

---

# ChainerRL Documentation

*Release 0.2.0*

Preferred Networks, Inc.

Jun 12, 2017



---

## Contents

---

<b>1</b>	<b>Installation</b>	<b>3</b>
1.1	How to install ChainerRL . . . . .	3
<b>2</b>	<b>API Reference</b>	<b>5</b>
2.1	Action values . . . . .	5
2.2	Agents . . . . .	6
2.3	Distributions . . . . .	15
2.4	Experiments . . . . .	17
<b>3</b>	<b>Indices and tables</b>	<b>19</b>



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.



### 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>=2.0.0
fastcache; python_version<'3.2'
functools; 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
```





## Action value interfaces

**max**  
Evaluate  $\max Q(s,a)$ .

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

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

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

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

**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*) – minimum 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 regularization 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 regularization 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, mini-
                             batch_size=32, update_interval=1, target_update_interval=10000,
                             phi=<function <lambda>>, target_update_method='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\_interval** (*int*) – Model update interval in step
- **target\_update\_interval** (*int*) – Target model update interval 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_interval=1, target_update_interval=10000,
                                clip_delta=True, phi=<function <lambda>>, target_update_method='hard',
                                soft_update_tau=0.01, n_times_update=1, average_q_decay=0.999,
                                average_loss_decay=0.99, batch_accumulator='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_interval=1,
                           target_update_interval=10000, clip_delta=True, phi=<function
                           <lambda>>, target_update_method='hard', soft_update_tau=0.01,
                           n_times_update=1, average_q_decay=0.999, average_loss_decay=0.99,
                           batch_accumulator='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\_interval** (*int*) – Model update interval in step
- **target\_update\_interval** (*int*) – Target model update interval 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_interval=1,
                           target_update_interval=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\_interval** (*int*) – Model update interval in step
- **target\_update\_interval** (*int*) – Target model update interval 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 regularization 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, re-
                           play_start_size=50000, minibatch_size=32, update_interval=1,
                           target_update_interval=10000, phi=<function <lambda>>, tar-
                           get_update_method=u'hard', soft_update_tau=0.01, n_times_update=1,
                           average_q_decay=0.999, average_loss_decay=0.99, log-
                           ger=<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\_interval** (*int*) – Model update interval in step
- **target\_update\_interval** (*int*) – Target model update interval 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.REINFORCE(model, optimizer, beta=0, phi=<function <lambda>>,
                                batchsize=1, act_deterministically=False, aver-
                                age_entropy_decay=0.999, backward_separately=False,
                                batch_states=<function batch_states>, logger=None)
```

William's episodic REINFORCE.

### Parameters

- **model** (*Policy*) – Model to train. It must be a callable that accepts observations as input and return action distributions (Distribution).
- **optimizer** (*chainer.Optimizer*) – optimizer used to train the model
- **beta** (*float*) – Weight coefficient for the entropy regularization term.
- **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.
- **batchsize** (*int*) – Number of episodes used for each update
- **backward\_separately** (*bool*) – If set true, call backward separately for each episode and accumulate only gradients.
- **average\_entropy\_decay** (*float*) – Decay rate of average entropy. Used only to record statistics.
- **batch\_states** (*callable*) – Method which makes a batch of observations. default is *chainerrl.misc.batch\_states*
- **logger** (*logging.Logger*) – Logger to be used.

**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\_interval=1*, *target\_update\_interval=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 target Q values, thus is an on-policy algorithm.

## Distributions

### Distribution interfaces

**class** `chainerrl.distribution.Distribution`

Batch of distributions of data.

**copy** (*x*)

Copy a distribution 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, min\_prob=0.0*)

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*)

Continuous deterministic distribution.

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

## Experiments

### Training and evaluation

```
chainerrl.experiments.train_agent_async(outdir, processes, make_env, profile=False,
                                       steps=80000000,      eval_interval=1000000,
                                       eval_n_runs=10,       max_episode_len=None,
                                       step_offset=0,        successful_score=None,
                                       eval_explorer=None,    agent=None,
                                       make_agent=None,       global_step_hooks=[],
                                       logger=None)
```

Train agent asynchronously using multiprocessing.

Either *agent* or *make\_agent* must be specified.

#### Parameters

- **outdir** (*str*) – Path to the directory to output things.
- **processes** (*int*) – Number of processes.
- **make\_env** (*callable*) – (process\_idx, test) -> Environment.
- **profile** (*bool*) – Profile if set True.
- **steps** (*int*) – Number of global time steps for training.
- **eval\_interval** (*int*) – Interval of evaluation. If set to None, the agent will not be evaluated at all.
- **eval\_n\_runs** (*int*) – Number of runs for each time of evaluation.
- **max\_episode\_len** (*int*) – Maximum episode length.
- **step\_offset** (*int*) – Time step from which training starts.
- **successful\_score** (*float*) – Finish training if the mean score is greater or equal to this value if not None
- **eval\_explorer** – Explorer used for evaluation.
- **agent** (*Agent*) – Agent to train.
- **make\_agent** (*callable*) – (process\_idx) -> Agent
- **global\_step\_hooks** (*list*) – List of callable objects that accepts (env, agent, step) as arguments. They are called every global step. See chainerrl.experiments.hooks.
- **logger** (*logging.Logger*) – Logger used in this function.

