TensorForce Documentation

Release 0.3.3

reinforce.io

Dec 30, 2017

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)

# 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_spec*, *actions_spec*, *batched_observe=1000*, *scope='base_agent'*)

Bases: `object`

Basic Reinforcement learning agent. An agent encapsulates execution logic of a particular reinforcement learning algorithm and defines the external interface to the environment.

The agent hence acts as an intermediate layer between environment and backend execution (value function or policy updates).

act(*states*, *deterministic=False*)

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.

Returns Scalar value of the action or dict of multiple actions the agent wants to execute.

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 on episode start. Updates internal episode and timestep 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_spec*, *actions_spec*, *device*, *session_config*, *scope*, *saver_spec*, *summary_spec*, *distributed_spec*, *optimizer*, *discount*, *variable_noise*, *states_preprocessing_spec*, *explorations_spec*, *reward_preprocessing_spec*)

Bases: `object`

Base class for all (TensorFlow-based) models.

create_output_operations(*states*, *internals*, *actions*, *terminal*, *reward*, *update*, *deterministic*)

Calls all the relevant TensorFlow functions for this model and hence creates all the TensorFlow operations involved.

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.
- **update** – Boolean tensor indicating whether this call happens during an update.
- **deterministic** – Boolean tensor indicating whether action should be chosen deterministically.

get_optimizer_kwargs(*states*, *internals*, *actions*, *terminal*, *reward*, *update*)

Returns the optimizer arguments including the time, the list of variables to optimize, and various argument-free functions (in particular `fn_loss` returning the combined 0-dim batch loss tensor) 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.
- **update** – Boolean tensor indicating whether this call happens during an update.

Returns Loss tensor of the size of the batch.

get_summaries()
Returns the TensorFlow summaries reported by the model
Returns List of summaries

get_variables(*include_non_trainable=False*)
Returns the TensorFlow variables used by the model.
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.

reset()
Resets the model to its initial state on episode start.
Returns Current episode and timestep counter, and a list containing the internal states initializations.

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.

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 were the model was saved.

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

tf_action_exploration(*action*, *exploration*, *action_spec*)
Applies optional exploration to the action.

tf_actions_and_internals(*states*, *internals*, *update*, *deterministic*)
Creates the TensorFlow operations for retrieving the actions (and posterior internal states) in reaction to the given input states (and prior internal 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.
- **deterministic** – Boolean tensor indicating whether action should be chosen deterministically.

Returns Actions and list of posterior internal state tensors.

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_loss_per_instance(*states*, *internals*, *actions*, *terminal*, *reward*, *update*)

Creates the TensorFlow operations for calculating the loss per batch instance of the given input states and actions.

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.
- **update** – Boolean tensor indicating whether this call happens during an update.

Returns Loss tensor.

tf_optimization(*states*, *internals*, *actions*, *terminal*, *reward*, *update*)

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.
- **update** – Boolean tensor indicating whether this call happens during an update.

Returns The optimization operation.

tf_preprocess_reward(*states*, *internals*, *terminal*, *reward*)

Applies optional pre-processing to the reward.

tf_preprocess_states(*states*)

Applies optional pre-processing to the states.

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.

MemoryAgent

class tensorforce.agents.**MemoryAgent**(*states_spec*, *actions_spec*, *batched_observe=1000*, *scope='memory_agent'*, *summary_spec=None*, *network_spec=None*, *discount=0.99*, *device=None*, *session_config=None*, *saver_spec=None*, *distributed_spec=None*, *optimizer=None*, *variable_noise=None*, *states_preprocessing_spec=None*, *explorations_spec=None*, *reward_preprocessing_spec=None*, *distributions_spec=None*, *entropy_regularization=None*, *batch_size=1000*, *memory=None*, *first_update=10000*, *update_frequency=4*, *repeat_update=1*)

Bases: tensorforce.agents.learning_agent.LearningAgent

The MemoryAgent class implements a replay memory from which it samples batches according to some sampling strategy to update the value function.

import_observations(*observations*)

Load an iterable of observation dicts into the replay memory.

Parameters **observations** – An iterable with each element containing an observation. Each observation requires keys 'state','action','reward','terminal', 'internal'. Use an empty list [] for 'internal' if internal state is irrelevant.

BatchAgent

class tensorforce.agents.**BatchAgent**(*states_spec*, *actions_spec*, *batched_observe=1000*, *summary_spec=None*, *network_spec=None*, *discount=0.99*, *device=None*, *session_config=None*, *scope='batch_agent'*, *saver_spec=None*, *distributed_spec=None*, *optimizer=None*, *variable_noise=None*, *states_preprocessing_spec=None*, *explorations_spec=None*, *reward_preprocessing_spec=None*, *distributions_spec=None*, *entropy_regularization=None*, *batch_size=1000*, *keep_last_timestep=True*)

Bases: tensorforce.agents.learning_agent.LearningAgent

The BatchAgent class implements a batch memory which generally implies on-policy experience collection and updates.

observe(*terminal*, *reward*)

Adds an observation and performs an update if the necessary conditions are satisfied, i.e. if one batch of experience has been collected as defined by the batch size.

In particular, note that episode control happens outside of the agent since the agent should be agnostic to how the training data is created.

Parameters

- **terminal** (*bool*) – Whether episode is terminated or not.
- **reward** (*float*) – The scalar reward value.

reset_batch()

Cleans up after a batch has been processed (observed). Resets all batch information to be ready for new observation data. Batch information contains:

- observed states
- internal-variables
- taken actions
- observed is-terminal signals/rewards
- total batch size

Deep-Q-Networks (DQN)

class tensorforce.agents.**DQNAgent**(*states_spec*, *actions_spec*, *batched_observe=None*, *scope='dqn'*, *summary_spec=None*, *network_spec=None*, *device=None*, *session_config=None*, *saver_spec=None*, *distributed_spec=None*, *optimizer=None*, *discount=0.99*, *variable_noise=None*, *states_preprocessing_spec=None*, *explorations_spec=None*, *reward_preprocessing_spec=None*, *distributions_spec=None*, *entropy_regularization=None*, *batch_size=32*, *memory=None*, *first_update=10000*, *update_frequency=4*, *repeat_update=1*, *target_sync_frequency=10000*, *target_update_weight=1.0*, *double_q_model=False*, *huber_loss=None*)

Bases: *tensorforce.agents.memory_agent.MemoryAgent*

Deep-Q-Network agent (DQN). The piece de resistance of deep reinforcement learning as described by Minh et al. (2015). Includes an option for double-DQN (DDQN; van Hasselt et al., 2015)

DQN chooses from one of a number of discrete actions by taking the maximum Q-value from the value function with one output neuron per available action. DQN uses a replay memory for experience playback.

Normalized Advantage Functions

class tensorforce.agents.**NAFAgent**(*states_spec*, *actions_spec*, *batched_observe=1000*, *scope='naf'*, *summary_spec=None*, *network_spec=None*, *device=None*, *session_config=None*, *saver_spec=None*, *distributed_spec=None*, *optimizer=None*, *discount=0.99*, *variable_noise=None*, *states_preprocessing_spec=None*, *explorations_spec=None*, *reward_preprocessing_spec=None*, *distributions_spec=None*, *entropy_regularization=None*, *batch_size=32*, *memory=None*, *first_update=10000*, *update_frequency=4*, *repeat_update=1*, *target_sync_frequency=10000*, *target_update_weight=1.0*, *double_q_model=False*, *huber_loss=None*)

Bases: *tensorforce.agents.memory_agent.MemoryAgent*

Normalized Advantage Functions (NAF) for continuous DQN: https://arxiv.org/abs/1603.00748

Deep-Q-learning from demonstration (DQFD)

class tensorforce.agents.**DQFDAgent**(*states_spec*, *actions_spec*, *batched_observe=1000*, *scope='dqfd'*, *summary_spec=None*, *network_spec=None*, *device=None*, *session_config=None*, *saver_spec=None*, *distributed_spec=None*, *optimizer=None*, *discount=0.99*, *variable_noise=None*, *states_preprocessing_spec=None*, *explorations_spec=None*, *reward_preprocessing_spec=None*, *distributions_spec=None*, *entropy_regularization=None*, *batch_size=32*, *memory=None*, *first_update=10000*, *update_frequency=4*, *repeat_update=1*, *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.memory_agent.MemoryAgent*

Deep Q-learning from demonstration (DQFD) agent (Hester et al., 2017). This agent uses DQN to pre-train from demonstration data via an additional supervised loss term.

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

observe(*reward*, *terminal*)
Adds observations, updates via sampling from memories according to update rate. DQFD samples from the online replay memory and the demo memory with the fractions controlled by a hyper parameter p called 'expert sampling ratio.

pretrain(*steps*)
Computes pre-train updates.
Parameters **steps** – Number of updates to execute.

set_demonstrations(*batch*)
Set all demonstrations from batch data. Expects a dict wherein each value contains an array containing all states, actions, rewards, terminals and internals respectively.
Parameters **batch** –

Vanilla Policy Gradient

class tensorforce.agents.**VPGAgent**(*states_spec*, *actions_spec*, *batched_observe=1000*, *scope='vpg'*, *summary_spec=None*, *network_spec=None*, *device=None*, *session_config=None*, *saver_spec=None*, *distributed_spec=None*, *optimizer=None*, *discount=0.99*, *variable_noise=None*, *states_preprocessing_spec=None*, *explorations_spec=None*, *reward_preprocessing_spec=None*, *distributions_spec=None*, *entropy_regularization=None*, *batch_size=1000*, *keep_last_timestep=True*, *baseline_mode=None*, *baseline=None*, *baseline_optimizer=None*, *gae_lambda=None*)
Bases: *tensorforce.agents.batch_agent.BatchAgent*

Vanilla Policy Gradient agent as described by [Sutton et al. (1999)] (https://papers.nips.cc/paper/1713-policy-gradient-methods-for-reinforcement-learning-with-function-approximation.pdf).

Trust Region Policy Optimization (TRPO)

class tensorforce.agents.**TRPOAgent**(*states_spec*, *actions_spec*, *batched_observe=1000*, *scope='trpo'*, *summary_spec=None*, *network_spec=None*, *device=None*, *session_config=None*, *saver_spec=None*, *distributed_spec=None*, *discount=0.99*, *variable_noise=None*, *states_preprocessing_spec=None*, *explorations_spec=None*, *reward_preprocessing_spec=None*, *distributions_spec=None*, *entropy_regularization=None*, *batch_size=1000*, *keep_last_timestep=True*, *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*)

Bases: *tensorforce.agents.batch_agent.BatchAgent*

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

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.

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

__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 ducandu plugin Observer 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 this step (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

class tensorforce.core.preprocessing.**Standardize**(*across_batch=False*, *scope='standardize'*, *summary_labels=()*)

Bases: *tensorforce.core.preprocessing.preprocessor.Preprocessor*

Standardize state. Subtract mean and divide by standard deviation.

Grayscale

class tensorforce.core.preprocessing.**Grayscale**(*weights=(0.299, 0.587, 0.114)*, *scope='grayscale'*, *summary_labels=()*)

Bases: *tensorforce.core.preprocessing.preprocessor.Preprocessor*

Turn 3D color state into grayscale.

ImageResize

class tensorforce.core.preprocessing.**ImageResize**(*width*, *height*, *scope='image_resize'*, *summary_labels=()*)

Bases: *tensorforce.core.preprocessing.preprocessor.Preprocessor*

Resize image to width x height.

Normalize

class tensorforce.core.preprocessing.**Normalize**(*scope='normalize'*, *summary_labels=()*)

Bases: *tensorforce.core.preprocessing.preprocessor.Preprocessor*

Normalize state. Subtract minimal value and divide by range.

Sequence

class tensorforce.core.preprocessing.**Sequence**(*length=2*, *scope='sequence'*, *summary_labels=()*)

Bases: *tensorforce.core.preprocessing.preprocessor.Preprocessor*

Concatenate `length` state vectors. Example: Used in Atari problems to create the Markov property.

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 summary_spec

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 "summary_spec" passed to the agent on initialization.

