ChainerRL Documentation

Release 0.1.0

Preferred Networks, Inc.

Mar 27, 2017

Contents

ChainerRL is a deep reinforcement learning library that implements various state-of-the-art deep reinforcement algorithms in Python using Chainer, a flexible deep learning framework.

CHAPTER 1

Installation

How to install ChainerRL

ChainerRL is tested with Python 2.7+ and 3.5.1+. For other requirements, see `requirements.txt`.

Listing 1.1: requirements.txt

```
cached-property
chainer
fastcache; python_version<'3.2'
funcsigs; python_version<'3.5'
future
gym>=0.7.3
numpy>=1.10.4
pillow
scipy
statistics; python_version<'3.4'
```

ChainerRL can be installed via PyPI:

```
pip install chainerrl
```

It can also be installed from the source code:

```
python setup.py install
```

For Windows users

ChainerRL contains `atari_py` as dependencies, and windows users may face errors while installing it. This problem is discussed in OpenAI gym issues, and one possible counter measure is to enable "Bash on Ubuntu on Windows" for Windows 10 users.

Refer Official install guilde to install "Bash on Ubuntu on Windows".

CHAPTER 2

API Reference

Action values

Action value interfaces

class chainerrl.action_value.**ActionValue**
Struct that holds state-fixed Q-functions and its subproducts.

Every operation it supports is done in a batch manner.

evaluate_actions(*actions*)
Evaluate Q(s,a) with a = given actions.

greedy_actions
Get argmax_a Q(s,a).

max
Evaluate max Q(s,a).

Action value implementations

class chainerrl.action_value.**DiscreteActionValue**(*q_values*, *q_values_formatter=<function <lambda>>*)
Qfunction output for discrete action space.

Parameters **q_values** (*ndarray or chainer.Variable*) – Array of Q values whose shape is (batchsize, n_actions)

class chainerrl.action_value.**QuadraticActionValue**(*mu*, *mat*, *v*, *min_action=None*, *max_action=None*)
Qfunction output for continuous action space.

See: http://arxiv.org/abs/1603.00748

Define a Q(s,a) with A(s,a) in a quadratic form.

Q(s,a) = V(s,a) + A(s,a) A(s,a) = -1/2 (u - mu(s))^T P(s) (u - mu(s))

Parameters

- **mu** (*chainer.Variable*) – mu(s), actions that maximize A(s,a)
- **mat** (*chainer.Variable*) – P(s), coefficient matrices of A(s,a). It must be positive definite.
- **v** (*chainer.Variable*) – V(s), values of s
- **min_action** (*ndarray*) – mininum action, not batched
- **max_action** (*ndarray*) – maximum action, not batched

class chainerrl.action_value.**SingleActionValue**(*evaluator*, *maximizer=None*)
ActionValue that can evaluate only a single action.

Agents

Agent interfaces

class chainerrl.agent.**Agent**
Abstract agent class.

act(*obs*)
Select an action for evaluation.

Returns action

Return type ~object

act_and_train(*obs*, *reward*)
Select an action for training.

Returns action

Return type ~object

get_statistics()
Get statistics of the agent.

Returns

List of two-item tuples. The first item in a tuple is a str that represents the name of item, while the second item is a value to be recorded.

Example: [('average_loss': 0), ('average_value': 1), ...]

load(*dirname*)
Load internal states.

Returns None

save(*dirname*)
Save internal states.

Returns None

stop_episode()
Prepare for a new episode.

Returns None

stop_episode_and_train (*state*, *reward*, *done=False*)
Observe consequences and prepare for a new episode.

Returns None

Agent implementations

class chainerrl.agents.**A3C**(*model*, *optimizer*, *t_max*, *gamma*, *beta=0.01*, *process_idx=0*, *phi=<function <lambda>>*, *pi_loss_coef=1.0*, *v_loss_coef=0.5*, *keep_loss_scale_same=False*, *normalize_grad_by_t_max=False*, *use_average_reward=False*, *average_reward_tau=0.01*, *act_deterministically=False*, *average_entropy_decay=0.999*, *average_value_decay=0.999*, *batch_states=<function batch_states>*)
A3C: Asynchronous Advantage Actor-Critic.

