ChainerRL Documentation

Release 0.7.0

Preferred Networks, Inc.

Jul 01, 2019

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

1.1 How to install ChainerRL

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

Listing 1: requirements.txt

```
cached-property
chainer>=4.0.0
fastcache; python_version<'3.2'
funcsigs; python_version<'3.5'
future
gym>=0.9.7
numpy>=1.10.4
pillow
scipy
statistics; python_version<'3.4'
```

ChainerRL can be installed via PyPI,

```
pip install chainerrl
```

or through the source code:

```
git clone https://github.com/chainer/chainerrl.git
cd chainerrl
python setup.py install
```

CHAPTER 2

API Reference

2.1 Action values

2.1.1 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).

params

Learnable parameters of this action value.

Returns tuple of chainer.Variable

2.1.2 Action value implementations

class chainerrl.action_value.**DiscreteActionValue**(*q_values*, *q_values_formatter=<function DiscreteActionValue.<lambda>>*)

Q-function 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*)

Q-function 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.

2.2 Agents

2.2.1 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

2.2.2 Agent implementations

class chainerrl.agents.**A2C**(*model*, *optimizer*, *gamma*, *num_processes*, *gpu=None*, *update_steps=5*, *phi=<function A2C.<lambda>>*, *pi_loss_coef=1.0*, *v_loss_coef=0.5*, *entropy_coeff=0.01*, *use_gae=False*, *tau=0.95*, *act_deterministically=False*, *average_actor_loss_decay=0.999*, *average_entropy_decay=0.999*, *average_value_decay=0.999*, *batch_states=<function batch_states>*)
A2C: Advantage Actor-Critic.

A2C is a synchronous, deterministic variant of Asynchronous Advantage Actor Critic (A3C).

See https://arxiv.org/abs/1708.05144

Parameters

- **model** (*A2CModel*) – Model to train
- **optimizer** (*chainer.Optimizer*) – optimizer used to train the model
- **gamma** (*float*) – Discount factor [0,1]
- **num_processes** (*int*) – The number of processes
- **gpu** (*int*) – GPU device id if not None nor negative.
- **update_steps** (*int*) – The number of update steps
- **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
- **entropy_coeff** (*float*) – Weight coefficient for the loss of the entropy
- **use_gae** (*bool*) – use generalized advantage estimation(GAE)
- **tau** (*float*) – gae parameter
- **average_actor_loss_decay** (*float*) – Decay rate of average actor 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.
- **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.**A3C**(*model*, *optimizer*, *t_max*, *gamma*, *beta=0.01*, *process_idx=0*, *phi=<function A3C.<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 ACER.<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.**CategoricalDoubleDQN**(*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 DQN.<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=<Logger chainerrl.agents.dqn (WARNING)>*, *batch_states=<function batch_states>*)

Categorical Double DQN.

class chainerrl.agents.**CategoricalDQN**(*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 DQN.<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=<Logger chainerrl.agents.dqn (WARNING)>*, *batch_states=<function batch_states>*)

Categorical DQN.

See https://arxiv.org/abs/1707.06887.

Arguments are the same as those of DQN except q_function must return DistributionalDiscreteActionValue and clip_delta is ignored.

class chainerrl.agents.**DDPG**(*model*, *actor_optimizer*, *critic_optimizer*, *replay_buffer*, *gamma*, *explorer*, *gpu=None*, *replay_start_size=50000*, *minibatch_size=32*, *update_interval=1*, *target_update_interval=10000*, *phi=<function DDPG.<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=<Logger chainerrl.agents.ddpg (WARNING)>*, *batch_states=<function batch_states>*, *burnin_action_func=None*)

Deep Deterministic Policy Gradients.

This can be used as SVG(0) by specifying a Gaussian 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*
- **burnin_action_func** (*callable or None*) – If not None, this callable object is used to select actions before the model is updated one or more times during training.

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 DQN.<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=<Logger chainerrl.agents.dqn (WARNING)>*, *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 DQN.<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=<Logger chainerrl.agents.dqn (WARNING)>*, *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.**IQN**(**args*, ***kwargs*)

Implicit Quantile Networks.

See https://arxiv.org/abs/1806.06923.

Parameters

- **quantile_thresholds_N** (*int*) – Number of quantile thresholds used in quantile regression.
- **quantile_thresholds_N_prime** (*int*) – Number of quantile thresholds used to sample from the return distribution at the next state.
- **quantile_thresholds_K** (*int*) – Number of quantile thresholds used to compute greedy actions.