"summary_spec" 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_spec=...,
    actions_spec=...,
    network_spec=...,
    summary_spec=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 "summary_spec" 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_spec=...,
    actions_spec=...,
    network_spec=...,
    summary_spec=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
)
```

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)

    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_spec*, *actions_spec*, *batched_observe=1000*, *scope='base_agent'*)

Bases: object

Basic Reinforcement learning agent. An agent encapsulates execution logic of a particular reinforcement learning algorithm and defines the external interface to the environment.

The agent hence acts as an intermediate layer between environment and backend execution (value function or policy updates).

act(*states*, *deterministic=False*)

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.

Returns Scalar value of the action or dict of multiple actions the agent wants to execute.

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.

static process_action_spec(*actions_spec*)

static process_state_spec(*states_spec*)

reset()
Reset the agent to its initial state on episode start. Updates internal episode and timestep 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.

should_stop()

tensorforce.agents.batch_agent module

class tensorforce.agents.batch_agent.**BatchAgent**(*states_spec*, *actions_spec*, *batched_observe=1000*, *summary_spec=None*, *network_spec=None*, *discount=0.99*, *device=None*, *session_config=None*, *scope='batch_agent'*, *saver_spec=None*, *distributed_spec=None*, *optimizer=None*, *variable_noise=None*, *states_preprocessing_spec=None*, *explorations_spec=None*, *reward_preprocessing_spec=None*, *distributions_spec=None*, *entropy_regularization=None*, *batch_size=1000*, *keep_last_timestep=True*)
Bases: tensorforce.agents.learning_agent.LearningAgent

The BatchAgent class implements a batch memory which generally implies on-policy experience collection and updates.

observe(*terminal*, *reward*)
Adds an observation and performs an update if the necessary conditions are satisfied, i.e. if one batch of experience has been collected as defined by the batch size.

In particular, note that episode control happens outside of the agent since the agent should be agnostic to how the training data is created.

Parameters

- **terminal** (*bool*) – Whether episode is terminated or not.
- **reward** (*float*) – The scalar reward value.

reset_batch()
Cleans up after a batch has been processed (observed). Resets all batch information to be ready for new observation data. Batch information contains:

- observed states
- internal-variables
- taken actions
- observed is-terminal signals/rewards
- total batch size

tensorforce.agents.constant_agent module

Random agent that always returns a random action. Useful to be able to get random agents with specific shapes.

class tensorforce.agents.constant_agent.**ConstantAgent**(*states_spec*, *actions_spec*, *batched_observe=1000*, *scope='constant'*, *action_values=None*)
Bases: *tensorforce.agents.agent.Agent*

Constant action agent for sanity checks. Returns a constant value at every step, useful to debug continuous problems.

initialize_model()

tensorforce.agents.ddqn_agent module

class tensorforce.agents.ddqn_agent.**DDQNAgent**(*states_spec*, *actions_spec*, *batched_observe=1000*, *scope='ddqn'*, *summary_spec=None*, *network_spec=None*, *device=None*, *session_config=None*, *saver_spec=None*, *distributed_spec=None*, *optimizer=None*, *discount=0.99*, *variable_noise=None*, *states_preprocessing_spec=None*, *explorations_spec=None*, *reward_preprocessing_spec=None*, *distributions_spec=None*, *entropy_regularization=None*, *batch_size=32*, *memory=None*, *first_update=10000*, *update_frequency=4*, *repeat_update=1*, *target_sync_frequency=10000*, *target_update_weight=1.0*, *huber_loss=None*)

Bases: *tensorforce.agents.memory_agent.MemoryAgent*

Double DQN Agent based on Van Hasselt et al.. Simple extension to DQN which improves stability.

initialize_model()

tensorforce.agents.dqfd_agent module

class tensorforce.agents.dqfd_agent.**DQFDAgent**(*states_spec*, *actions_spec*, *batched_observe=1000*, *scope='dqfd'*, *summary_spec=None*, *network_spec=None*, *device=None*, *session_config=None*, *saver_spec=None*, *distributed_spec=None*, *optimizer=None*, *discount=0.99*, *variable_noise=None*, *states_preprocessing_spec=None*, *explorations_spec=None*, *reward_preprocessing_spec=None*, *distributions_spec=None*, *entropy_regularization=None*, *batch_size=32*, *memory=None*, *first_update=10000*, *update_frequency=4*, *repeat_update=1*, *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.memory_agent.MemoryAgent*

Deep Q-learning from demonstration (DQFD) agent (Hester et al., 2017). This agent uses DQN to pre-train from demonstration data via an additional supervised loss term.

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

initialize_model()

observe(*reward*, *terminal*)
Adds observations, updates via sampling from memories according to update rate. DQFD samples from the online replay memory and the demo memory with the fractions controlled by a hyper parameter p called 'expert sampling ratio.

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

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

set_demonstrations(*batch*)
Set all demonstrations from batch data. Expects a dict wherein each value contains an array containing all states, actions, rewards, terminals and internals respectively.

Parameters **batch** –

tensorforce.agents.dqn_agent module

class tensorforce.agents.dqn_agent.**DQNAgent**(*states_spec*, *actions_spec*, *batched_observe=None*, *scope='dqn'*, *summary_spec=None*, *network_spec=None*, *device=None*, *session_config=None*, *saver_spec=None*, *distributed_spec=None*, *optimizer=None*, *discount=0.99*, *variable_noise=None*, *states_preprocessing_spec=None*, *explorations_spec=None*, *reward_preprocessing_spec=None*, *distributions_spec=None*, *entropy_regularization=None*, *batch_size=32*, *memory=None*, *first_update=10000*, *update_frequency=4*, *repeat_update=1*, *target_sync_frequency=10000*, *target_update_weight=1.0*, *double_q_model=False*, *huber_loss=None*)

Bases: *tensorforce.agents.memory_agent.MemoryAgent*

Deep-Q-Network agent (DQN). The piece de resistance of deep reinforcement learning as described by Minh et al. (2015). Includes an option for double-DQN (DDQN; van Hasselt et al., 2015)

DQN chooses from one of a number of discrete actions by taking the maximum Q-value from the value function with one output neuron per available action. DQN uses a replay memory for experience playback.

initialize_model()

tensorforce.agents.dqn_nstep_agent module

class tensorforce.agents.dqn_nstep_agent.**DQNNstepAgent**(*states_spec*, *actions_spec*, *batched_observe=1000*, *scope='dqn-nstep'*, *summary_spec=None*, *network_spec=None*, *device=None*, *session_config=None*, *saver_spec=None*, *distributed_spec=None*, *optimizer=None*, *discount=0.99*, *variable_noise=None*, *states_preprocessing_spec=None*, *exploration_spec=None*, *reward_preprocessing_spec=None*, *distributions_spec=None*, *entropy_regularization=None*, *batch_size=32*, *keep_last_timestep=True*, *target_sync_frequency=10000*, *target_update_weight=1.0*, *double_q_model=False*, *huber_loss=None*)

Bases: *tensorforce.agents.batch_agent.BatchAgent*

N-step Deep-Q-Network agent (DQN).

initialize_model()

tensorforce.agents.memory_agent module

class tensorforce.agents.memory_agent.**MemoryAgent**(*states_spec*, *actions_spec*, *batched_observe=1000*, *scope='memory_agent'*, *summary_spec=None*, *network_spec=None*, *discount=0.99*, *device=None*, *session_config=None*, *saver_spec=None*, *distributed_spec=None*, *optimizer=None*, *variable_noise=None*, *states_preprocessing_spec=None*, *explorations_spec=None*, *reward_preprocessing_spec=None*, *distributions_spec=None*, *entropy_regularization=None*, *batch_size=1000*, *memory=None*, *first_update=10000*, *update_frequency=4*, *repeat_update=1*)

Bases: tensorforce.agents.learning_agent.LearningAgent

The MemoryAgent class implements a replay memory from which it samples batches according to some sampling strategy to update the value function.

import_observations(*observations*)

Load an iterable of observation dicts into the replay memory.

Parameters **observations** – An iterable with each element containing an observation. Each observation requires keys 'state','action','reward','terminal', 'internal'. Use an empty list [] for 'internal' if internal state is irrelevant.

observe(*terminal*, *reward*)

tensorforce.agents.naf_agent module

class tensorforce.agents.naf_agent.**NAFAgent**(*states_spec*, *actions_spec*, *batched_observe=1000*, *scope='naf'*, *summary_spec=None*, *network_spec=None*, *device=None*, *session_config=None*, *saver_spec=None*, *distributed_spec=None*, *optimizer=None*, *discount=0.99*, *variable_noise=None*, *states_preprocessing_spec=None*, *explorations_spec=None*, *reward_preprocessing_spec=None*, *distributions_spec=None*, *entropy_regularization=None*, *batch_size=32*, *memory=None*, *first_update=10000*, *update_frequency=4*, *repeat_update=1*, *target_sync_frequency=10000*, *target_update_weight=1.0*, *double_q_model=False*, *huber_loss=None*)

Bases: *tensorforce.agents.memory_agent.MemoryAgent*

Normalized Advantage Functions (NAF) for continuous DQN: https://arxiv.org/abs/1603.00748

initialize_model()

tensorforce.agents.ppo_agent module

class tensorforce.agents.ppo_agent.**PPOAgent**(*states_spec*, *actions_spec*, *batched_observe=1000*, *scope='ppo'*, *summary_spec=None*, *network_spec=None*, *device=None*, *session_config=None*, *saver_spec=None*, *distributed_spec=None*, *discount=0.99*, *variable_noise=None*, *states_preprocessing_spec=None*, *explorations_spec=None*, *reward_preprocessing_spec=None*, *distributions_spec=None*, *entropy_regularization=0.01*, *batch_size=1000*, *keep_last_timestep=True*, *baseline_mode=None*, *baseline=None*, *baseline_optimizer=None*, *gae_lambda=None*, *likelihood_ratio_clipping=None*, *step_optimizer=None*, *optimization_steps=10*)

Bases: *tensorforce.agents.batch_agent.BatchAgent*

Proximal Policy Optimization agent ([Schulman et al., 2017] (https://openai-public.s3-us-west-2.amazonaws.com/blog/2017-07/ppo/ppo-arxiv.pdf).

initialize_model()

tensorforce.agents.random_agent module

class tensorforce.agents.random_agent.**RandomAgent**(*states_spec*, *actions_spec*, *batched_observe=1000*, *scope='random'*)

Bases: *tensorforce.agents.agent.Agent*

Random agent, useful as a baseline and sanity check.

initialize_model()

tensorforce.agents.trpo_agent module

class tensorforce.agents.trpo_agent.**TRPOAgent**(*states_spec*, *actions_spec*, *batched_observe=1000*, *scope='trpo'*, *summary_spec=None*, *network_spec=None*, *device=None*, *session_config=None*, *saver_spec=None*, *distributed_spec=None*, *discount=0.99*, *variable_noise=None*, *states_preprocessing_spec=None*, *explorations_spec=None*, *reward_preprocessing_spec=None*, *distributions_spec=None*, *entropy_regularization=None*, *batch_size=1000*, *keep_last_timestep=True*, *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*)

Bases: *tensorforce.agents.batch_agent.BatchAgent*

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

initialize_model()

tensorforce.agents.vpg_agent module

class tensorforce.agents.vpg_agent.**VPGAgent**(*states_spec*, *actions_spec*, *batched_observe=1000*, *scope='vpg'*, *summary_spec=None*, *network_spec=None*, *device=None*, *session_config=None*, *saver_spec=None*, *distributed_spec=None*, *optimizer=None*, *discount=0.99*, *variable_noise=None*, *states_preprocessing_spec=None*, *explorations_spec=None*, *reward_preprocessing_spec=None*, *distributions_spec=None*, *entropy_regularization=None*, *batch_size=1000*, *keep_last_timestep=True*, *baseline_mode=None*, *baseline=None*, *baseline_optimizer=None*, *gae_lambda=None*)

Bases: *tensorforce.agents.batch_agent.BatchAgent*

Vanilla Policy Gradient agent as described by [Sutton et al. (1999)] (https://papers.nips.cc/paper/1713-policy-gradient-methods-for-reinforcement-learning-with-function-approximation.pdf).

initialize_model()

Module contents

class tensorforce.agents.**Agent**(*states_spec*, *actions_spec*, *batched_observe=1000*, *scope='base_agent'*)

Bases: object