See http://arxiv.org/abs/1602.01783

Parameters

- **model** (*A3CModel*) – Model to train
- **optimizer** (*chainer.Optimizer*) – optimizer used to train the model
- **t_max** (*int*) – The model is updated after every t_max local steps
- **gamma** (*float*) – Discount factor [0,1]
- **beta** (*float*) – Weight coefficient for the entropy regularizaiton term.
- **process_idx** (*int*) – Index of the process.
- **phi** (*callable*) – Feature extractor function
- **pi_loss_coef** (*float*) – Weight coefficient for the loss of the policy
- **v_loss_coef** (*float*) – Weight coefficient for the loss of the value function
- **act_deterministically** (*bool*) – If set true, choose most probable actions in act method.
- **batch_states** (*callable*) – method which makes a batch of observations. default is *chainerrl.misc.batch_states.batch_states*

class chainerrl.agents.**ACER**(*model*, *optimizer*, *t_max*, *gamma*, *replay_buffer*, *beta=0.01*, *phi=<function <lambda>>*, *pi_loss_coef=1.0*, *Q_loss_coef=0.5*, *use_trust_region=True*, *trust_region_alpha=0.99*, *trust_region_delta=1*, *truncation_threshold=10*, *disable_online_update=False*, *n_times_replay=8*, *replay_start_size=10000*, *normalize_loss_by_steps=True*, *act_deterministically=False*, *use_Q_opc=False*, *average_entropy_decay=0.999*, *average_value_decay=0.999*, *average_kl_decay=0.999*, *logger=None*)
ACER (Actor-Critic with Experience Replay).

See http://arxiv.org/abs/1611.01224

Parameters

- **model** (*ACERModel*) – Model to train. It must be a callable that accepts observations as input and return three values: action distributions (Distribution), Q values (ActionValue) and state values (chainer.Variable).
- **optimizer** (*chainer.Optimizer*) – optimizer used to train the model

- **t_max** (*int*) – The model is updated after every t_max local steps
- **gamma** (*float*) – Discount factor [0,1]
- **replay_buffer** (*EpisodicReplayBuffer*) – Replay buffer to use. If set None, this agent won't use experience replay.
- **beta** (*float*) – Weight coefficient for the entropy regularizaiton term.
- **phi** (*callable*) – Feature extractor function
- **pi_loss_coef** (*float*) – Weight coefficient for the loss of the policy
- **Q_loss_coef** (*float*) – Weight coefficient for the loss of the value function
- **use_trust_region** (*bool*) – If set true, use efficient TRPO.
- **trust_region_alpha** (*float*) – Decay rate of the average model used for efficient TRPO.
- **trust_region_delta** (*float*) – Threshold used for efficient TRPO.
- **truncation_threshold** (*float or None*) – Threshold used to truncate larger importance weights. If set None, importance weights are not truncated.
- **disable_online_update** (*bool*) – If set true, disable online on-policy update and rely only on experience replay.
- **n_times_replay** (*int*) – Number of times experience replay is repeated per one time of online update.
- **replay_start_size** (*int*) – Experience replay is disabled if the number of transitions in the replay buffer is lower than this value.
- **normalize_loss_by_steps** (*bool*) – If set true, losses are normalized by the number of steps taken to accumulate the losses
- **act_deterministically** (*bool*) – If set true, choose most probable actions in act method.
- **use_Q_opc** (*bool*) – If set true, use Q_opc, a Q-value estimate without importance sampling, is used to compute advantage values for policy gradients. The original paper recommend to use in case of continuous action.
- **average_entropy_decay** (*float*) – Decay rate of average entropy. Used only to record statistics.
- **average_value_decay** (*float*) – Decay rate of average value. Used only to record statistics.
- **average_kl_decay** (*float*) – Decay rate of kl value. Used only to record statistics.

class chainerrl.agents.**AL**(**args*, ***kwargs*)

Advantage Learning.

