internals*, *update*)

tf_reward_estimation(*states*, *internals*, *terminal*, *reward*, *update*)

class tensorforce.models.**PGProbRatioModel**(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*, *variable_noise*, *states_preprocessing*, *actions_exploration*, *reward_preprocessing*, *update_mode*, *memory*, *optimizer*, *discount*, *network*, *distributions*, *entropy_regularization*, *baseline_mode*, *baseline*, *baseline_optimizer*, *gae_lambda*, *likelihood_ratio_clipping*)

Bases: *tensorforce.models.pg_model.PGModel*

Policy gradient model based on computing likelihood ratios, e.g. TRPO and PPO.

COMPONENT_BASELINE = 'baseline'

COMPONENT_DISTRIBUTION = 'distribution'

COMPONENT_NETWORK = 'network'

__init__(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*, *variable_noise*, *states_preprocessing*, *actions_exploration*, *reward_preprocessing*, *update_mode*, *memory*, *optimizer*, *discount*, *network*, *distributions*, *entropy_regularization*, *baseline_mode*, *baseline*, *baseline_optimizer*, *gae_lambda*, *likelihood_ratio_clipping*)

act(*states*, *internals*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)

Does a forward pass through the model to retrieve action (outputs) given inputs for state (and internal state, if applicable (e.g. RNNs))

Parameters

- **states** (*dict*) – Dict of state values (each key represents one state space component).
- **internals** (*dict*) – Dict of internal state values (each key represents one internal state component).
- **deterministic** (*bool*) – If True, will not apply exploration after actions are calculated.
- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).

Returns

- Actual action-outputs (batched if state input is a batch).

Return type tuple

as_local_model()

baseline_optimizer_arguments(*states*, *internals*, *reward*)

Returns the baseline optimizer arguments including the time, the list of variables to optimize, and various functions which the optimizer might require to perform an update step.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **reward** – Reward tensor.

Returns Baseline optimizer arguments as dict.

close()

create_act_operations(*states*, *internals*, *deterministic*, *independent*)

create_distributions()

create_observe_operations(*terminal*, *reward*)

create_operations(*states*, *internals*, *actions*, *terminal*, *reward*, *deterministic*, *independent*)

get_component(*component_name*)
Looks up a component by its name.

Parameters **component_name** – The name of the component to look up.

Returns The component for the provided name or None if there is no such component.

get_components()

get_feed_dict(*states=None*, *internals=None*, *actions=None*, *terminal=None*, *reward=None*, *deterministic=None*, *independent=None*)

get_savable_components()
Returns the list of all of the components this model consists of that can be individually saved and restored. For instance the network or distribution.

Returns List of util.SavableComponent

get_summaries()

get_variables(*include_submodules=False*, *include_nontrainable=False*)

import_experience(*states*, *internals*, *actions*, *terminal*, *reward*)
Stores experiences.

initialize(*custom_getter*)

observe(*terminal*, *reward*)
Adds an observation (reward and is-terminal) to the model without updating its trainable variables.

Parameters

- **terminal** (*bool*) – Whether the episode has terminated.
- **reward** (*float*) – The observed reward value.

Returns The value of the model-internal episode counter.

optimizer_arguments(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*)

reset()
Resets the model to its initial state on episode start. This should also reset all preprocessor(s).

Returns Current episode, timestep counter and the shallow-copied list of internal state initialization Tensors.

Return type tuple

restore(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters

- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

restore_component(*component_name*, *save_path*)
Restores a component's parameters from a save location.

Parameters

- **component_name** – The component to restore.
- **save_path** – The save location.

save(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters

- **directory** – Optional checkpoint directory.
- **append_timestep** – Appends the current timestep to the checkpoint file if true.

Returns Checkpoint path where the model was saved.

save_component(*component_name*, *save_path*)
Saves a component of this model to the designated location.

Parameters

- **component_name** – The component to save.
- **save_path** – The location to save to.

Returns Checkpoint path where the component was saved.

setup()
Sets up the TensorFlow model graph and initializes (and enters) the TensorFlow session.

tf_action_exploration(*action*, *exploration*, *action_spec*)
Applies optional exploration to the action (post-processor for action outputs).

Parameters

- **action** (*tf.Tensor*) – The original output action tensor (to be post-processed).
- **exploration** (Exploration) – The Exploration object to use.
- **action_spec** (*dict*) – Dict specifying the action space.

Returns The post-processed action output tensor.

tf_actions_and_internals(*states*, *internals*, *deterministic*)

tf_baseline_loss(*states*, *internals*, *reward*, *update*, *reference=None*)
Creates the TensorFlow operations for calculating the baseline loss of a batch.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **reward** – Reward tensor.
- **update** – Boolean tensor indicating whether this call happens during an update.
- **reference** – Optional reference tensor(s), in case of a comparative loss.

Returns Loss tensor.

tf_discounted_cumulative_reward(*terminal*, *reward*, *discount*, *final_reward=0.0*)
Creates the TensorFlow operations for calculating the discounted cumulative rewards for a given sequence of rewards.

Parameters

- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **discount** – Discount factor.
- **final_reward** – Last reward value in the sequence.

Returns Discounted cumulative reward tensor.

tf_import_experience(*states*, *internals*, *actions*, *terminal*, *reward*)

Imports experiences into the TensorFlow memory structure. Can be used to import off-policy data.

Parameters

- **states** – Dict of state values to import with keys as state names and values as values to set.
- **internals** – Internal values to set, can be fetched from agent via agent.current_internals if no values available.
- **actions** – Dict of action values to import with keys as action names and values as values to set.
- **terminal** – Terminal value(s)
- **reward** – Reward value(s)

tf_initialize()

tf_kl_divergence(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)

tf_loss(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)

Creates the TensorFlow operations for calculating the full loss of a batch.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **actions** – Dict of action tensors.
- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **next_states** – Dict of successor state tensors.
- **next_internals** – List of posterior internal state tensors.
- **update** – Boolean tensor indicating whether this call happens during an update.
- **reference** – Optional reference tensor(s), in case of a comparative loss.

Returns Loss tensor.

tf_loss_per_instance(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)

tf_observe_timestep(*states*, *internals*, *actions*, *terminal*, *reward*)

tf_optimization(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states=None*, *next_internals=None*)

tf_preprocess(*states*, *actions*, *reward*)

tf_reference(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*)

tf_regularization_losses(*states*, *internals*, *update*)

tf_reward_estimation(*states*, *internals*, *terminal*, *reward*, *update*)

class tensorforce.models.**DPGTargetModel**(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*, *variable_noise*, *states_preprocessing*, *actions_exploration*, *reward_preprocessing*, *update_mode*, *memory*, *optimizer*, *discount*, *network*, *distributions*, *entropy_regularization*, *critic_network*, *critic_optimizer*, *target_sync_frequency*, *target_update_weight*)

Bases: *tensorforce.models.distribution_model.DistributionModel*

Policy gradient model log likelihood model with target network (e.g. DDPG)

COMPONENT_CRITIC = 'critic'

COMPONENT_DISTRIBUTION = 'distribution'

COMPONENT_NETWORK = 'network'

COMPONENT_TARGET_DISTRIBUTION = 'target_distribution'

COMPONENT_TARGET_NETWORK = 'target_network'

__init__(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*, *variable_noise*, *states_preprocessing*, *actions_exploration*, *reward_preprocessing*, *update_mode*, *memory*, *optimizer*, *discount*, *network*, *distributions*, *entropy_regularization*, *critic_network*, *critic_optimizer*, *target_sync_frequency*, *target_update_weight*)

act(*states*, *internals*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)

Does a forward pass through the model to retrieve action (outputs) given inputs for state (and internal state, if applicable (e.g. RNNs))

Parameters

- **states** (*dict*) – Dict of state values (each key represents one state space component).
- **internals** (*dict*) – Dict of internal state values (each key represents one internal state component).
- **deterministic** (*bool*) – If True, will not apply exploration after actions are calculated.
- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).

Returns

- Actual action-outputs (batched if state input is a batch).

Return type tuple

as_local_model()

close()

create_act_operations(*states*, *internals*, *deterministic*, *independent*)

create_distributions()

create_observe_operations(*terminal*, *reward*)

create_operations(*states*, *internals*, *actions*, *terminal*, *reward*, *deterministic*, *independent*)

get_component(*component_name*)
Looks up a component by its name.

Parameters **component_name** – The name of the component to look up.

Returns The component for the provided name or None if there is no such component.

get_components()

get_feed_dict(*states=None*, *internals=None*, *actions=None*, *terminal=None*, *reward=None*, *deterministic=None*, *independent=None*)

get_savable_components()
Returns the list of all of the components this model consists of that can be individually saved and restored. For instance the network or distribution.

Returns List of util.SavableComponent

get_summaries()

get_variables(*include_submodules=False*, *include_nontrainable=False*)

import_experience(*states*, *internals*, *actions*, *terminal*, *reward*)
Stores experiences.

initialize(*custom_getter*)

observe(*terminal*, *reward*)
Adds an observation (reward and is-terminal) to the model without updating its trainable variables.

Parameters

- **terminal** (*bool*) – Whether the episode has terminated.
- **reward** (*float*) – The observed reward value.

Returns The value of the model-internal episode counter.

optimizer_arguments(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*)

reset()
Resets the model to its initial state on episode start. This should also reset all preprocessor(s).

Returns Current episode, timestep counter and the shallow-copied list of internal state initialization Tensors.

Return type tuple

restore(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters

- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

restore_component(*component_name*, *save_path*)
Restores a component's parameters from a save location.

Parameters

- **component_name** – The component to restore.
- **save_path** – The save location.

save(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters

- **directory** – Optional checkpoint directory.
- **append_timestep** – Appends the current timestep to the checkpoint file if true.

Returns Checkpoint path where the model was saved.

save_component(*component_name*, *save_path*)
Saves a component of this model to the designated location.

Parameters

- **component_name** – The component to save.
- **save_path** – The location to save to.

Returns Checkpoint path where the component was saved.

setup()
Sets up the TensorFlow model graph and initializes (and enters) the TensorFlow session.

tf_action_exploration(*action*, *exploration*, *action_spec*)
Applies optional exploration to the action (post-processor for action outputs).

Parameters

- **action** (*tf.Tensor*) – The original output action tensor (to be post-processed).
- **exploration** (Exploration) – The Exploration object to use.
- **action_spec** (*dict*) – Dict specifying the action space.

Returns The post-processed action output tensor.

tf_actions_and_internals(*states*, *internals*, *deterministic*)

tf_discounted_cumulative_reward(*terminal*, *reward*, *discount*, *final_reward=0.0*)
Creates the TensorFlow operations for calculating the discounted cumulative rewards for a given sequence of rewards.

Parameters

- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **discount** – Discount factor.
- **final_reward** – Last reward value in the sequence.

Returns Discounted cumulative reward tensor.

tf_import_experience(*states*, *internals*, *actions*, *terminal*, *reward*)
Imports experiences into the TensorFlow memory structure. Can be used to import off-policy data.

Parameters

- **states** – Dict of state values to import with keys as state names and values as values to set.
- **internals** – Internal values to set, can be fetched from agent via agent.current_internals if no values available.

- **actions** – Dict of action values to import with keys as action names and values as values to set.
- **terminal** – Terminal value(s)
- **reward** – Reward value(s)

tf_initialize()

tf_kl_divergence(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)

tf_loss(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)

Creates the TensorFlow operations for calculating the full loss of a batch.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **actions** – Dict of action tensors.
- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **next_states** – Dict of successor state tensors.
- **next_internals** – List of posterior internal state tensors.
- **update** – Boolean tensor indicating whether this call happens during an update.
- **reference** – Optional reference tensor(s), in case of a comparative loss.

Returns Loss tensor.

tf_loss_per_instance(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)

tf_observe_timestep(*states*, *internals*, *actions*, *terminal*, *reward*)

tf_optimization(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states=None*, *next_internals=None*)

tf_predict_target_q(*states*, *internals*, *terminal*, *actions*, *reward*, *update*)

tf_preprocess(*states*, *actions*, *reward*)

tf_reference(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*)

Creates the TensorFlow operations for obtaining the reference tensor(s), in case of a comparative loss.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **actions** – Dict of action tensors.
- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **next_states** – Dict of successor state tensors.
- **next_internals** – List of posterior internal state tensors.
- **update** – Boolean tensor indicating whether this call happens during an update.

Returns Reference tensor(s).

tf_regularization_losses(*states*, *internals*, *update*)

tf_target_actions_and_internals(*states*, *internals*, *deterministic=True*)

class tensorforce.models.**PGLogProbModel**(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*, *variable_noise*, *states_preprocessing*, *actions_exploration*, *reward_preprocessing*, *update_mode*, *memory*, *optimizer*, *discount*, *network*, *distributions*, *entropy_regularization*, *baseline_mode*, *baseline*, *baseline_optimizer*, *gae_lambda*)

Bases: *tensorforce.models.pg_model.PGModel*

Policy gradient model based on computing log likelihoods, e.g. VPG.

COMPONENT_BASELINE = 'baseline'

COMPONENT_DISTRIBUTION = 'distribution'

COMPONENT_NETWORK = 'network'

__init__(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*, *variable_noise*, *states_preprocessing*, *actions_exploration*, *reward_preprocessing*, *update_mode*, *memory*, *optimizer*, *discount*, *network*, *distributions*, *entropy_regularization*, *baseline_mode*, *baseline*, *baseline_optimizer*, *gae_lambda*)

act(*states*, *internals*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)

Does a forward pass through the model to retrieve action (outputs) given inputs for state (and internal state, if applicable (e.g. RNNs))

Parameters

- **states** (*dict*) – Dict of state values (each key represents one state space component).
- **internals** (*dict*) – Dict of internal state values (each key represents one internal state component).
- **deterministic** (*bool*) – If True, will not apply exploration after actions are calculated.
- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).

Returns

- Actual action-outputs (batched if state input is a batch).

Return type tuple

as_local_model()

baseline_optimizer_arguments(*states*, *internals*, *reward*)

Returns the baseline optimizer arguments including the time, the list of variables to optimize, and various functions which the optimizer might require to perform an update step.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **reward** – Reward tensor.

Returns Baseline optimizer arguments as dict.

close()

create_act_operations(*states*, *internals*, *deterministic*, *independent*)

create_distributions()

create_observe_operations(*terminal*, *reward*)

create_operations(*states*, *internals*, *actions*, *terminal*, *reward*, *deterministic*, *independent*)

get_component(*component_name*)
Looks up a component by its name.

Parameters **component_name** – The name of the component to look up.

Returns The component for the provided name or None if there is no such component.

get_components()

get_feed_dict(*states=None*, *internals=None*, *actions=None*, *terminal=None*, *reward=None*, *deterministic=None*, *independent=None*)

get_savable_components()
Returns the list of all of the components this model consists of that can be individually saved and restored. For instance the network or distribution.

Returns List of util.SavableComponent

get_summaries()

get_variables(*include_submodules=False*, *include_nontrainable=False*)

import_experience(*states*, *internals*, *actions*, *terminal*, *reward*)
Stores experiences.

initialize(*custom_getter*)

observe(*terminal*, *reward*)
Adds an observation (reward and is-terminal) to the model without updating its trainable variables.

Parameters

- **terminal** (*bool*) – Whether the episode has terminated.
- **reward** (*float*) – The observed reward value.

Returns The value of the model-internal episode counter.

optimizer_arguments(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*)

reset()
Resets the model to its initial state on episode start. This should also reset all preprocessor(s).

Returns Current episode, timestep counter and the shallow-copied list of internal state initialization Tensors.

Return type tuple

restore(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters

- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

restore_component(*component_name*, *save_path*)
Restores a component's parameters from a save location.

Parameters

- **component_name** – The component to restore.
- **save_path** – The save location.

save(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters

- **directory** – Optional checkpoint directory.
- **append_timestep** – Appends the current timestep to the checkpoint file if true.

Returns Checkpoint path where the model was saved.

save_component(*component_name*, *save_path*)
Saves a component of this model to the designated location.

Parameters

- **component_name** – The component to save.
- **save_path** – The location to save to.

Returns Checkpoint path where the component was saved.

setup()
Sets up the TensorFlow model graph and initializes (and enters) the TensorFlow session.

tf_action_exploration(*action*, *exploration*, *action_spec*)
Applies optional exploration to the action (post-processor for action outputs).

Parameters

- **action** (*tf.Tensor*) – The original output action tensor (to be post-processed).
- **exploration** (Exploration) – The Exploration object to use.
- **action_spec** (*dict*) – Dict specifying the action space.

Returns The post-processed action output tensor.

tf_actions_and_internals(*states*, *internals*, *deterministic*)

tf_baseline_loss(*states*, *internals*, *reward*, *update*, *reference=None*)
Creates the TensorFlow operations for calculating the baseline loss of a batch.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **reward** – Reward tensor.
- **update** – Boolean tensor indicating whether this call happens during an update.
- **reference** – Optional reference tensor(s), in case of a comparative loss.

Returns Loss tensor.

tf_discounted_cumulative_reward(*terminal*, *reward*, *discount*, *final_reward=0.0*)
Creates the TensorFlow operations for calculating the discounted cumulative rewards for a given sequence of rewards.

Parameters

- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **discount** – Discount factor.
- **final_reward** – Last reward value in the sequence.

Returns Discounted cumulative reward tensor.

tf_import_experience(*states*, *internals*, *actions*, *terminal*, *reward*)
Imports experiences into the TensorFlow memory structure. Can be used to import off-policy data.

Parameters

- **states** – Dict of state values to import with keys as state names and values as values to set.
- **internals** – Internal values to set, can be fetched from agent via agent.current_internals if no values available.
- **actions** – Dict of action values to import with keys as action names and values as values to set.
- **terminal** – Terminal value(s)
- **reward** – Reward value(s)

tf_initialize()

tf_kl_divergence(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)

tf_loss(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)
Creates the TensorFlow operations for calculating the full loss of a batch.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **actions** – Dict of action tensors.
- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **next_states** – Dict of successor state tensors.
- **next_internals** – List of posterior internal state tensors.
- **update** – Boolean tensor indicating whether this call happens during an update.
- **reference** – Optional reference tensor(s), in case of a comparative loss.

Returns Loss tensor.

tf_loss_per_instance(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)

tf_observe_timestep(*states*, *internals*, *actions*, *terminal*, *reward*)

tf_optimization(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states=None*, *next_internals=None*)

tf_preprocess(*states*, *actions*, *reward*)

tf_reference(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*)
Creates the TensorFlow operations for obtaining the reference tensor(s), in case of a comparative loss.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **actions** – Dict of action tensors.
- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **next_states** – Dict of successor state tensors.
- **next_internals** – List of posterior internal state tensors.
- **update** – Boolean tensor indicating whether this call happens during an update.

Returns Reference tensor(s).

tf_regularization_losses(*states*, *internals*, *update*)

tf_reward_estimation(*states*, *internals*, *terminal*, *reward*, *update*)

class tensorforce.models.**QModel**(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*, *variable_noise*, *states_preprocessing*, *actions_exploration*, *reward_preprocessing*, *update_mode*, *memory*, *optimizer*, *discount*, *network*, *distributions*, *entropy_regularization*, *target_sync_frequency*, *target_update_weight*, *double_q_model*, *huber_loss*)

Bases: *tensorforce.models.distribution_model.DistributionModel*

Q-value model.

COMPONENT_DISTRIBUTION = 'distribution'

COMPONENT_NETWORK = 'network'

COMPONENT_TARGET_DISTRIBUTION = 'target_distribution'

COMPONENT_TARGET_NETWORK = 'target_network'

__init__(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*, *variable_noise*, *states_preprocessing*, *actions_exploration*, *reward_preprocessing*, *update_mode*, *memory*, *optimizer*, *discount*, *network*, *distributions*, *entropy_regularization*, *target_sync_frequency*, *target_update_weight*, *double_q_model*, *huber_loss*)

act(*states*, *internals*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)
Does a forward pass through the model to retrieve action (outputs) given inputs for state (and internal state, if applicable (e.g. RNNs))

Parameters

- **states** (*dict*) – Dict of state values (each key represents one state space component).
- **internals** (*dict*) – Dict of internal state values (each key represents one internal state component).

- **deterministic** (*bool*) – If True, will not apply exploration after actions are calculated.
- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).

Returns

- Actual action-outputs (batched if state input is a batch).

Return type tuple

as_local_model()

close()

create_act_operations(*states*, *internals*, *deterministic*, *independent*)

create_distributions()

create_observe_operations(*terminal*, *reward*)

create_operations(*states*, *internals*, *actions*, *terminal*, *reward*, *deterministic*, *independent*)

get_component(*component_name*)
Looks up a component by its name.

Parameters **component_name** – The name of the component to look up.

Returns The component for the provided name or None if there is no such component.

get_components()

get_feed_dict(*states=None*, *internals=None*, *actions=None*, *terminal=None*, *reward=None*, *deterministic=None*, *independent=None*)

get_savable_components()
Returns the list of all of the components this model consists of that can be individually saved and restored. For instance the network or distribution.

Returns List of util.SavableComponent

get_summaries()

get_variables(*include_submodules=False*, *include_nontrainable=False*)

import_experience(*states*, *internals*, *actions*, *terminal*, *reward*)
Stores experiences.

initialize(*custom_getter*)

observe(*terminal*, *reward*)
Adds an observation (reward and is-terminal) to the model without updating its trainable variables.

Parameters

- **terminal** (*bool*) – Whether the episode has terminated.
- **reward** (*float*) – The observed reward value.

Returns The value of the model-internal episode counter.

optimizer_arguments(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*)

reset()
Resets the model to its initial state on episode start. This should also reset all preprocessor(s).

Returns Current episode, timestep counter and the shallow-copied list of internal state initialization Tensors.

Return type tuple

restore(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters

- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

restore_component(*component_name*, *save_path*)
Restores a component's parameters from a save location.

Parameters

- **component_name** – The component to restore.
- **save_path** – The save location.

save(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters

- **directory** – Optional checkpoint directory.
- **append_timestep** – Appends the current timestep to the checkpoint file if true.