Basic Reinforcement learning agent. An agent encapsulates execution logic of a particular reinforcement learning algorithm and defines the external interface to the environment.

The agent hence acts as an intermediate layer between environment and backend execution (value function or policy updates).

act(*states*, *deterministic=False*)

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.

Returns Scalar value of the action or dict of multiple actions the agent wants to execute.

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.

static process_action_spec(*actions_spec*)

static process_state_spec(*states_spec*)

reset()

Reset the agent to its initial state on episode start. Updates internal episode and timestep 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.

should_stop()

class tensorforce.agents.**ConstantAgent**(*states_spec*, *actions_spec*, *batched_observe=1000*, *scope='constant'*, *action_values=None*)
Bases: *tensorforce.agents.agent.Agent*

Constant action agent for sanity checks. Returns a constant value at every step, useful to debug continuous problems.

initialize_model()

class tensorforce.agents.**RandomAgent**(*states_spec*, *actions_spec*, *batched_observe=1000*, *scope='random'*)
Bases: *tensorforce.agents.agent.Agent*

Random agent, useful as a baseline and sanity check.

initialize_model()

class tensorforce.agents.**LearningAgent**(*states_spec*, *actions_spec*, *batched_observe=1000*, *scope='dqn'*, *summary_spec=None*, *network_spec=None*, *discount=0.99*, *device=None*, *session_config=None*, *saver_spec=None*, *distributed_spec=None*, *optimizer=None*, *variable_noise=None*, *states_preprocessing_spec=None*, *explorations_spec=None*, *reward_preprocessing_spec=None*, *distributions_spec=None*, *entropy_regularization=None*)
Bases: *tensorforce.agents.agent.Agent*

An Agent that actually learns by optimizing the parameters of its tensorflow model.

class tensorforce.agents.**BatchAgent**(*states_spec*, *actions_spec*, *batched_observe=1000*, *summary_spec=None*, *network_spec=None*, *discount=0.99*, *device=None*, *session_config=None*, *scope='batch_agent'*, *saver_spec=None*, *distributed_spec=None*, *optimizer=None*, *variable_noise=None*, *states_preprocessing_spec=None*, *explorations_spec=None*, *reward_preprocessing_spec=None*, *distributions_spec=None*, *entropy_regularization=None*, *batch_size=1000*, *keep_last_timestep=True*)
Bases: tensorforce.agents.learning_agent.LearningAgent

The `BatchAgent` class implements a batch memory which generally implies on-policy experience collection and updates.

observe(*terminal*, *reward*)

Adds an observation and performs an update if the necessary conditions are satisfied, i.e. if one batch of experience has been collected as defined by the batch size.

In particular, note that episode control happens outside of the agent since the agent should be agnostic to how the training data is created.

Parameters

- **terminal** (*bool*) – Whether episode is terminated or not.
- **reward** (*float*) – The scalar reward value.

reset_batch()

Cleans up after a batch has been processed (observed). Resets all batch information to be ready for new observation data. Batch information contains:

- observed states
- internal-variables
- taken actions
- observed is-terminal signals/rewards
- total batch size

class tensorforce.agents.**MemoryAgent**(*states_spec*, *actions_spec*, *batched_observe=1000*, *scope='memory_agent'*, *summary_spec=None*, *network_spec=None*, *discount=0.99*, *device=None*, *session_config=None*, *saver_spec=None*, *distributed_spec=None*, *optimizer=None*, *variable_noise=None*, *states_preprocessing_spec=None*, *explorations_spec=None*, *reward_preprocessing_spec=None*, *distributions_spec=None*, *entropy_regularization=None*, *batch_size=1000*, *memory=None*, *first_update=10000*, *update_frequency=4*, *repeat_update=1*)

Bases: `tensorforce.agents.learning_agent.LearningAgent`

The `MemoryAgent` class implements a replay memory from which it samples batches according to some sampling strategy to update the value function.

import_observations(*observations*)

Load an iterable of observation dicts into the replay memory.

Parameters **observations** – An iterable with each element containing an observation. Each observation requires keys 'state','action','reward','terminal', 'internal'. Use an empty list [] for 'internal' if internal state is irrelevant.

observe(*terminal*, *reward*)

class tensorforce.agents.**VPGAgent**(*states_spec*, *actions_spec*, *batched_observe=1000*, *scope='vpg'*, *summary_spec=None*, *network_spec=None*, *device=None*, *session_config=None*, *saver_spec=None*, *distributed_spec=None*, *optimizer=None*, *discount=0.99*, *variable_noise=None*, *states_preprocessing_spec=None*, *explorations_spec=None*, *reward_preprocessing_spec=None*, *distributions_spec=None*, *entropy_regularization=None*, *batch_size=1000*, *keep_last_timestep=True*, *baseline_mode=None*, *baseline=None*, *baseline_optimizer=None*, *gae_lambda=None*)

Bases: *tensorforce.agents.batch_agent.BatchAgent*

Vanilla Policy Gradient agent as described by [Sutton et al. (1999)] (https://papers.nips.cc/paper/1713-policy-gradient-methods-for-reinforcement-learning-with-function-approximation.pdf).

initialize_model()

class tensorforce.agents.**TRPOAgent**(*states_spec*, *actions_spec*, *batched_observe=1000*, *scope='trpo'*, *summary_spec=None*, *network_spec=None*, *device=None*, *session_config=None*, *saver_spec=None*, *distributed_spec=None*, *discount=0.99*, *variable_noise=None*, *states_preprocessing_spec=None*, *explorations_spec=None*, *reward_preprocessing_spec=None*, *distributions_spec=None*, *entropy_regularization=None*, *batch_size=1000*, *keep_last_timestep=True*, *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*)

Bases: *tensorforce.agents.batch_agent.BatchAgent*

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

initialize_model()

class tensorforce.agents.**PPOAgent**(*states_spec*, *actions_spec*, *batched_observe=1000*, *scope='ppo'*, *summary_spec=None*, *network_spec=None*, *device=None*, *session_config=None*, *saver_spec=None*, *distributed_spec=None*, *discount=0.99*, *variable_noise=None*, *states_preprocessing_spec=None*, *explorations_spec=None*, *reward_preprocessing_spec=None*, *distributions_spec=None*, *entropy_regularization=0.01*, *batch_size=1000*, *keep_last_timestep=True*, *baseline_mode=None*, *baseline=None*, *baseline_optimizer=None*, *gae_lambda=None*, *likelihood_ratio_clipping=None*, *step_optimizer=None*, *optimization_steps=10*)

Bases: *tensorforce.agents.batch_agent.BatchAgent*

Proximal Policy Optimization agent ([Schulman et al., 2017] (https://openai-public.s3-us-west-2.amazonaws.com/blog/2017-07/ppo/ppo-arxiv.pdf).

initialize_model()

class tensorforce.agents.**DQNAgent**(*states_spec*, *actions_spec*, *batched_observe=None*, *scope='dqn'*, *summary_spec=None*, *network_spec=None*, *device=None*, *session_config=None*, *saver_spec=None*, *distributed_spec=None*, *optimizer=None*, *discount=0.99*, *variable_noise=None*, *states_preprocessing_spec=None*, *explorations_spec=None*, *reward_preprocessing_spec=None*, *distributions_spec=None*, *entropy_regularization=None*, *batch_size=32*, *memory=None*, *first_update=10000*, *update_frequency=4*, *repeat_update=1*, *target_sync_frequency=10000*, *target_update_weight=1.0*, *double_q_model=False*, *huber_loss=None*)

Bases: *tensorforce.agents.memory_agent.MemoryAgent*

Deep-Q-Network agent (DQN). The piece de resistance of deep reinforcement learning as described by Minh et al. (2015). Includes an option for double-DQN (DDQN; van Hasselt et al., 2015)

DQN chooses from one of a number of discrete actions by taking the maximum Q-value from the value function with one output neuron per available action. DQN uses a replay memory for experience playback.

initialize_model()

class tensorforce.agents.**DDQNAgent**(*states_spec*, *actions_spec*, *batched_observe=1000*, *scope='ddqn'*, *summary_spec=None*, *network_spec=None*, *device=None*, *session_config=None*, *saver_spec=None*, *distributed_spec=None*, *optimizer=None*, *discount=0.99*, *variable_noise=None*, *states_preprocessing_spec=None*, *explorations_spec=None*, *reward_preprocessing_spec=None*, *distributions_spec=None*, *entropy_regularization=None*, *batch_size=32*, *memory=None*, *first_update=10000*, *update_frequency=4*, *repeat_update=1*, *target_sync_frequency=10000*, *target_update_weight=1.0*, *huber_loss=None*)

Bases: *tensorforce.agents.memory_agent.MemoryAgent*

Double DQN Agent based on Van Hasselt et al.. Simple extension to DQN which improves stability.

initialize_model()

class tensorforce.agents.**DQNNstepAgent**(*states_spec*, *actions_spec*, *batched_observe=1000*, *scope='dqn-nstep'*, *summary_spec=None*, *network_spec=None*, *device=None*, *session_config=None*, *saver_spec=None*, *distributed_spec=None*, *optimizer=None*, *discount=0.99*, *variable_noise=None*, *states_preprocessing_spec=None*, *explorations_spec=None*, *reward_preprocessing_spec=None*, *distributions_spec=None*, *entropy_regularization=None*, *batch_size=32*, *keep_last_timestep=True*, *target_sync_frequency=10000*, *target_update_weight=1.0*, *double_q_model=False*, *huber_loss=None*)

Bases: *tensorforce.agents.batch_agent.BatchAgent*

N-step Deep-Q-Network agent (DQN).

initialize_model()

class tensorforce.agents.**DQFDAgent**(*states_spec*, *actions_spec*, *batched_observe=1000*, *scope='dqfd'*, *summary_spec=None*, *network_spec=None*, *device=None*, *session_config=None*, *saver_spec=None*, *distributed_spec=None*, *optimizer=None*, *discount=0.99*, *variable_noise=None*, *states_preprocessing_spec=None*, *explorations_spec=None*, *reward_preprocessing_spec=None*, *distributions_spec=None*, *entropy_regularization=None*, *batch_size=32*, *memory=None*, *first_update=10000*, *update_frequency=4*, *repeat_update=1*, *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.memory_agent.MemoryAgent*

Deep Q-learning from demonstration (DQFD) agent (Hester et al., 2017). This agent uses DQN to pre-train from demonstration data via an additional supervised loss term.

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

initialize_model()

observe(*reward*, *terminal*)

Adds observations, updates via sampling from memories according to update rate. DQFD samples from the online replay memory and the demo memory with the fractions controlled by a hyper parameter p called 'expert sampling ratio.

pretrain(*steps*)

Computes pre-train updates.

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

set_demonstrations(*batch*)

Set all demonstrations from batch data. Expects a dict wherein each value contains an array containing all states, actions, rewards, terminals and internals respectively.

Parameters **batch** –

class tensorforce.agents.**NAFAgent**(*states_spec*, *actions_spec*, *batched_observe=1000*, *scope='naf'*, *summary_spec=None*, *network_spec=None*, *device=None*, *session_config=None*, *saver_spec=None*, *distributed_spec=None*, *optimizer=None*, *discount=0.99*, *variable_noise=None*, *states_preprocessing_spec=None*, *explorations_spec=None*, *reward_preprocessing_spec=None*, *distributions_spec=None*, *entropy_regularization=None*, *batch_size=32*, *memory=None*, *first_update=10000*, *update_frequency=4*, *repeat_update=1*, *target_sync_frequency=10000*, *target_update_weight=1.0*, *double_q_model=False*, *huber_loss=None*)

Bases: *tensorforce.agents.memory_agent.MemoryAgent*

Normalized Advantage Functions (NAF) for continuous DQN: https://arxiv.org/abs/1603.00748

initialize_model()

tensorforce.contrib package

Submodules

tensorforce.contrib.ale module

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

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*

action_names

actions

close()

current_state

execute(*actions*)

is_terminal

reset()

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

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.

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

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.

actions

close()

execute(*actions*)

reset()

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*

static action_from_space(*space*)

actions

close()

execute(*actions*)

reset()

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/.

actions

close()

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

execute(*actions*)

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

reset()

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})"

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 is encoded via msgpack-numpy.