For other arguments, see chainerrl.agents.DQN.

class chainerrl.agents.**NSQ**(*q_function*, *optimizer*, *t_max*, *gamma*, *i_target*, *explorer*, *phi=<function NSQ.<lambda>>*, *average_q_decay=0.999*, *logger=<Logger chainerrl.agents.nsq (WARNING)>*, *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 PCL.<lambda>>*, *pi_loss_coef=1.0*, *v_loss_coef=0.5*, *rollout_len=10*, *batch-size=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_interval=1*, *target_update_interval=10000*, *phi=<function PGT.<lambda>>*, *target_update_method='hard'*, *soft_update_tau=0.01*, *n_times_update=1*, *average_q_decay=0.999*, *average_loss_decay=0.99*, *logger=<Logger chainerrl.agents.pgt (WARNING)>*, *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.**PPO**(*model*, *optimizer*, *obs_normalizer=None*, *gpu=None*, *gamma=0.99*, *lambd=0.95*, *phi=<function PPO.<lambda>>*, *value_func_coef=1.0*, *entropy_coef=0.01*, *update_interval=2048*, *minibatch_size=64*, *epochs=10*, *clip_eps=0.2*, *clip_eps_vf=None*, *standardize_advantages=True*, *batch_states=<function batch_states>*, *recurrent=False*, *max_recurrent_sequence_len=None*, *act_deterministically=False*, *value_stats_window=1000*, *entropy_stats_window=1000*, *value_loss_stats_window=100*, *policy_loss_stats_window=100*)

Proximal Policy Optimization

See https://arxiv.org/abs/1707.06347

Parameters

- **model** (*A3CModel*) – Model to train. Recurrent models are not supported. state s |-> (pi(s, _), v(s))
- **optimizer** (*chainer.Optimizer*) – Optimizer used to train the model
- **gpu** (*int*) – GPU device id if not None nor negative
- **gamma** (*float*) – Discount factor [0, 1]
- **lambd** (*float*) – Lambda-return factor [0, 1]
- **phi** (*callable*) – Feature extractor function
- **value_func_coef** (*float*) – Weight coefficient for loss of value function (0, inf)
- **entropy_coef** (*float*) – Weight coefficient for entropy bonus [0, inf)
- **update_interval** (*int*) – Model update interval in step
- **minibatch_size** (*int*) – Minibatch size
- **epochs** (*int*) – Training epochs in an update
- **clip_eps** (*float*) – Epsilon for pessimistic clipping of likelihood ratio to update policy
- **clip_eps_vf** (*float*) – Epsilon for pessimistic clipping of value to update value function. If it is `None`, value function is not clipped on updates.
- **standardize_advantages** (*bool*) – Use standardized advantages on updates
- **recurrent** (*bool*) – If set to True, *model* is assumed to implement *chainerrl.links.StatelessRecurrent* and update in a recurrent manner.
- **max_recurrent_sequence_len** (*int*) – Maximum length of consecutive sequences of transitions in a minibatch for updatig the model. This value is used only when *recurrent* is True. A smaller value will encourage a minibatch to contain more and shorter sequences.
- **act_deterministically** (*bool*) – If set to True, choose most probable actions in the act method instead of sampling from distributions.
- **value_stats_window** (*int*) – Window size used to compute statistics of value predictions.
- **entropy_stats_window** (*int*) – Window size used to compute statistics of entropy of action distributions.
- **value_loss_stats_window** (*int*) – Window size used to compute statistics of loss values regarding the value function.
- **policy_loss_stats_window** (*int*) – Window size used to compute statistics of loss values regarding the policy.