See: http://arxiv.org/abs/1512.04860.

Parameters **alpha** (*float*) – Weight of (persistent) advantages. Convergence is guaranteed only for alpha in [0, 1).

For other arguments, see DQN.

class chainerrl.agents.**DDPG**(*model*, *actor_optimizer*, *critic_optimizer*, *replay_buffer*, *gamma*, *explorer*, *gpu=None*, *replay_start_size=50000*, *minibatch_size=32*, *update_frequency=1*, *target_update_frequency=10000*, *phi=<function <lambda>>*, *target_update_method=u'hard'*, *soft_update_tau=0.01*, *n_times_update=1*, *average_q_decay=0.999*, *average_loss_decay=0.99*, *episodic_update=False*, *episodic_update_len=None*, *logger=<logging.Logger object>*, *batch_states=<function batch_states>*)

Deep Deterministic Policy Gradients.

This can be used as SVG(0) by specifying a Gaussina policy instead of a deterministic policy.

Parameters

- **model** (*DDPGModel*) – DDPG model that contains both a policy and a Q-function
- **actor_optimizer** (*Optimizer*) – Optimizer setup with the policy
- **critic_optimizer** (*Optimizer*) – Optimizer setup with the Q-function
- **replay_buffer** (*ReplayBuffer*) – Replay buffer
- **gamma** (*float*) – Discount factor
- **explorer** (*Explorer*) – Explorer that specifies an exploration strategy.
- **gpu** (*int*) – GPU device id if not None nor negative.
- **replay_start_size** (*int*) – if the replay buffer's size is less than replay_start_size, skip update
- **minibatch_size** (*int*) – Minibatch size
- **update_frequency** (*int*) – Model update frequency in step
- **target_update_frequency** (*int*) – Target model update frequency in step
- **phi** (*callable*) – Feature extractor applied to observations
- **target_update_method** (*str*) – 'hard' or 'soft'.
- **soft_update_tau** (*float*) – Tau of soft target update.
- **n_times_update** (*int*) – Number of repetition of update
- **average_q_decay** (*float*) – Decay rate of average Q, only used for recording statistics
- **average_loss_decay** (*float*) – Decay rate of average loss, only used for recording statistics
- **batch_accumulator** (*str*) – 'mean' or 'sum'
- **episodic_update** (*bool*) – Use full episodes for update if set True
- **episodic_update_len** (*int or None*) – Subsequences of this length are used for update if set int and episodic_update=True
- **logger** (*Logger*) – Logger used
- **batch_states** (*callable*) – method which makes a batch of observations. default is *chainerrl.misc.batch_states.batch_states*

class chainerrl.agents.**DoubleDQN**(*q_function*, *optimizer*, *replay_buffer*, *gamma*, *explorer*, *gpu=None*, *replay_start_size=50000*, *minibatch_size=32*, *update_frequency=1*, *target_update_frequency=10000*, *clip_delta=True*, *phi=<function <lambda>>*, *target_update_method=u'hard'*, *soft_update_tau=0.01*, *n_times_update=1*, *average_q_decay=0.999*, *average_loss_decay=0.99*, *batch_accumulator=u'mean'*, *episodic_update=False*, *episodic_update_len=None*, *logger=<logging.Logger object>*, *batch_states=<function batch_states>*)

Double DQN.

See: http://arxiv.org/abs/1509.06461.

class chainerrl.agents.**DoublePAL**(**args*, ***kwargs*)

class chainerrl.agents.**DPP**(**args*, ***kwargs*)

Dynamic Policy Programming with softmax operator.

Parameters **eta** (*float*) – Positive constant.