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*

close()
Same as disconnect method.

connect()
Starts the server tcp connection on the given host:port.

current_state

disconnect()
Ends our server tcp connection.

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.

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.

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 ducandu plugin Observer objects placed in the Game (these could be camera pixels or other observations, e.g. a x/y/z position of some game actor).

actions()

connect()

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*)

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.

get_summaries()

get_variables(*include_non_trainable=False*)

tf_predict(*states*, *update*)

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.

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_non_trainable=False*)
Returns the TensorFlow variables used by the baseline.

Returns List of variables

tf_loss(*states*, *reward*, *update*)
Creates the TensorFlow operations for calculating the L2 loss between predicted state values and actual rewards.

Parameters

- **states** – State tensors
- **reward** – Reward tensor
- **update** – Boolean tensor indicating whether this call happens during an update.

Returns Loss tensor

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

Returns State value tensor

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.

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.

tensorforce.core.baselines.network_baseline module

class tensorforce.core.baselines.network_baseline.**NetworkBaseline**(*network_spec*, *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.

get_summaries()

get_variables(*include_non_trainable=False*)

tf_predict(*states*, *update*)

tf_regularization_loss()

Creates the TensorFlow operations for the baseline regularization loss.

Returns Regularization loss tensor

Module contents

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

Bases: object

Base class for baseline value functions.

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_non_trainable=False*)

Returns the TensorFlow variables used by the baseline.

Returns List of variables

tf_loss(*states*, *reward*, *update*)

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

Parameters

- **states** – State tensors
- **reward** – Reward tensor
- **update** – Boolean tensor indicating whether this call happens during an update.

Returns Loss tensor

tf_predict(*states*, *update*)

Creates the TensorFlow operations for predicting the value function of given states. :param states: State tensors :param update: Boolean tensor indicating whether this call happens during an update.

Returns State value tensor

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.

get_summaries()

get_variables(*include_non_trainable=False*)

tf_predict(*states*, *update*)

tf_regularization_loss()

class tensorforce.core.baselines.**NetworkBaseline**(*network_spec*, *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.

get_summaries()

get_variables(*include_non_trainable=False*)

tf_predict(*states*, *update*)

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

Returns Regularization loss tensor

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.

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.

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 actions.

get_summaries()

get_variables(*include_non_trainable=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*)

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

get_summaries()

get_variables(*include_non_trainable=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

get_summaries()

get_variables(*include_non_trainable=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*)

tensorforce.core.distributions.distribution module

class tensorforce.core.distributions.distribution.**Distribution**(*scope='distribution'*, *summary_labels=None*)

Bases: object

Base class for policy distributions.

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_non_trainable=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
- **be returned.** (*should*) –

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.

get_summaries()

get_variables(*include_non_trainable=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**(*scope='distribution'*, *summary_labels=None*)

Bases: object

Base class for policy distributions.

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_non_trainable=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
- **be returned.** (*should*) –

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 actions.

get_summaries()

get_variables(*include_non_trainable=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.**Categorical**(*shape*, *num_actions*, *probabilities=None*, *scope='categorical'*, *summary_labels=()*)

Bases: *tensorforce.core.distributions.distribution.Distribution*

Categorical distribution, for discrete actions

get_summaries()

get_variables(*include_non_trainable=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.**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.

get_summaries()

get_variables(*include_non_trainable=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

get_summaries()

get_variables(*include_non_trainable=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.

tf_explore(*episode*, *timestep*, *action_shape*)

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.

tf_explore(*episode*, *timestep*, *action_shape*)

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.

tf_explore(*episode=0*, *timestep=0*, *action_shape=(1,)*)

tensorforce.core.explorations.exploration module

class tensorforce.core.explorations.exploration.**Exploration**(*scope='exploration'*, *summary_labels=None*)

Bases: object

Abstract exploration object.

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_shape*)
Creates exploration value, e.g. compute an epsilon for epsilon-greedy or sample normal noise.

tensorforce.core.explorations.linear_decay module

class tensorforce.core.explorations.linear_decay.**LinearDecay**(*scope='exploration'*, *summary_labels=None*)

Bases: *tensorforce.core.explorations.exploration.Exploration*

Linear decay based on episode number.

tf_explore(*episode*, *timestep*, *action_shape*)

tensorforce.core.explorations.ornstein_uhlenbeck_process module

class tensorforce.core.explorations.ornstein_uhlenbeck_process.**OrnsteinUhlenbeckProcess**(*sigr*
mu=
thet
scop
sum
mar

Bases: *tensorforce.core.explorations.exploration.Exploration*

Explores via an Ornstein-Uhlenbeck process.

tf_explore(*episode*, *timestep*, *action_shape*)

Module contents

class tensorforce.core.explorations.**Exploration**(*scope='exploration'*, *summary_labels=None*)

Bases: object

Abstract exploration object.

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_shape*)
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.

tf_explore(*episode*, *timestep*, *action_shape*)

class tensorforce.core.explorations.**LinearDecay**(*scope='exploration'*, *summary_labels=None*)

Bases: *tensorforce.core.explorations.exploration.Exploration*

Linear decay based on episode number.

tf_explore(*episode*, *timestep*, *action_shape*)

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.

tf_explore(*episode=0*, *timestep=0*, *action_shape=(1,)*)

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.

tf_explore(*episode*, *timestep*, *action_shape*)

tensorforce.core.memories package

Submodules

tensorforce.core.memories.memory module

class tensorforce.core.memories.memory.**Memory**(*states_spec*, *actions_spec*)

Bases: object

Abstract memory class.

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

Inserts a single experience to the memory.

Parameters

- **states** –
- **internals** –
- **actions** –
- **terminal** –
- **reward** –

Returns:

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

Creates a memory from a specification dict.

get_batch(*batch_size*, *next_states=False*)

Samples a batch from the memory.

Parameters

- **batch_size** – The batch size
- **next_states** – A boolean flag indicating whether 'next_states' values should be included

Returns: A dict containing states, internal states, actions, terminals, rewards (and next states)

set_memory(*states*, *internals*, *actions*, *terminals*, *rewards*)

Deletes memory content and sets content to provided observations.

Parameters

- **states** –
- **internals** –

- **actions** –
- **terminals** –
- **rewards** –

update_batch(*loss_per_instance*)
Updates loss values for sampling strategies based on loss functions.

Parameters **loss_per_instance** –

tensorforce.core.memories.naive_prioritized_replay module

class tensorforce.core.memories.naive_prioritized_replay.**NaivePrioritizedReplay**(*states_spec*, *actions_spec*, *capacity*, *prioritization_weight=1.*

Bases: *tensorforce.core.memories.memory.Memory*

Prioritised replay sampling based on loss per experience.

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

get_batch(*batch_size*, *next_states=False*)
Samples a batch of the specified size according to priority.

Parameters

- **batch_size** – The batch size
- **next_states** – A boolean flag indicating whether 'next_states' values should be included

Returns: A dict containing states, actions, rewards, terminals, internal states (and next states)

update_batch(*loss_per_instance*)
Computes priorities according to loss.

Parameters **loss_per_instance** –

tensorforce.core.memories.naive_prioritized_replay.**random**() → x in the interval [0, 1).

tensorforce.core.memories.prioritized_replay module

class tensorforce.core.memories.prioritized_replay.**PrioritizedReplay**(*states_spec*, *actions_spec*, *capacity*, *prioritization_weight=1.0*, *prioritization_constant=0.0*)

Bases: *tensorforce.core.memories.memory.Memory*

Prioritised replay sampling based on loss per experience.

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

get_batch(*batch_size*, *next_states=False*)

Samples a batch of the specified size according to priority.

Parameters

- **batch_size** – The batch size
- **next_states** – A boolean flag indicating whether 'next_states' values should be included

Returns: A dict containing states, actions, rewards, terminals, internal states (and next states)

update_batch(*loss_per_instance*)

Computes priorities according to loss.

Parameters **loss_per_instance** –

class tensorforce.core.memories.prioritized_replay.**SumTree**(*capacity*)

Bases: object

Sum tree data structure where data is stored in leaves and each node on the tree contains a sum of the children.

Items and priorities are stored in leaf nodes, while internal nodes store the sum of priorities from all its descendants. Internally a single list stores the internal nodes followed by leaf nodes.

See:

- Binary heap trees
- Section B.2.1 in the prioritized replay paper
- The CNTK implementation

Usage: tree = SumTree(100) tree.push('item1', priority=0.5) tree.push('item2', priority=0.6) item, priority = tree[0] batch = tree.sample_minibatch(2)

move(*external_index*, *new_priority*)

Change the priority of a leaf node

put(*item*, *priority=None*)

Stores a transition in replay memory.

If the memory is full, the oldest entry is replaced.

sample_minibatch(*batch_size*)
Sample minibatch of size batch_size.

tensorforce.core.memories.replay module

class tensorforce.core.memories.replay.**Replay**(*states_spec*, *actions_spec*, *capacity*, *random_sampling=True*)
Bases: *tensorforce.core.memories.memory.Memory*

Replay memory to store observations and sample mini batches for training from.

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

get_batch(*batch_size*, *next_states=False*, *keep_terminal_states=True*)
Samples a batch of the specified size by selecting a random start/end point and returning the contained sequence or random indices depending on the field 'random_sampling'.

Parameters

- **batch_size** – The batch size
- **next_states** – A boolean flag indicating whether 'next_states' values should be included
- **keep_terminal_states** – A boolean flag indicating whether to keep terminal states when next_states are requested. In this case, the next state is not from the same episode and should probably not be used to learn a model of the environment. However, if the environment produces sparse rewards (i.e. only one reward at the end of the episode) we cannot exclude terminal states, as otherwise there would never be a reward to learn from.