Returns Checkpoint path where the model was saved.

save_component(*component_name*, *save_path*)
Saves a component of this model to the designated location.

Parameters

- **component_name** – The component to save.
- **save_path** – The location to save to.

Returns Checkpoint path where the component was saved.

setup()
Sets up the TensorFlow model graph and initializes (and enters) the TensorFlow session.

target_optimizer_arguments()
Returns the target optimizer arguments including the time, the list of variables to optimize, and various functions which the optimizer might require to perform an update step.

Returns Target optimizer arguments as dict.

tf_action_exploration(*action*, *exploration*, *action_spec*)
Applies optional exploration to the action (post-processor for action outputs).

Parameters

- **action** (*tf.Tensor*) – The original output action tensor (to be post-processed).
- **exploration** (Exploration) – The Exploration object to use.
- **action_spec** (*dict*) – Dict specifying the action space.

Returns The post-processed action output tensor.

tf_actions_and_internals(*states*, *internals*, *deterministic*)

tf_discounted_cumulative_reward(*terminal*, *reward*, *discount*, *final_reward=0.0*)
Creates the TensorFlow operations for calculating the discounted cumulative rewards for a given sequence of rewards.

Parameters

- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **discount** – Discount factor.
- **final_reward** – Last reward value in the sequence.

Returns Discounted cumulative reward tensor.

tf_import_experience(*states*, *internals*, *actions*, *terminal*, *reward*)
Imports experiences into the TensorFlow memory structure. Can be used to import off-policy data.

Parameters

- **states** – Dict of state values to import with keys as state names and values as values to set.
- **internals** – Internal values to set, can be fetched from agent via agent.current_internals if no values available.
- **actions** – Dict of action values to import with keys as action names and values as values to set.
- **terminal** – Terminal value(s)
- **reward** – Reward value(s)

tf_initialize()

tf_kl_divergence(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)

tf_loss(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)
Creates the TensorFlow operations for calculating the full loss of a batch.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **actions** – Dict of action tensors.
- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **next_states** – Dict of successor state tensors.
- **next_internals** – List of posterior internal state tensors.
- **update** – Boolean tensor indicating whether this call happens during an update.
- **reference** – Optional reference tensor(s), in case of a comparative loss.

Returns Loss tensor.

tf_loss_per_instance(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)

tf_observe_timestep(*states*, *internals*, *actions*, *terminal*, *reward*)

tf_optimization(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states=None*, *next_internals=None*)

tf_preprocess(*states*, *actions*, *reward*)

tf_q_delta(*q_value*, *next_q_value*, *terminal*, *reward*)
Creates the deltas (or advantage) of the Q values.

Returns A list of deltas per action

tf_q_value(*embedding*, *distr_params*, *action*, *name*)

tf_reference(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*)
Creates the TensorFlow operations for obtaining the reference tensor(s), in case of a comparative loss.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **actions** – Dict of action tensors.
- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **next_states** – Dict of successor state tensors.
- **next_internals** – List of posterior internal state tensors.
- **update** – Boolean tensor indicating whether this call happens during an update.

Returns Reference tensor(s).

tf_regularization_losses(*states*, *internals*, *update*)

class tensorforce.models.**QNstepModel**(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*, *variable_noise*, *states_preprocessing*, *actions_exploration*, *reward_preprocessing*, *update_mode*, *memory*, *optimizer*, *discount*, *network*, *distributions*, *entropy_regularization*, *target_sync_frequency*, *target_update_weight*, *double_q_model*, *huber_loss*)

Bases: *tensorforce.models.q_model.QModel*

Deep Q network using n-step rewards as described in Asynchronous Methods for Reinforcement Learning.

COMPONENT_DISTRIBUTION = 'distribution'

COMPONENT_NETWORK = 'network'

COMPONENT_TARGET_DISTRIBUTION = 'target_distribution'

COMPONENT_TARGET_NETWORK = 'target_network'

__init__(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*, *variable_noise*, *states_preprocessing*, *actions_exploration*, *reward_preprocessing*, *update_mode*, *memory*, *optimizer*, *discount*, *network*, *distributions*, *entropy_regularization*, *target_sync_frequency*, *target_update_weight*, *double_q_model*, *huber_loss*)

act(*states*, *internals*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)
Does a forward pass through the model to retrieve action (outputs) given inputs for state (and internal state, if applicable (e.g. RNNs))

Parameters

- **states** (*dict*) – Dict of state values (each key represents one state space component).
- **internals** (*dict*) – Dict of internal state values (each key represents one internal state component).
- **deterministic** (*bool*) – If True, will not apply exploration after actions are calculated.
- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).

Returns

- Actual action-outputs (batched if state input is a batch).

Return type tuple

as_local_model()

close()

create_act_operations(*states*, *internals*, *deterministic*, *independent*)

create_distributions()

create_observe_operations(*terminal*, *reward*)

create_operations(*states*, *internals*, *actions*, *terminal*, *reward*, *deterministic*, *independent*)

get_component(*component_name*)
Looks up a component by its name.

Parameters **component_name** – The name of the component to look up.

Returns The component for the provided name or None if there is no such component.

get_components()

get_feed_dict(*states=None*, *internals=None*, *actions=None*, *terminal=None*, *reward=None*, *deterministic=None*, *independent=None*)

get_savable_components()
Returns the list of all of the components this model consists of that can be individually saved and restored. For instance the network or distribution.

Returns List of util.SavableComponent

get_summaries()

get_variables(*include_submodules=False*, *include_nontrainable=False*)

import_experience(*states*, *internals*, *actions*, *terminal*, *reward*)
Stores experiences.

initialize(*custom_getter*)

observe(*terminal*, *reward*)
Adds an observation (reward and is-terminal) to the model without updating its trainable variables.

Parameters

- **terminal** (*bool*) – Whether the episode has terminated.
- **reward** (*float*) – The observed reward value.

Returns The value of the model-internal episode counter.

optimizer_arguments(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*)

reset()
Resets the model to its initial state on episode start. This should also reset all preprocessor(s).

Returns Current episode, timestep counter and the shallow-copied list of internal state initialization Tensors.

Return type tuple

restore(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters

- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

restore_component(*component_name*, *save_path*)
Restores a component's parameters from a save location.

Parameters

- **component_name** – The component to restore.
- **save_path** – The save location.

save(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters

- **directory** – Optional checkpoint directory.
- **append_timestep** – Appends the current timestep to the checkpoint file if true.

Returns Checkpoint path where the model was saved.

save_component(*component_name*, *save_path*)
Saves a component of this model to the designated location.

Parameters

- **component_name** – The component to save.
- **save_path** – The location to save to.

Returns Checkpoint path where the component was saved.

setup()
Sets up the TensorFlow model graph and initializes (and enters) the TensorFlow session.

target_optimizer_arguments()
Returns the target optimizer arguments including the time, the list of variables to optimize, and various functions which the optimizer might require to perform an update step.

Returns Target optimizer arguments as dict.

tf_action_exploration(*action*, *exploration*, *action_spec*)
Applies optional exploration to the action (post-processor for action outputs).

Parameters

- **action** (*tf.Tensor*) – The original output action tensor (to be post-processed).
- **exploration** (Exploration) – The Exploration object to use.
- **action_spec** (*dict*) – Dict specifying the action space.

Returns The post-processed action output tensor.

tf_actions_and_internals(*states*, *internals*, *deterministic*)

tf_discounted_cumulative_reward(*terminal*, *reward*, *discount*, *final_reward=0.0*)
Creates the TensorFlow operations for calculating the discounted cumulative rewards for a given sequence of rewards.

Parameters

- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **discount** – Discount factor.
- **final_reward** – Last reward value in the sequence.

Returns Discounted cumulative reward tensor.

tf_import_experience(*states*, *internals*, *actions*, *terminal*, *reward*)
Imports experiences into the TensorFlow memory structure. Can be used to import off-policy data.

Parameters

- **states** – Dict of state values to import with keys as state names and values as values to set.
- **internals** – Internal values to set, can be fetched from agent via agent.current_internals if no values available.
- **actions** – Dict of action values to import with keys as action names and values as values to set.
- **terminal** – Terminal value(s)
- **reward** – Reward value(s)

tf_initialize()

tf_kl_divergence(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)

tf_loss(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)
Creates the TensorFlow operations for calculating the full loss of a batch.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **actions** – Dict of action tensors.
- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.

- **next_states** – Dict of successor state tensors.
- **next_internals** – List of posterior internal state tensors.
- **update** – Boolean tensor indicating whether this call happens during an update.
- **reference** – Optional reference tensor(s), in case of a comparative loss.

Returns Loss tensor.

tf_loss_per_instance(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)

tf_observe_timestep(*states*, *internals*, *actions*, *terminal*, *reward*)

tf_optimization(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states=None*, *next_internals=None*)

tf_preprocess(*states*, *actions*, *reward*)

tf_q_delta(*q_value*, *next_q_value*, *terminal*, *reward*)

tf_q_value(*embedding*, *distr_params*, *action*, *name*)

tf_reference(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*)

Creates the TensorFlow operations for obtaining the reference tensor(s), in case of a comparative loss.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **actions** – Dict of action tensors.
- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **next_states** – Dict of successor state tensors.
- **next_internals** – List of posterior internal state tensors.
- **update** – Boolean tensor indicating whether this call happens during an update.

Returns Reference tensor(s).

tf_regularization_losses(*states*, *internals*, *update*)

class tensorforce.models.**QNAFModel**(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*, *variable_noise*, *states_preprocessing*, *actions_exploration*, *reward_preprocessing*, *update_mode*, *memory*, *optimizer*, *discount*, *network*, *distributions*, *entropy_regularization*, *target_sync_frequency*, *target_update_weight*, *double_q_model*, *huber_loss*)

Bases: *tensorforce.models.q_model.QModel*

COMPONENT_DISTRIBUTION = 'distribution'

COMPONENT_NETWORK = 'network'

COMPONENT_TARGET_DISTRIBUTION = 'target_distribution'

COMPONENT_TARGET_NETWORK = 'target_network'

__init__(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*, *variable_noise*, *states_preprocessing*, *actions_exploration*, *reward_preprocessing*, *update_mode*, *memory*, *optimizer*, *discount*, *network*, *distributions*, *entropy_regularization*, *target_sync_frequency*, *target_update_weight*, *double_q_model*, *huber_loss*)

act(*states*, *internals*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)
Does a forward pass through the model to retrieve action (outputs) given inputs for state (and internal state, if applicable (e.g. RNNs))

Parameters

- **states** (*dict*) – Dict of state values (each key represents one state space component).
- **internals** (*dict*) – Dict of internal state values (each key represents one internal state component).
- **deterministic** (*bool*) – If True, will not apply exploration after actions are calculated.
- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).

Returns

- Actual action-outputs (batched if state input is a batch).

Return type tuple

as_local_model()

close()

create_act_operations(*states*, *internals*, *deterministic*, *independent*)

create_distributions()

create_observe_operations(*terminal*, *reward*)

create_operations(*states*, *internals*, *actions*, *terminal*, *reward*, *deterministic*, *independent*)

get_component(*component_name*)
Looks up a component by its name.

Parameters **component_name** – The name of the component to look up.

Returns The component for the provided name or None if there is no such component.

get_components()

get_feed_dict(*states=None*, *internals=None*, *actions=None*, *terminal=None*, *reward=None*, *deterministic=None*, *independent=None*)

get_savable_components()
Returns the list of all of the components this model consists of that can be individually saved and restored. For instance the network or distribution.

Returns List of util.SavableComponent

get_summaries()

get_variables(*include_submodules=False*, *include_nontrainable=False*)

import_experience(*states*, *internals*, *actions*, *terminal*, *reward*)
Stores experiences.

initialize(*custom_getter*)

observe(*terminal*, *reward*)
Adds an observation (reward and is-terminal) to the model without updating its trainable variables.

Parameters

- **terminal** (*bool*) – Whether the episode has terminated.
- **reward** (*float*) – The observed reward value.

Returns The value of the model-internal episode counter.

optimizer_arguments(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*)

reset()
Resets the model to its initial state on episode start. This should also reset all preprocessor(s).

Returns Current episode, timestep counter and the shallow-copied list of internal state initialization Tensors.

Return type tuple

restore(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters

- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

restore_component(*component_name*, *save_path*)
Restores a component's parameters from a save location.

Parameters

- **component_name** – The component to restore.
- **save_path** – The save location.

save(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters

- **directory** – Optional checkpoint directory.
- **append_timestep** – Appends the current timestep to the checkpoint file if true.

Returns Checkpoint path where the model was saved.

save_component(*component_name*, *save_path*)
Saves a component of this model to the designated location.

Parameters

- **component_name** – The component to save.
- **save_path** – The location to save to.

Returns Checkpoint path where the component was saved.

setup()
Sets up the TensorFlow model graph and initializes (and enters) the TensorFlow session.

target_optimizer_arguments()
Returns the target optimizer arguments including the time, the list of variables to optimize, and various functions which the optimizer might require to perform an update step.

Returns Target optimizer arguments as dict.

tf_action_exploration(*action*, *exploration*, *action_spec*)
Applies optional exploration to the action (post-processor for action outputs).

Parameters

- **action** (*tf.Tensor*) – The original output action tensor (to be post-processed).
- **exploration** (Exploration) – The Exploration object to use.
- **action_spec** (*dict*) – Dict specifying the action space.

Returns The post-processed action output tensor.

tf_actions_and_internals(*states*, *internals*, *deterministic*)

tf_discounted_cumulative_reward(*terminal*, *reward*, *discount*, *final_reward=0.0*)
Creates the TensorFlow operations for calculating the discounted cumulative rewards for a given sequence of rewards.

Parameters

- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **discount** – Discount factor.
- **final_reward** – Last reward value in the sequence.

Returns Discounted cumulative reward tensor.

tf_import_experience(*states*, *internals*, *actions*, *terminal*, *reward*)
Imports experiences into the TensorFlow memory structure. Can be used to import off-policy data.

Parameters

- **states** – Dict of state values to import with keys as state names and values as values to set.
- **internals** – Internal values to set, can be fetched from agent via agent.current_internals if no values available.
- **actions** – Dict of action values to import with keys as action names and values as values to set.
- **terminal** – Terminal value(s)
- **reward** – Reward value(s)

tf_initialize()

tf_kl_divergence(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)

tf_loss(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)
Creates the TensorFlow operations for calculating the full loss of a batch.

Parameters

- **states** – Dict of state tensors.

- **internals** – List of prior internal state tensors.
- **actions** – Dict of action tensors.
- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **next_states** – Dict of successor state tensors.
- **next_internals** – List of posterior internal state tensors.
- **update** – Boolean tensor indicating whether this call happens during an update.
- **reference** – Optional reference tensor(s), in case of a comparative loss.

Returns Loss tensor.

tf_loss_per_instance(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)

tf_observe_timestep(*states*, *internals*, *actions*, *terminal*, *reward*)

tf_optimization(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states=None*, *next_internals=None*)

tf_preprocess(*states*, *actions*, *reward*)

tf_q_delta(*q_value*, *next_q_value*, *terminal*, *reward*)
Creates the deltas (or advantage) of the Q values.

Returns A list of deltas per action

tf_q_value(*embedding*, *distr_params*, *action*, *name*)

tf_reference(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*)
Creates the TensorFlow operations for obtaining the reference tensor(s), in case of a comparative loss.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **actions** – Dict of action tensors.
- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **next_states** – Dict of successor state tensors.
- **next_internals** – List of posterior internal state tensors.
- **update** – Boolean tensor indicating whether this call happens during an update.

Returns Reference tensor(s).

tf_regularization_losses(*states*, *internals*, *update*)

class tensorforce.models.**QDemoModel**(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*, *variable_noise*, *states_preprocessing*, *actions_exploration*, *reward_preprocessing*, *update_mode*, *memory*, *optimizer*, *discount*, *network*, *distributions*, *entropy_regularization*, *target_sync_frequency*, *target_update_weight*, *double_q_model*, *huber_loss*, *expert_margin*, *supervised_weight*, *demo_memory_capacity*, *demo_batch_size*)

Bases: *tensorforce.models.q_model.QModel*

Model for deep Q-learning from demonstration. Principal structure similar to double deep Q-networks but uses additional loss terms for demo data.

COMPONENT_DISTRIBUTION = 'distribution'

COMPONENT_NETWORK = 'network'

COMPONENT_TARGET_DISTRIBUTION = 'target_distribution'

COMPONENT_TARGET_NETWORK = 'target_network'

__init__(*states*, *actions*, *scope*, *device*, *saver*, *summarizer*, *execution*, *batching_capacity*, *variable_noise*, *states_preprocessing*, *actions_exploration*, *reward_preprocessing*, *update_mode*, *memory*, *optimizer*, *discount*, *network*, *distributions*, *entropy_regularization*, *target_sync_frequency*, *target_update_weight*, *double_q_model*, *huber_loss*, *expert_margin*, *supervised_weight*, *demo_memory_capacity*, *demo_batch_size*)

act(*states*, *internals*, *deterministic=False*, *independent=False*, *fetch_tensors=None*)
Does a forward pass through the model to retrieve action (outputs) given inputs for state (and internal state, if applicable (e.g. RNNs))

Parameters

- **states** (*dict*) – Dict of state values (each key represents one state space component).
- **internals** (*dict*) – Dict of internal state values (each key represents one internal state component).
- **deterministic** (*bool*) – If True, will not apply exploration after actions are calculated.
- **independent** (*bool*) – If true, action is not followed by observe (and hence not included in updates).

Returns

- Actual action-outputs (batched if state input is a batch).

Return type tuple

as_local_model()

close()

create_act_operations(*states*, *internals*, *deterministic*, *independent*)

create_distributions()

create_observe_operations(*terminal*, *reward*)

create_operations(*states*, *internals*, *actions*, *terminal*, *reward*, *deterministic*, *independent*)

demo_update()
Performs a demonstration update by calling the demo optimization operation. Note that the batch data does not have to be fetched from the demo memory as this is now part of the TensorFlow operation of the demo update.

get_component(*component_name*)
Looks up a component by its name.

Parameters **component_name** – The name of the component to look up.

Returns The component for the provided name or None if there is no such component.

get_components()

get_feed_dict(*states=None*, *internals=None*, *actions=None*, *terminal=None*, *reward=None*, *deterministic=None*, *independent=None*)

get_savable_components()
Returns the list of all of the components this model consists of that can be individually saved and restored. For instance the network or distribution.

Returns List of util.SavableComponent

get_summaries()

get_variables(*include_submodules=False*, *include_nontrainable=False*)
Returns the TensorFlow variables used by the model.

Returns List of variables.

import_demo_experience(*states*, *internals*, *actions*, *terminal*, *reward*)
Stores demonstrations in the demo memory.

import_experience(*states*, *internals*, *actions*, *terminal*, *reward*)
Stores experiences.

initialize(*custom_getter*)

observe(*terminal*, *reward*)
Adds an observation (reward and is-terminal) to the model without updating its trainable variables.

Parameters

- **terminal** (*bool*) – Whether the episode has terminated.
- **reward** (*float*) – The observed reward value.

Returns The value of the model-internal episode counter.

optimizer_arguments(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*)

reset()
Resets the model to its initial state on episode start. This should also reset all preprocessor(s).

Returns Current episode, timestep counter and the shallow-copied list of internal state initialization Tensors.

Return type tuple

restore(*directory=None*, *file=None*)
Restore TensorFlow model. If no checkpoint file is given, the latest checkpoint is restored. If no checkpoint directory is given, the model's default saver directory is used (unless file specifies the entire path).

Parameters

- **directory** – Optional checkpoint directory.
- **file** – Optional checkpoint file, or path if directory not given.

restore_component(*component_name*, *save_path*)
Restores a component's parameters from a save location.

Parameters

- **component_name** – The component to restore.
- **save_path** – The save location.

save(*directory=None*, *append_timestep=True*)
Save TensorFlow model. If no checkpoint directory is given, the model's default saver directory is used. Optionally appends current timestep to prevent overwriting previous checkpoint files. Turn off to be able to load model from the same given path argument as given here.

Parameters

- **directory** – Optional checkpoint directory.
- **append_timestep** – Appends the current timestep to the checkpoint file if true.

Returns Checkpoint path where the model was saved.

save_component(*component_name*, *save_path*)
Saves a component of this model to the designated location.

Parameters

- **component_name** – The component to save.
- **save_path** – The location to save to.

Returns Checkpoint path where the component was saved.

setup()
Sets up the TensorFlow model graph and initializes (and enters) the TensorFlow session.

target_optimizer_arguments()
Returns the target optimizer arguments including the time, the list of variables to optimize, and various functions which the optimizer might require to perform an update step.

Returns Target optimizer arguments as dict.

tf_action_exploration(*action*, *exploration*, *action_spec*)
Applies optional exploration to the action (post-processor for action outputs).

Parameters

- **action** (*tf.Tensor*) – The original output action tensor (to be post-processed).
- **exploration** (Exploration) – The Exploration object to use.
- **action_spec** (*dict*) – Dict specifying the action space.

Returns The post-processed action output tensor.

tf_actions_and_internals(*states*, *internals*, *deterministic*)

tf_combined_loss(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)
Combines Q-loss and demo loss.

tf_demo_loss(*states*, *actions*, *terminal*, *reward*, *internals*, *update*, *reference=None*)
Extends the q-model loss via the dqfd large-margin loss.

tf_demo_optimization(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*)

tf_discounted_cumulative_reward(*terminal*, *reward*, *discount*, *final_reward=0.0*)
Creates the TensorFlow operations for calculating the discounted cumulative rewards for a given sequence of rewards.

Parameters

- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **discount** – Discount factor.

- **final_reward** – Last reward value in the sequence.

Returns Discounted cumulative reward tensor.

tf_import_demo_experience(*states*, *internals*, *actions*, *terminal*, *reward*)
Imports a single experience to memory.

tf_import_experience(*states*, *internals*, *actions*, *terminal*, *reward*)
Imports experiences into the TensorFlow memory structure. Can be used to import off-policy data.