Statistics:

average_value: Average of value predictions on non-terminal states. It's updated on (batch_)act_and_train.

average_entropy: Average of entropy of action distributions on non-terminal states. It's updated on (batch_)act_and_train.

average_value_loss: Average of losses regarding the value function. It's updated after the model is updated.

average_policy_loss: Average of losses regarding the policy. It's updated after the model is updated.

n_updates: Number of model updates so far. explained_variance: Explained variance computed from the last batch.

class chainerrl.agents.**REINFORCE**(*model*, *optimizer*, *beta=0*, *phi=<function REINFORCE.<lambda>>*, *batchsize=1*, *act_deterministically=False*, *average_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 regularizaiton 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 DQN.<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=<Logger chainerrl.agents.dqn (WARNING)>*, *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.

class chainerrl.agents.**TD3**(*policy*, *q_func1*, *q_func2*, *policy_optimizer*, *q_func1_optimizer*, *q_func2_optimizer*, *replay_buffer*, *gamma*, *explorer*, *gpu=None*, *replay_start_size=10000*, *minibatch_size=100*, *update_interval=1*, *phi=<function TD3.<lambda>>*, *soft_update_tau=0.005*, *n_times_update=1*, *logger=<Logger chainerrl.agents.td3 (WARNING)>*, *batch_states=<function batch_states>*, *burnin_action_func=None*, *policy_update_delay=2*, *target_policy_smoothing_func=<function default_target_policy_smoothing_func>*)

Twin Delayed Deep Deterministic Policy Gradients (TD3).

See http://arxiv.org/abs/1802.09477

Parameters

- **policy** (*Policy*) – Policy.
- **q_func1** (*Link*) – First Q-function that takes state-action pairs as input and outputs predicted Q-values.
- **q_func2** (*Link*) – Second Q-function that takes state-action pairs as input and outputs predicted Q-values.
- **policy_optimizer** (*Optimizer*) – Optimizer setup with the policy
- **q_func1_optimizer** (*Optimizer*) – Optimizer setup with the first Q-function.
- **q_func2_optimizer** (*Optimizer*) – Optimizer setup with the second 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
- **phi** (*callable*) – Feature extractor applied to observations
- **soft_update_tau** (*float*) – Tau of soft target update.
- **logger** (*Logger*) – Logger used

- **batch_states** (*callable*) – method which makes a batch of observations. default is *chainerrl.misc.batch_states.batch_states*
- **burnin_action_func** (*callable or None*) – If not None, this callable object is used to select actions before the model is updated one or more times during training.
- **policy_update_delay** (*int*) – Delay of policy updates. Policy is updated once in *policy_update_delay* times of Q-function updates.
- **target_policy_smoothing_func** (*callable*) – Callable that takes a batch of actions as input and outputs a noisy version of it. It is used for target policy smoothing when computing target Q-values.

class chainerrl.agents.**TRPO**(*policy*, *vf*, *vf_optimizer*, *obs_normalizer=None*, *gamma=0.99*, *lambd=0.95*, *phi=<function TRPO.<lambda>>*, *entropy_coef=0.01*, *update_interval=2048*, *max_kl=0.01*, *vf_epochs=3*, *vf_batch_size=64*, *standardize_advantages=True*, *line_search_max_backtrack=10*, *conjugate_gradient_max_iter=10*, *conjugate_gradient_damping=0.01*, *act_deterministically=False*, *value_stats_window=1000*, *entropy_stats_window=1000*, *kl_stats_window=100*, *policy_step_size_stats_window=100*, *logger=<Logger chainerrl.agents.trpo (WARNING)>*)

Trust Region Policy Optimization.

A given stochastic policy is optimized by the TRPO algorithm. A given value function is also trained to predict by the TD(lambda) algorithm and used for Generalized Advantage Estimation (GAE).

Since the policy is optimized via the conjugate gradient method and line search while the value function is optimized via SGD, these two models should be separate.

Since TRPO requires second-order derivatives to compute Hessian-vector products, Chainer v3.0.0 or newer is required. In addition, your policy must contain only functions that support second-order derivatives.

See https://arxiv.org/abs/1502.05477 for TRPO. See https://arxiv.org/abs/1506.02438 for GAE.

Parameters

- **policy** (*Policy*) – Stochastic policy. Its forward computation must contain only functions that support second-order derivatives. Recurrent models are not supported.
- **vf** (*ValueFunction*) – Value function. Recurrent models are not supported.
- **vf_optimizer** (*chainer.Optimizer*) – Optimizer for the value function.
- **obs_normalizer** (*chainerrl.links.EmpiricalNormalization or None*) – If set to chainerrl.links.EmpiricalNormalization, it is used to normalize observations based on the empirical mean and standard deviation of observations. These statistics are updated after computing advantages and target values and before updating the policy and the value function.
- **gamma** (*float*) – Discount factor [0, 1]
- **lambd** (*float*) – Lambda-return factor [0, 1]
- **phi** (*callable*) – Feature extractor function
- **entropy_coef** (*float*) – Weight coefficient for entropoy bonus [0, inf)
- **update_interval** (*int*) – Interval steps of TRPO iterations. Every after this amount of steps, this agent updates the policy and the value function using data from these steps.
- **vf_epochs** (*int*) – Number of epochs for which the value function is trained on each TRPO iteration.