For other arguments, see DQN.

class chainerrl.agents.**DQN**(*q_function*, *optimizer*, *replay_buffer*, *gamma*, *explorer*, *gpu=None*, *replay_start_size=50000*, *minibatch_size=32*, *update_frequency=1*, *target_update_frequency=10000*, *clip_delta=True*, *phi=<function <lambda>>*, *target_update_method=u'hard'*, *soft_update_tau=0.01*, *n_times_update=1*, *average_q_decay=0.999*, *average_loss_decay=0.99*, *batch_accumulator=u'mean'*, *episodic_update=False*, *episodic_update_len=None*, *logger=<logging.Logger object>*, *batch_states=<function batch_states>*)

Deep Q-Network algorithm.

Parameters

- **q_function** (*StateQFunction*) – Q-function
- **optimizer** (*Optimizer*) – Optimizer that is already setup
- **replay_buffer** (*ReplayBuffer*) – Replay buffer
- **gamma** (*float*) – Discount factor
- **explorer** (*Explorer*) – Explorer that specifies an exploration strategy.
- **gpu** (*int*) – GPU device id if not None nor negative.
- **replay_start_size** (*int*) – if the replay buffer's size is less than replay_start_size, skip update
- **minibatch_size** (*int*) – Minibatch size
- **update_frequency** (*int*) – Model update frequency in step
- **target_update_frequency** (*int*) – Target model update frequency in step
- **clip_delta** (*bool*) – Clip delta if set True
- **phi** (*callable*) – Feature extractor applied to observations
- **target_update_method** (*str*) – 'hard' or 'soft'.
- **soft_update_tau** (*float*) – Tau of soft target update.
- **n_times_update** (*int*) – Number of repetition of update

- **average_q_decay** (*float*) – Decay rate of average Q, only used for recording statistics
- **average_loss_decay** (*float*) – Decay rate of average loss, only used for recording statistics
- **batch_accumulator** (*str*) – 'mean' or 'sum'
- **episodic_update** (*bool*) – Use full episodes for update if set True
- **episodic_update_len** (*int or None*) – Subsequences of this length are used for update if set int and episodic_update=True
- **logger** (*Logger*) – Logger used
- **batch_states** (*callable*) – method which makes a batch of observations. default is *chainerrl.misc.batch_states.batch_states*

class chainerrl.agents.**DQN**(*q_function*, *optimizer*, *replay_buffer*, *gamma*, *explorer*, *gpu=None*, *replay_start_size=50000*, *minibatch_size=32*, *update_frequency=1*, *target_update_frequency=10000*, *clip_delta=True*, *phi=<function <lambda>>*, *target_update_method=u'hard'*, *soft_update_tau=0.01*, *n_times_update=1*, *average_q_decay=0.999*, *average_loss_decay=0.99*, *batch_accumulator=u'mean'*, *episodic_update=False*, *episodic_update_len=None*, *logger=<logging.Logger object>*, *batch_states=<function batch_states>*)

Deep Q-Network algorithm.

Parameters

- **q_function** (*StateQFunction*) – Q-function
- **optimizer** (*Optimizer*) – Optimizer that is already setup
- **replay_buffer** (*ReplayBuffer*) – Replay buffer
- **gamma** (*float*) – Discount factor
- **explorer** (*Explorer*) – Explorer that specifies an exploration strategy.
- **gpu** (*int*) – GPU device id if not None nor negative.
- **replay_start_size** (*int*) – if the replay buffer's size is less than replay_start_size, skip update
- **minibatch_size** (*int*) – Minibatch size
- **update_frequency** (*int*) – Model update frequency in step
- **target_update_frequency** (*int*) – Target model update frequency in step
- **clip_delta** (*bool*) – Clip delta if set True
- **phi** (*callable*) – Feature extractor applied to observations
- **target_update_method** (*str*) – 'hard' or 'soft'.
- **soft_update_tau** (*float*) – Tau of soft target update.
- **n_times_update** (*int*) – Number of repetition of update
- **average_q_decay** (*float*) – Decay rate of average Q, only used for recording statistics
- **average_loss_decay** (*float*) – Decay rate of average loss, only used for recording statistics
- **batch_accumulator** (*str*) – 'mean' or 'sum'
- **episodic_update** (*bool*) – Use full episodes for update if set True