Parameters

- **states** – Dict of state values to import with keys as state names and values as values to set.
- **internals** – Internal values to set, can be fetched from agent via agent.current_internals if no values available.
- **actions** – Dict of action values to import with keys as action names and values as values to set.
- **terminal** – Terminal value(s)
- **reward** – Reward value(s)

tf_initialize()

tf_kl_divergence(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)

tf_loss(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)
Creates the TensorFlow operations for calculating the full loss of a batch.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **actions** – Dict of action tensors.
- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **next_states** – Dict of successor state tensors.
- **next_internals** – List of posterior internal state tensors.
- **update** – Boolean tensor indicating whether this call happens during an update.
- **reference** – Optional reference tensor(s), in case of a comparative loss.

Returns Loss tensor.

tf_loss_per_instance(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*, *reference=None*)

tf_observe_timestep(*states*, *internals*, *actions*, *terminal*, *reward*)

tf_optimization(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states=None*, *next_internals=None*)

tf_preprocess(*states*, *actions*, *reward*)

tf_q_delta(*q_value*, *next_q_value*, *terminal*, *reward*)
Creates the deltas (or advantage) of the Q values.

Returns A list of deltas per action

tf_q_value(*embedding*, *distr_params*, *action*, *name*)

tf_reference(*states*, *internals*, *actions*, *terminal*, *reward*, *next_states*, *next_internals*, *update*)
Creates the TensorFlow operations for obtaining the reference tensor(s), in case of a comparative loss.

Parameters

- **states** – Dict of state tensors.
- **internals** – List of prior internal state tensors.
- **actions** – Dict of action tensors.
- **terminal** – Terminal boolean tensor.
- **reward** – Reward tensor.
- **next_states** – Dict of successor state tensors.
- **next_internals** – List of posterior internal state tensors.
- **update** – Boolean tensor indicating whether this call happens during an update.

Returns Reference tensor(s).

tf_regularization_losses(*states*, *internals*, *update*)

tensorforce.tests package

Submodules

tensorforce.tests.base_agent_test module

class tensorforce.tests.base_agent_test.**BaseAgentTest**
Bases: *tensorforce.tests.base_test.BaseTest*

Base class for tests of fundamental Agent functionality, i.e. various action types and shapes and internal states.

__init__
x.**init**(...) initializes x; see help(type(x)) for signature

agent = None

base_test_pass(*name*, *environment*, *network*, ***kwargs*)
Basic test loop, requires an Agent to achieve a certain performance on an environment.

Parameters

- **name** (*str*) – The name of the test.
- **environment** (Environment) – The Environment object to use for the test.
- **network** (LayerBasedNetwork) – The Network to use for the agent's model.
- **kwargs** (*any*) – Agent arguments.

base_test_run(*name*, *environment*, *network*, ***kwargs*)
Run test, tests whether algorithm can run and update without compilation errors, not whether it passes.

Parameters

- **name** (*str*) – The name of the test.
- **environment** (Environment) – The Environment object to use for the test.

- **network** (LayerBasedNetwork) – The Network to use for the agent's model.
- **kwargs** (*any*) – Agent arguments.

config = None

exclude_bool = False

exclude_bounded = False

exclude_float = False

exclude_int = False

exclude_lstm = False

exclude_multi = False

multi_config = None

pass_threshold = 0.8

pre_run(*agent*, *environment*)
Called before Runner.run.

requires_network = True

test_bool()
Tests the case of one boolean action.

test_bounded_float()
Tests the case of one bounded float action, i.e. with min and max value.

test_float()
Tests the case of one float action.

test_int()
Tests the case of one integer action.

test_lstm()
Tests the case of using internal states via an LSTM layer (for one integer action).

test_multi()
Tests the case of multiple actions of different type and shape.

tensorforce.tests.base_test module

class tensorforce.tests.base_test.**BaseTest**
Bases: object

Base class for tests of Agent functionality.

__init__
x.**init**(...) initializes x; see help(type(x)) for signature

agent = None

base_test_pass(*name*, *environment*, *network*, ***kwargs*)
Basic test loop, requires an Agent to achieve a certain performance on an environment.

Parameters

- **name** (*str*) – The name of the test.
- **environment** (Environment) – The Environment object to use for the test.

- **network** (LayerBasedNetwork) – The Network to use for the agent's model.
- **kwargs** (*any*) – Agent arguments.

base_test_run(*name*, *environment*, *network*, ***kwargs*)
Run test, tests whether algorithm can run and update without compilation errors, not whether it passes.

Parameters

- **name** (*str*) – The name of the test.
- **environment** (Environment) – The Environment object to use for the test.
- **network** (LayerBasedNetwork) – The Network to use for the agent's model.
- **kwargs** (*any*) – Agent arguments.

pass_threshold = 0.8

pre_run(*agent*, *environment*)
Called before Runner.run.

requires_network = True

tensorforce.tests.test_constant_agent module

class tensorforce.tests.test_constant_agent.**TestConstantAgent**(*methodName='runTest'*)
Bases: *tensorforce.tests.base_agent_test.BaseAgentTest*, unittest.case.TestCase

__init__(*methodName='runTest'*)
Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

addCleanup(*function*, **args*, ***kwargs*)
Add a function, with arguments, to be called when the test is completed. Functions added are called on a LIFO basis and are called after tearDown on test failure or success.

Cleanup items are called even if setUp fails (unlike tearDown).

addTypeEqualityFunc(*typeobj*, *function*)
Add a type specific assertEqual style function to compare a type.

This method is for use by TestCase subclasses that need to register their own type equality functions to provide nicer error messages.

Parameters

- **typeobj** – The data type to call this function on when both values are of the same type in assertEqual().
- **function** – The callable taking two arguments and an optional msg= argument that raises self.failureException with a useful error message when the two arguments are not equal.

agent
alias of ConstantAgent

assertAlmostEqual(*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is more than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

If the two objects compare equal then they will automatically compare almost equal.

assertAlmostEquals(*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is more than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

If the two objects compare equal then they will automatically compare almost equal.

assertDictContainsSubset(*expected*, *actual*, *msg=None*)
Checks whether actual is a superset of expected.

assertDictEqual(*d1*, *d2*, *msg=None*)

assertEqual(*first*, *second*, *msg=None*)
Fail if the two objects are unequal as determined by the '==' operator.

assertEquals(*first*, *second*, *msg=None*)
Fail if the two objects are unequal as determined by the '==' operator.

assertFalse(*expr*, *msg=None*)
Check that the expression is false.

assertGreater(*a*, *b*, *msg=None*)
Just like self.assertTrue(a > b), but with a nicer default message.

assertGreaterEqual(*a*, *b*, *msg=None*)
Just like self.assertTrue(a >= b), but with a nicer default message.

assertIn(*member*, *container*, *msg=None*)
Just like self.assertTrue(a in b), but with a nicer default message.

assertIs(*expr1*, *expr2*, *msg=None*)
Just like self.assertTrue(a is b), but with a nicer default message.

assertIsInstance(*obj*, *cls*, *msg=None*)
Same as self.assertTrue(isinstance(obj, cls)), with a nicer default message.

assertIsNone(*obj*, *msg=None*)
Same as self.assertTrue(obj is None), with a nicer default message.

assertIsNot(*expr1*, *expr2*, *msg=None*)
Just like self.assertTrue(a is not b), but with a nicer default message.

assertIsNotNone(*obj*, *msg=None*)
Included for symmetry with assertIsNone.

assertItemsEqual(*expected_seq*, *actual_seq*, *msg=None*)
An unordered sequence specific comparison. It asserts that actual_seq and expected_seq have the same element counts. Equivalent to:

Asserts that each element has the same count in both sequences. .. rubric:: Example

- [0, 1, 1] and [1, 0, 1] compare equal.
- [0, 0, 1] and [0, 1] compare unequal.

assertLess(*a*, *b*, *msg=None*)
Just like self.assertTrue(a < b), but with a nicer default message.

assertLessEqual(*a*, *b*, *msg=None*)
Just like self.assertTrue(a <= b), but with a nicer default message.

assertListEqual(*list1*, *list2*, *msg=None*)
A list-specific equality assertion.

Parameters

- **list1** – The first list to compare.
- **list2** – The second list to compare.
- **msg** – Optional message to use on failure instead of a list of differences.

assertMultiLineEqual(*first*, *second*, *msg=None*)
Assert that two multi-line strings are equal.

assertNotAlmostEqual(*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is less than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

Objects that are equal automatically fail.

assertNotAlmostEquals(*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is less than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

Objects that are equal automatically fail.

assertNotEqual(*first*, *second*, *msg=None*)
Fail if the two objects are equal as determined by the '!=' operator.

assertNotEquals(*first*, *second*, *msg=None*)
Fail if the two objects are equal as determined by the '!=' operator.

assertNotIn(*member*, *container*, *msg=None*)
Just like self.assertTrue(a not in b), but with a nicer default message.

assertNotIsInstance(*obj*, *cls*, *msg=None*)
Included for symmetry with assertIsInstance.

assertNotRegexpMatches(*text*, *unexpected_regexp*, *msg=None*)
Fail the test if the text matches the regular expression.

assertRaises(*excClass*, *callableObj=None*, **args*, ***kwargs*)
Fail unless an exception of class excClass is raised by callableObj when invoked with arguments args and keyword arguments kwargs. If a different type of exception is raised, it will not be caught, and the test case will be deemed to have suffered an error, exactly as for an unexpected exception.

If called with callableObj omitted or None, will return a context object used like this:

The context manager keeps a reference to the exception as the 'exception' attribute. This allows you to inspect the exception after the assertion:

assertRaisesRegexp(*expected_exception*, *expected_regexp*, *callable_obj=None*, **args*, ***kwargs*)
Asserts that the message in a raised exception matches a regexp.

Parameters

- **expected_exception** – Exception class expected to be raised.
- **expected_regexp** – Regexp (re pattern object or string) expected to be found in error message.
- **callable_obj** – Function to be called.
- **args** – Extra args.
- **kwargs** – Extra kwargs.

assertRegexpMatches(*text*, *expected_regexp*, *msg=None*)
Fail the test unless the text matches the regular expression.

assertSequenceEqual(*seq1*, *seq2*, *msg=None*, *seq_type=None*)
An equality assertion for ordered sequences (like lists and tuples).

For the purposes of this function, a valid ordered sequence type is one which can be indexed, has a length, and has an equality operator.

Parameters

- **seq1** – The first sequence to compare.
- **seq2** – The second sequence to compare.
- **seq_type** – The expected datatype of the sequences, or None if no datatype should be enforced.
- **msg** – Optional message to use on failure instead of a list of differences.

assertSetEqual(*set1*, *set2*, *msg=None*)
A set-specific equality assertion.

Parameters

- **set1** – The first set to compare.
- **set2** – The second set to compare.
- **msg** – Optional message to use on failure instead of a list of differences.

assertSetEqual uses ducktyping to support different types of sets, and is optimized for sets specifically (parameters must support a difference method).

assertTrue(*expr*, *msg=None*)
Check that the expression is true.

assertTupleEqual(*tuple1*, *tuple2*, *msg=None*)
A tuple-specific equality assertion.

Parameters

- **tuple1** – The first tuple to compare.
- **tuple2** – The second tuple to compare.
- **msg** – Optional message to use on failure instead of a list of differences.

assert_(*expr*, *msg=None*)
Check that the expression is true.

base_test_pass(*name*, *environment*, *network*, ***kwargs*)
Basic test loop, requires an Agent to achieve a certain performance on an environment.

Parameters

- **name** (*str*) – The name of the test.
- **environment** (Environment) – The Environment object to use for the test.
- **network** (LayerBasedNetwork) – The Network to use for the agent's model.
- **kwargs** (*any*) – Agent arguments.

base_test_run(*name*, *environment*, *network*, ***kwargs*)
Run test, tests whether algorithm can run and update without compilation errors, not whether it passes.

Parameters

- **name** (*str*) – The name of the test.
- **environment** (Environment) – The Environment object to use for the test.
- **network** (LayerBasedNetwork) – The Network to use for the agent's model.
- **kwargs** (*any*) – Agent arguments.

config = {'action_values': {'action': 1.0}}

countTestCases()

debug()
Run the test without collecting errors in a TestResult

defaultTestResult()

doCleanups()
Execute all cleanup functions. Normally called for you after tearDown.

exclude_bool = True

exclude_bounded = False

exclude_float = False

exclude_int = True

exclude_lstm = True

exclude_multi = True

fail(*msg=None*)
Fail immediately, with the given message.

failIf(**args*, ***kwargs*)

failIfAlmostEqual(**args*, ***kwargs*)

failIfEqual(**args*, ***kwargs*)

failUnless(**args*, ***kwargs*)

failUnlessAlmostEqual(**args*, ***kwargs*)

failUnlessEqual(**args*, ***kwargs*)

failUnlessRaises(**args*, ***kwargs*)

failureException
alias of AssertionError

id()

longMessage = False

maxDiff = 640

multi_config = None

pass_threshold = 0.8

pre_run(*agent*, *environment*)
Called before `Runner.run`.

requires_network = False

run(*result=None*)

setUp()
Hook method for setting up the test fixture before exercising it.

setUpClass()
Hook method for setting up class fixture before running tests in the class.

shortDescription()
Returns a one-line description of the test, or None if no description has been provided.

The default implementation of this method returns the first line of the specified test method's docstring.

skipTest(*reason*)
Skip this test.

tearDown()
Hook method for deconstructing the test fixture after testing it.

tearDownClass()
Hook method for deconstructing the class fixture after running all tests in the class.

test_bool()
Tests the case of one boolean action.

test_bounded_float()
Tests the case of one bounded float action, i.e. with min and max value.

test_float()
Tests the case of one float action.

test_int()
Tests the case of one integer action.

test_lstm()
Tests the case of using internal states via an LSTM layer (for one integer action).

test_multi()
Tests the case of multiple actions of different type and shape.

tensorforce.tests.test_ddqn_agent module

tensorforce.tests.test_dqfd_agent module

class tensorforce.tests.test_dqfd_agent.**TestDQFDAgent**(*methodName='runTest'*)
Bases: *tensorforce.tests.base_agent_test.BaseAgentTest*, unittest.case.TestCase

__init__(*methodName='runTest'*)
Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

addCleanup (*function*, **args*, ***kwargs*)
Add a function, with arguments, to be called when the test is completed. Functions added are called on a LIFO basis and are called after tearDown on test failure or success.

Cleanup items are called even if setUp fails (unlike tearDown).

addTypeEqualityFunc (*typeobj*, *function*)
Add a type specific assertEqual style function to compare a type.

This method is for use by TestCase subclasses that need to register their own type equality functions to provide nicer error messages.

Parameters

- **typeobj** – The data type to call this function on when both values are of the same type in assertEqual().
- **function** – The callable taking two arguments and an optional msg= argument that raises self.failureException with a useful error message when the two arguments are not equal.

agent
alias of `DQFDAgent`

assertAlmostEqual (*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is more than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

If the two objects compare equal then they will automatically compare almost equal.

assertAlmostEquals (*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is more than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

If the two objects compare equal then they will automatically compare almost equal.

assertDictContainsSubset (*expected*, *actual*, *msg=None*)
Checks whether actual is a superset of expected.

assertDictEqual (*d1*, *d2*, *msg=None*)

assertEqual (*first*, *second*, *msg=None*)
Fail if the two objects are unequal as determined by the '==' operator.

assertEquals (*first*, *second*, *msg=None*)
Fail if the two objects are unequal as determined by the '==' operator.

assertFalse (*expr*, *msg=None*)
Check that the expression is false.

assertGreater (*a*, *b*, *msg=None*)
Just like self.assertTrue(a > b), but with a nicer default message.

assertGreaterEqual (*a*, *b*, *msg=None*)
Just like self.assertTrue(a >= b), but with a nicer default message.

assertIn(*member*, *container*, *msg=None*)
Just like self.assertTrue(a in b), but with a nicer default message.

assertIs(*expr1*, *expr2*, *msg=None*)
Just like self.assertTrue(a is b), but with a nicer default message.

assertIsInstance(*obj*, *cls*, *msg=None*)
Same as self.assertTrue(isinstance(obj, cls)), with a nicer default message.

assertIsNone(*obj*, *msg=None*)
Same as self.assertTrue(obj is None), with a nicer default message.

assertIsNot(*expr1*, *expr2*, *msg=None*)
Just like self.assertTrue(a is not b), but with a nicer default message.

assertIsNotNone(*obj*, *msg=None*)
Included for symmetry with assertIsNone.

assertItemsEqual(*expected_seq*, *actual_seq*, *msg=None*)
An unordered sequence specific comparison. It asserts that actual_seq and expected_seq have the same element counts. Equivalent to:

Asserts that each element has the same count in both sequences. .. rubric:: Example

- [0, 1, 1] and [1, 0, 1] compare equal.
- [0, 0, 1] and [0, 1] compare unequal.

assertLess(*a*, *b*, *msg=None*)
Just like self.assertTrue(a < b), but with a nicer default message.

assertLessEqual(*a*, *b*, *msg=None*)
Just like self.assertTrue(a <= b), but with a nicer default message.

assertListEqual(*list1*, *list2*, *msg=None*)
A list-specific equality assertion.

Parameters

- **list1** – The first list to compare.
- **list2** – The second list to compare.
- **msg** – Optional message to use on failure instead of a list of differences.

assertMultiLineEqual(*first*, *second*, *msg=None*)
Assert that two multi-line strings are equal.

assertNotAlmostEqual(*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is less than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

Objects that are equal automatically fail.

assertNotAlmostEquals(*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is less than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

Objects that are equal automatically fail.

assertNotEqual(*first*, *second*, *msg=None*)
Fail if the two objects are equal as determined by the '!=' operator.

assertNotEquals(*first*, *second*, *msg=None*)
Fail if the two objects are equal as determined by the '!=' operator.

assertNotIn(*member*, *container*, *msg=None*)
Just like self.assertTrue(a not in b), but with a nicer default message.

assertNotIsInstance(*obj*, *cls*, *msg=None*)
Included for symmetry with assertIsInstance.

assertNotRegexpMatches(*text*, *unexpected_regexp*, *msg=None*)
Fail the test if the text matches the regular expression.

assertRaises(*excClass*, *callableObj=None*, **args*, ***kwargs*)
Fail unless an exception of class excClass is raised by callableObj when invoked with arguments args and keyword arguments kwargs. If a different type of exception is raised, it will not be caught, and the test case will be deemed to have suffered an error, exactly as for an unexpected exception.

If called with callableObj omitted or None, will return a context object used like this:

The context manager keeps a reference to the exception as the 'exception' attribute. This allows you to inspect the exception after the assertion:

assertRaisesRegexp(*expected_exception*, *expected_regexp*, *callable_obj=None*, **args*, ***kwargs*)
Asserts that the message in a raised exception matches a regexp.

Parameters

- **expected_exception** – Exception class expected to be raised.
- **expected_regexp** – Regexp (re pattern object or string) expected to be found in error message.
- **callable_obj** – Function to be called.
- **args** – Extra args.
- **kwargs** – Extra kwargs.

assertRegexpMatches(*text*, *expected_regexp*, *msg=None*)
Fail the test unless the text matches the regular expression.

assertSequenceEqual(*seq1*, *seq2*, *msg=None*, *seq_type=None*)
An equality assertion for ordered sequences (like lists and tuples).

For the purposes of this function, a valid ordered sequence type is one which can be indexed, has a length, and has an equality operator.

Parameters

- **seq1** – The first sequence to compare.
- **seq2** – The second sequence to compare.
- **seq_type** – The expected datatype of the sequences, or None if no datatype should be enforced.
- **msg** – Optional message to use on failure instead of a list of differences.

assertSetEqual(*set1*, *set2*, *msg=None*)
A set-specific equality assertion.

Parameters

- **set1** – The first set to compare.
- **set2** – The second set to compare.
- **msg** – Optional message to use on failure instead of a list of differences.

assertSetEqual uses ducktyping to support different types of sets, and is optimized for sets specifically (parameters must support a difference method).

assertTrue(*expr*, *msg=None*)
Check that the expression is true.

assertTupleEqual(*tuple1*, *tuple2*, *msg=None*)
A tuple-specific equality assertion.

Parameters

- **tuple1** – The first tuple to compare.
- **tuple2** – The second tuple to compare.
- **msg** – Optional message to use on failure instead of a list of differences.

assert_(*expr*, *msg=None*)
Check that the expression is true.

base_test_pass(*name*, *environment*, *network*, ***kwargs*)
Basic test loop, requires an Agent to achieve a certain performance on an environment.

Parameters

- **name** (*str*) – The name of the test.
- **environment** (Environment) – The Environment object to use for the test.
- **network** (LayerBasedNetwork) – The Network to use for the agent's model.
- **kwargs** (*any*) – Agent arguments.

base_test_run(*name*, *environment*, *network*, ***kwargs*)
Run test, tests whether algorithm can run and update without compilation errors, not whether it passes.

Parameters

- **name** (*str*) – The name of the test.
- **environment** (Environment) – The Environment object to use for the test.
- **network** (LayerBasedNetwork) – The Network to use for the agent's model.
- **kwargs** (*any*) – Agent arguments.

config = {'update_mode': {'frequency': 4, 'batch_size': 8, 'unit': 'timesteps'}, '

countTestCases()

debug()
Run the test without collecting errors in a TestResult

defaultTestResult()

doCleanups()
Execute all cleanup functions. Normally called for you after tearDown.

exclude_bool = False

exclude_bounded = True

exclude_float = True

exclude_int = False

exclude_lstm = False

exclude_multi = False

fail(*msg=None*)
Fail immediately, with the given message.

failIf(**args*, ***kwargs*)

failIfAlmostEqual(**args*, ***kwargs*)

failIfEqual(**args*, ***kwargs*)

failUnless(**args*, ***kwargs*)

failUnlessAlmostEqual(**args*, ***kwargs*)

failUnlessEqual(**args*, ***kwargs*)

failUnlessRaises(**args*, ***kwargs*)

failureException
alias of `AssertionError`

id()

longMessage = False

maxDiff = 640

multi_config = None

pass_threshold = 0.8

pre_run(*agent*, *environment*)

requires_network = True

run(*result=None*)

setUp()
Hook method for setting up the test fixture before exercising it.

setUpClass()
Hook method for setting up class fixture before running tests in the class.

shortDescription()
Returns a one-line description of the test, or None if no description has been provided.