Returns: A dict containing states, actions, rewards, terminals, internal states (and next states)

set_memory(*states*, *internals*, *actions*, *terminal*, *reward*)
Convenience function to set whole batches as memory content to bypass calling the insert function for every single experience.

Parameters

- **states** –
- **internals** –
- **actions** –
- **terminal** –
- **reward** –

Returns:

update_batch(*loss_per_instance*)

Module contents

class tensorforce.core.memories.**Memory**(*states_spec*, *actions_spec*)
Bases: object

Abstract memory class.

add_observation(*states*, *internals*, *actions*, *terminal*, *reward*)
Inserts a single experience to the memory.

Parameters

- **states** –
- **internals** –
- **actions** –
- **terminal** –
- **reward** –

Returns:

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

get_batch(*batch_size*, *next_states=False*)
Samples a batch from the memory.

Parameters

- **batch_size** – The batch size
- **next_states** – A boolean flag indicating whether 'next_states' values should be included

Returns: A dict containing states, internal states, actions, terminals, rewards (and next states)

set_memory(*states*, *internals*, *actions*, *terminals*, *rewards*)
Deletes memory content and sets content to provided observations.

Parameters

- **states** –
- **internals** –
- **actions** –
- **terminals** –
- **rewards** –

update_batch(*loss_per_instance*)
Updates loss values for sampling strategies based on loss functions.

Parameters **loss_per_instance** –

class tensorforce.core.memories.**Replay**(*states_spec*, *actions_spec*, *capacity*, *random_sampling=True*)
Bases: *tensorforce.core.memories.memory.Memory*

Replay memory to store observations and sample mini batches for training from.

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

get_batch(*batch_size*, *next_states=False*, *keep_terminal_states=True*)
Samples a batch of the specified size by selecting a random start/end point and returning the contained sequence or random indices depending on the field 'random_sampling'.

Parameters

- **batch_size** – The batch size
- **next_states** – A boolean flag indicating whether 'next_states' values should be included

- **keep_terminal_states** – A boolean flag indicating whether to keep terminal states when `next_states` are requested. In this case, the next state is not from the same episode and should probably not be used to learn a model of the environment. However, if the environment produces sparse rewards (i.e. only one reward at the end of the episode) we cannot exclude terminal states, as otherwise there would never be a reward to learn from.

Returns: A dict containing states, actions, rewards, terminals, internal states (and next states)

set_memory(*states*, *internals*, *actions*, *terminal*, *reward*)
Convenience function to set whole batches as memory content to bypass calling the insert function for every single experience.

Parameters

- **states** –
- **internals** –
- **actions** –
- **terminal** –
- **reward** –

Returns:

update_batch(*loss_per_instance*)

class tensorforce.core.memories.**PrioritizedReplay**(*states_spec*, *actions_spec*, *capacity*, *prioritization_weight=1.0*, *prioritization_constant=0.0*)
Bases: *tensorforce.core.memories.memory.Memory*

Prioritised replay sampling based on loss per experience.

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

get_batch(*batch_size*, *next_states=False*)
Samples a batch of the specified size according to priority.

Parameters

- **batch_size** – The batch size
- **next_states** – A boolean flag indicating whether 'next_states' values should be included

Returns: A dict containing states, actions, rewards, terminals, internal states (and next states)

update_batch(*loss_per_instance*)
Computes priorities according to loss.

Parameters **loss_per_instance** –

class tensorforce.core.memories.**NaivePrioritizedReplay**(*states_spec*, *actions_spec*, *capacity*, *prioritization_weight=1.0*)
Bases: *tensorforce.core.memories.memory.Memory*

Prioritised replay sampling based on loss per experience.

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

get_batch(*batch_size*, *next_states=False*)
Samples a batch of the specified size according to priority.

Parameters

- **batch_size** – The batch size
- **next_states** – A boolean flag indicating whether 'next_states' values should be included

Returns: A dict containing states, actions, rewards, terminals, internal states (and next states)

update_batch(*loss_per_instance*)
Computes priorities according to loss.

Parameters **loss_per_instance** –

tensorforce.core.networks package

Submodules

tensorforce.core.networks.layer module

Collection of custom layer implementations. We prefer not to use contrib-layers to retain full control over shapes and internal states.

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.

get_summaries()

get_variables(*include_non_trainable=False*)

tf_apply(*x*, *update*)

tf_regularization_loss()

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.

get_summaries()

get_variables(*include_non_trainable=False*)

tf_apply(*x*, *update*)

tf_regularization_loss()

class tensorforce.core.networks.layer.**Dense**(*size=None*, *bias=True*, *activation='tanh'*, *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.

get_summaries()

get_variables(*include_non_trainable=False*)

tf_apply(*x*, *update*)

tf_regularization_loss()

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.

tf_apply(*x*, *update*)

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

get_summaries()

get_variables(*include_non_trainable=False*)

tf_apply(*x*, *update*)

tf_regularization_loss()

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.

tf_apply(*x*, *update*)

tf_regularization_loss()

class tensorforce.core.networks.layer.**Flatten**(*scope='flatten'*, *summary_labels=()*)

Bases: *tensorforce.core.networks.layer.Layer*

Flatten layer reshaping the input.

tf_apply(*x*, *update*)

class tensorforce.core.networks.layer.**InternalLstm**(*size*, *dropout=None*, *scope='internal_lstm'*, *summary_labels=()*)

Bases: *tensorforce.core.networks.layer.Layer*

Long short-term memory layer for internal state management.

internals_init()

internals_input()

tf_apply(*x*, *update*, *state*)

class tensorforce.core.networks.layer.**Layer**(*num_internals=0*, *scope='layer'*, *summary_labels=None*)

Bases: object

Base class for network layers.

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_non_trainable=False*)
Returns the TensorFlow variables used by the layer.

Returns List of variables.

internals_init()
Returns the TensorFlow tensors for internal state initializations.

Returns List of internal state initialization tensors.

internals_input()
Returns the TensorFlow placeholders for internal state inputs.

Returns List of internal state input placeholders.

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.

tf_apply(*x*, *update=False*)

tf_regularization_loss()

class tensorforce.core.networks.layer.**Lstm**(*size*, *dropout=None*, *scope='lstm'*, *summary_labels=()*, *return_final_state=True*)

Bases: *tensorforce.core.networks.layer.Layer*

tf_apply(*x*, *update*, *sequence_length=None*)

class tensorforce.core.networks.layer.**Nonlinearity**(*name='relu'*, *scope='nonlinearity'*, *summary_labels=()*)

Bases: *tensorforce.core.networks.layer.Layer*

Non-linearity layer applying a non-linear transformation.

tf_apply(*x*, *update*)

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.

tf_apply(*x*, *update*)

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.

add_layer(*layer*)

get_summaries()

get_variables(*include_non_trainable=False*)

internals_init()

internals_input()

tf_regularization_loss()

class tensorforce.core.networks.network.**LayeredNetwork**(*layers_spec*, *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.

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

tf_apply(*x*, *internals*, *update*, *return_internals=False*)

class tensorforce.core.networks.network.**Network**(*scope='network'*, *summary_labels=None*)

Bases: object

Base class for neural networks.

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

get_summaries()
Returns the TensorFlow summaries reported by the network.

Returns List of summaries

get_variables(*include_non_trainable=False*)
Returns the TensorFlow variables used by the network.

Returns List of variables

internals_init()
Returns the TensorFlow tensors for internal state initializations.

Returns List of internal state initialization tensors

internals_input()
Returns the TensorFlow placeholders for internal state inputs.

Returns List of internal state input placeholders

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**(*num_internals=0*, *scope='layer'*, *summary_labels=None*)

Bases: object

Base class for network layers.

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_non_trainable=False*)
Returns the TensorFlow variables used by the layer.

Returns List of variables.

internals_init()
Returns the TensorFlow tensors for internal state initializations.

Returns List of internal state initialization tensors.

internals_input()
Returns the TensorFlow placeholders for internal state inputs.

Returns List of internal state input placeholders.

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.**Nonlinearity**(*name='relu'*, *scope='nonlinearity'*, *summary_labels=()*)
Bases: *tensorforce.core.networks.layer.Layer*

Non-linearity layer applying a non-linear transformation.

tf_apply(*x*, *update*)

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.

tf_apply(*x*, *update*)

class tensorforce.core.networks.**Flatten**(*scope='flatten'*, *summary_labels=()*)
Bases: *tensorforce.core.networks.layer.Layer*

Flatten layer reshaping the input.

tf_apply(*x*, *update*)

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.

tf_apply(*x*, *update*)

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.

tf_apply(*x*, *update*)

tf_regularization_loss()

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.

tf_apply(*x*, *update=False*)

tf_regularization_loss()

class tensorforce.core.networks.**Dense**(*size=None*, *bias=True*, *activation='tanh'*, *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.

get_summaries()

get_variables(*include_non_trainable=False*)

tf_apply(*x*, *update*)

tf_regularization_loss()

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

get_summaries()

get_variables(*include_non_trainable=False*)

tf_apply(*x*, *update*)

tf_regularization_loss()

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.

get_summaries()

get_variables(*include_non_trainable=False*)

tf_apply(*x*, *update*)

tf_regularization_loss()

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.

get_summaries()

get_variables(*include_non_trainable=False*)

tf_apply(*x*, *update*)

tf_regularization_loss()

class tensorforce.core.networks.**InternalLstm**(*size*, *dropout=None*, *scope='internal_lstm'*, *summary_labels=()*)

Bases: *tensorforce.core.networks.layer.Layer*

Long short-term memory layer for internal state management.

internals_init()

internals_input()

tf_apply(*x*, *update*, *state*)

class tensorforce.core.networks.**Lstm**(*size*, *dropout=None*, *scope='lstm'*, *summary_labels=()*, *return_final_state=True*)

Bases: *tensorforce.core.networks.layer.Layer*

tf_apply(*x*, *update*, *sequence_length=None*)

class tensorforce.core.networks.**Network**(*scope='network'*, *summary_labels=None*)

Bases: object

Base class for neural networks.

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

get_summaries()
Returns the TensorFlow summaries reported by the network.

Returns List of summaries

get_variables(*include_non_trainable=False*)
Returns the TensorFlow variables used by the network.

Returns List of variables

internals_init()
Returns the TensorFlow tensors for internal state initializations.

Returns List of internal state initialization tensors

internals_input()
Returns the TensorFlow placeholders for internal state inputs.

Returns List of internal state input placeholders

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.

add_layer(*layer*)

get_summaries()

get_variables(*include_non_trainable=False*)

internals_init()

internals_input()

tf_regularization_loss()

class tensorforce.core.networks.**LayeredNetwork**(*layers_spec*, *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.

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

tf_apply(*x*, *internals*, *update*, *return_internals=False*)

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}
```