- **vf_batch_size** (*int*) – Batch size of SGD for the value function.
- **standardize_advantages** (*bool*) – Use standardized advantages on updates
- **line_search_max_backtrack** (*int*) – Maximum number of backtracking in line search to tune step sizes of policy updates.
- **conjugate_gradient_max_iter** (*int*) – Maximum number of iterations in the conjugate gradient method.
- **conjugate_gradient_damping** (*float*) – Damping factor used in the conjugate gradient method.
- **act_deterministically** (*bool*) – If set to True, choose most probable actions in the act method instead of sampling from distributions.
- **value_stats_window** (*int*) – Window size used to compute statistics of value predictions.
- **entropy_stats_window** (*int*) – Window size used to compute statistics of entropy of action distributions.
- **kl_stats_window** (*int*) – Window size used to compute statistics of KL divergence between old and new policies.
- **policy_step_size_stats_window** (*int*) – Window size used to compute statistics of step sizes of policy updates.

Statistics:

average_value: Average of value predictions on non-terminal states. It's updated before the value function is updated.

average_entropy: Average of entropy of action distributions on non-terminal states. It's updated on act_and_train.

average_kl: Average of KL divergence between old and new policies. It's updated after the policy is updated.

average_policy_step_size: Average of step sizes of policy updates It's updated after the policy is updated.

2.3 Distributions

2.3.1 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

2.3.2 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.
- **beta** (*float*) – inverse of the temperature parameter of softmax distribution
- **min_prob** (*float*) – minimum probability across all labels

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.

2.4 Experiments

2.4.1 Training and evaluation

chainerrl.experiments.**train_agent_async**(*outdir*, *processes*, *make_env*, *profile=False*, *steps=80000000*, *eval_interval=1000000*, *eval_n_steps=None*, *eval_n_episodes=10*, *max_episode_len=None*, *step_offset=0*, *successful_score=None*, *agent=None*, *make_agent=None*, *global_step_hooks=[]*, *save_best_so_far_agent=True*, *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_steps** (*int*) – Number of eval timesteps at each eval phase
- **eval_n_episodes** (*int*) – Number of eval episodes at each eval phase
- **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
- **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.
- **save_best_so_far_agent** (*bool*) – If set to True, after each evaluation, if the score (= mean return of evaluation episodes) exceeds the best-so-far score, the current agent is saved.
- **logger** (*logging.Logger*) – Logger used in this function.

Returns Trained agent.

chainerrl.experiments.**train_agent_with_evaluation**(*agent*, *env*, *steps*, *eval_n_steps*, *eval_n_episodes*, *eval_interval*, *outdir*, *train_max_episode_len=None*, *step_offset=0*, *eval_max_episode_len=None*, *eval_env=None*, *successful_score=None*, *step_hooks=[]*, *save_best_so_far_agent=True*, *logger=None*)

Train an agent while periodically evaluating it.

Parameters

- **agent** – A chainerrl.agent.Agent
- **env** – Environment train the agent against.
- **steps** (*int*) – Total number of timesteps for training.
- **eval_n_steps** (*int*) – Number of timesteps at each evaluation phase.
- **eval_n_episodes** (*int*) – Number of episodes at each evaluation phase.
- **eval_interval** (*int*) – Interval of evaluation.
- **outdir** (*str*) – Path to the directory to output data.
- **train_max_episode_len** (*int*) – Maximum episode length during training.
- **step_offset** (*int*) – Time step from which training starts.
- **eval_max_episode_len** (*int or None*) – Maximum episode length of evaluation runs. If None, train_max_episode_len is used instead.
- **eval_env** – Environment used for evaluation.
- **successful_score** (*float*) – Finish training if the mean score is greater than 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.
- **save_best_so_far_agent** (*bool*) – If set to True, after each evaluation phase, if the score (= mean return of evaluation episodes) exceeds the best-so-far score, the current agent is saved.
- **logger** (*logging.Logger*) – Logger used in this function.