The default implementation of this method returns the first line of the specified test method's docstring.

skipTest(*reason*)
Skip this test.

tearDown()
Hook method for deconstructing the test fixture after testing it.

tearDownClass()
Hook method for deconstructing the class fixture after running all tests in the class.

test_bool()
Tests the case of one boolean action.

test_bounded_float()
Tests the case of one bounded float action, i.e. with min and max value.

test_float()
Tests the case of one float action.

test_int()
Tests the case of one integer action.

test_lstm()
Tests the case of using internal states via an LSTM layer (for one integer action).

test_multi()
Tests the case of multiple actions of different type and shape.

tensorforce.tests.test_dqn_agent module

class tensorforce.tests.test_dqn_agent.**TestDQNAgent**(*methodName='runTest'*)
Bases: *tensorforce.tests.base_agent_test.BaseAgentTest*, unittest.case.TestCase

__init__(*methodName='runTest'*)
Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

addCleanup(*function*, **args*, ***kwargs*)
Add a function, with arguments, to be called when the test is completed. Functions added are called on a LIFO basis and are called after tearDown on test failure or success.

Cleanup items are called even if setUp fails (unlike tearDown).

addTypeEqualityFunc(*typeobj*, *function*)
Add a type specific assertEqual style function to compare a type.

This method is for use by TestCase subclasses that need to register their own type equality functions to provide nicer error messages.

Parameters

- **typeobj** – The data type to call this function on when both values are of the same type in assertEqual().
- **function** – The callable taking two arguments and an optional msg= argument that raises self.failureException with a useful error message when the two arguments are not equal.

agent
alias of DQNAgent

assertAlmostEqual(*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is more than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

If the two objects compare equal then they will automatically compare almost equal.

assertAlmostEquals(*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal

places (default 7) and comparing to zero, or by comparing that the between the two objects is more than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

If the two objects compare equal then they will automatically compare almost equal.

assertDictContainsSubset (*expected*, *actual*, *msg=None*)
Checks whether actual is a superset of expected.

assertDictEqual (*d1*, *d2*, *msg=None*)

assertEqual (*first*, *second*, *msg=None*)
Fail if the two objects are unequal as determined by the '==' operator.

assertEquals (*first*, *second*, *msg=None*)
Fail if the two objects are unequal as determined by the '==' operator.

assertFalse (*expr*, *msg=None*)
Check that the expression is false.

assertGreater (*a*, *b*, *msg=None*)
Just like self.assertTrue(a > b), but with a nicer default message.

assertGreaterEqual (*a*, *b*, *msg=None*)
Just like self.assertTrue(a >= b), but with a nicer default message.

assertIn (*member*, *container*, *msg=None*)
Just like self.assertTrue(a in b), but with a nicer default message.

assertIs (*expr1*, *expr2*, *msg=None*)
Just like self.assertTrue(a is b), but with a nicer default message.

assertIsInstance (*obj*, *cls*, *msg=None*)
Same as self.assertTrue(isinstance(obj, cls)), with a nicer default message.

assertIsNone (*obj*, *msg=None*)
Same as self.assertTrue(obj is None), with a nicer default message.

assertIsNot (*expr1*, *expr2*, *msg=None*)
Just like self.assertTrue(a is not b), but with a nicer default message.

assertIsNotNone (*obj*, *msg=None*)
Included for symmetry with assertIsNone.

assertItemsEqual (*expected_seq*, *actual_seq*, *msg=None*)
An unordered sequence specific comparison. It asserts that actual_seq and expected_seq have the same element counts. Equivalent to:

Asserts that each element has the same count in both sequences. .. rubric:: Example

- [0, 1, 1] and [1, 0, 1] compare equal.
- [0, 0, 1] and [0, 1] compare unequal.

assertLess (*a*, *b*, *msg=None*)
Just like self.assertTrue(a < b), but with a nicer default message.

assertLessEqual (*a*, *b*, *msg=None*)
Just like self.assertTrue(a <= b), but with a nicer default message.

assertListEqual (*list1*, *list2*, *msg=None*)
A list-specific equality assertion.

Parameters

- **list1** – The first list to compare.
- **list2** – The second list to compare.
- **msg** – Optional message to use on failure instead of a list of differences.

assertMultiLineEqual (*first*, *second*, *msg=None*)
Assert that two multi-line strings are equal.

assertNotAlmostEqual (*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is less than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

Objects that are equal automatically fail.

assertNotAlmostEquals (*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is less than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

Objects that are equal automatically fail.

assertNotEqual (*first*, *second*, *msg=None*)
Fail if the two objects are equal as determined by the '!=' operator.

assertNotEquals (*first*, *second*, *msg=None*)
Fail if the two objects are equal as determined by the '!=' operator.

assertNotIn (*member*, *container*, *msg=None*)
Just like self.assertTrue(a not in b), but with a nicer default message.

assertNotIsInstance (*obj*, *cls*, *msg=None*)
Included for symmetry with assertIsInstance.

assertNotRegexpMatches (*text*, *unexpected_regexp*, *msg=None*)
Fail the test if the text matches the regular expression.

assertRaises (*excClass*, *callableObj=None*, **args*, ***kwargs*)
Fail unless an exception of class excClass is raised by callableObj when invoked with arguments args and keyword arguments kwargs. If a different type of exception is raised, it will not be caught, and the test case will be deemed to have suffered an error, exactly as for an unexpected exception.

If called with callableObj omitted or None, will return a context object used like this:

The context manager keeps a reference to the exception as the 'exception' attribute. This allows you to inspect the exception after the assertion:

assertRaisesRegexp (*expected_exception*, *expected_regexp*, *callable_obj=None*, **args*, ***kwargs*)
Asserts that the message in a raised exception matches a regexp.

Parameters

- **expected_exception** – Exception class expected to be raised.
- **expected_regexp** – Regexp (re pattern object or string) expected to be found in error message.

- **callable_obj** – Function to be called.
- **args** – Extra args.
- **kwargs** – Extra kwargs.

assertRegexpMatches(*text*, *expected_regexp*, *msg=None*)
Fail the test unless the text matches the regular expression.

assertSequenceEqual(*seq1*, *seq2*, *msg=None*, *seq_type=None*)
An equality assertion for ordered sequences (like lists and tuples).

For the purposes of this function, a valid ordered sequence type is one which can be indexed, has a length, and has an equality operator.

Parameters

- **seq1** – The first sequence to compare.
- **seq2** – The second sequence to compare.
- **seq_type** – The expected datatype of the sequences, or None if no datatype should be enforced.
- **msg** – Optional message to use on failure instead of a list of differences.

assertSetEqual(*set1*, *set2*, *msg=None*)
A set-specific equality assertion.

Parameters

- **set1** – The first set to compare.
- **set2** – The second set to compare.
- **msg** – Optional message to use on failure instead of a list of differences.

assertSetEqual uses ducktyping to support different types of sets, and is optimized for sets specifically (parameters must support a difference method).

assertTrue(*expr*, *msg=None*)
Check that the expression is true.

assertTupleEqual(*tuple1*, *tuple2*, *msg=None*)
A tuple-specific equality assertion.

Parameters

- **tuple1** – The first tuple to compare.
- **tuple2** – The second tuple to compare.
- **msg** – Optional message to use on failure instead of a list of differences.

assert_(*expr*, *msg=None*)
Check that the expression is true.

base_test_pass(*name*, *environment*, *network*, ***kwargs*)
Basic test loop, requires an Agent to achieve a certain performance on an environment.

Parameters

- **name** (*str*) – The name of the test.
- **environment** (Environment) – The Environment object to use for the test.
- **network** (LayerBasedNetwork) – The Network to use for the agent's model.

- **kwargs** (*any*) – Agent arguments.

base_test_run(*name*, *environment*, *network*, ***kwargs*)
Run test, tests whether algorithm can run and update without compilation errors, not whether it passes.

Parameters

- **name** (*str*) – The name of the test.
- **environment** (Environment) – The Environment object to use for the test.
- **network** (LayerBasedNetwork) – The Network to use for the agent's model.
- **kwargs** (*any*) – Agent arguments.

config = {'update_mode': {'frequency': 8, 'batch_size': 8, 'unit': 'timesteps'}, '

countTestCases()

debug()
Run the test without collecting errors in a TestResult

defaultTestResult()

doCleanups()
Execute all cleanup functions. Normally called for you after tearDown.

exclude_bool = False

exclude_bounded = True

exclude_float = True

exclude_int = False

exclude_lstm = False

exclude_multi = False

fail(*msg=None*)
Fail immediately, with the given message.

failIf(**args*, ***kwargs*)

failIfAlmostEqual(**args*, ***kwargs*)

failIfEqual(**args*, ***kwargs*)

failUnless(**args*, ***kwargs*)

failUnlessAlmostEqual(**args*, ***kwargs*)

failUnlessEqual(**args*, ***kwargs*)

failUnlessRaises(**args*, ***kwargs*)

failureException
alias of AssertionError

id()

longMessage = False

maxDiff = 640

multi_config = None

pass_threshold = 0.8

pre_run(*agent*, *environment*)
Called before `Runner.run`.

requires_network = True

run(*result=None*)

setUp()
Hook method for setting up the test fixture before exercising it.

setUpClass()
Hook method for setting up class fixture before running tests in the class.

shortDescription()
Returns a one-line description of the test, or None if no description has been provided.

The default implementation of this method returns the first line of the specified test method's docstring.

skipTest(*reason*)
Skip this test.

tearDown()
Hook method for deconstructing the test fixture after testing it.

tearDownClass()
Hook method for deconstructing the class fixture after running all tests in the class.

test_bool()
Tests the case of one boolean action.

test_bounded_float()
Tests the case of one bounded float action, i.e. with min and max value.

test_float()
Tests the case of one float action.

test_int()
Tests the case of one integer action.

test_lstm()
Tests the case of using internal states via an LSTM layer (for one integer action).

test_multi()
Tests the case of multiple actions of different type and shape.

tensorforce.tests.test_dqn_memories module

tensorforce.tests.test_dqn_nstep_agent module

class tensorforce.tests.test_dqn_nstep_agent.**TestDQNNstepAgent**(*methodName='runTest'*)
Bases: *tensorforce.tests.base_agent_test.BaseAgentTest*, unittest.case.TestCase

__init__(*methodName='runTest'*)
Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

addCleanup(*function*, **args*, ***kwargs*)
Add a function, with arguments, to be called when the test is completed. Functions added are called on a LIFO basis and are called after tearDown on test failure or success.

Cleanup items are called even if setUp fails (unlike tearDown).

addTypeEqualityFunc(*typeobj*, *function*)
Add a type specific assertEqual style function to compare a type.

This method is for use by TestCase subclasses that need to register their own type equality functions to provide nicer error messages.

Parameters

- **typeobj** – The data type to call this function on when both values are of the same type in assertEqual().
- **function** – The callable taking two arguments and an optional msg= argument that raises self.failureException with a useful error message when the two arguments are not equal.

agent
alias of `DQNNstepAgent`

assertAlmostEqual(*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is more than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

If the two objects compare equal then they will automatically compare almost equal.

assertAlmostEquals(*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is more than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

If the two objects compare equal then they will automatically compare almost equal.

assertDictContainsSubset(*expected*, *actual*, *msg=None*)
Checks whether actual is a superset of expected.

assertDictEqual(*d1*, *d2*, *msg=None*)

assertEqual(*first*, *second*, *msg=None*)
Fail if the two objects are unequal as determined by the '==' operator.

assertEquals(*first*, *second*, *msg=None*)
Fail if the two objects are unequal as determined by the '==' operator.

assertFalse(*expr*, *msg=None*)
Check that the expression is false.

assertGreater(*a*, *b*, *msg=None*)
Just like self.assertTrue(a > b), but with a nicer default message.

assertGreaterEqual(*a*, *b*, *msg=None*)
Just like self.assertTrue(a >= b), but with a nicer default message.

assertIn(*member*, *container*, *msg=None*)
Just like self.assertTrue(a in b), but with a nicer default message.

assertIs(*expr1*, *expr2*, *msg=None*)
Just like self.assertTrue(a is b), but with a nicer default message.

assertIsInstance(*obj*, *cls*, *msg=None*)
Same as self.assertTrue(isinstance(obj, cls)), with a nicer default message.

assertIsNone(*obj*, *msg=None*)
Same as self.assertTrue(obj is None), with a nicer default message.

assertIsNot(*expr1*, *expr2*, *msg=None*)
Just like self.assertTrue(a is not b), but with a nicer default message.

assertIsNotNone(*obj*, *msg=None*)
Included for symmetry with assertIsNone.

assertItemsEqual(*expected_seq*, *actual_seq*, *msg=None*)
An unordered sequence specific comparison. It asserts that actual_seq and expected_seq have the same element counts. Equivalent to:

Asserts that each element has the same count in both sequences. .. rubric:: Example

- [0, 1, 1] and [1, 0, 1] compare equal.
- [0, 0, 1] and [0, 1] compare unequal.

assertLess(*a*, *b*, *msg=None*)
Just like self.assertTrue(a < b), but with a nicer default message.

assertLessEqual(*a*, *b*, *msg=None*)
Just like self.assertTrue(a <= b), but with a nicer default message.

assertListEqual(*list1*, *list2*, *msg=None*)
A list-specific equality assertion.

Parameters

- **list1** – The first list to compare.
- **list2** – The second list to compare.
- **msg** – Optional message to use on failure instead of a list of differences.

assertMultiLineEqual(*first*, *second*, *msg=None*)
Assert that two multi-line strings are equal.

assertNotAlmostEqual(*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is less than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

Objects that are equal automatically fail.

assertNotAlmostEquals(*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is less than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

Objects that are equal automatically fail.

assertNotEqual(*first*, *second*, *msg=None*)
Fail if the two objects are equal as determined by the '!=' operator.

assertNotEquals(*first*, *second*, *msg=None*)
Fail if the two objects are equal as determined by the '!=' operator.

assertNotIn(*member*, *container*, *msg=None*)
Just like self.assertTrue(a not in b), but with a nicer default message.

assertNotIsInstance(*obj*, *cls*, *msg=None*)
Included for symmetry with assertIsInstance.

assertNotRegexpMatches(*text*, *unexpected_regexp*, *msg=None*)
Fail the test if the text matches the regular expression.

assertRaises(*excClass*, *callableObj=None*, **args*, ***kwargs*)
Fail unless an exception of class excClass is raised by callableObj when invoked with arguments args and keyword arguments kwargs. If a different type of exception is raised, it will not be caught, and the test case will be deemed to have suffered an error, exactly as for an unexpected exception.

If called with callableObj omitted or None, will return a context object used like this:

The context manager keeps a reference to the exception as the 'exception' attribute. This allows you to inspect the exception after the assertion:

assertRaisesRegexp(*expected_exception*, *expected_regexp*, *callable_obj=None*, **args*, ***kwargs*)
Asserts that the message in a raised exception matches a regexp.

Parameters

- **expected_exception** – Exception class expected to be raised.
- **expected_regexp** – Regexp (re pattern object or string) expected to be found in error message.
- **callable_obj** – Function to be called.
- **args** – Extra args.
- **kwargs** – Extra kwargs.

assertRegexpMatches(*text*, *expected_regexp*, *msg=None*)
Fail the test unless the text matches the regular expression.

assertSequenceEqual(*seq1*, *seq2*, *msg=None*, *seq_type=None*)
An equality assertion for ordered sequences (like lists and tuples).

For the purposes of this function, a valid ordered sequence type is one which can be indexed, has a length, and has an equality operator.

Parameters

- **seq1** – The first sequence to compare.
- **seq2** – The second sequence to compare.
- **seq_type** – The expected datatype of the sequences, or None if no datatype should be enforced.
- **msg** – Optional message to use on failure instead of a list of differences.

assertSetEqual(*set1*, *set2*, *msg=None*)
A set-specific equality assertion.

Parameters

- **set1** – The first set to compare.
- **set2** – The second set to compare.

- **msg** – Optional message to use on failure instead of a list of differences.

assertSetEqual uses ducktyping to support different types of sets, and is optimized for sets specifically (parameters must support a difference method).

assertTrue(*expr*, *msg=None*)
Check that the expression is true.

assertTupleEqual(*tuple1*, *tuple2*, *msg=None*)
A tuple-specific equality assertion.

Parameters

- **tuple1** – The first tuple to compare.
- **tuple2** – The second tuple to compare.
- **msg** – Optional message to use on failure instead of a list of differences.

assert_(*expr*, *msg=None*)
Check that the expression is true.

base_test_pass(*name*, *environment*, *network*, ***kwargs*)
Basic test loop, requires an Agent to achieve a certain performance on an environment.

Parameters

- **name** (*str*) – The name of the test.
- **environment** (Environment) – The Environment object to use for the test.
- **network** (LayerBasedNetwork) – The Network to use for the agent's model.
- **kwargs** (*any*) – Agent arguments.

base_test_run(*name*, *environment*, *network*, ***kwargs*)
Run test, tests whether algorithm can run and update without compilation errors, not whether it passes.

Parameters

- **name** (*str*) – The name of the test.
- **environment** (Environment) – The Environment object to use for the test.
- **network** (LayerBasedNetwork) – The Network to use for the agent's model.
- **kwargs** (*any*) – Agent arguments.

config = {'update_mode': {'frequency': 4, 'batch_size': 4, 'unit': 'episodes'}, 'o

countTestCases()

debug()
Run the test without collecting errors in a TestResult

defaultTestResult()

doCleanups()
Execute all cleanup functions. Normally called for you after tearDown.

exclude_bool = False

exclude_bounded = True

exclude_float = True

exclude_int = False

exclude_lstm = False

exclude_multi = True

fail(*msg=None*)
Fail immediately, with the given message.

failIf(**args*, ***kwargs*)

failIfAlmostEqual(**args*, ***kwargs*)

failIfEqual(**args*, ***kwargs*)

failUnless(**args*, ***kwargs*)

failUnlessAlmostEqual(**args*, ***kwargs*)

failUnlessEqual(**args*, ***kwargs*)

failUnlessRaises(**args*, ***kwargs*)

failureException
alias of `AssertionError`

id()

longMessage = False

maxDiff = 640

multi_config = None

pass_threshold = 0.8

pre_run(*agent*, *environment*)
Called before `Runner.run`.

requires_network = True

run(*result=None*)

setUp()
Hook method for setting up the test fixture before exercising it.

setUpClass()
Hook method for setting up class fixture before running tests in the class.

shortDescription()
Returns a one-line description of the test, or None if no description has been provided.

The default implementation of this method returns the first line of the specified test method's docstring.

skipTest(*reason*)
Skip this test.

tearDown()
Hook method for deconstructing the test fixture after testing it.

tearDownClass()
Hook method for deconstructing the class fixture after running all tests in the class.

test_bool()
Tests the case of one boolean action.

test_bounded_float()
Tests the case of one bounded float action, i.e. with min and max value.

test_float()
Tests the case of one float action.

test_int()
Tests the case of one integer action.

test_lstm()
Tests the case of using internal states via an LSTM layer (for one integer action).

test_multi()
Tests the case of multiple actions of different type and shape.

tensorforce.tests.test_naf_agent module

class tensorforce.tests.test_naf_agent.**TestNAFAgent**(*methodName='runTest'*)
Bases: *tensorforce.tests.base_agent_test.BaseAgentTest*, unittest.case.TestCase

__init__(*methodName='runTest'*)
Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

addCleanup(*function*, **args*, ***kwargs*)
Add a function, with arguments, to be called when the test is completed. Functions added are called on a LIFO basis and are called after tearDown on test failure or success.

Cleanup items are called even if setUp fails (unlike tearDown).

addTypeEqualityFunc(*typeobj*, *function*)
Add a type specific assertEqual style function to compare a type.

This method is for use by TestCase subclasses that need to register their own type equality functions to provide nicer error messages.

Parameters

- **typeobj** – The data type to call this function on when both values are of the same type in assertEqual().
- **function** – The callable taking two arguments and an optional msg= argument that raises self.failureException with a useful error message when the two arguments are not equal.

agent
alias of NAFAgent

assertAlmostEqual(*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is more than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

If the two objects compare equal then they will automatically compare almost equal.

assertAlmostEquals(*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is more than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

If the two objects compare equal then they will automatically compare almost equal.

assertDictContainsSubset(*expected*, *actual*, *msg=None*)
Checks whether actual is a superset of expected.

assertDictEqual(*d1*, *d2*, *msg=None*)

assertEqual(*first*, *second*, *msg=None*)
Fail if the two objects are unequal as determined by the '==' operator.

assertEquals(*first*, *second*, *msg=None*)
Fail if the two objects are unequal as determined by the '==' operator.

assertFalse(*expr*, *msg=None*)
Check that the expression is false.

assertGreater(*a*, *b*, *msg=None*)
Just like self.assertTrue(a > b), but with a nicer default message.

assertGreaterEqual(*a*, *b*, *msg=None*)
Just like self.assertTrue(a >= b), but with a nicer default message.

assertIn(*member*, *container*, *msg=None*)
Just like self.assertTrue(a in b), but with a nicer default message.

assertIs(*expr1*, *expr2*, *msg=None*)
Just like self.assertTrue(a is b), but with a nicer default message.

assertIsInstance(*obj*, *cls*, *msg=None*)
Same as self.assertTrue(isinstance(obj, cls)), with a nicer default message.

assertIsNone(*obj*, *msg=None*)
Same as self.assertTrue(obj is None), with a nicer default message.

assertIsNot(*expr1*, *expr2*, *msg=None*)
Just like self.assertTrue(a is not b), but with a nicer default message.

assertIsNotNone(*obj*, *msg=None*)
Included for symmetry with assertIsNone.

assertItemsEqual(*expected_seq*, *actual_seq*, *msg=None*)
An unordered sequence specific comparison. It asserts that actual_seq and expected_seq have the same element counts. Equivalent to:

Asserts that each element has the same count in both sequences. .. rubric:: Example

- [0, 1, 1] and [1, 0, 1] compare equal.
- [0, 0, 1] and [0, 1] compare unequal.

assertLess(*a*, *b*, *msg=None*)
Just like self.assertTrue(a < b), but with a nicer default message.

assertLessEqual(*a*, *b*, *msg=None*)
Just like self.assertTrue(a <= b), but with a nicer default message.

assertListEqual(*list1*, *list2*, *msg=None*)
A list-specific equality assertion.