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.

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'.

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.

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.

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.

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}
```

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'.

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*, *summaries=None*, *summary_labels=None*)
Bases: *tensorforce.core.optimizers.meta_optimizer.MetaOptimizer*

The multi-shep meta optimizer repeatedly applies the optimization step proposed by another optimizer a number of times.

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*, *summaries=None*, *summary_labels=None*)
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.

tf_step(*time*, *variables*, *fn_loss*, ***kwargs*)
Creates the TensorFlow operations for performing an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.
- **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*, *summaries=None*, *summary_labels=None*)
Bases: *tensorforce.core.optimizers.meta_optimizer.MetaOptimizer*

The global optimizer applies an optimizer to the local variables. In addition, it also applies the update 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.

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*, ***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.

get_variables()

tensorforce.core.optimizers.multi_step module

class tensorforce.core.optimizers.multi_step.**MultiStep**(*optimizer*, *num_steps=5*, *summaries=None*, *summary_labels=None*)

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.

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.natural_gradient module

class tensorforce.core.optimizers.natural_gradient.**NaturalGradient**(*learning_rate*, *cg_max_iterations=20*, *cg_damping=0.001*, *cg_unroll_loop=False*, *summaries=None*, *summary_labels=None*)

Bases: *tensorforce.core.optimizers.optimizer.Optimizer*

Natural gradient optimizer.

tf_step(*time*, *variables*, *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.
- **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*, *summaries=None*, *summary_labels=None*)

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.

tf_step(*time*, *variables*, *fn_loss*, *fn_reference=None*, *fn_compare=None*, ***kwargs*)

Creates the TensorFlow operations for performing an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.
- **fn_loss** – A callable returning the loss of the current model.
- **fn_reference** – A callable returning the reference values necessary for comparison.
- **fn_compare** – A callable comparing the current model to the reference model given by its values.
- ****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**(*summaries=None*, *summary_labels=None*)

Bases: object

Generic TensorFlow optimizer which minimizes 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.

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_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:
- **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.
- **fn_reference** (–) – A callable returning the reference values necessary for comparison.
- **fn_compare** (–) – A callable comparing the current model to the reference model given by its values.
- **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*)

Bases: *tensorforce.core.optimizers.optimizer.Optimizer*

The synchronization optimizer updates variables periodically to the value of a corresponding set of source variables.

get_variables()

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*, *summaries=None*, *summary_labels=None*, ***kwargs*)

Bases: *tensorforce.core.optimizers.optimizer.Optimizer*

Wrapper class for TensorFlow optimizers.

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.

tf_optimizers = {'nadam': <sphinx.ext.autodoc._MockObject object>, 'adam': <sphinx.e

tf_step(*time*, *variables*, *fn_loss*, ***kwargs*)

Creates the TensorFlow operations for performing an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.

- **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**(*summaries=None*, *summary_labels=None*)
Bases: object

Generic TensorFlow optimizer which minimizes 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.

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_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:
- **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.
- **fn_reference** (–) – A callable returning the reference values necessary for comparison.
- **fn_compare** (–) – A callable comparing the current model to the reference model given by its values.
- **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*, ***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.

get_variables()

class tensorforce.core.optimizers.**TFOptimizer**(*optimizer*, *summaries=None*, *summary_labels=None*, ***kwargs*)
Bases: *tensorforce.core.optimizers.optimizer.Optimizer*

Wrapper class for TensorFlow optimizers.

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.

tf_optimizers = {'nadam': <sphinx.ext.autodoc._MockObject object>, 'adam': <sphinx.e

tf_step(*time*, *variables*, *fn_loss*, ***kwargs*)
Creates the TensorFlow operations for performing an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.
- **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*, *summaries=None*, *summary_labels=None*)
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.

tf_step(*time*, *variables*, *fn_loss*, ***kwargs*)
Creates the TensorFlow operations for performing an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.
- **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*, *summaries=None*, *summary_labels=None*)

Bases: *tensorforce.core.optimizers.optimizer.Optimizer*

Natural gradient optimizer.

tf_step(*time*, *variables*, *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.
- **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.**MultiStep**(*optimizer*, *num_steps=5*, *summaries=None*, *summary_labels=None*)

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.

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.**OptimizedStep**(*optimizer*, *ls_max_iterations=10*, *ls_accept_ratio=0.9*, *ls_mode='exponential'*, *ls_parameter=0.5*, *ls_unroll_loop=False*, *summaries=None*, *summary_labels=None*)

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.

tf_step(*time*, *variables*, *fn_loss*, *fn_reference=None*, *fn_compare=None*, ***kwargs*)

Creates the TensorFlow operations for performing an optimization step.

Parameters

- **time** – Time tensor.
- **variables** – List of variables to optimize.
- **fn_loss** – A callable returning the loss of the current model.
- **fn_reference** – A callable returning the reference values necessary for comparison.
- **fn_compare** – A callable comparing the current model to the reference model given by its values.
- ****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*)

Bases: *tensorforce.core.optimizers.optimizer.Optimizer*

The synchronization optimizer updates variables periodically to the value of a corresponding set of source variables.

get_variables()

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.

class tensorforce.core.optimizers.**ClippedStep**(*optimizer*, *clipping_value*, *summaries=None*, *summary_labels=None*)

Bases: *tensorforce.core.optimizers.meta_optimizer.MetaOptimizer*

The multi-shep meta optimizer repeatedly applies the optimization step proposed by another optimizer a number of times.

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.**GlobalOptimizer**(*optimizer*, *summaries=None*, *summary_labels=None*)

Bases: *tensorforce.core.optimizers.meta_optimizer.MetaOptimizer*

The global optimizer applies an optimizer to the local variables. In addition, it also applies the update 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.

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.preprocessing package

Submodules

tensorforce.core.preprocessing.clip module

class tensorforce.core.preprocessing.clip.**Clip**(*min_value*, *max_value*, *scope='clip'*, *summary_labels=()*)

Bases: *tensorforce.core.preprocessing.preprocessor.Preprocessor*

Clip by min/max.

tf_process(*tensor*)

tensorforce.core.preprocessing.divide module

class tensorforce.core.preprocessing.divide.**Divide**(*scale*, *scope='divide'*, *summary_labels=()*)

Bases: *tensorforce.core.preprocessing.preprocessor.Preprocessor*

Divide state by scale.

tf_process(*tensor*)

tensorforce.core.preprocessing.grayscale module

class tensorforce.core.preprocessing.grayscale.**Grayscale**(*weights=(0.299, 0.587, 0.114)*, *scope='grayscale'*, *summary_labels=()*)

Bases: *tensorforce.core.preprocessing.preprocessor.Preprocessor*

Turn 3D color state into grayscale.

processed_shape(*shape*)

tf_process(*tensor*)

tensorforce.core.preprocessing.image_resize module

class tensorforce.core.preprocessing.image_resize.**ImageResize**(*width*, *height*, *scope='image_resize'*, *summary_labels=()*)

Bases: *tensorforce.core.preprocessing.preprocessor.Preprocessor*

Resize image to width x height.

processed_shape(*shape*)

tf_process(*tensor*)

tensorforce.core.preprocessing.normalize module

class tensorforce.core.preprocessing.normalize.**Normalize**(*scope='normalize'*, *summary_labels=()*)

Bases: *tensorforce.core.preprocessing.preprocessor.Preprocessor*

Normalize state. Subtract minimal value and divide by range.

tf_process(*tensor*)

tensorforce.core.preprocessing.preprocessor module

class tensorforce.core.preprocessing.preprocessor.**Preprocessor**(*scope='preprocessor'*, *summary_labels=None*)

Bases: object

get_variables()

Returns the TensorFlow variables used by the preprocessor.

Returns List of variables.

processed_shape(*shape*)

Shape of preprocessed state given original shape.

Parameters **shape** – original shape.

Returns: processed tensor shape

reset()

tf_process(*tensor*)
Process state.

Parameters **tensor** – tensor to process.

Returns: processed tensor.

tensorforce.core.preprocessing.preprocessor_stack module

class tensorforce.core.preprocessing.preprocessor_stack.**PreprocessorStack**
Bases: object

static from_spec(*spec*)
Creates a preprocessing stack from a specification dict.

get_variables()

process(*tensor*)
Process state.

Parameters **tensor** – tensor to process

Returns: processed state

processed_shape(*shape*)
Shape of preprocessed state given original shape.

Parameters **shape** – original state shape

Returns: processed state shape

reset()

tensorforce.core.preprocessing.running_standardize module

class tensorforce.core.preprocessing.running_standardize.**RunningStandardize**(*axis=None*, *reset_after_batch=True*, *scope='running_stan*, *summary_labels=()*)
Bases: *tensorforce.core.preprocessing.preprocessor.Preprocessor*

Standardize state w.r.t past states. Subtract mean and divide by standard deviation of sequence of past states.

reset()

tf_process(*tensor*)

tensorforce.core.preprocessing.sequence module

class tensorforce.core.preprocessing.sequence.**Sequence**(*length=2*, *scope='sequence'*, *summary_labels=()*)
Bases: *tensorforce.core.preprocessing.preprocessor.Preprocessor*

Concatenate `length` state vectors. Example: Used in Atari problems to create the Markov property.

processed_shape(*shape*)

reset()

tf_process(*tensor*)

tensorforce.core.preprocessing.standardize module

class tensorforce.core.preprocessing.standardize.**Standardize**(*across_batch=False*, *scope='standardize'*, *summary_labels=()*)

Bases: *tensorforce.core.preprocessing.preprocessor.Preprocessor*

Standardize state. Subtract mean and divide by standard deviation.

tf_process(*tensor*)

Module contents

class tensorforce.core.preprocessing.**Preprocessor**(*scope='preprocessor'*, *summary_labels=None*)

Bases: object

get_variables()

Returns the TensorFlow variables used by the preprocessor.

Returns List of variables.

processed_shape(*shape*)

Shape of preprocessed state given original shape.

Parameters **shape** – original shape.

Returns: processed tensor shape

reset()

tf_process(*tensor*)

Process state.

Parameters **tensor** – tensor to process.

Returns: processed tensor.

class tensorforce.core.preprocessing.**Sequence**(*length=2*, *scope='sequence'*, *summary_labels=()*)

Bases: *tensorforce.core.preprocessing.preprocessor.Preprocessor*

Concatenate `length` state vectors. Example: Used in Atari problems to create the Markov property.

processed_shape(*shape*)

reset()

tf_process(*tensor*)

class tensorforce.core.preprocessing.**Standardize**(*across_batch=False*, *scope='standardize'*, *summary_labels=()*)

Bases: *tensorforce.core.preprocessing.preprocessor.Preprocessor*

Standardize state. Subtract mean and divide by standard deviation.

tf_process(*tensor*)

class tensorforce.core.preprocessing.**RunningStandardize**(*axis=None*, *reset_after_batch=True*, *scope='running_standardize'*, *summary_labels=()*)

Bases: *tensorforce.core.preprocessing.preprocessor.Preprocessor*

Standardize state w.r.t past states. Subtract mean and divide by standard deviation of sequence of past states.

reset()

tf_process(*tensor*)

class tensorforce.core.preprocessing.**Normalize**(*scope='normalize'*, *summary_labels=()*)

Bases: *tensorforce.core.preprocessing.preprocessor.Preprocessor*

Normalize state. Subtract minimal value and divide by range.

tf_process(*tensor*)

class tensorforce.core.preprocessing.**Grayscale**(*weights=(0.299, 0.587, 0.114)*, *scope='grayscale'*, *summary_labels=()*)

Bases: *tensorforce.core.preprocessing.preprocessor.Preprocessor*

Turn 3D color state into grayscale.

processed_shape(*shape*)

tf_process(*tensor*)

class tensorforce.core.preprocessing.**ImageResize**(*width*, *height*, *scope='image_resize'*, *summary_labels=()*)

Bases: *tensorforce.core.preprocessing.preprocessor.Preprocessor*

Resize image to width x height.

processed_shape(*shape*)

tf_process(*tensor*)

class tensorforce.core.preprocessing.**PreprocessorStack**

Bases: object

static from_spec(*spec*)

Creates a preprocessing stack from a specification dict.

get_variables()

process(*tensor*)

Process state.

Parameters **tensor** – tensor to process

Returns: processed state

processed_shape(*shape*)

Shape of preprocessed state given original shape.

Parameters **shape** – original state shape

Returns: processed state shape

reset()

class tensorforce.core.preprocessing.**Divide**(*scale*, *scope='divide'*, *summary_labels=()*)
Bases: *tensorforce.core.preprocessing.preprocessor.Preprocessor*

Divide state by scale.

tf_process(*tensor*)

class tensorforce.core.preprocessing.**Clip**(*min_value*, *max_value*, *scope='clip'*, *summary_labels=()*)
Bases: *tensorforce.core.preprocessing.preprocessor.Preprocessor*

Clip by min/max.

tf_process(*tensor*)

Module contents

tensorforce.environments package

Submodules

tensorforce.environments.environment module

class tensorforce.environments.environment.**Environment**
Bases: object

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.

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

class tensorforce.environments.minimal_test.**MinimalTest**(*specification*)
Bases: *tensorforce.environments.environment.Environment*

actions

close()

execute(*actions*)

reset()

states

tensorforce.environments.minimal_test.**random**() → x in the interval [0, 1).

Module contents

class tensorforce.environments.**Environment**