2.4.2 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

2.5 Links

2.5.1 Link interfaces

class chainerrl.links.**StatelessRecurrent**

Stateless recurrent link interface.

This class defines the interface of a recurrent link ChainerRL can handle.

In most casese, you can just use ChainerRL's existing containers like *chainerrl.links.StatelessRecurrentChainList*, *chainerrl.links.StatelessRecurrentSequential*, and *chainerrl.links.StatelessRecurrentBranched* to define a recurrent link. You can use Chainer's recurrent links such as L.NStepLSTM inside the containers.

To write your own recurrent link, you need to implement the interface.

concatenate_recurrent_states(*split_recurrent_states*)

Concatenate recurrent states into a batch.

This method can be used to make a batched recurrent state from separate recurrent states obtained via the *get_recurrent_state_at* method.

Parameters **split_recurrent_states** (*object*) – Recurrent states to concatenate.

Returns Batched recurrent_state.

Return type object

get_recurrent_state_at(*recurrent_state*, *indices*)

Get a recurrent state at given indices.

This method can be used to save a recurrent state so that you can reuse it when you replay past sequences.

Parameters **indices** (*int or array-like of ints*) – Which recurrent state to get.

Returns Recurrent state of given indices.

Return type object

mask_recurrent_state_at(*recurrent_state*, *indices*)

Return a recurrent state masked at given indices.

This method can be used to initialize a recurrent state only for a certain sequence, not all the sequences.

Parameters

- **recurrent_state** (*object*) – Batched recurrent state.

- **indices** (*int or array-like of ints*) – Which recurrent state to mask.

Returns New batched recurrent state.

Return type object

n_step_forward(*x*, *recurrent_state*)
Multi-step batch forward computation.

This method sequentially applies layers as chainer.Sequential does.

Parameters

- **x** (*list*) – Input sequences. Each sequence should be a variable whose first axis corresponds to time or a tuple of such variables.
- **recurrent_state** (*object*) – Batched recurrent state. If set to None, it is initialized.
- **output_mode** (*str*) – If set to 'concat', the output value is concatenated into a single large batch, which can be suitable for loss computation. If set to 'split', the output value is a list of output sequences.

Returns

Output sequences. See the description of the *output_mode* argument.

object: New batched recurrent state.

Return type object

2.5.2 Link implementations

class chainerrl.links.**Branched**(**links*)
Link that calls forward functions of child links in parallel.

When either the *__call__* method of this link are called, all the argeuments are forwarded to each child link's *__call__* method.

The returned values from the child links are returned as a tuple.

Parameters ***links** – Child links. Each link should be callable.

class chainerrl.links.**EmpiricalNormalization**(*shape*, *batch_axis=0*, *eps=0.01*, *dtype=<class 'numpy.float32'>*, *until=None*, *clip_threshold=None*)
Normalize mean and variance of values based on emprical values.

Parameters

- **shape** (*int or tuple of int*) – Shape of input values except batch axis.
- **batch_axis** (*int*) – Batch axis.
- **eps** (*float*) – Small value for stability.
- **dtype** (*dtype*) – Dtype of input values.
- **until** (*int or None*) – If this arg is specified, the link learns input values until the sum of batch sizes exceeds it.

class chainerrl.links.**FactorizedNoisyLinear**(*mu_link*, *sigma_scale=0.4*)
Linear layer in Factorized Noisy Network

Parameters

- **mu_link** (*L.Linear*) – Linear link that computes mean of output.

- **sigma_scale** (*float*) – The hyperparameter sigma_0 in the original paper. Scaling factor of the initial weights of noise-scaling parameters.

class chainerrl.links.**MLP**(*in_size*, *out_size*, *hidden_sizes*, *nonlinearity=<function relu>*, *last_wscale=1*)
Multi-Layer Perceptron

class chainerrl.links.**MLPBN**(*in_size*, *out_size*, *hidden_sizes*, *normalize_input=True*, *normalize_output=False*, *nonlinearity=<function relu>*, *last_wscale=1*)
Multi-Layer Perceptron with Batch Normalization.