Parameters

- **list1** – The first list to compare.
- **list2** – The second list to compare.
- **msg** – Optional message to use on failure instead of a list of differences.

assertMultiLineEqual (*first*, *second*, *msg=None*)
Assert that two multi-line strings are equal.

assertNotAlmostEqual (*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is less than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

Objects that are equal automatically fail.

assertNotAlmostEquals (*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is less than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

Objects that are equal automatically fail.

assertNotEqual (*first*, *second*, *msg=None*)
Fail if the two objects are equal as determined by the '!=' operator.

assertNotEquals (*first*, *second*, *msg=None*)
Fail if the two objects are equal as determined by the '!=' operator.

assertNotIn (*member*, *container*, *msg=None*)
Just like self.assertTrue(a not in b), but with a nicer default message.

assertNotIsInstance (*obj*, *cls*, *msg=None*)
Included for symmetry with assertIsInstance.

assertNotRegexpMatches (*text*, *unexpected_regexp*, *msg=None*)
Fail the test if the text matches the regular expression.

assertRaises (*excClass*, *callableObj=None*, **args*, ***kwargs*)
Fail unless an exception of class excClass is raised by callableObj when invoked with arguments args and keyword arguments kwargs. If a different type of exception is raised, it will not be caught, and the test case will be deemed to have suffered an error, exactly as for an unexpected exception.

If called with callableObj omitted or None, will return a context object used like this:

The context manager keeps a reference to the exception as the 'exception' attribute. This allows you to inspect the exception after the assertion:

assertRaisesRegexp (*expected_exception*, *expected_regexp*, *callable_obj=None*, **args*, ***kwargs*)
Asserts that the message in a raised exception matches a regexp.

Parameters

- **expected_exception** – Exception class expected to be raised.
- **expected_regexp** – Regexp (re pattern object or string) expected to be found in error message.
- **callable_obj** – Function to be called.
- **args** – Extra args.
- **kwargs** – Extra kwargs.

assertRegexpMatches(*text*, *expected_regexp*, *msg=None*)
Fail the test unless the text matches the regular expression.

assertSequenceEqual(*seq1*, *seq2*, *msg=None*, *seq_type=None*)
An equality assertion for ordered sequences (like lists and tuples).

For the purposes of this function, a valid ordered sequence type is one which can be indexed, has a length, and has an equality operator.

Parameters

- **seq1** – The first sequence to compare.
- **seq2** – The second sequence to compare.
- **seq_type** – The expected datatype of the sequences, or None if no datatype should be enforced.
- **msg** – Optional message to use on failure instead of a list of differences.

assertSetEqual(*set1*, *set2*, *msg=None*)
A set-specific equality assertion.

Parameters

- **set1** – The first set to compare.
- **set2** – The second set to compare.
- **msg** – Optional message to use on failure instead of a list of differences.

assertSetEqual uses ducktyping to support different types of sets, and is optimized for sets specifically (parameters must support a difference method).

assertTrue(*expr*, *msg=None*)
Check that the expression is true.

assertTupleEqual(*tuple1*, *tuple2*, *msg=None*)
A tuple-specific equality assertion.

Parameters

- **tuple1** – The first tuple to compare.
- **tuple2** – The second tuple to compare.
- **msg** – Optional message to use on failure instead of a list of differences.

assert_(*expr*, *msg=None*)
Check that the expression is true.

base_test_pass(*name*, *environment*, *network*, ***kwargs*)
Basic test loop, requires an Agent to achieve a certain performance on an environment.

Parameters

- **name** (*str*) – The name of the test.
- **environment** (Environment) – The Environment object to use for the test.
- **network** (LayerBasedNetwork) – The Network to use for the agent's model.
- **kwargs** (*any*) – Agent arguments.

base_test_run(*name*, *environment*, *network*, ***kwargs*)
Run test, tests whether algorithm can run and update without compilation errors, not whether it passes.

Parameters

- **name** (*str*) – The name of the test.
- **environment** (Environment) – The Environment object to use for the test.
- **network** (LayerBasedNetwork) – The Network to use for the agent's model.
- **kwargs** (*any*) – Agent arguments.

config = {'update_mode': {'frequency': 4, 'batch_size': 8, 'unit': 'timesteps'}, '

countTestCases()

debug()
Run the test without collecting errors in a TestResult

defaultTestResult()

doCleanups()
Execute all cleanup functions. Normally called for you after tearDown.

exclude_bool = True

exclude_bounded = True

exclude_float = False

exclude_int = True

exclude_lstm = True

exclude_multi = True

fail(*msg=None*)
Fail immediately, with the given message.

failIf(**args*, ***kwargs*)

failIfAlmostEqual(**args*, ***kwargs*)

failIfEqual(**args*, ***kwargs*)

failUnless(**args*, ***kwargs*)

failUnlessAlmostEqual(**args*, ***kwargs*)

failUnlessEqual(**args*, ***kwargs*)

failUnlessRaises(**args*, ***kwargs*)

failureException
alias of AssertionError

id()

longMessage = False

maxDiff = 640

multi_config = None

pass_threshold = 0.8

pre_run(*agent*, *environment*)
Called before Runner.run.

requires_network = True

run(*result=None*)

setUp()
Hook method for setting up the test fixture before exercising it.

setUpClass()
Hook method for setting up class fixture before running tests in the class.

shortDescription()
Returns a one-line description of the test, or None if no description has been provided.

The default implementation of this method returns the first line of the specified test method's docstring.

skipTest(*reason*)
Skip this test.

tearDown()
Hook method for deconstructing the test fixture after testing it.

tearDownClass()
Hook method for deconstructing the class fixture after running all tests in the class.

test_bool()
Tests the case of one boolean action.

test_bounded_float()
Tests the case of one bounded float action, i.e. with min and max value.

test_float()
Tests the case of one float action.

test_int()
Tests the case of one integer action.

test_lstm()
Tests the case of using internal states via an LSTM layer (for one integer action).

test_multi()
Tests the case of multiple actions of different type and shape.

tensorforce.tests.test_ppo_agent module

class tensorforce.tests.test_ppo_agent.**TestPPOAgent**(*methodName='runTest'*)
Bases: *tensorforce.tests.base_agent_test.BaseAgentTest*, unittest.case.TestCase

__init__(*methodName='runTest'*)
Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

addCleanup(*function*, **args*, ***kwargs*)
Add a function, with arguments, to be called when the test is completed. Functions added are called on a LIFO basis and are called after tearDown on test failure or success.

Cleanup items are called even if setUp fails (unlike tearDown).

addTypeEqualityFunc(*typeobj*, *function*)
Add a type specific assertEqual style function to compare a type.

This method is for use by TestCase subclasses that need to register their own type equality functions to provide nicer error messages.

Parameters

- **typeobj** – The data type to call this function on when both values are of the same type in assertEqual().
- **function** – The callable taking two arguments and an optional msg= argument that raises self.failureException with a useful error message when the two arguments are not equal.

agent
alias of `PPOAgent`

assertAlmostEqual(*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is more than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

If the two objects compare equal then they will automatically compare almost equal.

assertAlmostEquals(*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is more than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

If the two objects compare equal then they will automatically compare almost equal.

assertDictContainsSubset(*expected*, *actual*, *msg=None*)
Checks whether actual is a superset of expected.

assertDictEqual(*d1*, *d2*, *msg=None*)

assertEqual(*first*, *second*, *msg=None*)
Fail if the two objects are unequal as determined by the '==' operator.

assertEquals(*first*, *second*, *msg=None*)
Fail if the two objects are unequal as determined by the '==' operator.

assertFalse(*expr*, *msg=None*)
Check that the expression is false.

assertGreater(*a*, *b*, *msg=None*)
Just like self.assertTrue(a > b), but with a nicer default message.

assertGreaterEqual(*a*, *b*, *msg=None*)
Just like self.assertTrue(a >= b), but with a nicer default message.

assertIn(*member*, *container*, *msg=None*)
Just like self.assertTrue(a in b), but with a nicer default message.

assertIs(*expr1*, *expr2*, *msg=None*)
Just like self.assertTrue(a is b), but with a nicer default message.

assertIsInstance(*obj*, *cls*, *msg=None*)
Same as self.assertTrue(isinstance(obj, cls)), with a nicer default message.

assertIsNone(*obj*, *msg=None*)
Same as self.assertTrue(obj is None), with a nicer default message.

assertIsNot(*expr1*, *expr2*, *msg=None*)
Just like self.assertTrue(a is not b), but with a nicer default message.

assertIsNotNone(*obj*, *msg=None*)
Included for symmetry with assertIsNone.

assertItemsEqual(*expected_seq*, *actual_seq*, *msg=None*)
An unordered sequence specific comparison. It asserts that actual_seq and expected_seq have the same element counts. Equivalent to:

Asserts that each element has the same count in both sequences. .. rubric:: Example

- [0, 1, 1] and [1, 0, 1] compare equal.
- [0, 0, 1] and [0, 1] compare unequal.

assertLess(*a*, *b*, *msg=None*)
Just like self.assertTrue(a < b), but with a nicer default message.

assertLessEqual(*a*, *b*, *msg=None*)
Just like self.assertTrue(a <= b), but with a nicer default message.

assertListEqual(*list1*, *list2*, *msg=None*)
A list-specific equality assertion.

Parameters

- **list1** – The first list to compare.
- **list2** – The second list to compare.
- **msg** – Optional message to use on failure instead of a list of differences.

assertMultiLineEqual(*first*, *second*, *msg=None*)
Assert that two multi-line strings are equal.

assertNotAlmostEqual(*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is less than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

Objects that are equal automatically fail.

assertNotAlmostEquals(*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is less than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

Objects that are equal automatically fail.

assertNotEqual(*first*, *second*, *msg=None*)
Fail if the two objects are equal as determined by the '!=' operator.

assertNotEquals(*first*, *second*, *msg=None*)
Fail if the two objects are equal as determined by the '!=' operator.

assertNotIn(*member*, *container*, *msg=None*)
Just like self.assertTrue(a not in b), but with a nicer default message.

assertNotIsInstance(*obj*, *cls*, *msg=None*)
Included for symmetry with assertIsInstance.

assertNotRegexpMatches(*text*, *unexpected_regexp*, *msg=None*)
Fail the test if the text matches the regular expression.

assertRaises(*excClass*, *callableObj=None*, **args*, ***kwargs*)
Fail unless an exception of class excClass is raised by callableObj when invoked with arguments args and keyword arguments kwargs. If a different type of exception is raised, it will not be caught, and the test case will be deemed to have suffered an error, exactly as for an unexpected exception.

If called with callableObj omitted or None, will return a context object used like this:

The context manager keeps a reference to the exception as the 'exception' attribute. This allows you to inspect the exception after the assertion:

assertRaisesRegexp(*expected_exception*, *expected_regexp*, *callable_obj=None*, **args*, ***kwargs*)
Asserts that the message in a raised exception matches a regexp.

Parameters

- **expected_exception** – Exception class expected to be raised.
- **expected_regexp** – Regexp (re pattern object or string) expected to be found in error message.
- **callable_obj** – Function to be called.
- **args** – Extra args.
- **kwargs** – Extra kwargs.

assertRegexpMatches(*text*, *expected_regexp*, *msg=None*)
Fail the test unless the text matches the regular expression.

assertSequenceEqual(*seq1*, *seq2*, *msg=None*, *seq_type=None*)
An equality assertion for ordered sequences (like lists and tuples).

For the purposes of this function, a valid ordered sequence type is one which can be indexed, has a length, and has an equality operator.

Parameters

- **seq1** – The first sequence to compare.
- **seq2** – The second sequence to compare.
- **seq_type** – The expected datatype of the sequences, or None if no datatype should be enforced.
- **msg** – Optional message to use on failure instead of a list of differences.

assertSetEqual(*set1*, *set2*, *msg=None*)
A set-specific equality assertion.

Parameters

- **set1** – The first set to compare.
- **set2** – The second set to compare.
- **msg** – Optional message to use on failure instead of a list of differences.

assertSetEqual uses ducktyping to support different types of sets, and is optimized for sets specifically (parameters must support a difference method).

assertTrue(*expr*, *msg=None*)
Check that the expression is true.

assertTupleEqual(*tuple1*, *tuple2*, *msg=None*)
A tuple-specific equality assertion.

Parameters

- **tuple1** – The first tuple to compare.
- **tuple2** – The second tuple to compare.
- **msg** – Optional message to use on failure instead of a list of differences.

assert_(*expr*, *msg=None*)
Check that the expression is true.

base_test_pass(*name*, *environment*, *network*, ***kwargs*)
Basic test loop, requires an Agent to achieve a certain performance on an environment.

Parameters

- **name** (*str*) – The name of the test.
- **environment** (Environment) – The Environment object to use for the test.
- **network** (LayerBasedNetwork) – The Network to use for the agent's model.
- **kwargs** (*any*) – Agent arguments.

base_test_run(*name*, *environment*, *network*, ***kwargs*)
Run test, tests whether algorithm can run and update without compilation errors, not whether it passes.

Parameters

- **name** (*str*) – The name of the test.
- **environment** (Environment) – The Environment object to use for the test.
- **network** (LayerBasedNetwork) – The Network to use for the agent's model.
- **kwargs** (*any*) – Agent arguments.

config = {'update_mode': {'frequency': 4, 'batch_size': 4, 'unit': 'episodes'}, 'st

countTestCases()

debug()
Run the test without collecting errors in a TestResult

defaultTestResult()

doCleanups()
Execute all cleanup functions. Normally called for you after tearDown.

exclude_bool = False

exclude_bounded = False

exclude_float = False

exclude_int = False

exclude_lstm = False

exclude_multi = False

fail(*msg=None*)
Fail immediately, with the given message.

failIf(**args*, ***kwargs*)

failIfAlmostEqual(**args*, ***kwargs*)

failIfEqual(**args*, ***kwargs*)

failUnless(**args*, ***kwargs*)

failUnlessAlmostEqual(**args*, ***kwargs*)

failUnlessEqual(**args*, ***kwargs*)

failUnlessRaises(**args*, ***kwargs*)

failureException
alias of `AssertionError`

id()

longMessage = False

maxDiff = 640

multi_config = None

pass_threshold = 0.8

pre_run(*agent*, *environment*)
Called before `Runner.run`.

requires_network = True

run(*result=None*)

setUp()
Hook method for setting up the test fixture before exercising it.

setUpClass()
Hook method for setting up class fixture before running tests in the class.

shortDescription()
Returns a one-line description of the test, or None if no description has been provided.

The default implementation of this method returns the first line of the specified test method's docstring.

skipTest(*reason*)
Skip this test.

tearDown()
Hook method for deconstructing the test fixture after testing it.

tearDownClass()
Hook method for deconstructing the class fixture after running all tests in the class.

test_bool()
Tests the case of one boolean action.

test_bounded_float()
Tests the case of one bounded float action, i.e. with min and max value.

test_float()
Tests the case of one float action.

test_int()
Tests the case of one integer action.

test_lstm()
Tests the case of using internal states via an LSTM layer (for one integer action).

test_multi()
Tests the case of multiple actions of different type and shape.

tensorforce.tests.test_quickstart_example module

class tensorforce.tests.test_quickstart_example.**TestQuickstartExample**(*methodName='runTest'*)
Bases: unittest.case.TestCase

__init__(*methodName='runTest'*)
Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

addCleanup(*function*, **args*, ***kwargs*)
Add a function, with arguments, to be called when the test is completed. Functions added are called on a LIFO basis and are called after tearDown on test failure or success.

Cleanup items are called even if setUp fails (unlike tearDown).

addTypeEqualityFunc(*typeobj*, *function*)
Add a type specific assertEqual style function to compare a type.

This method is for use by TestCase subclasses that need to register their own type equality functions to provide nicer error messages.

Parameters

- **typeobj** – The data type to call this function on when both values are of the same type in assertEqual().
- **function** – The callable taking two arguments and an optional msg= argument that raises self.failureException with a useful error message when the two arguments are not equal.

assertAlmostEqual(*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is more than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

If the two objects compare equal then they will automatically compare almost equal.

assertAlmostEquals(*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is more than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

If the two objects compare equal then they will automatically compare almost equal.

assertDictContainsSubset(*expected*, *actual*, *msg=None*)
Checks whether actual is a superset of expected.

assertDictEqual(*d1*, *d2*, *msg=None*)

assertEqual(*first*, *second*, *msg=None*)
Fail if the two objects are unequal as determined by the '==' operator.

assertEquals(*first*, *second*, *msg=None*)
Fail if the two objects are unequal as determined by the '==' operator.

assertFalse(*expr*, *msg=None*)
Check that the expression is false.

assertGreater(*a*, *b*, *msg=None*)
Just like self.assertTrue(a > b), but with a nicer default message.

assertGreaterEqual(*a*, *b*, *msg=None*)
Just like self.assertTrue(a >= b), but with a nicer default message.

assertIn(*member*, *container*, *msg=None*)
Just like self.assertTrue(a in b), but with a nicer default message.

assertIs(*expr1*, *expr2*, *msg=None*)
Just like self.assertTrue(a is b), but with a nicer default message.

assertIsInstance(*obj*, *cls*, *msg=None*)
Same as self.assertTrue(isinstance(obj, cls)), with a nicer default message.

assertIsNone(*obj*, *msg=None*)
Same as self.assertTrue(obj is None), with a nicer default message.

assertIsNot(*expr1*, *expr2*, *msg=None*)
Just like self.assertTrue(a is not b), but with a nicer default message.

assertIsNotNone(*obj*, *msg=None*)
Included for symmetry with assertIsNone.

assertItemsEqual(*expected_seq*, *actual_seq*, *msg=None*)
An unordered sequence specific comparison. It asserts that actual_seq and expected_seq have the same element counts. Equivalent to:

Asserts that each element has the same count in both sequences. .. rubric:: Example

- [0, 1, 1] and [1, 0, 1] compare equal.
- [0, 0, 1] and [0, 1] compare unequal.

assertLess(*a*, *b*, *msg=None*)
Just like self.assertTrue(a < b), but with a nicer default message.

assertLessEqual(*a*, *b*, *msg=None*)
Just like self.assertTrue(a <= b), but with a nicer default message.

assertListEqual(*list1*, *list2*, *msg=None*)
A list-specific equality assertion.

Parameters

- **list1** – The first list to compare.
- **list2** – The second list to compare.
- **msg** – Optional message to use on failure instead of a list of differences.

assertMultiLineEqual(*first*, *second*, *msg=None*)
Assert that two multi-line strings are equal.

assertNotAlmostEqual(*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is less than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

Objects that are equal automatically fail.

assertNotAlmostEquals (*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is less than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

Objects that are equal automatically fail.

assertNotEqual (*first*, *second*, *msg=None*)
Fail if the two objects are equal as determined by the '!=' operator.

assertNotEquals (*first*, *second*, *msg=None*)
Fail if the two objects are equal as determined by the '!=' operator.

assertNotIn (*member*, *container*, *msg=None*)
Just like self.assertTrue(a not in b), but with a nicer default message.

assertNotIsInstance (*obj*, *cls*, *msg=None*)
Included for symmetry with assertIsInstance.

assertNotRegexpMatches (*text*, *unexpected_regexp*, *msg=None*)
Fail the test if the text matches the regular expression.

assertRaises (*excClass*, *callableObj=None*, **args*, ***kwargs*)
Fail unless an exception of class excClass is raised by callableObj when invoked with arguments args and keyword arguments kwargs. If a different type of exception is raised, it will not be caught, and the test case will be deemed to have suffered an error, exactly as for an unexpected exception.

If called with callableObj omitted or None, will return a context object used like this:

The context manager keeps a reference to the exception as the 'exception' attribute. This allows you to inspect the exception after the assertion:

assertRaisesRegexp (*expected_exception*, *expected_regexp*, *callable_obj=None*, **args*, ***kwargs*)
Asserts that the message in a raised exception matches a regexp.

Parameters

- **expected_exception** – Exception class expected to be raised.
- **expected_regexp** – Regexp (re pattern object or string) expected to be found in error message.
- **callable_obj** – Function to be called.
- **args** – Extra args.
- **kwargs** – Extra kwargs.

assertRegexpMatches (*text*, *expected_regexp*, *msg=None*)
Fail the test unless the text matches the regular expression.

assertSequenceEqual (*seq1*, *seq2*, *msg=None*, *seq_type=None*)
An equality assertion for ordered sequences (like lists and tuples).

For the purposes of this function, a valid ordered sequence type is one which can be indexed, has a length, and has an equality operator.

Parameters

- **seq1** – The first sequence to compare.
- **seq2** – The second sequence to compare.
- **seq_type** – The expected datatype of the sequences, or None if no datatype should be enforced.
- **msg** – Optional message to use on failure instead of a list of differences.

assertSetEqual(*set1*, *set2*, *msg=None*)
A set-specific equality assertion.

Parameters

- **set1** – The first set to compare.
- **set2** – The second set to compare.
- **msg** – Optional message to use on failure instead of a list of differences.

assertSetEqual uses ducktyping to support different types of sets, and is optimized for sets specifically (parameters must support a difference method).

assertTrue(*expr*, *msg=None*)
Check that the expression is true.

assertTupleEqual(*tuple1*, *tuple2*, *msg=None*)
A tuple-specific equality assertion.