Bases: object

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.

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.execution package

Submodules

tensorforce.execution.runner module

class tensorforce.execution.runner.**Runner**(*agent*, *environment*, *repeat_actions=1*, *history=None*)

Bases: object

Simple runner for non-realtime single-process execution.

reset(*history=None*)

run(*timesteps=None*, *episodes=None*, *max_episode_timesteps=None*, *deterministic=False*, *episode_finished=None*)
Runs the agent on the environment.

Parameters

- **timesteps** – Number of timesteps
- **episodes** – Number of episodes
- **max_episode_timesteps** – Max number of timesteps per episode
- **deterministic** – Deterministic flag
- **episode_finished** – Function handler taking a Runner argument and returning a boolean indicating whether to continue execution. For instance, useful for reporting intermediate performance or integrating termination conditions.

tensorforce.execution.threaded_runner module

Runner for non-realtime threaded execution of multiple agents.

class tensorforce.execution.threaded_runner.**ThreadedRunner**(*agents*, *environments*, *repeat_actions=1*, *save_path=None*, *save_episodes=None*)

Bases: object

run(*episodes=-1*, *max_timesteps=-1*, *episode_finished=None*, *summary_report=None*, *summary_interval=0*)

tensorforce.execution.threaded_runner.**WorkerAgentGenerator**(*agent_class*)
Worker Agent generator, recieves an Agent class and creates a Worker Agent class that inherits from that Agent.

Module contents

class tensorforce.execution.**Runner**(*agent*, *environment*, *repeat_actions=1*, *history=None*)

Bases: object

Simple runner for non-realtime single-process execution.

reset(*history=None*)

run(*timesteps=None*, *episodes=None*, *max_episode_timesteps=None*, *deterministic=False*, *episode_finished=None*)
Runs the agent on the environment.

Parameters

- **timesteps** – Number of timesteps
- **episodes** – Number of episodes
- **max_episode_timesteps** – Max number of timesteps per episode
- **deterministic** – Deterministic flag
- **episode_finished** – Function handler taking a `Runner` argument and returning a boolean indicating whether to continue execution. For instance, useful for reporting intermediate performance or integrating termination conditions.

class tensorforce.execution.**ThreadedRunner**(*agents*, *environments*, *repeat_actions=1*, *save_path=None*, *save_episodes=None*)

Bases: `object`

run(*episodes=-1*, *max_timesteps=-1*, *episode_finished=None*, *summary_report=None*, *summary_interval=0*)

tensorforce.models package

Submodules

tensorforce.models.constant_model module

class tensorforce.models.constant_model.**ConstantModel**(*states_spec*, *actions_spec*, *device*, *session_config*, *scope*, *saver_spec*, *summary_spec*, *distributed_spec*, *optimizer*, *discount*, *variable_noise*, *states_preprocessing_spec*, *explorations_spec*, *reward_preprocessing_spec*, *action_values*)

Bases: *tensorforce.models.model.Model*

Utility class to return constant actions of a desired shape and with given bounds.

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

tf_loss_per_instance(*states*, *internals*, *actions*, *terminal*, *reward*, *update*)

tensorforce.models.distribution_model module

class tensorforce.models.distribution_model.**DistributionModel**(*states_spec*, *actions_spec*, *network_spec*, *device*, *session_config*, *scope*, *saver_spec*, *summary_spec*, *distributed_spec*, *optimizer*, *discount*, *variable_noise*, *states_preprocessing_spec*, *explorations_spec*, *reward_preprocessing_spec*, *distributions_spec*, *entropy_regularization*)

Bases: *tensorforce.models.model.Model*

Base class for models using distributions parameterized by a neural network.

create_distributions()

static get_distributions_summaries(*distributions*)

static get_distributions_variables(*distributions*, *include_non_trainable=False*)

get_optimizer_kwargs(*states*, *internals*, *actions*, *terminal*, *reward*, *update*)

get_summaries()

get_variables(*include_non_trainable=False*)

initialize(*custom_getter*)

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

tf_kl_divergence(*states*, *internals*, *update*)

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 give 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.
- `action_exploration(action, exploration, action_spec)` for action postprocessing (e.g. exploration), returning the processed batch of actions.
- `preprocess_reward(states, internals, terminal, reward)` for reward preprocessing (e.g. reward normalization), returning the processed batch of rewards.
- `create_output_operations(states, internals, actions, terminal, reward, deterministic)` for further output operations, 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_spec*, *actions_spec*, *device*, *session_config*, *scope*, *saver_spec*, *summary_spec*, *distributed_spec*, *optimizer*, *discount*, *variable_noise*, *states_preprocessing_spec*, *explorations_spec*, *reward_preprocessing_spec*)

Bases: `object`

Base class for all (TensorFlow-based) models.

act(*states*, *internals*, *deterministic=False*)

close()

create_output_operations(*states*, *internals*, *actions*, *terminal*, *reward*, *update*, *deterministic*)

Calls all the relevant TensorFlow functions for this model and hence creates all the TensorFlow operations involved.

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.
- **update** – Boolean tensor indicating whether this call happens during an update.
- **deterministic** – Boolean tensor indicating whether action should be chosen deterministically.

get_optimizer_kwargs(*states*, *internals*, *actions*, *terminal*, *reward*, *update*)

Returns the optimizer arguments including the time, the list of variables to optimize, and various argument-free functions (in particular `fn_loss` returning the combined 0-dim batch loss tensor) 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.
- **update** – Boolean tensor indicating whether this call happens during an update.

Returns Loss tensor of the size of the batch.

get_summaries()
Returns the TensorFlow summaries reported by the model

Returns List of summaries

get_variables(*include_non_trainable=False*)
Returns the TensorFlow variables used by the model.

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*)

reset()
Resets the model to its initial state on episode start.

Returns Current episode and timestep counter, and a list containing the internal states initializations.

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.

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 were the model was saved.

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

tf_action_exploration(*action*, *exploration*, *action_spec*)
Applies optional exploration to the action.

tf_actions_and_internals(*states*, *internals*, *update*, *deterministic*)
Creates the TensorFlow operations for retrieving the actions (and posterior internal states) in reaction to the given input states (and prior internal 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.
- **deterministic** – Boolean tensor indicating whether action should be chosen deterministically.

Returns Actions and list of posterior internal state tensors.

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_loss(*states*, *internals*, *actions*, *terminal*, *reward*, *update*)

tf_loss_per_instance(*states*, *internals*, *actions*, *terminal*, *reward*, *update*)
Creates the TensorFlow operations for calculating the loss per batch instance of the given input states and actions.

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.
- **update** – Boolean tensor indicating whether this call happens during an update.

Returns Loss tensor.

tf_optimization(*states*, *internals*, *actions*, *terminal*, *reward*, *update*)
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.
- **update** – Boolean tensor indicating whether this call happens during an update.

Returns The optimization operation.

tf_preprocess_reward(*states*, *internals*, *terminal*, *reward*)
Applies optional pre-processing to the reward.

tf_preprocess_states(*states*)
Applies optional pre-processing to the states.

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.

update(*states*, *internals*, *actions*, *terminal*, *reward*, *return_loss_per_instance=False*)

tensorforce.models.pg_log_prob_model module

class tensorforce.models.pg_log_prob_model.**PGLogProbModel**(*states_spec*, *actions_spec*, *network_spec*, *device*, *session_config*, *scope*, *saver_spec*, *summary_spec*, *distributed_spec*, *optimizer*, *discount*, *variable_noise*, *states_preprocessing_spec*, *explorations_spec*, *reward_preprocessing_spec*, *distributions_spec*, *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.

tf_pg_loss_per_instance(*states*, *internals*, *actions*, *terminal*, *reward*, *update*)

tensorforce.models.pg_model module

class tensorforce.models.pg_model.**PGModel**(*states_spec*, *actions_spec*, *network_spec*, *device*, *session_config*, *scope*, *saver_spec*, *summary_spec*, *distributed_spec*, *optimizer*, *discount*, *variable_noise*, *states_preprocessing_spec*, *explorations_spec*, *reward_preprocessing_spec*, *distributions_spec*, *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`.

get_summaries()

get_variables(*include_non_trainable=False*)

initialize(*custom_getter*)

tf_loss_per_instance(*states*, *internals*, *actions*, *terminal*, *reward*, *update*)

tf_optimization(*states*, *internals*, *actions*, *terminal*, *reward*, *update*)

tf_pg_loss_per_instance(*states*, *internals*, *actions*, *terminal*, *reward*, *update*)

Creates the TensorFlow operations for calculating the (policy-gradient-specific) loss per batch instance of the given input states and actions, after the specified reward/advantage calculations.

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.
- **update** – Boolean tensor indicating whether this call happens during an update.

Returns Loss tensor.

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_spec*, *actions_spec*, *network_spec*, *device*, *session_config*, *scope*, *saver_spec*, *summary_spec*, *distributed_spec*, *optimizer*, *discount*, *variable_noise*, *states_preprocessing_spec*, *explorations_spec*, *reward_preprocessing_spec*, *distributions_spec*, *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.

get_optimizer_kwargs(*states*, *actions*, *terminal*, *reward*, *internals*, *update*)

initialize(*custom_getter*)

tf_compare(*states*, *internals*, *actions*, *terminal*, *reward*, *update*, *reference*)

tf_pg_loss_per_instance(*states*, *internals*, *actions*, *terminal*, *reward*, *update*)

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

tensorforce.models.q_demo_model module

class tensorforce.models.q_demo_model.**QDemoModel**(*states_spec*, *actions_spec*, *device*, *session_config*, *scope*, *saver_spec*, *summary_spec*, *distributed_spec*, *optimizer*, *discount*, *variable_noise*, *states_preprocessing_spec*, *explorations_spec*, *reward_preprocessing_spec*, *network_spec*, *distributions_spec*, *entropy_regularization*, *target_sync_frequency*, *target_update_weight*, *double_q_model*, *huber_loss*, *random_sampling_fix*, *expert_margin*, *supervised_weight*)

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.

create_output_operations(*states*, *internals*, *actions*, *terminal*, *reward*, *update*, *deterministic*)

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

initialize(*custom_getter*)

tf_demo_loss(*states*, *actions*, *terminal*, *reward*, *internals*, *update*)

tf_demo_optimization(*states*, *internals*, *actions*, *terminal*, *reward*, *update*)

tensorforce.models.q_model module

class tensorforce.models.q_model.**QModel**(*states_spec*, *actions_spec*, *network_spec*, *device*, *session_config*, *scope*, *saver_spec*, *summary_spec*, *distributed_spec*, *optimizer*, *discount*, *variable_noise*, *states_preprocessing_spec*, *explorations_spec*, *reward_preprocessing_spec*, *distributions_spec*, *entropy_regularization*, *target_sync_frequency*, *target_update_weight*, *double_q_model*, *huber_loss*, *random_sampling_fix*)

Bases: *tensorforce.models.distribution_model.DistributionModel*

Q-value model.

get_summaries()

get_variables(*include_non_trainable=False*)

initialize(*custom_getter*)

tf_loss_per_instance(*states*, *internals*, *actions*, *terminal*, *reward*, *update*)

tf_optimization(*states*, *internals*, *actions*, *terminal*, *reward*, *update*)

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*)

update(*states*, *internals*, *actions*, *terminal*, *reward*, *return_loss_per_instance=False*)

tensorforce.models.q_naf_model module

class tensorforce.models.q_naf_model.**QNAFModel**(*states_spec*, *actions_spec*, *network_spec*, *device*, *session_config*, *scope*, *saver_spec*, *summary_spec*, *distributed_spec*, *optimizer*, *discount*, *variable_noise*, *states_preprocessing_spec*, *explorations_spec*, *reward_preprocessing_spec*, *distributions_spec*, *entropy_regularization*, *target_sync_frequency*, *target_update_weight*, *double_q_model*, *huber_loss*, *random_sampling_fix*)

Bases: *tensorforce.models.q_model.QModel*

get_variables(*include_non_trainable=False*)

initialize(*custom_getter*)

tf_loss_per_instance(*states*, *internals*, *actions*, *terminal*, *reward*, *update*)

tf_q_value(*embedding*, *distr_params*, *action*, *name*)

tf_regularization_losses(*states*, *internals*, *update*)

tensorforce.models.q_nstep_model module

class tensorforce.models.q_nstep_model.**QNstepModel**(*states_spec*, *actions_spec*, *network_spec*, *device*, *session_config*, *scope*, *saver_spec*, *summary_spec*, *distributed_spec*, *optimizer*, *discount*, *variable_noise*, *states_preprocessing_spec*, *explorations_spec*, *reward_preprocessing_spec*, *distributions_spec*, *entropy_regularization*, *target_sync_frequency*, *target_update_weight*, *double_q_model*, *huber_loss*, *random_sampling_fix*)

Bases: *tensorforce.models.q_model.QModel*

Deep Q network using n-step rewards as described in Asynchronous Methods for Reinforcement Learning.

tf_q_delta(*q_value*, *next_q_value*, *terminal*, *reward*)

tensorforce.models.random_model module

class tensorforce.models.random_model.**RandomModel**(*states_spec*, *actions_spec*, *device*, *session_config*, *scope*, *saver_spec*, *summary_spec*, *distributed_spec*, *optimizer*, *discount*, *variable_noise*, *states_preprocessing_spec*, *explorations_spec*, *reward_preprocessing_spec*)

Bases: *tensorforce.models.model.Model*

Utility class to return random actions of a desired shape and with given bounds.

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

tf_loss_per_instance(*states*, *internals*, *actions*, *terminal*, *reward*, *update*)

Module contents

class tensorforce.models.**Model**(*states_spec*, *actions_spec*, *device*, *session_config*, *scope*, *saver_spec*, *summary_spec*, *distributed_spec*, *optimizer*, *discount*, *variable_noise*, *states_preprocessing_spec*, *explorations_spec*, *reward_preprocessing_spec*)

Bases: object

Base class for all (TensorFlow-based) models.

act(*states*, *internals*, *deterministic=False*)

close()

create_output_operations(*states*, *internals*, *actions*, *terminal*, *reward*, *update*, *deterministic*)
Calls all the relevant TensorFlow functions for this model and hence creates all the TensorFlow operations involved.