Parameters

- **in_size** (*int*) – Input size.
- **out_size** (*int*) – Output size.
- **hidden_sizes** (*list of ints*) – Sizes of hidden channels.
- **normalize_input** (*bool*) – If set to True, Batch Normalization is applied to inputs.
- **normalize_output** (*bool*) – If set to True, Batch Normalization is applied to outputs.
- **nonlinearity** (*callable*) – Nonlinearity between layers. It must accept a Variable as an argument and return a Variable with the same shape. Nonlinearities with learnable parameters such as PReLU are not supported.
- **last_wscale** (*float*) – Scale of weight initialization of the last layer.

class chainerrl.links.**NIPSDQNHead**(*n_input_channels=4*, *n_output_channels=256*, *activation=<function relu>*, *bias=0.1*)
DQN's head (NIPS workshop version)

class chainerrl.links.**NatureDQNHead**(*n_input_channels=4*, *n_output_channels=512*, *activation=<function relu>*, *bias=0.1*)
DQN's head (Nature version)

class chainerrl.links.**Sequence**(**layers*)
Sequential callable Link that consists of other Links.

class chainerrl.links.**StatelessRecurrentBranched**(**links*)
Stateless recurrent parallel link.

This is a recurrent analog to chainerrl.links.Branched. It bundles multiple links that implements *StatelessRecurrent*.

Parameters ***links** – Child links. Each link should be recurrent and callable.

class chainerrl.links.**StatelessRecurrentChainList**(**links*)
ChainList that auutomatically handles recurrent states.

This link extends chainer.ChainList by adding the *recurrent_children* property that returns all the recurrent child links and implementing recurrent state manimulation methods required for the StatelessRecurrent interface.

A recurrent state for this link is defined as a tuple of recurrent states of child recurrent links.

class chainerrl.links.**StatelessRecurrentSequential**(**layers*)
Sequential model that can contain stateless recurrent links.

This link a stateless recurrent analog to chainer.Sequential. It supports the stateless recurrent interface by automatically detecting recurrent links and handles recurrent states properly.

For non-recurrent layers (non-link callables or non-recurrent callable links), this link automatically concatenates the input to the layers for efficient computation.

Parameters ***layers** – Callable objects.

2.5.3 Link utility functions

chainerrl.links.**to_factorized_noisy**(*link*, **args*, ***kwargs*)
Add noisiness to components of given link

Currently this function supports L.Linear (with and without bias)

2.6 Using recurrent models

2.6.1 Recurrent model interface

class chainerrl.recurrent.**Recurrent**
Interface of recurrent and stateful models.

This is an interface of recurrent and stateful models. ChainerRL supports recurrent neural network models as stateful models that implement this interface.

To implement this interface, you need to implement three abstract methods of it: get_state, set_state and reset_state.

get_state()
Get the current state of this model.

Returns Any object that represents a state of this model.

reset_state()
Reset the state of this model to the initial state.

For typical RL models, this method is expected to be called before every episode.

set_state(*state*)
Overwrite the state of this model with a given state.

Parameters **state** (*object*) – Any object that represents a state of this model.

update_state(**args*, ***kwargs*)
Update this model's state as if self.__call__ is called.

Unlike __call__, stateless objects may do nothing.

2.6.2 Utilities

chainerrl.recurrent.**state_kept**(*link*)
Keeps the previous state of a given link.

This is a context manager that saves saves the current state of the link before entering the context, and then restores the saved state after escaping the context.

This will just ignore non-Recurrent links.

```
# Suppose the link is in a state A
assert link.get_state() is A

with state_kept(link):
    # The link is still in a state A
    assert link.get_state() is A

    # After evaluating the link, it may be in a different state
```

(continues on next page)

(continued from previous page)

```
    y1 = link(x1)
    assert link.get_state() is not A

# After escaping from the context, the link is in a state A again
# because of the context manager
assert link.get_state() is A
```

chainerrl.recurrent.**state_reset**(*link*)

Reset the state while keeping the previous state of a given link.

This is a context manager that saves saves the current state of the link and reset it to the initial state before entering the context, and then restores the saved state after escaping the context.

This will just ignore non-Recurrent links.

```
# Suppose the link is in a non-initial state A
assert link.get_state() is A

with state_reset(link):
    # The link's state has been reset to the initial state
    assert link.get_state() is InitialState

    # After evaluating the link, it may be in a different state
    y1 = link(x1)
    assert link.get_state() is not InitialState

# After escaping from the context, the link is in a state A again
# because of the context manager
assert link.get_state() is A
```

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Index

N

P

Q

R

S

T

U