Parameters

- **tuple1** – The first tuple to compare.
- **tuple2** – The second tuple to compare.
- **msg** – Optional message to use on failure instead of a list of differences.

assert_(*expr*, *msg=None*)
Check that the expression is true.

countTestCases()

debug()
Run the test without collecting errors in a TestResult

defaultTestResult()

doCleanups()
Execute all cleanup functions. Normally called for you after tearDown.

fail(*msg=None*)
Fail immediately, with the given message.

failIf(**args*, ***kwargs*)

failIfAlmostEqual(**args*, ***kwargs*)

failIfEqual(**args*, ***kwargs*)

failUnless(**args*, ***kwargs*)

failUnlessAlmostEqual(**args*, ***kwargs*)

failUnlessEqual(**args*, ***kwargs*)

failUnlessRaises(**args*, ***kwargs*)

failureException
alias of AssertionError

id()

longMessage = False

maxDiff = 640

run(*result=None*)

setUp()
Hook method for setting up the test fixture before exercising it.

setUpClass()
Hook method for setting up class fixture before running tests in the class.

shortDescription()
Returns a one-line description of the test, or None if no description has been provided.

The default implementation of this method returns the first line of the specified test method's docstring.

skipTest(*reason*)
Skip this test.

tearDown()
Hook method for deconstructing the test fixture after testing it.

tearDownClass()
Hook method for deconstructing the class fixture after running all tests in the class.

test_example()

tensorforce.tests.test_random_agent module

class tensorforce.tests.test_random_agent.**TestRandomAgent**(*methodName='runTest'*)
Bases: *tensorforce.tests.base_agent_test.BaseAgentTest*, unittest.case.TestCase

__init__(*methodName='runTest'*)
Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

addCleanup(*function*, **args*, ***kwargs*)
Add a function, with arguments, to be called when the test is completed. Functions added are called on a LIFO basis and are called after tearDown on test failure or success.

Cleanup items are called even if setUp fails (unlike tearDown).

addTypeEqualityFunc(*typeobj*, *function*)
Add a type specific assertEqual style function to compare a type.

This method is for use by TestCase subclasses that need to register their own type equality functions to provide nicer error messages.

Parameters

- **typeobj** – The data type to call this function on when both values are of the same type in assertEqual().
- **function** – The callable taking two arguments and an optional msg= argument that raises self.failureException with a useful error message when the two arguments are not equal.

agent
alias of `RandomAgent`

assertAlmostEqual(*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is more than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

If the two objects compare equal then they will automatically compare almost equal.

assertAlmostEquals(*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is more than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

If the two objects compare equal then they will automatically compare almost equal.

assertDictContainsSubset(*expected*, *actual*, *msg=None*)
Checks whether actual is a superset of expected.

assertDictEqual(*d1*, *d2*, *msg=None*)

assertEqual(*first*, *second*, *msg=None*)
Fail if the two objects are unequal as determined by the '==' operator.

assertEquals(*first*, *second*, *msg=None*)
Fail if the two objects are unequal as determined by the '==' operator.

assertFalse(*expr*, *msg=None*)
Check that the expression is false.

assertGreater(*a*, *b*, *msg=None*)
Just like self.assertTrue(a > b), but with a nicer default message.

assertGreaterEqual(*a*, *b*, *msg=None*)
Just like self.assertTrue(a >= b), but with a nicer default message.

assertIn(*member*, *container*, *msg=None*)
Just like self.assertTrue(a in b), but with a nicer default message.

assertIs(*expr1*, *expr2*, *msg=None*)
Just like self.assertTrue(a is b), but with a nicer default message.

assertIsInstance(*obj*, *cls*, *msg=None*)
Same as self.assertTrue(isinstance(obj, cls)), with a nicer default message.

assertIsNone(*obj*, *msg=None*)
Same as self.assertTrue(obj is None), with a nicer default message.

assertIsNot(*expr1*, *expr2*, *msg=None*)
Just like self.assertTrue(a is not b), but with a nicer default message.

assertIsNotNone(*obj*, *msg=None*)
Included for symmetry with assertIsNone.

assertItemsEqual(*expected_seq*, *actual_seq*, *msg=None*)
An unordered sequence specific comparison. It asserts that actual_seq and expected_seq have the same element counts. Equivalent to:

Asserts that each element has the same count in both sequences. .. rubric:: Example

- [0, 1, 1] and [1, 0, 1] compare equal.
- [0, 0, 1] and [0, 1] compare unequal.

assertLess(*a*, *b*, *msg=None*)
Just like self.assertTrue(a < b), but with a nicer default message.

assertLessEqual(*a*, *b*, *msg=None*)
Just like self.assertTrue(a <= b), but with a nicer default message.

assertListEqual(*list1*, *list2*, *msg=None*)
A list-specific equality assertion.

Parameters

- **list1** – The first list to compare.
- **list2** – The second list to compare.
- **msg** – Optional message to use on failure instead of a list of differences.

assertMultiLineEqual(*first*, *second*, *msg=None*)
Assert that two multi-line strings are equal.

assertNotAlmostEqual(*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is less than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

Objects that are equal automatically fail.

assertNotAlmostEquals(*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is less than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

Objects that are equal automatically fail.

assertNotEqual(*first*, *second*, *msg=None*)
Fail if the two objects are equal as determined by the '!=' operator.

assertNotEquals(*first*, *second*, *msg=None*)
Fail if the two objects are equal as determined by the '!=' operator.

assertNotIn(*member*, *container*, *msg=None*)
Just like self.assertTrue(a not in b), but with a nicer default message.

assertNotIsInstance(*obj*, *cls*, *msg=None*)
Included for symmetry with assertIsInstance.

assertNotRegexpMatches(*text*, *unexpected_regexp*, *msg=None*)
Fail the test if the text matches the regular expression.

assertRaises(*excClass*, *callableObj=None*, **args*, ***kwargs*)
Fail unless an exception of class excClass is raised by callableObj when invoked with arguments args and keyword arguments kwargs. If a different type of exception is raised, it will not be caught, and the test case will be deemed to have suffered an error, exactly as for an unexpected exception.

If called with callableObj omitted or None, will return a context object used like this:

The context manager keeps a reference to the exception as the 'exception' attribute. This allows you to inspect the exception after the assertion:

assertRaisesRegexp(*expected_exception*, *expected_regexp*, *callable_obj=None*, **args*, ***kwargs*)
Asserts that the message in a raised exception matches a regexp.

Parameters

- **expected_exception** – Exception class expected to be raised.
- **expected_regexp** – Regexp (re pattern object or string) expected to be found in error message.
- **callable_obj** – Function to be called.
- **args** – Extra args.
- **kwargs** – Extra kwargs.

assertRegexpMatches(*text*, *expected_regexp*, *msg=None*)
Fail the test unless the text matches the regular expression.

assertSequenceEqual(*seq1*, *seq2*, *msg=None*, *seq_type=None*)
An equality assertion for ordered sequences (like lists and tuples).

For the purposes of this function, a valid ordered sequence type is one which can be indexed, has a length, and has an equality operator.

Parameters

- **seq1** – The first sequence to compare.
- **seq2** – The second sequence to compare.
- **seq_type** – The expected datatype of the sequences, or None if no datatype should be enforced.
- **msg** – Optional message to use on failure instead of a list of differences.

assertSetEqual(*set1*, *set2*, *msg=None*)
A set-specific equality assertion.

Parameters

- **set1** – The first set to compare.
- **set2** – The second set to compare.
- **msg** – Optional message to use on failure instead of a list of differences.

assertSetEqual uses ducktyping to support different types of sets, and is optimized for sets specifically (parameters must support a difference method).

assertTrue(*expr*, *msg=None*)
Check that the expression is true.

assertTupleEqual(*tuple1*, *tuple2*, *msg=None*)
A tuple-specific equality assertion.

Parameters

- **tuple1** – The first tuple to compare.
- **tuple2** – The second tuple to compare.
- **msg** – Optional message to use on failure instead of a list of differences.

assert_(*expr*, *msg=None*)
Check that the expression is true.

base_test_pass(*name*, *environment*, *network*, ***kwargs*)
Basic test loop, requires an Agent to achieve a certain performance on an environment.

Parameters

- **name** (*str*) – The name of the test.
- **environment** (Environment) – The Environment object to use for the test.
- **network** (LayerBasedNetwork) – The Network to use for the agent's model.
- **kwargs** (*any*) – Agent arguments.

base_test_run(*name*, *environment*, *network*, ***kwargs*)
Run test, tests whether algorithm can run and update without compilation errors, not whether it passes.

Parameters

- **name** (*str*) – The name of the test.
- **environment** (Environment) – The Environment object to use for the test.
- **network** (LayerBasedNetwork) – The Network to use for the agent's model.
- **kwargs** (*any*) – Agent arguments.

config = {}

countTestCases()

debug()
Run the test without collecting errors in a TestResult

defaultTestResult()

doCleanups()
Execute all cleanup functions. Normally called for you after tearDown.

exclude_bool = False

exclude_bounded = False

exclude_float = False

exclude_int = False

exclude_lstm = True

exclude_multi = False

fail(*msg=None*)
Fail immediately, with the given message.

failIf(**args*, ***kwargs*)

failIfAlmostEqual(**args*, ***kwargs*)

failIfEqual(**args*, ***kwargs*)

failUnless(**args*, ***kwargs*)

failUnlessAlmostEqual(**args*, ***kwargs*)

failUnlessEqual(**args*, ***kwargs*)

failUnlessRaises(**args*, ***kwargs*)

failureException
alias of AssertionError

id()

longMessage = False

maxDiff = 640

multi_config = None

pass_threshold = 0.0

pre_run(*agent*, *environment*)
Called before Runner.run.

requires_network = False

run(*result=None*)

setUp()
Hook method for setting up the test fixture before exercising it.

setUpClass()
Hook method for setting up class fixture before running tests in the class.

shortDescription()
Returns a one-line description of the test, or None if no description has been provided.

The default implementation of this method returns the first line of the specified test method's docstring.

skipTest(*reason*)
Skip this test.

tearDown()
Hook method for deconstructing the test fixture after testing it.

tearDownClass()
Hook method for deconstructing the class fixture after running all tests in the class.

test_bool()
Tests the case of one boolean action.

test_bounded_float()
Tests the case of one bounded float action, i.e. with min and max value.

test_float()
Tests the case of one float action.

test_int()
Tests the case of one integer action.

test_lstm()
Tests the case of using internal states via an LSTM layer (for one integer action).

test_multi()
Tests the case of multiple actions of different type and shape.

tensorforce.tests.test_reward_estimation module

class tensorforce.tests.test_reward_estimation.**TestRewardEstimation**(*methodName='runTest'*)
Bases: unittest.case.TestCase

__init__ (*methodName='runTest'*)
Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

addCleanup (*function*, **args*, ***kwargs*)
Add a function, with arguments, to be called when the test is completed. Functions added are called on a LIFO basis and are called after tearDown on test failure or success.

Cleanup items are called even if setUp fails (unlike tearDown).

addTypeEqualityFunc (*typeobj*, *function*)
Add a type specific assertEqual style function to compare a type.

This method is for use by TestCase subclasses that need to register their own type equality functions to provide nicer error messages.

Parameters

- **typeobj** – The data type to call this function on when both values are of the same type in assertEqual().
- **function** – The callable taking two arguments and an optional msg= argument that raises self.failureException with a useful error message when the two arguments are not equal.

assertAlmostEqual (*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is more than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

If the two objects compare equal then they will automatically compare almost equal.

assertAlmostEquals (*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is more than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

If the two objects compare equal then they will automatically compare almost equal.

assertDictContainsSubset (*expected*, *actual*, *msg=None*)
Checks whether actual is a superset of expected.

assertDictEqual (*d1*, *d2*, *msg=None*)

assertEqual (*first*, *second*, *msg=None*)
Fail if the two objects are unequal as determined by the '==' operator.

assertEquals (*first*, *second*, *msg=None*)
Fail if the two objects are unequal as determined by the '==' operator.

assertFalse (*expr*, *msg=None*)
Check that the expression is false.

assertGreater (*a*, *b*, *msg=None*)
Just like self.assertTrue(a > b), but with a nicer default message.

assertGreaterEqual (*a*, *b*, *msg=None*)
Just like self.assertTrue(a >= b), but with a nicer default message.

assertIn(*member*, *container*, *msg=None*)
Just like self.assertTrue(a in b), but with a nicer default message.

assertIs(*expr1*, *expr2*, *msg=None*)
Just like self.assertTrue(a is b), but with a nicer default message.

assertIsInstance(*obj*, *cls*, *msg=None*)
Same as self.assertTrue(isinstance(obj, cls)), with a nicer default message.

assertIsNone(*obj*, *msg=None*)
Same as self.assertTrue(obj is None), with a nicer default message.

assertIsNot(*expr1*, *expr2*, *msg=None*)
Just like self.assertTrue(a is not b), but with a nicer default message.

assertIsNotNone(*obj*, *msg=None*)
Included for symmetry with assertIsNone.

assertItemsEqual(*expected_seq*, *actual_seq*, *msg=None*)
An unordered sequence specific comparison. It asserts that actual_seq and expected_seq have the same element counts. Equivalent to:

Asserts that each element has the same count in both sequences. .. rubric:: Example

- [0, 1, 1] and [1, 0, 1] compare equal.
- [0, 0, 1] and [0, 1] compare unequal.

assertLess(*a*, *b*, *msg=None*)
Just like self.assertTrue(a < b), but with a nicer default message.

assertLessEqual(*a*, *b*, *msg=None*)
Just like self.assertTrue(a <= b), but with a nicer default message.

assertListEqual(*list1*, *list2*, *msg=None*)
A list-specific equality assertion.

Parameters

- **list1** – The first list to compare.
- **list2** – The second list to compare.
- **msg** – Optional message to use on failure instead of a list of differences.

assertMultiLineEqual(*first*, *second*, *msg=None*)
Assert that two multi-line strings are equal.

assertNotAlmostEqual(*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is less than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

Objects that are equal automatically fail.

assertNotAlmostEquals(*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is less than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

Objects that are equal automatically fail.

assertNotEqual(*first*, *second*, *msg=None*)
Fail if the two objects are equal as determined by the '!=' operator.

assertNotEquals(*first*, *second*, *msg=None*)
Fail if the two objects are equal as determined by the '!=' operator.

assertNotIn(*member*, *container*, *msg=None*)
Just like self.assertTrue(a not in b), but with a nicer default message.

assertNotIsInstance(*obj*, *cls*, *msg=None*)
Included for symmetry with assertIsInstance.

assertNotRegexpMatches(*text*, *unexpected_regexp*, *msg=None*)
Fail the test if the text matches the regular expression.

assertRaises(*excClass*, *callableObj=None*, **args*, ***kwargs*)
Fail unless an exception of class excClass is raised by callableObj when invoked with arguments args and keyword arguments kwargs. If a different type of exception is raised, it will not be caught, and the test case will be deemed to have suffered an error, exactly as for an unexpected exception.

If called with callableObj omitted or None, will return a context object used like this:

The context manager keeps a reference to the exception as the 'exception' attribute. This allows you to inspect the exception after the assertion:

assertRaisesRegexp(*expected_exception*, *expected_regexp*, *callable_obj=None*, **args*, ***kwargs*)
Asserts that the message in a raised exception matches a regexp.

Parameters

- **expected_exception** – Exception class expected to be raised.
- **expected_regexp** – Regexp (re pattern object or string) expected to be found in error message.
- **callable_obj** – Function to be called.
- **args** – Extra args.
- **kwargs** – Extra kwargs.

assertRegexpMatches(*text*, *expected_regexp*, *msg=None*)
Fail the test unless the text matches the regular expression.

assertSequenceEqual(*seq1*, *seq2*, *msg=None*, *seq_type=None*)
An equality assertion for ordered sequences (like lists and tuples).

For the purposes of this function, a valid ordered sequence type is one which can be indexed, has a length, and has an equality operator.

Parameters

- **seq1** – The first sequence to compare.
- **seq2** – The second sequence to compare.
- **seq_type** – The expected datatype of the sequences, or None if no datatype should be enforced.
- **msg** – Optional message to use on failure instead of a list of differences.

assertSetEqual(*set1*, *set2*, *msg=None*)
A set-specific equality assertion.

Parameters

- **set1** – The first set to compare.
- **set2** – The second set to compare.
- **msg** – Optional message to use on failure instead of a list of differences.

assertSetEqual uses ducktyping to support different types of sets, and is optimized for sets specifically (parameters must support a difference method).

assertTrue(*expr*, *msg=None*)
Check that the expression is true.

assertTupleEqual(*tuple1*, *tuple2*, *msg=None*)
A tuple-specific equality assertion.

Parameters

- **tuple1** – The first tuple to compare.
- **tuple2** – The second tuple to compare.
- **msg** – Optional message to use on failure instead of a list of differences.

assert_(*expr*, *msg=None*)
Check that the expression is true.

countTestCases()

debug()
Run the test without collecting errors in a TestResult

defaultTestResult()

doCleanups()
Execute all cleanup functions. Normally called for you after tearDown.

fail(*msg=None*)
Fail immediately, with the given message.

failIf(**args*, ***kwargs*)

failIfAlmostEqual(**args*, ***kwargs*)

failIfEqual(**args*, ***kwargs*)

failUnless(**args*, ***kwargs*)

failUnlessAlmostEqual(**args*, ***kwargs*)

failUnlessEqual(**args*, ***kwargs*)

failUnlessRaises(**args*, ***kwargs*)

failureException
alias of `AssertionError`

id()

longMessage = False

maxDiff = 640

run(*result=None*)

setUp()
Hook method for setting up the test fixture before exercising it.

setUpClass()
Hook method for setting up class fixture before running tests in the class.

shortDescription()
Returns a one-line description of the test, or None if no description has been provided.

The default implementation of this method returns the first line of the specified test method's docstring.

skipTest(*reason*)
Skip this test.

tearDown()
Hook method for deconstructing the test fixture after testing it.

tearDownClass()
Hook method for deconstructing the class fixture after running all tests in the class.

test_baseline()

test_basic()

test_gae()

tensorforce.tests.test_trpo_agent module

class tensorforce.tests.test_trpo_agent.**TestTRPOAgent**(*methodName='runTest'*)
Bases: *tensorforce.tests.base_agent_test.BaseAgentTest*, unittest.case.TestCase

__init__(*methodName='runTest'*)
Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

addCleanup(*function*, **args*, ***kwargs*)
Add a function, with arguments, to be called when the test is completed. Functions added are called on a LIFO basis and are called after tearDown on test failure or success.

Cleanup items are called even if setUp fails (unlike tearDown).

addTypeEqualityFunc(*typeobj*, *function*)
Add a type specific assertEqual style function to compare a type.

This method is for use by TestCase subclasses that need to register their own type equality functions to provide nicer error messages.

Parameters

- **typeobj** – The data type to call this function on when both values are of the same type in assertEqual().
- **function** – The callable taking two arguments and an optional msg= argument that raises self.failureException with a useful error message when the two arguments are not equal.

agent
alias of TRPOAgent

assertAlmostEqual(*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is more than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

If the two objects compare equal then they will automatically compare almost equal.

assertAlmostEquals(*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is more than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

If the two objects compare equal then they will automatically compare almost equal.

assertDictContainsSubset(*expected*, *actual*, *msg=None*)
Checks whether actual is a superset of expected.

assertDictEqual(*d1*, *d2*, *msg=None*)

assertEqual(*first*, *second*, *msg=None*)
Fail if the two objects are unequal as determined by the '==' operator.

assertEquals(*first*, *second*, *msg=None*)
Fail if the two objects are unequal as determined by the '==' operator.

assertFalse(*expr*, *msg=None*)
Check that the expression is false.

assertGreater(*a*, *b*, *msg=None*)
Just like self.assertTrue(a > b), but with a nicer default message.

assertGreaterEqual(*a*, *b*, *msg=None*)
Just like self.assertTrue(a >= b), but with a nicer default message.

assertIn(*member*, *container*, *msg=None*)
Just like self.assertTrue(a in b), but with a nicer default message.

assertIs(*expr1*, *expr2*, *msg=None*)
Just like self.assertTrue(a is b), but with a nicer default message.

assertIsInstance(*obj*, *cls*, *msg=None*)
Same as self.assertTrue(isinstance(obj, cls)), with a nicer default message.

assertIsNone(*obj*, *msg=None*)
Same as self.assertTrue(obj is None), with a nicer default message.

assertIsNot(*expr1*, *expr2*, *msg=None*)
Just like self.assertTrue(a is not b), but with a nicer default message.

assertIsNotNone(*obj*, *msg=None*)
Included for symmetry with assertIsNone.

assertItemsEqual(*expected_seq*, *actual_seq*, *msg=None*)
An unordered sequence specific comparison. It asserts that actual_seq and expected_seq have the same element counts. Equivalent to:

Asserts that each element has the same count in both sequences. .. rubric:: Example

- [0, 1, 1] and [1, 0, 1] compare equal.
- [0, 0, 1] and [0, 1] compare unequal.

assertLess(*a*, *b*, *msg=None*)
Just like self.assertTrue(a < b), but with a nicer default message.

assertLessEqual(*a*, *b*, *msg=None*)
Just like self.assertTrue(a <= b), but with a nicer default message.

assertListEqual(*list1*, *list2*, *msg=None*)
A list-specific equality assertion.

Parameters

- **list1** – The first list to compare.
- **list2** – The second list to compare.
- **msg** – Optional message to use on failure instead of a list of differences.

assertMultiLineEqual(*first*, *second*, *msg=None*)
Assert that two multi-line strings are equal.

assertNotAlmostEqual(*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is less than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

Objects that are equal automatically fail.

assertNotAlmostEquals(*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is less than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

Objects that are equal automatically fail.

assertNotEqual(*first*, *second*, *msg=None*)
Fail if the two objects are equal as determined by the '!=' operator.

assertNotEquals(*first*, *second*, *msg=None*)
Fail if the two objects are equal as determined by the '!=' operator.

assertNotIn(*member*, *container*, *msg=None*)
Just like self.assertTrue(a not in b), but with a nicer default message.

assertNotIsInstance(*obj*, *cls*, *msg=None*)
Included for symmetry with assertIsInstance.

assertNotRegexpMatches(*text*, *unexpected_regexp*, *msg=None*)
Fail the test if the text matches the regular expression.

assertRaises(*excClass*, *callableObj=None*, **args*, ***kwargs*)
Fail unless an exception of class excClass is raised by callableObj when invoked with arguments args and keyword arguments kwargs. If a different type of exception is raised, it will not be caught, and the test case will be deemed to have suffered an error, exactly as for an unexpected exception.