- **episodic_update_len** (*int or None*) – Subsequences of this length are used for update if set int and episodic_update=True
- **logger** (*Logger*) – Logger used
- **batch_states** (*callable*) – method which makes a batch of observations. default is *chainerrl.misc.batch_states.batch_states*

class chainerrl.agents.**NSQ**(*q_function*, *optimizer*, *t_max*, *gamma*, *i_target*, *explorer*, *phi=<function <lambda>>*, *average_q_decay=0.999*, *logger=<logging.Logger object>*, *batch_states=<function batch_states>*)

Asynchronous N-step Q-Learning.

See http://arxiv.org/abs/1602.01783

Parameters

- **q_function** (*A3CModel*) – Model to train
- **optimizer** (*chainer.Optimizer*) – optimizer used to train the model
- **t_max** (*int*) – The model is updated after every t_max local steps
- **gamma** (*float*) – Discount factor [0,1]
- **i_target** (*intn*) – The target model is updated after every i_target global steps
- **explorer** (*Explorer*) – Explorer to use in training
- **phi** (*callable*) – Feature extractor function
- **average_q_decay** (*float*) – Decay rate of average Q, only used for recording statistics
- **batch_states** (*callable*) – method which makes a batch of observations. default is *chainerrl.misc.batch_states.batch_states*

class chainerrl.agents.**PAL**(**args*, ***kwargs*)

Persistent Advantage Learning.

See: http://arxiv.org/abs/1512.04860.

Parameters **alpha** (*float*) – Weight of (persistent) advantages. Convergence is guaranteed only for alpha in [0, 1).

For other arguments, see DQN.

class chainerrl.agents.**PCL**(*model*, *optimizer*, *replay_buffer=None*, *t_max=None*, *gamma=0.99*, *tau=0.01*, *phi=<function <lambda>>*, *pi_loss_coef=1.0*, *v_loss_coef=0.5*, *rollout_len=10*, *batchsize=1*, *disable_online_update=False*, *n_times_replay=1*, *replay_start_size=100*, *normalize_loss_by_steps=True*, *act_deterministically=False*, *average_loss_decay=0.999*, *average_entropy_decay=0.999*, *average_value_decay=0.999*, *explorer=None*, *logger=None*, *batch_states=<function batch_states>*, *backprop_future_values=True*, *train_async=False*)

PCL (Path Consistency Learning).

Not only the batch PCL algorithm proposed in the paper but also its asynchronous variant is implemented.

See https://arxiv.org/abs/1702.08892

Parameters

- **model** (*chainer.Link*) – Model to train. It must be a callable that accepts a batch of observations as input and return two values:

  - action distributions (Distribution)
  - state values (chainer.Variable)
- **optimizer** (*chainer.Optimizer*) – optimizer used to train the model
- **t_max** (*int or None*) – The model is updated after every t_max local steps. If set None, the model is updated after every episode.
- **gamma** (*float*) – Discount factor [0,1]
- **tau** (*float*) – Weight coefficient for the entropy regularizaiton term.
- **phi** (*callable*) – Feature extractor function
- **pi_loss_coef** (*float*) – Weight coefficient for the loss of the policy
- **v_loss_coef** (*float*) – Weight coefficient for the loss of the value function
- **rollout_len** (*int*) – Number of rollout steps
- **batchsize** (*int*) – Number of episodes or sub-trajectories used for an update. The total number of transitions used will be (batchsize x t_max).
- **disable_online_update** (*bool*) – If set true, disable online on-policy update and rely only on experience replay.
- **n_times_replay** (*int*) – Number of times experience replay is repeated per one time of online update.
- **replay_start_size** (*int*) – Experience replay is disabled if the number of transitions in the replay buffer is lower than this value.
- **normalize_loss_by_steps** (*bool*) – If set true, losses are normalized by the number of steps taken to accumulate the losses
- **act_deterministically** (*bool*) – If set true, choose most probable actions in act method.
- **average_loss_decay** (*float*) – Decay rate of average loss. Used only to record statistics.
- **average_entropy_decay** (*float*) – Decay rate of average entropy. Used only to record statistics.
- **average_value_decay** (*float*) – Decay rate of average value. Used only to record statistics.
- **explorer** (*Explorer or None*) – If not None, this explorer is used for selecting actions.
- **logger** (*None or Logger*) – Logger to be used
- **batch_states** (*callable*) – Method which makes a batch of observations. default is *chainerrl.misc.batch_states.batch_states*
- **backprop_future_values** (*bool*) – If set True, value gradients are computed not only wrt V(s_t) but also V(s_{t+d}).
- **train_async** (*bool*) – If set True, use a process-local model to compute gradients and update the globally shared model.

class chainerrl.agents.**PGT**(*model*, *actor_optimizer*, *critic_optimizer*, *replay_buffer*, *gamma*, *explorer*, *beta=0.01*, *act_deterministically=False*, *gpu=-1*, *replay_start_size=50000*, *minibatch_size=32*, *update_frequency=1*, *target_update_frequency=10000*, *phi=<function <lambda>>*, *target_update_method=u'hard'*, *soft_update_tau=0.01*, *n_times_update=1*, *average_q_decay=0.999*, *average_loss_decay=0.99*, *logger=<logging.Logger object>*, *batch_states=<function batch_states>*)

Policy Gradient Theorem with an approximate policy and a Q-function.

This agent is almost the same with DDPG except that it uses the likelihood ratio gradient estimation instead of value gradients.

Parameters

- **model** (*chainer.Chain*) – Chain that contains both a policy and a Q-function
- **actor_optimizer** (*Optimizer*) – Optimizer setup with the policy
- **critic_optimizer** (*Optimizer*) – Optimizer setup with the Q-function
- **replay_buffer** (*ReplayBuffer*) – Replay buffer
- **gamma** (*float*) – Discount factor
- **explorer** (*Explorer*) – Explorer that specifies an exploration strategy.
- **gpu** (*int*) – GPU device id. -1 for CPU.
- **replay_start_size** (*int*) – if the replay buffer's size is less than replay_start_size, skip update
- **minibatch_size** (*int*) – Minibatch size
- **update_frequency** (*int*) – Model update frequency in step
- **target_update_frequency** (*int*) – Target model update frequency in step
- **phi** (*callable*) – Feature extractor applied to observations
- **target_update_method** (*str*) – 'hard' or 'soft'.
- **soft_update_tau** (*float*) – Tau of soft target update.
- **n_times_update** (*int*) – Number of repetition of update
- **average_q_decay** (*float*) – Decay rate of average Q, only used for recording statistics
- **average_loss_decay** (*float*) – Decay rate of average loss, only used for recording statistics
- **batch_accumulator** (*str*) – 'mean' or 'sum'
- **logger** (*Logger*) – Logger used
- **beta** (*float*) – Coefficient for entropy regularization
- **act_deterministically** (*bool*) – Act deterministically by selecting most probable actions in test time
- **batch_states** (*callable*) – method which makes a batch of observations. default is *chainerrl.misc.batch_states.batch_states*

class chainerrl.agents.**ResidualDQN**(**args*, ***kwargs*)

DQN that allows maxQ also backpropagate gradients.

class chainerrl.agents.**SARSA**(*q_function*, *optimizer*, *replay_buffer*, *gamma*, *explorer*, *gpu=None*, *replay_start_size=50000*, *minibatch_size=32*, *update_frequency=1*, *target_update_frequency=10000*, *clip_delta=True*, *phi=<function <lambda>>*, *target_update_method=u'hard'*, *soft_update_tau=0.01*, *n_times_update=1*, *average_q_decay=0.999*, *average_loss_decay=0.99*, *batch_accumulator=u'mean'*, *episodic_update=False*, *episodic_update_len=None*, *logger=<logging.Logger object>*, *batch_states=<function batch_states>*)

SARSA.