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.
- **update** – Boolean tensor indicating whether this call happens during an update.
- **deterministic** – Boolean tensor indicating whether action should be chosen deterministically.

get_optimizer_kwargs(*states*, *internals*, *actions*, *terminal*, *reward*, *update*)
Returns the optimizer arguments including the time, the list of variables to optimize, and various argument-free functions (in particular `fn_loss` returning the combined 0-dim batch loss tensor) 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.
- **update** – Boolean tensor indicating whether this call happens during an update.

Returns Loss tensor of the size of the batch.

get_summaries()
Returns the TensorFlow summaries reported by the model

Returns List of summaries

get_variables(*include_non_trainable=False*)
Returns the TensorFlow variables used by the model.

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*)

reset()
Resets the model to its initial state on episode start.

Returns Current episode and timestep counter, and a list containing the internal states initializations.

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.

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 were the model was saved.

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

tf_action_exploration(*action*, *exploration*, *action_spec*)
Applies optional exploration to the action.

tf_actions_and_internals(*states*, *internals*, *update*, *deterministic*)
Creates the TensorFlow operations for retrieving the actions (and posterior internal states) in reaction to the given input states (and prior internal 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.
- **deterministic** – Boolean tensor indicating whether action should be chosen deterministically.

Returns Actions and list of posterior internal state tensors.

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_loss(*states*, *internals*, *actions*, *terminal*, *reward*, *update*)

tf_loss_per_instance(*states*, *internals*, *actions*, *terminal*, *reward*, *update*)
Creates the TensorFlow operations for calculating the loss per batch instance of the given input states and actions.

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.
- **update** – Boolean tensor indicating whether this call happens during an update.

Returns Loss tensor.

tf_optimization(*states*, *internals*, *actions*, *terminal*, *reward*, *update*)
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.
- **update** – Boolean tensor indicating whether this call happens during an update.

Returns The optimization operation.

tf_preprocess_reward(*states*, *internals*, *terminal*, *reward*)
Applies optional pre-processing to the reward.

tf_preprocess_states(*states*)
Applies optional pre-processing to the states.

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.

update(*states*, *internals*, *actions*, *terminal*, *reward*, *return_loss_per_instance=False*)

class tensorforce.models.**DistributionModel**(*states_spec*, *actions_spec*, *network_spec*, *device*, *session_config*, *scope*, *saver_spec*, *summary_spec*, *distributed_spec*, *optimizer*, *discount*, *variable_noise*, *states_preprocessing_spec*, *explorations_spec*, *reward_preprocessing_spec*, *distributions_spec*, *entropy_regularization*)

Bases: *tensorforce.models.model.Model*

Base class for models using distributions parameterized by a neural network.

create_distributions()

static get_distributions_summaries(*distributions*)

static get_distributions_variables(*distributions*, *include_non_trainable=False*)

get_optimizer_kwargs(*states*, *internals*, *actions*, *terminal*, *reward*, *update*)

get_summaries()

get_variables(*include_non_trainable=False*)

initialize(*custom_getter*)

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

tf_kl_divergence(*states*, *internals*, *update*)

tf_regularization_losses(*states*, *internals*, *update*)

class tensorforce.models.**PGModel**(*states_spec*, *actions_spec*, *network_spec*, *device*, *session_config*, *scope*, *saver_spec*, *summary_spec*, *distributed_spec*, *optimizer*, *discount*, *variable_noise*, *states_preprocessing_spec*, *explorations_spec*, *reward_preprocessing_spec*, *distributions_spec*, *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`.

get_summaries()

get_variables(*include_non_trainable=False*)

initialize(*custom_getter*)

tf_loss_per_instance(*states*, *internals*, *actions*, *terminal*, *reward*, *update*)

tf_optimization(*states*, *internals*, *actions*, *terminal*, *reward*, *update*)

tf_pg_loss_per_instance(*states*, *internals*, *actions*, *terminal*, *reward*, *update*)

Creates the TensorFlow operations for calculating the (policy-gradient-specific) loss per batch instance of the given input states and actions, after the specified reward/advantage calculations.

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.
- **update** – Boolean tensor indicating whether this call happens during an update.

Returns Loss tensor.

tf_regularization_losses(*states*, *internals*, *update*)

tf_reward_estimation(*states*, *internals*, *terminal*, *reward*, *update*)

class tensorforce.models.**PGProbRatioModel**(*states_spec*, *actions_spec*, *network_spec*, *device*, *session_config*, *scope*, *saver_spec*, *summary_spec*, *distributed_spec*, *optimizer*, *discount*, *variable_noise*, *states_preprocessing_spec*, *explorations_spec*, *reward_preprocessing_spec*, *distributions_spec*, *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.

get_optimizer_kwargs(*states*, *actions*, *terminal*, *reward*, *internals*, *update*)

initialize(*custom_getter*)

tf_compare(*states*, *internals*, *actions*, *terminal*, *reward*, *update*, *reference*)

tf_pg_loss_per_instance(*states*, *internals*, *actions*, *terminal*, *reward*, *update*)

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

class tensorforce.models.**PGLogProbModel**(*states_spec*, *actions_spec*, *network_spec*, *device*, *session_config*, *scope*, *saver_spec*, *summary_spec*, *distributed_spec*, *optimizer*, *discount*, *variable_noise*, *states_preprocessing_spec*, *explorations_spec*, *reward_preprocessing_spec*, *distributions_spec*, *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.

tf_pg_loss_per_instance(*states*, *internals*, *actions*, *terminal*, *reward*, *update*)

class tensorforce.models.**QModel**(*states_spec*, *actions_spec*, *network_spec*, *device*, *session_config*, *scope*, *saver_spec*, *summary_spec*, *distributed_spec*, *optimizer*, *discount*, *variable_noise*, *states_preprocessing_spec*, *explorations_spec*, *reward_preprocessing_spec*, *distributions_spec*, *entropy_regularization*, *target_sync_frequency*, *target_update_weight*, *double_q_model*, *huber_loss*, *random_sampling_fix*)

Bases: *tensorforce.models.distribution_model.DistributionModel*

Q-value model.

get_summaries()

get_variables(*include_non_trainable=False*)

initialize(*custom_getter*)

tf_loss_per_instance(*states*, *internals*, *actions*, *terminal*, *reward*, *update*)

tf_optimization(*states*, *internals*, *actions*, *terminal*, *reward*, *update*)

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*)

update(*states*, *internals*, *actions*, *terminal*, *reward*, *return_loss_per_instance=False*)

class tensorforce.models.**QNstepModel**(*states_spec*, *actions_spec*, *network_spec*, *device*, *session_config*, *scope*, *saver_spec*, *summary_spec*, *distributed_spec*, *optimizer*, *discount*, *variable_noise*, *states_preprocessing_spec*, *explorations_spec*, *reward_preprocessing_spec*, *distributions_spec*, *entropy_regularization*, *target_sync_frequency*, *target_update_weight*, *double_q_model*, *huber_loss*, *random_sampling_fix*)

Bases: *tensorforce.models.q_model.QModel*

Deep Q network using n-step rewards as described in Asynchronous Methods for Reinforcement Learning.

tf_q_delta(*q_value*, *next_q_value*, *terminal*, *reward*)

class tensorforce.models.**QNAFModel**(*states_spec*, *actions_spec*, *network_spec*, *device*, *session_config*, *scope*, *saver_spec*, *summary_spec*, *distributed_spec*, *optimizer*, *discount*, *variable_noise*, *states_preprocessing_spec*, *explorations_spec*, *reward_preprocessing_spec*, *distributions_spec*, *entropy_regularization*, *target_sync_frequency*, *target_update_weight*, *double_q_model*, *huber_loss*, *random_sampling_fix*)

Bases: *tensorforce.models.q_model.QModel*

get_variables(*include_non_trainable=False*)

initialize(*custom_getter*)

tf_loss_per_instance(*states*, *internals*, *actions*, *terminal*, *reward*, *update*)

tf_q_value(*embedding*, *distr_params*, *action*, *name*)

tf_regularization_losses(*states*, *internals*, *update*)

class tensorforce.models.**QDemoModel**(*states_spec*, *actions_spec*, *device*, *session_config*, *scope*, *saver_spec*, *summary_spec*, *distributed_spec*, *optimizer*, *discount*, *variable_noise*, *states_preprocessing_spec*, *explorations_spec*, *reward_preprocessing_spec*, *network_spec*, *distributions_spec*, *entropy_regularization*, *target_sync_frequency*, *target_update_weight*, *double_q_model*, *huber_loss*, *random_sampling_fix*, *expert_margin*, *supervised_weight*)

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.

create_output_operations(*states*, *internals*, *actions*, *terminal*, *reward*, *update*, *deterministic*)

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

initialize(*custom_getter*)

tf_demo_loss(*states*, *actions*, *terminal*, *reward*, *internals*, *update*)

tf_demo_optimization(*states*, *internals*, *actions*, *terminal*, *reward*, *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.

config = None

exclude_bool = False

exclude_bounded = False

exclude_float = False

exclude_int = False

exclude_lstm = False

exclude_multi = False

multi_config = None

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.

agent = None

base_test_pass(*name*, *environment*, *network_spec*, ***kwargs*)
Basic test loop, requires an Agent to achieve a certain performance on an environment.

base_test_run(*name*, *environment*, *network_spec*, ***kwargs*)
Run test, tests whether algorithm can run and update without compilation errors, not whether it passes.

deterministic = None

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

agent
alias of `ConstantAgent`

config = {'action_values': {'action': 1.0}}

deterministic = False

exclude_bool = True

exclude_bounded = True

exclude_int = True

exclude_lstm = True

exclude_multi = True

requires_network = False

tensorforce.tests.test_ddqn_agent module

class tensorforce.tests.test_ddqn_agent.**TestDDQNAgent**(*methodName='runTest'*)
Bases: *tensorforce.tests.base_agent_test.BaseAgentTest*, unittest.case.TestCase

agent
alias of `DDQNAgent`

config = {'optimizer': {'learning_rate': 0.002, 'type': 'adam'}, 'repeat_update':

deterministic = True

exclude_bounded = True

exclude_float = True

multi_config = {'optimizer': {'learning_rate': 0.01, 'type': 'adam'}, 'repeat_updat

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

agent
alias of DQFDAgent

config = {'demo_sampling_ratio': 0.2, 'memory': {'capacity': 1000, 'type': 'replay

deterministic = True

exclude_bounded = True

exclude_float = True

multi_config = {'optimizer': {'learning_rate': 0.01, 'type': 'adam'}, 'target_sync_

pre_run(*agent*, *environment*)

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

agent
alias of DQNAgent

config = {'optimizer': {'learning_rate': 0.002, 'type': 'adam'}, 'repeat_update':

deterministic = True

exclude_bounded = True

exclude_float = True

multi_config = {'optimizer': {'learning_rate': 0.01, 'type': 'adam'}, 'repeat_updat

tensorforce.tests.test_dqn_memories module

class tensorforce.tests.test_dqn_memories.**TestDQNMemories**(*methodName='runTest'*)
Bases: *tensorforce.tests.base_test.BaseTest*, unittest.case.TestCase

agent
alias of DQNAgent

deterministic = True

test_naive_prioritized_replay()

test_prioritized_replay()

test_replay()

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

agent
alias of DQNNstepAgent

config = {'optimizer': {'learning_rate': 0.01, 'type': 'adam'}, 'batch_size': 8}

deterministic = True

exclude_bounded = True

exclude_float = True

exclude_multi = True

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

agent
alias of NAFAgent

config = {'optimizer': {'learning_rate': 0.001, 'type': 'adam'}, 'repeat_update':

deterministic = True

exclude_bool = True

exclude_bounded = True

exclude_int = True

exclude_lstm = True

exclude_multi = True

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

agent
alias of PPOAgent

config = {'batch_size': 8}

deterministic = False

multi_config = {'step_optimizer': {'learning_rate': 0.001, 'type': 'adam'}, 'batch_

tensorforce.tests.test_quickstart_example module

class tensorforce.tests.test_quickstart_example.**TestQuickstartExample**(*methodName='runTest'*)
Bases: unittest.case.TestCase

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

agent
alias of RandomAgent

config = {}

deterministic = False

exclude_lstm = True

pass_threshold = 0.0

requires_network = False

tensorforce.tests.test_reward_estimation module

class tensorforce.tests.test_reward_estimation.**TestRewardEstimation**(*methodName='runTest'*)
Bases: unittest.case.TestCase

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

agent
alias of TRPOAgent

config = {'learning_rate': 0.005, 'batch_size': 16}

deterministic = False

multi_config = {'learning_rate': 0.1, 'batch_size': 64}

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

execute(*action*)

get_state()

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

agent
alias of VPGAgent

config = {'batch_size': 8}

deterministic = False

multi_config = {'optimizer': {'learning_rate': 0.01, 'type': 'adam'}, 'batch_size':

tensorforce.tests.test_vpg_baselines module

class tensorforce.tests.test_vpg_baselines.**TestVPGBaselines**(*methodName='runTest'*)
Bases: *tensorforce.tests.base_test.BaseTest*, unittest.case.TestCase

agent
alias of VPGAgent

deterministic = False

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

agent
alias of VPGAgent

deterministic = False

test_adam()

test_clipped_step()

test_evolutionary()

test_multi_step()

test_natural_gradient()

test_optimized_step()

Module contents

1.6.2 Submodules

1.6.3 tensorforce.exception module

exception tensorforce.exception.**TensorForceError**
Bases: exceptions.Exception

TensorForce error

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

build_metagraph_list()
Convert MetaParams into TF Summary Format and create summary_op

Parameters **None** –

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

tensorforce.util.**cumulative_discount**(*values*, *terminals*, *discount*, *cumulative_start=0.0*)
Compute cumulative discounts. :param values: Values to discount :param terminals: Booleans indicating terminal states :param discount: Discount factor :param cumulative_start: Float or ndarray, estimated reward for state t + 1. Default 0.0

Returns The cumulative discounted rewards.

Return type dicounted_values

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.**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.**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.**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

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

A

B

C

D

E

F

G

I

L

M

Q

R

S

T

U

V

W