If called with callableObj omitted or None, will return a context object used like this:

The context manager keeps a reference to the exception as the 'exception' attribute. This allows you to inspect the exception after the assertion:

assertRaisesRegexp(*expected_exception*, *expected_regexp*, *callable_obj=None*, **args*, ***kwargs*)
Asserts that the message in a raised exception matches a regexp.

Parameters

- **expected_exception** – Exception class expected to be raised.
- **expected_regexp** – Regexp (re pattern object or string) expected to be found in error message.
- **callable_obj** – Function to be called.
- **args** – Extra args.
- **kwargs** – Extra kwargs.

assertRegexpMatches(*text*, *expected_regexp*, *msg=None*)
Fail the test unless the text matches the regular expression.

assertSequenceEqual(*seq1*, *seq2*, *msg=None*, *seq_type=None*)
An equality assertion for ordered sequences (like lists and tuples).

For the purposes of this function, a valid ordered sequence type is one which can be indexed, has a length, and has an equality operator.

Parameters

- **seq1** – The first sequence to compare.
- **seq2** – The second sequence to compare.
- **seq_type** – The expected datatype of the sequences, or None if no datatype should be enforced.
- **msg** – Optional message to use on failure instead of a list of differences.

assertSetEqual(*set1*, *set2*, *msg=None*)
A set-specific equality assertion.

Parameters

- **set1** – The first set to compare.
- **set2** – The second set to compare.
- **msg** – Optional message to use on failure instead of a list of differences.

assertSetEqual uses ducktyping to support different types of sets, and is optimized for sets specifically (parameters must support a difference method).

assertTrue(*expr*, *msg=None*)
Check that the expression is true.

assertTupleEqual(*tuple1*, *tuple2*, *msg=None*)
A tuple-specific equality assertion.

Parameters

- **tuple1** – The first tuple to compare.
- **tuple2** – The second tuple to compare.
- **msg** – Optional message to use on failure instead of a list of differences.

assert_(*expr*, *msg=None*)
Check that the expression is true.

base_test_pass(*name*, *environment*, *network*, ***kwargs*)
Basic test loop, requires an Agent to achieve a certain performance on an environment.

Parameters

- **name** (*str*) – The name of the test.
- **environment** (Environment) – The Environment object to use for the test.
- **network** (LayerBasedNetwork) – The Network to use for the agent's model.
- **kwargs** (*any*) – Agent arguments.

base_test_run(*name*, *environment*, *network*, ***kwargs*)
Run test, tests whether algorithm can run and update without compilation errors, not whether it passes.

Parameters

- **name** (*str*) – The name of the test.
- **environment** (Environment) – The Environment object to use for the test.
- **network** (LayerBasedNetwork) – The Network to use for the agent's model.
- **kwargs** (*any*) – Agent arguments.

config = {'update_mode': {'frequency': 4, 'batch_size': 4, 'unit': 'episodes'}, 'l

countTestCases()

debug()
Run the test without collecting errors in a TestResult

defaultTestResult()

doCleanups()
Execute all cleanup functions. Normally called for you after tearDown.

exclude_bool = False

exclude_bounded = False

exclude_float = False

exclude_int = False

exclude_lstm = False

exclude_multi = False

fail(*msg=None*)
Fail immediately, with the given message.

failIf(**args*, ***kwargs*)

failIfAlmostEqual(**args*, ***kwargs*)

failIfEqual(**args*, ***kwargs*)

failUnless(**args*, ***kwargs*)

failUnlessAlmostEqual(**args*, ***kwargs*)

failUnlessEqual(**args*, ***kwargs*)

failUnlessRaises(**args*, ***kwargs*)

failureException
alias of AssertionError

id()

longMessage = False

maxDiff = 640

multi_config = None

pass_threshold = 0.8

pre_run(*agent*, *environment*)
Called before `Runner.run`.

requires_network = True

run(*result=None*)

setUp()
Hook method for setting up the test fixture before exercising it.

setUpClass()
Hook method for setting up class fixture before running tests in the class.

shortDescription()
Returns a one-line description of the test, or None if no description has been provided.

The default implementation of this method returns the first line of the specified test method's docstring.

skipTest(*reason*)
Skip this test.

tearDown()
Hook method for deconstructing the test fixture after testing it.

tearDownClass()
Hook method for deconstructing the class fixture after running all tests in the class.

test_bool()
Tests the case of one boolean action.

test_bounded_float()
Tests the case of one bounded float action, i.e. with min and max value.

test_float()
Tests the case of one float action.

test_int()
Tests the case of one integer action.

test_lstm()
Tests the case of using internal states via an LSTM layer (for one integer action).

test_multi()
Tests the case of multiple actions of different type and shape.

tensorforce.tests.test_tutorial_code module

class tensorforce.tests.test_tutorial_code.**TestTutorialCode**(*methodName='runTest'*)
Bases: `unittest.case.TestCase`

Validation of random code snippets as to be notified when old blog posts need to be changed.

class MyClient(**args*, ***kwargs*)
Bases: `object`

__init__(**args*, ***kwargs*)

execute(*action*)

get_state()

__init__ (*methodName='runTest'*)
Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

addCleanup (*function*, **args*, ***kwargs*)
Add a function, with arguments, to be called when the test is completed. Functions added are called on a LIFO basis and are called after tearDown on test failure or success.

Cleanup items are called even if setUp fails (unlike tearDown).

addTypeEqualityFunc (*typeobj*, *function*)
Add a type specific assertEqual style function to compare a type.

This method is for use by TestCase subclasses that need to register their own type equality functions to provide nicer error messages.

Parameters

- **typeobj** – The data type to call this function on when both values are of the same type in assertEqual().
- **function** – The callable taking two arguments and an optional msg= argument that raises self.failureException with a useful error message when the two arguments are not equal.

assertAlmostEqual (*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is more than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

If the two objects compare equal then they will automatically compare almost equal.

assertAlmostEquals (*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is more than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

If the two objects compare equal then they will automatically compare almost equal.

assertDictContainsSubset (*expected*, *actual*, *msg=None*)
Checks whether actual is a superset of expected.

assertDictEqual (*d1*, *d2*, *msg=None*)

assertEqual (*first*, *second*, *msg=None*)
Fail if the two objects are unequal as determined by the '==' operator.

assertEquals (*first*, *second*, *msg=None*)
Fail if the two objects are unequal as determined by the '==' operator.

assertFalse (*expr*, *msg=None*)
Check that the expression is false.

assertGreater (*a*, *b*, *msg=None*)
Just like self.assertTrue(a > b), but with a nicer default message.

assertGreaterEqual (*a*, *b*, *msg=None*)
Just like self.assertTrue(a >= b), but with a nicer default message.

assertIn(*member*, *container*, *msg=None*)
Just like self.assertTrue(a in b), but with a nicer default message.

assertIs(*expr1*, *expr2*, *msg=None*)
Just like self.assertTrue(a is b), but with a nicer default message.

assertIsInstance(*obj*, *cls*, *msg=None*)
Same as self.assertTrue(isinstance(obj, cls)), with a nicer default message.

assertIsNone(*obj*, *msg=None*)
Same as self.assertTrue(obj is None), with a nicer default message.

assertIsNot(*expr1*, *expr2*, *msg=None*)
Just like self.assertTrue(a is not b), but with a nicer default message.

assertIsNotNone(*obj*, *msg=None*)
Included for symmetry with assertIsNone.

assertItemsEqual(*expected_seq*, *actual_seq*, *msg=None*)
An unordered sequence specific comparison. It asserts that actual_seq and expected_seq have the same element counts. Equivalent to:

Asserts that each element has the same count in both sequences. .. rubric:: Example

- [0, 1, 1] and [1, 0, 1] compare equal.
- [0, 0, 1] and [0, 1] compare unequal.

assertLess(*a*, *b*, *msg=None*)
Just like self.assertTrue(a < b), but with a nicer default message.

assertLessEqual(*a*, *b*, *msg=None*)
Just like self.assertTrue(a <= b), but with a nicer default message.

assertListEqual(*list1*, *list2*, *msg=None*)
A list-specific equality assertion.

Parameters

- **list1** – The first list to compare.
- **list2** – The second list to compare.
- **msg** – Optional message to use on failure instead of a list of differences.

assertMultiLineEqual(*first*, *second*, *msg=None*)
Assert that two multi-line strings are equal.

assertNotAlmostEqual(*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is less than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

Objects that are equal automatically fail.

assertNotAlmostEquals(*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is less than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

Objects that are equal automatically fail.

assertNotEqual(*first*, *second*, *msg=None*)
Fail if the two objects are equal as determined by the '!=' operator.

assertNotEquals(*first*, *second*, *msg=None*)
Fail if the two objects are equal as determined by the '!=' operator.

assertNotIn(*member*, *container*, *msg=None*)
Just like self.assertTrue(a not in b), but with a nicer default message.

assertNotIsInstance(*obj*, *cls*, *msg=None*)
Included for symmetry with assertIsInstance.

assertNotRegexpMatches(*text*, *unexpected_regexp*, *msg=None*)
Fail the test if the text matches the regular expression.

assertRaises(*excClass*, *callableObj=None*, **args*, ***kwargs*)
Fail unless an exception of class excClass is raised by callableObj when invoked with arguments args and keyword arguments kwargs. If a different type of exception is raised, it will not be caught, and the test case will be deemed to have suffered an error, exactly as for an unexpected exception.

If called with callableObj omitted or None, will return a context object used like this:

The context manager keeps a reference to the exception as the 'exception' attribute. This allows you to inspect the exception after the assertion:

assertRaisesRegexp(*expected_exception*, *expected_regexp*, *callable_obj=None*, **args*, ***kwargs*)
Asserts that the message in a raised exception matches a regexp.

Parameters

- **expected_exception** – Exception class expected to be raised.
- **expected_regexp** – Regexp (re pattern object or string) expected to be found in error message.
- **callable_obj** – Function to be called.
- **args** – Extra args.
- **kwargs** – Extra kwargs.

assertRegexpMatches(*text*, *expected_regexp*, *msg=None*)
Fail the test unless the text matches the regular expression.

assertSequenceEqual(*seq1*, *seq2*, *msg=None*, *seq_type=None*)
An equality assertion for ordered sequences (like lists and tuples).

For the purposes of this function, a valid ordered sequence type is one which can be indexed, has a length, and has an equality operator.

Parameters

- **seq1** – The first sequence to compare.
- **seq2** – The second sequence to compare.
- **seq_type** – The expected datatype of the sequences, or None if no datatype should be enforced.
- **msg** – Optional message to use on failure instead of a list of differences.

assertSetEqual(*set1*, *set2*, *msg=None*)
A set-specific equality assertion.

Parameters

- **set1** – The first set to compare.
- **set2** – The second set to compare.
- **msg** – Optional message to use on failure instead of a list of differences.

assertSetEqual uses ducktyping to support different types of sets, and is optimized for sets specifically (parameters must support a difference method).

assertTrue(*expr*, *msg=None*)
Check that the expression is true.

assertTupleEqual(*tuple1*, *tuple2*, *msg=None*)
A tuple-specific equality assertion.

Parameters

- **tuple1** – The first tuple to compare.
- **tuple2** – The second tuple to compare.
- **msg** – Optional message to use on failure instead of a list of differences.

assert_(*expr*, *msg=None*)
Check that the expression is true.

countTestCases()

debug()
Run the test without collecting errors in a TestResult

defaultTestResult()

doCleanups()
Execute all cleanup functions. Normally called for you after tearDown.

fail(*msg=None*)
Fail immediately, with the given message.

failIf(**args*, ***kwargs*)

failIfAlmostEqual(**args*, ***kwargs*)

failIfEqual(**args*, ***kwargs*)

failUnless(**args*, ***kwargs*)

failUnlessAlmostEqual(**args*, ***kwargs*)

failUnlessEqual(**args*, ***kwargs*)

failUnlessRaises(**args*, ***kwargs*)

failureException
alias of `AssertionError`

id()

longMessage = False

maxDiff = 640

run(*result=None*)

setUp()
Hook method for setting up the test fixture before exercising it.

setUpClass()
Hook method for setting up class fixture before running tests in the class.

shortDescription()
Returns a one-line description of the test, or None if no description has been provided.

The default implementation of this method returns the first line of the specified test method's docstring.

skipTest(*reason*)
Skip this test.

tearDown()
Hook method for deconstructing the test fixture after testing it.

tearDownClass()
Hook method for deconstructing the class fixture after running all tests in the class.

test_blogpost_introduction()
Test of introduction blog post examples.

test_blogpost_introduction_runner()

test_reinforceio_homepage()
Code example from the homepage and README.md.

tensorforce.tests.test_vpg_agent module

class tensorforce.tests.test_vpg_agent.**TestVPGAgent**(*methodName='runTest'*)
Bases: *tensorforce.tests.base_agent_test.BaseAgentTest*, unittest.case.TestCase

__init__(*methodName='runTest'*)
Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

addCleanup(*function*, **args*, ***kwargs*)
Add a function, with arguments, to be called when the test is completed. Functions added are called on a LIFO basis and are called after tearDown on test failure or success.

Cleanup items are called even if setUp fails (unlike tearDown).

addTypeEqualityFunc(*typeobj*, *function*)
Add a type specific assertEqual style function to compare a type.

This method is for use by TestCase subclasses that need to register their own type equality functions to provide nicer error messages.

Parameters

- **typeobj** – The data type to call this function on when both values are of the same type in assertEqual().
- **function** – The callable taking two arguments and an optional msg= argument that raises self.failureException with a useful error message when the two arguments are not equal.

agent
alias of VPGAgent

assertAlmostEqual(*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal

places (default 7) and comparing to zero, or by comparing that the between the two objects is more than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

If the two objects compare equal then they will automatically compare almost equal.

assertAlmostEquals(*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is more than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

If the two objects compare equal then they will automatically compare almost equal.

assertDictContainsSubset(*expected*, *actual*, *msg=None*)
Checks whether actual is a superset of expected.

assertDictEqual(*d1*, *d2*, *msg=None*)

assertEqual(*first*, *second*, *msg=None*)
Fail if the two objects are unequal as determined by the '==' operator.

assertEquals(*first*, *second*, *msg=None*)
Fail if the two objects are unequal as determined by the '==' operator.

assertFalse(*expr*, *msg=None*)
Check that the expression is false.

assertGreater(*a*, *b*, *msg=None*)
Just like self.assertTrue(a > b), but with a nicer default message.

assertGreaterEqual(*a*, *b*, *msg=None*)
Just like self.assertTrue(a >= b), but with a nicer default message.

assertIn(*member*, *container*, *msg=None*)
Just like self.assertTrue(a in b), but with a nicer default message.

assertIs(*expr1*, *expr2*, *msg=None*)
Just like self.assertTrue(a is b), but with a nicer default message.

assertIsInstance(*obj*, *cls*, *msg=None*)
Same as self.assertTrue(isinstance(obj, cls)), with a nicer default message.

assertIsNone(*obj*, *msg=None*)
Same as self.assertTrue(obj is None), with a nicer default message.

assertIsNot(*expr1*, *expr2*, *msg=None*)
Just like self.assertTrue(a is not b), but with a nicer default message.

assertIsNotNone(*obj*, *msg=None*)
Included for symmetry with assertIsNone.

assertItemsEqual(*expected_seq*, *actual_seq*, *msg=None*)
An unordered sequence specific comparison. It asserts that actual_seq and expected_seq have the same element counts. Equivalent to:

Asserts that each element has the same count in both sequences. .. rubric:: Example

- [0, 1, 1] and [1, 0, 1] compare equal.
- [0, 0, 1] and [0, 1] compare unequal.

assertLess(*a*, *b*, *msg=None*)
Just like self.assertTrue(a < b), but with a nicer default message.

assertLessEqual(*a*, *b*, *msg=None*)
Just like self.assertTrue(a <= b), but with a nicer default message.

assertListEqual(*list1*, *list2*, *msg=None*)
A list-specific equality assertion.

Parameters

- **list1** – The first list to compare.
- **list2** – The second list to compare.
- **msg** – Optional message to use on failure instead of a list of differences.

assertMultiLineEqual(*first*, *second*, *msg=None*)
Assert that two multi-line strings are equal.

assertNotAlmostEqual(*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is less than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

Objects that are equal automatically fail.

assertNotAlmostEquals(*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is less than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

Objects that are equal automatically fail.

assertNotEqual(*first*, *second*, *msg=None*)
Fail if the two objects are equal as determined by the '!=' operator.

assertNotEquals(*first*, *second*, *msg=None*)
Fail if the two objects are equal as determined by the '!=' operator.

assertNotIn(*member*, *container*, *msg=None*)
Just like self.assertTrue(a not in b), but with a nicer default message.

assertNotIsInstance(*obj*, *cls*, *msg=None*)
Included for symmetry with assertIsInstance.

assertNotRegexpMatches(*text*, *unexpected_regexp*, *msg=None*)
Fail the test if the text matches the regular expression.

assertRaises(*excClass*, *callableObj=None*, **args*, ***kwargs*)
Fail unless an exception of class excClass is raised by callableObj when invoked with arguments args and keyword arguments kwargs. If a different type of exception is raised, it will not be caught, and the test case will be deemed to have suffered an error, exactly as for an unexpected exception.

If called with callableObj omitted or None, will return a context object used like this:

The context manager keeps a reference to the exception as the 'exception' attribute. This allows you to inspect the exception after the assertion:

assertRaisesRegexp(*expected_exception*, *expected_regexp*, *callable_obj=None*, **args*, ***kwargs*)
Asserts that the message in a raised exception matches a regexp.

Parameters

- **expected_exception** – Exception class expected to be raised.
- **expected_regexp** – Regexp (re pattern object or string) expected to be found in error message.
- **callable_obj** – Function to be called.
- **args** – Extra args.
- **kwargs** – Extra kwargs.

assertRegexpMatches(*text*, *expected_regexp*, *msg=None*)
Fail the test unless the text matches the regular expression.

assertSequenceEqual(*seq1*, *seq2*, *msg=None*, *seq_type=None*)
An equality assertion for ordered sequences (like lists and tuples).

For the purposes of this function, a valid ordered sequence type is one which can be indexed, has a length, and has an equality operator.

Parameters

- **seq1** – The first sequence to compare.
- **seq2** – The second sequence to compare.
- **seq_type** – The expected datatype of the sequences, or None if no datatype should be enforced.
- **msg** – Optional message to use on failure instead of a list of differences.

assertSetEqual(*set1*, *set2*, *msg=None*)
A set-specific equality assertion.

Parameters

- **set1** – The first set to compare.
- **set2** – The second set to compare.
- **msg** – Optional message to use on failure instead of a list of differences.

assertSetEqual uses ducktyping to support different types of sets, and is optimized for sets specifically (parameters must support a difference method).

assertTrue(*expr*, *msg=None*)
Check that the expression is true.

assertTupleEqual(*tuple1*, *tuple2*, *msg=None*)
A tuple-specific equality assertion.

Parameters

- **tuple1** – The first tuple to compare.
- **tuple2** – The second tuple to compare.
- **msg** – Optional message to use on failure instead of a list of differences.

assert_(*expr*, *msg=None*)
Check that the expression is true.

base_test_pass(*name*, *environment*, *network*, ***kwargs*)
Basic test loop, requires an Agent to achieve a certain performance on an environment.

Parameters

- **name** (*str*) – The name of the test.
- **environment** (Environment) – The Environment object to use for the test.
- **network** (LayerBasedNetwork) – The Network to use for the agent's model.
- **kwargs** (*any*) – Agent arguments.

base_test_run(*name*, *environment*, *network*, ***kwargs*)
Run test, tests whether algorithm can run and update without compilation errors, not whether it passes.

Parameters

- **name** (*str*) – The name of the test.
- **environment** (Environment) – The Environment object to use for the test.
- **network** (LayerBasedNetwork) – The Network to use for the agent's model.
- **kwargs** (*any*) – Agent arguments.

config = {'update_mode': {'frequency': 4, 'batch_size': 4, 'unit': 'episodes'}, 'o

countTestCases()

debug()
Run the test without collecting errors in a TestResult

defaultTestResult()

doCleanups()
Execute all cleanup functions. Normally called for you after tearDown.

exclude_bool = False

exclude_bounded = False

exclude_float = False

exclude_int = False

exclude_lstm = False

exclude_multi = False

fail(*msg=None*)
Fail immediately, with the given message.

failIf(**args*, ***kwargs*)

failIfAlmostEqual(**args*, ***kwargs*)

failIfEqual(**args*, ***kwargs*)

failUnless(**args*, ***kwargs*)

failUnlessAlmostEqual(**args*, ***kwargs*)

failUnlessEqual(**args*, ***kwargs*)

failUnlessRaises(**args*, ***kwargs*)

failureException
alias of AssertionError

id()

longMessage = False

maxDiff = 640

multi_config = None

pass_threshold = 0.8

pre_run(*agent*, *environment*)
Called before `Runner.run`.

requires_network = True

run(*result=None*)

setUp()
Hook method for setting up the test fixture before exercising it.

setUpClass()
Hook method for setting up class fixture before running tests in the class.

shortDescription()
Returns a one-line description of the test, or None if no description has been provided.

The default implementation of this method returns the first line of the specified test method's docstring.

skipTest(*reason*)
Skip this test.

tearDown()
Hook method for deconstructing the test fixture after testing it.

tearDownClass()
Hook method for deconstructing the class fixture after running all tests in the class.

test_bool()
Tests the case of one boolean action.

test_bounded_float()
Tests the case of one bounded float action, i.e. with min and max value.

test_float()
Tests the case of one float action.

test_int()
Tests the case of one integer action.

test_lstm()
Tests the case of using internal states via an LSTM layer (for one integer action).

test_multi()
Tests the case of multiple actions of different type and shape.

tensorforce.tests.test_vpg_baselines module

class tensorforce.tests.test_vpg_baselines.**TestVPGBaselines**(*methodName='runTest'*)
Bases: *tensorforce.tests.base_test.BaseTest*, unittest.case.TestCase

__init__(*methodName='runTest'*)
Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

addCleanup (*function*, **args*, ***kwargs*)
Add a function, with arguments, to be called when the test is completed. Functions added are called on a LIFO basis and are called after tearDown on test failure or success.