Unlike DQN, this agent uses actions that have been actually taken to compute tareget Q values, thus is an on-policy algorithm.

Distributions

Distribution interfaces

class chainerrl.distribution.**Distribution**

Batch of distributions of data.

copy(*x*)

Copy a distribion unchained from the computation graph.

Returns Distribution

entropy

Entropy of distributions.

Returns chainer.Variable

kl

Compute KL divergence D_KL(P|Q).

Parameters **distrib** (Distribution) – Distribution Q.

Returns chainer.Variable

log_prob(*x*)

Compute log p(x).

Returns chainer.Variable

most_probable

Most probable data points.

Returns chainer.Variable

params

Learnable parameters of this distribution.

Returns tuple of chainer.Variable

prob(*x*)

Compute p(x).

Returns chainer.Variable

sample()

Sample from distributions.

Returns chainer.Variable

Distribution implementations

class chainerrl.distribution.**GaussianDistribution**(*mean*, *var*)
Gaussian distribution.

class chainerrl.distribution.**SoftmaxDistribution**(*logits*, *beta=1.0*)
Softmax distribution.

Parameters **logits** (*ndarray or chainer.Variable*) – Logits for softmax distribution.

class chainerrl.distribution.**MellowmaxDistribution**(*values*, *omega=8.0*)
Maximum entropy mellowmax distribution.

See: http://arxiv.org/abs/1612.05628

Parameters **values** (*ndarray or chainer.Variable*) – Values to apply mellowmax.

class chainerrl.distribution.**ContinuousDeterministicDistribution**(*x*)
Continous deterministic distribution.

This distribution is supposed to be used in continuous deterministic policies.

Experiments

chainerrl.experiments.**train_agent_async**(*outdir*, *processes*, *make_env*, *profile=False*, *steps=80000000*, *eval_frequency=1000000*, *eval_n_runs=10*, *gamma=0.99*, *max_episode_len=None*, *step_offset=0*, *successful_score=None*, *eval_explorer=None*, *agent=None*, *make_agent=None*)
Train agent asynchronously.

One of agent and make_agent must be specified.

Parameters

- **agent** (Agent) – Agent to train
- **make_agent** (*callable*) – (process_idx) -> Agent
- **processes** (*int*) – Number of processes.
- **make_env** (*callable*) – (process_idx, test) -> env
- **model_opt** (*callable*) – () -> (models, optimizers)
- **profile** (*bool*) – Profile if set True
- **steps** (*int*) – Number of global time steps for training

chainerrl.experiments.**train_agent_with_evaluation**(*agent*, *env*, *steps*, *eval_n_runs*, *eval_frequency*, *outdir*, *max_episode_len=None*, *step_offset=0*, *eval_explorer=None*, *eval_max_episode_len=None*, *eval_env=None*, *successful_score=None*, *render=False*)
Run a DQN-like agent.

Parameters

- **agent** – Agent.
- **env** – Environment.
- **steps** (*int*) – Number of total time steps for training.
- **eval_n_runs** (*int*) – Number of runs for each time of evaluation.
- **eval_frequency** (*int*) – Interval of evaluation.
- **outdir** (*str*) – Path to the directory to output things.
- **max_episode_len** (*int*) – Maximum episode length.
- **step_offset** (*int*) – Time step from which training starts.
- **eval_explorer** – Explorer used for evaluation.
- **eval_env** – Environment used for evaluation.
- **successful_score** (*float*) – Finish training if the mean score is greater or equal to this value if not None

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Index