**Returns** Trained agent.

```
chainerrl.experiments.train_agent_with_evaluation(agent, env, steps, eval_n_runs,
                                                  eval_interval,      outdir,
                                                  max_episode_len=None,
                                                  step_offset=0, eval_explorer=None,
                                                  eval_max_episode_len=None,
                                                  eval_env=None,      success-
                                                  ful_score=None,    step_hooks=[],
                                                  logger=None)
```

Train an agent while regularly evaluating it.

#### Parameters

- **agent** – Agent to train.
- **env** – Environment train the agent against.
- **steps** (*int*) – Number of total time steps for training.
- **eval\_n\_runs** (*int*) – Number of runs for each time of evaluation.
- **eval\_interval** (*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
- **step\_hooks** (*list*) – List of callable objects that accepts (env, agent, step) as arguments. They are called every step. See `chainerrl.experiments.hooks`.
- **logger** (*logging.Logger*) – Logger used in this function.

## Training hooks

**class** `chainerrl.experiments.StepHook`

Hook function that will be called in training.

This class is for clarifying the interface required for Hook functions. You don't need to inherit this class to define your own hooks. Any callable that accepts (env, agent, step) as arguments can be used as a hook.

**class** `chainerrl.experiments.LinearInterpolationHook` (*total\_steps*, *start\_value*, *stop\_value*, *setter*)

Hook that will set a linearly interpolated value.

You can use this hook to decay the learning rate by using a setter function as follows:

```
def lr_setter(env, agent, value):
    agent.optimizer.lr = value

hook = LinearInterpolationHook(10 ** 6, 1e-3, 0, lr_setter)
```

### Parameters

- **total\_steps** (*int*) – Number of total steps.
- **start\_value** (*float*) – Start value.
- **stop\_value** (*float*) – Stop value.
- **setter** (*callable*) – (env, agent, value) -> None

## CHAPTER 3

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`





## A

A3C (class in chainerrl.agents), 7  
ACER (class in chainerrl.agents), 7  
act() (chainerrl.agent.Agent method), 6  
act\_and\_train() (chainerrl.agent.Agent method), 6  
ActionValue (class in chainerrl.action\_value), 5  
Agent (class in chainerrl.agent), 6  
AL (class in chainerrl.agents), 8

## C

ContinuousDeterministicDistribution (class in chainerrl.distribution), 16  
copy() (chainerrl.distribution.Distribution method), 15

## D

DDPG (class in chainerrl.agents), 8  
DiscreteActionValue (class in chainerrl.action\_value), 5  
Distribution (class in chainerrl.distribution), 15  
DoubleDQN (class in chainerrl.agents), 9  
DoublePAL (class in chainerrl.agents), 10  
DPP (class in chainerrl.agents), 10  
DQN (class in chainerrl.agents), 10, 11

## E

entropy (chainerrl.distribution.Distribution attribute), 15  
evaluate\_actions() (chainerrl.action\_value.ActionValue method), 5

## G

GaussianDistribution (class in chainerrl.distribution), 16  
get\_statistics() (chainerrl.agent.Agent method), 6  
greedy\_actions (chainerrl.action\_value.ActionValue attribute), 5

## K

kl (chainerrl.distribution.Distribution attribute), 15

## L

LinearInterpolationHook (class in chainerrl.experiments), 18

load() (chainerrl.agent.Agent method), 6  
log\_prob() (chainerrl.distribution.Distribution method), 16

## M

max (chainerrl.action\_value.ActionValue attribute), 5  
MellowmaxDistribution (class in chainerrl.distribution), 16  
most\_probable (chainerrl.distribution.Distribution attribute), 16

## N

NSQ (class in chainerrl.agents), 12

## P

PAL (class in chainerrl.agents), 12  
params (chainerrl.distribution.Distribution attribute), 16  
PCL (class in chainerrl.agents), 12  
PGT (class in chainerrl.agents), 13  
prob() (chainerrl.distribution.Distribution method), 16

## Q

QuadraticActionValue (class in chainerrl.action\_value), 5

## R

REINFORCE (class in chainerrl.agents), 14  
ResidualDQN (class in chainerrl.agents), 15

## S

sample() (chainerrl.distribution.Distribution method), 16  
SARSA (class in chainerrl.agents), 15  
save() (chainerrl.agent.Agent method), 6  
SingleActionValue (class in chainerrl.action\_value), 6  
SoftmaxDistribution (class in chainerrl.distribution), 16  
StepHook (class in chainerrl.experiments), 18  
stop\_episode() (chainerrl.agent.Agent method), 6  
stop\_episode\_and\_train() (chainerrl.agent.Agent method), 6

## T

`train_agent_async()` (in module `chainerrl.experiments`),

[17](#)

`train_agent_with_evaluation()` (in module `chainerrl.experiments`), [17](#)