Cleanup items are called even if setUp fails (unlike tearDown).

addTypeEqualityFunc (*typeobj*, *function*)
Add a type specific assertEqual style function to compare a type.

This method is for use by TestCase subclasses that need to register their own type equality functions to provide nicer error messages.

Parameters

- **typeobj** – The data type to call this function on when both values are of the same type in assertEqual().
- **function** – The callable taking two arguments and an optional msg= argument that raises self.failureException with a useful error message when the two arguments are not equal.

agent
alias of `VPGAgent`

assertAlmostEqual (*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is more than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

If the two objects compare equal then they will automatically compare almost equal.

assertAlmostEquals (*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is more than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

If the two objects compare equal then they will automatically compare almost equal.

assertDictContainsSubset (*expected*, *actual*, *msg=None*)
Checks whether actual is a superset of expected.

assertDictEqual (*d1*, *d2*, *msg=None*)

assertEqual (*first*, *second*, *msg=None*)
Fail if the two objects are unequal as determined by the '==' operator.

assertEquals (*first*, *second*, *msg=None*)
Fail if the two objects are unequal as determined by the '==' operator.

assertFalse (*expr*, *msg=None*)
Check that the expression is false.

assertGreater (*a*, *b*, *msg=None*)
Just like self.assertTrue(a > b), but with a nicer default message.

assertGreaterEqual (*a*, *b*, *msg=None*)
Just like self.assertTrue(a >= b), but with a nicer default message.

assertIn(*member*, *container*, *msg=None*)
Just like self.assertTrue(a in b), but with a nicer default message.

assertIs(*expr1*, *expr2*, *msg=None*)
Just like self.assertTrue(a is b), but with a nicer default message.

assertIsInstance(*obj*, *cls*, *msg=None*)
Same as self.assertTrue(isinstance(obj, cls)), with a nicer default message.

assertIsNone(*obj*, *msg=None*)
Same as self.assertTrue(obj is None), with a nicer default message.

assertIsNot(*expr1*, *expr2*, *msg=None*)
Just like self.assertTrue(a is not b), but with a nicer default message.

assertIsNotNone(*obj*, *msg=None*)
Included for symmetry with assertIsNone.

assertItemsEqual(*expected_seq*, *actual_seq*, *msg=None*)
An unordered sequence specific comparison. It asserts that actual_seq and expected_seq have the same element counts. Equivalent to:

Asserts that each element has the same count in both sequences. .. rubric:: Example

- [0, 1, 1] and [1, 0, 1] compare equal.
- [0, 0, 1] and [0, 1] compare unequal.

assertLess(*a*, *b*, *msg=None*)
Just like self.assertTrue(a < b), but with a nicer default message.

assertLessEqual(*a*, *b*, *msg=None*)
Just like self.assertTrue(a <= b), but with a nicer default message.

assertListEqual(*list1*, *list2*, *msg=None*)
A list-specific equality assertion.

Parameters

- **list1** – The first list to compare.
- **list2** – The second list to compare.
- **msg** – Optional message to use on failure instead of a list of differences.

assertMultiLineEqual(*first*, *second*, *msg=None*)
Assert that two multi-line strings are equal.

assertNotAlmostEqual(*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is less than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

Objects that are equal automatically fail.

assertNotAlmostEquals(*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is less than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

Objects that are equal automatically fail.

assertNotEqual(*first*, *second*, *msg=None*)
Fail if the two objects are equal as determined by the '!=' operator.

assertNotEquals(*first*, *second*, *msg=None*)
Fail if the two objects are equal as determined by the '!=' operator.

assertNotIn(*member*, *container*, *msg=None*)
Just like self.assertTrue(a not in b), but with a nicer default message.

assertNotIsInstance(*obj*, *cls*, *msg=None*)
Included for symmetry with assertIsInstance.

assertNotRegexpMatches(*text*, *unexpected_regexp*, *msg=None*)
Fail the test if the text matches the regular expression.

assertRaises(*excClass*, *callableObj=None*, **args*, ***kwargs*)
Fail unless an exception of class excClass is raised by callableObj when invoked with arguments args and keyword arguments kwargs. If a different type of exception is raised, it will not be caught, and the test case will be deemed to have suffered an error, exactly as for an unexpected exception.

If called with callableObj omitted or None, will return a context object used like this:

The context manager keeps a reference to the exception as the 'exception' attribute. This allows you to inspect the exception after the assertion:

assertRaisesRegexp(*expected_exception*, *expected_regexp*, *callable_obj=None*, **args*, ***kwargs*)
Asserts that the message in a raised exception matches a regexp.

Parameters

- **expected_exception** – Exception class expected to be raised.
- **expected_regexp** – Regexp (re pattern object or string) expected to be found in error message.
- **callable_obj** – Function to be called.
- **args** – Extra args.
- **kwargs** – Extra kwargs.

assertRegexpMatches(*text*, *expected_regexp*, *msg=None*)
Fail the test unless the text matches the regular expression.

assertSequenceEqual(*seq1*, *seq2*, *msg=None*, *seq_type=None*)
An equality assertion for ordered sequences (like lists and tuples).

For the purposes of this function, a valid ordered sequence type is one which can be indexed, has a length, and has an equality operator.

Parameters

- **seq1** – The first sequence to compare.
- **seq2** – The second sequence to compare.
- **seq_type** – The expected datatype of the sequences, or None if no datatype should be enforced.
- **msg** – Optional message to use on failure instead of a list of differences.

assertSetEqual(*set1*, *set2*, *msg=None*)
A set-specific equality assertion.

Parameters

- **set1** – The first set to compare.
- **set2** – The second set to compare.
- **msg** – Optional message to use on failure instead of a list of differences.

assertSetEqual uses ducktyping to support different types of sets, and is optimized for sets specifically (parameters must support a difference method).

assertTrue(*expr*, *msg=None*)
Check that the expression is true.

assertTupleEqual(*tuple1*, *tuple2*, *msg=None*)
A tuple-specific equality assertion.

Parameters

- **tuple1** – The first tuple to compare.
- **tuple2** – The second tuple to compare.
- **msg** – Optional message to use on failure instead of a list of differences.

assert_(*expr*, *msg=None*)
Check that the expression is true.

base_test_pass(*name*, *environment*, *network*, ***kwargs*)
Basic test loop, requires an Agent to achieve a certain performance on an environment.

Parameters

- **name** (*str*) – The name of the test.
- **environment** (Environment) – The Environment object to use for the test.
- **network** (LayerBasedNetwork) – The Network to use for the agent's model.
- **kwargs** (*any*) – Agent arguments.

base_test_run(*name*, *environment*, *network*, ***kwargs*)
Run test, tests whether algorithm can run and update without compilation errors, not whether it passes.

Parameters

- **name** (*str*) – The name of the test.
- **environment** (Environment) – The Environment object to use for the test.
- **network** (LayerBasedNetwork) – The Network to use for the agent's model.
- **kwargs** (*any*) – Agent arguments.

countTestCases()

debug()
Run the test without collecting errors in a TestResult

defaultTestResult()

doCleanups()
Execute all cleanup functions. Normally called for you after tearDown.

fail(*msg=None*)
Fail immediately, with the given message.

failIf(**args*, ***kwargs*)

failIfAlmostEqual(**args*, ***kwargs*)

failIfEqual(**args*, ***kwargs*)

failUnless(**args*, ***kwargs*)

failUnlessAlmostEqual(**args*, ***kwargs*)

failUnlessEqual(**args*, ***kwargs*)

failUnlessRaises(**args*, ***kwargs*)

failureException
alias of `AssertionError`

id()

longMessage = False

maxDiff = 640

pass_threshold = 0.8

pre_run(*agent*, *environment*)
Called before `Runner.run`.

requires_network = True

run(*result=None*)

setUp()
Hook method for setting up the test fixture before exercising it.

setUpClass()
Hook method for setting up class fixture before running tests in the class.

shortDescription()
Returns a one-line description of the test, or None if no description has been provided.

The default implementation of this method returns the first line of the specified test method's docstring.

skipTest(*reason*)
Skip this test.

tearDown()
Hook method for deconstructing the test fixture after testing it.

tearDownClass()
Hook method for deconstructing the class fixture after running all tests in the class.

test_baseline_no_optimizer()

test_gae_baseline()

test_multi_baseline()

test_network_baseline()

test_states_baseline()

tensorforce.tests.test_vpg_optimizers module

class tensorforce.tests.test_vpg_optimizers.**TestVPGOptimizers**(*methodName='runTest'*)
Bases: *tensorforce.tests.base_test.BaseTest*, unittest.case.TestCase

__init__ (*methodName='runTest'*)
Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

addCleanup (*function*, **args*, ***kwargs*)
Add a function, with arguments, to be called when the test is completed. Functions added are called on a LIFO basis and are called after tearDown on test failure or success.

Cleanup items are called even if setUp fails (unlike tearDown).

addTypeEqualityFunc (*typeobj*, *function*)
Add a type specific assertEqual style function to compare a type.

This method is for use by TestCase subclasses that need to register their own type equality functions to provide nicer error messages.

Parameters

- **typeobj** – The data type to call this function on when both values are of the same type in assertEqual().
- **function** – The callable taking two arguments and an optional msg= argument that raises self.failureException with a useful error message when the two arguments are not equal.

agent
alias of `VPGAgent`

assertAlmostEqual (*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is more than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

If the two objects compare equal then they will automatically compare almost equal.

assertAlmostEquals (*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is more than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

If the two objects compare equal then they will automatically compare almost equal.

assertDictContainsSubset (*expected*, *actual*, *msg=None*)
Checks whether actual is a superset of expected.

assertDictEqual (*d1*, *d2*, *msg=None*)

assertEqual (*first*, *second*, *msg=None*)
Fail if the two objects are unequal as determined by the '==' operator.

assertEquals (*first*, *second*, *msg=None*)
Fail if the two objects are unequal as determined by the '==' operator.

assertFalse (*expr*, *msg=None*)
Check that the expression is false.

assertGreater (*a*, *b*, *msg=None*)
Just like self.assertTrue(a > b), but with a nicer default message.

assertGreaterEqual(*a*, *b*, *msg=None*)
Just like self.assertTrue(a >= b), but with a nicer default message.

assertIn(*member*, *container*, *msg=None*)
Just like self.assertTrue(a in b), but with a nicer default message.

assertIs(*expr1*, *expr2*, *msg=None*)
Just like self.assertTrue(a is b), but with a nicer default message.

assertIsInstance(*obj*, *cls*, *msg=None*)
Same as self.assertTrue(isinstance(obj, cls)), with a nicer default message.

assertIsNone(*obj*, *msg=None*)
Same as self.assertTrue(obj is None), with a nicer default message.

assertIsNot(*expr1*, *expr2*, *msg=None*)
Just like self.assertTrue(a is not b), but with a nicer default message.

assertIsNotNone(*obj*, *msg=None*)
Included for symmetry with assertIsNone.

assertItemsEqual(*expected_seq*, *actual_seq*, *msg=None*)
An unordered sequence specific comparison. It asserts that actual_seq and expected_seq have the same element counts. Equivalent to:

Asserts that each element has the same count in both sequences. .. rubric:: Example

- [0, 1, 1] and [1, 0, 1] compare equal.
- [0, 0, 1] and [0, 1] compare unequal.

assertLess(*a*, *b*, *msg=None*)
Just like self.assertTrue(a < b), but with a nicer default message.

assertLessEqual(*a*, *b*, *msg=None*)
Just like self.assertTrue(a <= b), but with a nicer default message.

assertListEqual(*list1*, *list2*, *msg=None*)
A list-specific equality assertion.

Parameters

- **list1** – The first list to compare.
- **list2** – The second list to compare.
- **msg** – Optional message to use on failure instead of a list of differences.

assertMultiLineEqual(*first*, *second*, *msg=None*)
Assert that two multi-line strings are equal.

assertNotAlmostEqual(*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is less than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

Objects that are equal automatically fail.

assertNotAlmostEquals(*first*, *second*, *places=None*, *msg=None*, *delta=None*)
Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the between the two objects is less than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

Objects that are equal automatically fail.

assertNotEqual(*first*, *second*, *msg=None*)
Fail if the two objects are equal as determined by the '!=' operator.

assertNotEquals(*first*, *second*, *msg=None*)
Fail if the two objects are equal as determined by the '!=' operator.

assertNotIn(*member*, *container*, *msg=None*)
Just like self.assertTrue(a not in b), but with a nicer default message.

assertNotIsInstance(*obj*, *cls*, *msg=None*)
Included for symmetry with assertIsInstance.

assertNotRegexpMatches(*text*, *unexpected_regexp*, *msg=None*)
Fail the test if the text matches the regular expression.

assertRaises(*excClass*, *callableObj=None*, **args*, ***kwargs*)
Fail unless an exception of class excClass is raised by callableObj when invoked with arguments args and keyword arguments kwargs. If a different type of exception is raised, it will not be caught, and the test case will be deemed to have suffered an error, exactly as for an unexpected exception.

If called with callableObj omitted or None, will return a context object used like this:

The context manager keeps a reference to the exception as the 'exception' attribute. This allows you to inspect the exception after the assertion:

assertRaisesRegexp(*expected_exception*, *expected_regexp*, *callable_obj=None*, **args*, ***kwargs*)
Asserts that the message in a raised exception matches a regexp.

Parameters

- **expected_exception** – Exception class expected to be raised.
- **expected_regexp** – Regexp (re pattern object or string) expected to be found in error message.
- **callable_obj** – Function to be called.
- **args** – Extra args.
- **kwargs** – Extra kwargs.

assertRegexpMatches(*text*, *expected_regexp*, *msg=None*)
Fail the test unless the text matches the regular expression.

assertSequenceEqual(*seq1*, *seq2*, *msg=None*, *seq_type=None*)
An equality assertion for ordered sequences (like lists and tuples).

For the purposes of this function, a valid ordered sequence type is one which can be indexed, has a length, and has an equality operator.

Parameters

- **seq1** – The first sequence to compare.
- **seq2** – The second sequence to compare.
- **seq_type** – The expected datatype of the sequences, or None if no datatype should be enforced.
- **msg** – Optional message to use on failure instead of a list of differences.

assertSetEqual(*set1*, *set2*, *msg=None*)
A set-specific equality assertion.

Parameters

- **set1** – The first set to compare.
- **set2** – The second set to compare.
- **msg** – Optional message to use on failure instead of a list of differences.

assertSetEqual uses ducktyping to support different types of sets, and is optimized for sets specifically (parameters must support a difference method).

assertTrue(*expr*, *msg=None*)
Check that the expression is true.

assertTupleEqual(*tuple1*, *tuple2*, *msg=None*)
A tuple-specific equality assertion.

Parameters

- **tuple1** – The first tuple to compare.
- **tuple2** – The second tuple to compare.
- **msg** – Optional message to use on failure instead of a list of differences.

assert_(*expr*, *msg=None*)
Check that the expression is true.

base_test_pass(*name*, *environment*, *network*, ***kwargs*)
Basic test loop, requires an Agent to achieve a certain performance on an environment.

Parameters

- **name** (*str*) – The name of the test.
- **environment** (Environment) – The Environment object to use for the test.
- **network** (LayerBasedNetwork) – The Network to use for the agent's model.
- **kwargs** (*any*) – Agent arguments.

base_test_run(*name*, *environment*, *network*, ***kwargs*)
Run test, tests whether algorithm can run and update without compilation errors, not whether it passes.

Parameters

- **name** (*str*) – The name of the test.
- **environment** (Environment) – The Environment object to use for the test.
- **network** (LayerBasedNetwork) – The Network to use for the agent's model.
- **kwargs** (*any*) – Agent arguments.

countTestCases()

debug()
Run the test without collecting errors in a TestResult

defaultTestResult()

doCleanups()
Execute all cleanup functions. Normally called for you after tearDown.

fail(*msg=None*)
Fail immediately, with the given message.

failIf(**args*, ***kwargs*)

failIfAlmostEqual(**args*, ***kwargs*)

failIfEqual(**args*, ***kwargs*)

failUnless(**args*, ***kwargs*)

failUnlessAlmostEqual(**args*, ***kwargs*)

failUnlessEqual(**args*, ***kwargs*)

failUnlessRaises(**args*, ***kwargs*)

failureException
alias of `AssertionError`

id()

longMessage = False

maxDiff = 640

pass_threshold = 0.8

pre_run(*agent*, *environment*)
Called before `Runner.run`.

requires_network = True

run(*result=None*)

setUp()
Hook method for setting up the test fixture before exercising it.

setUpClass()
Hook method for setting up class fixture before running tests in the class.

shortDescription()
Returns a one-line description of the test, or None if no description has been provided.

The default implementation of this method returns the first line of the specified test method's docstring.

skipTest(*reason*)
Skip this test.

tearDown()
Hook method for deconstructing the test fixture after testing it.

tearDownClass()
Hook method for deconstructing the class fixture after running all tests in the class.

test_adam()

test_clipped_step()

test_evolutionary()

test_multi_step()

test_natural_gradient()

test_optimized_step()

test_subsampling_step()

Module contents

1.6.2 Submodules

1.6.3 tensorforce.exception module

exception tensorforce.exception.**TensorForceError**
Bases: exceptions.Exception

TensorForce error

__init__
x.**init**(...) initializes x; see help(type(x)) for signature

args

message

1.6.4 tensorforce.meta_parameter_recorder module

class tensorforce.meta_parameter_recorder.**MetaParameterRecorder**(*current_frame*)
Bases: object

Class to record MetaParameters as well as Summary/Description for TensorBoard (TEXT & FILE will come later).

General:

- format_type: used to configure data conversion for TensorBoard=0, TEXT & JSON (not Implemented), etc

__init__(*current_frame*)
Init the MetaPrameterRecord with "Agent" parameters by passing inspect.currentframe() from Agent Class.

The Init will search back to find the parent class to capture all passed parameters and store them in "self.meta_params".

NOTE: Currently only optimized for TensorBoard output.

TODO: Add JSON Export, TEXT EXPORT

Parameters **current_frame** – Frame value from class to obtain metaparameters[= inspect.currentframe()]

build_metagraph_list()
Convert MetaParams into TF Summary Format and create summary_op.

Returns Merged TF Op for TEXT summary elements, should only be executed once to reduce data duplication.

convert_data_to_string(*data*, *indent=0*, *format_type=0*, *separator=None*, *eol=None*)

convert_dictionary_to_string(*data*, *indent=0*, *format_type=0*, *separator=None*, *eol=None*)

convert_list_to_string(*data*, *indent=0*, *format_type=0*, *eol=None*, *count=True*)

convert_ndarray_to_md(*data*, *format_type=0*, *eol=None*)

merge_custom(*custom_dict*)

text_output(*format_type=1*)

1.6.5 tensorforce.util module

class tensorforce.util.**SavableComponent**
Bases: object

Component that can save and restore its own state.

__init__
x.**init**(...) initializes x; see help(type(x)) for signature

get_savable_variables()
Returns the list of all the variables this component is responsible to save and restore.

Returns The list of variables that will be saved or restored.

register_saver_ops()
Registers the saver operations to the graph in context.

restore(*sess*, *save_path*)
Restores the values of the managed variables from disk location.

Parameters

- **sess** – The session for which to save the managed variables.
- **save_path** – The path used to save the data to.

save(*sess*, *save_path*, *timestep=None*)
Saves this component's managed variables.

Parameters

- **sess** – The session for which to save the managed variables.
- **save_path** – The path to save data to.
- **timestep** – Optional, the timestep to append to the file name.

Returns Checkpoint path where the model was saved.

tensorforce.util.**get_object**(*obj*, *predefined_objects=None*, *default_object=None*, *kwargs=None*)
Utility method to map some kind of object specification to its content, e.g. optimizer or baseline specifications to the respective classes.

Parameters

- **obj** – A specification dict (value for key 'type' optionally specifies the object, options as follows), a module path (e.g., my_module.MyClass), a key in predefined_objects, or a callable (e.g., the class type object).
- **predefined_objects** – Dict containing predefined set of objects, accessible via their key
- **default_object** – Default object is no other is specified
- **kwargs** – Arguments for object creation

Returns: The retrieved object

tensorforce.util.**map_tensors**(*fn*, *tensors*)

tensorforce.util.**np_dtype**(*dtype*)
Translates dtype specifications in configurations to numpy data types. :param dtype: String describing a numerical type (e.g. 'float') or numerical type primitive.

Returns: Numpy data type

tensorforce.util.**prepare_kwargs**(*raw*, *string_parameter='name'*)
Utility method to convert raw string/diction input into a dictionary to pass into a function. Always returns a dictionary.

Parameters **raw** – string or dictionary, string is assumed to be the name of the activation activation function. Dictionary will be passed through unchanged.

Returns: kwargs dictionary for **kwargs

tensorforce.util.**prod**(*xs*)
Computes the product along the elements in an iterable. Returns 1 for empty iterable.

Parameters **xs** – Iterable containing numbers.

Returns: Product along iterable.

tensorforce.util.**rank**(*x*)

tensorforce.util.**shape**(*x*, *unknown=-1*)

tensorforce.util.**strip_name_scope**(*name*, *base_scope*)

tensorforce.util.**tf_dtype**(*dtype*)
Translates dtype specifications in configurations to tensorflow data types.

Parameters **dtype** – String describing a numerical type (e.g. 'float'), numpy data type, or numerical type primitive.

Returns: TensorFlow data type

1.6.6 Module contents

exception tensorforce.**TensorForceError**
Bases: exceptions.Exception

TensorForce error

__init__
x.**init**(...) initializes x; see help(type(x)) for signature

args

message

CHAPTER 2

More information

You can find more information at our TensorForce GitHub repository.

We have a seperate repository available for benchmarking our algorithm implementations [here](https://github.com/reinforceio/tensorforce-benchmark).

Python Module Index

t

Index

Symbols

A

B

C

D

E

F

G

I

L

M

N

O

P

Q

R

S

T

U

V

W