vinci Documentation

Release 1.0

Pierre Manceron

Nov 21, 2017

Contents

Vinci is a generic Deep Reinforcement Learning framework.

Contents:

CHAPTER 1

agent

CHAPTER 2

ddpg

class rl.agents.ddpg.**DDPGAgent**(*actor*, *critic*, *memory*, *gamma=0.99*, *batch_size=32*, *train_interval=1*, *memory_interval=1*, *critic_gradient_clip=100*, *random_process=None*, *custom_model_objects=None*, *warmup_actor_steps=200*, *warmup_critic_steps=200*, *invert_gradients=False*, *gradient_inverter_min=-1.0*, *gradient_inverter_max=1.0*, *actor_reset_threshold=0.3*, *reset_controlers=False*, *actor_learning_rate=0.001*, *critic_learning_rate=0.0001*, *target_critic_update=0.01*, *target_actor_update=0.01*, *critic_regularization=0.01*, ***kwargs*)

Deep Deterministic Policy Gradient Agent as defined in https://arxiv.org/abs/1509.02971.

Parameters

- **actor** (*keras.model*) – The actor network
- **critic** (*keras.model*) – The critic network
- **env** (*gym.env*) – The gym environment
- **memory** (*rl.memory.Memory*) – The memory object
- **gamma** (*float*) – Discount factor
- **batch_size** (*int*) – Size of the minibatches
- **train_interval** (*int*) – Train only at multiples of this number
- **memory_interval** (*int*) – Add experiences to memory only at multiples of this number
- **critic_gradient_clip** – Delta to which the rewards are clipped (via Huber loss, see https://github.com/devsisters/DQN-tensorflow/issues/16)
- **random_process** – The noise used to perform exploration
- **custom_model_objects** –
- **target_critic_update** (*float*) – Target critic update factor
- **target_actor_update** (*float*) – Target actor update factor

- **invert_gradients** (*bool*) – Use gradient inverting as defined in https://arxiv.org/abs/1511.04143

backward()
Backward method of the DDPG agent

backward_offline(*train_actor=True*, *train_critic=True*)
Offline Backward method of the DDPG agent

Parameters

- **offline** (*bool*) – Add the new experiences to memory
- **train_actor** (*bool*) – Activate of Deactivate training of the actor
- **train_critic** (*bool*) – Activate of Deactivate training of the critic

checkpoint()
Save the weights

load_memory(*memory*)
Loads the given memory as the replay buffer

restore_checkpoint(*actor=True*, *critic=True*, *checkpoint_id=0*)
Restore from checkpoint

save(*name='DDPG'*)
Save the model as an HDF5 file

train_actor(*batch*, *sgd_iterations=1*, *can_reset_actor=False*)
Fit the actor network

train_controllers(*train_critic=True*, *train_actor=True*, *can_reset_actor=False*, *hard_update_target_critic=False*, *hard_update_target_actor=False*)
Fit the actor and critic networks

Parameters

- **train_critic** (*bool*) – Whether to fit the critic
- **train_actor** (*bool*) – Whether to fit the actor
- **can_reset_actor** (*bool*) –

train_critic(*batch*, *sgd_iterations=1*)
Fit the critic network

CHAPTER 3

hooks

class rl.hooks.hook.**Hook**(*agent_id='default'*, *experiment_id='default'*)

The abstract Hook class. A hook is designed to be a callable running on an agent object. It shouldn't return anything and instead exports the data itself (e.g. pickle, image). It is run at the end of **each step**.

The hook API relies on the following agent attributes, always available:

- agent.training: boolean: Whether the agent is in training mode
- agent.step: int: the step number. Begins to 1.
- agent.reward: The reward of the current step
- agent.episode: int: The current episode. Begins to 1.
- agent.episode_step: int: The step count in the current episode. Begins to 1.
- agent.done: Whether the episode is terminated
- agent.step_summaries: A list of summaries of the current step

These variables may also be available: * agent.episode_reward: The cumulated reward of the current episode * agent.observation: The observation at the beginning of the step * agent.observation_1: The observation at the end of the step * agent.action: The action taken during the step * agent.policy * agent.goal * agent.achievement * agent.error

Parameters

- **agent** – the RL agent
- **episodic** – Whether the hook will use episode information

agent_init()

Callback that is called when the agent is initialized

experiment_init()

Callback that is called when the experiment is initialized

experiments_init()

Callback that is called when the experiments object is initialized

class rl.hooks.hook.**ValidationHook**(**args*, ***kwargs*)
Perform validation of the hooks variables at runtime

CHAPTER 4

memory

class rl.memory.**Batch**(*state0*, *action*, *reward*, *state1*, *terminal1*)

action
Alias for field number 1

reward
Alias for field number 2

state0
Alias for field number 0

state1
Alias for field number 3

terminal1
Alias for field number 4

class rl.memory.**Experience**(*state0*, *action*, *reward*, *state1*, *terminal1*)

action
Alias for field number 1

reward
Alias for field number 2

state0
Alias for field number 0

state1
Alias for field number 3

terminal1
Alias for field number 4

class rl.memory.**Memory**(*env*)
Abstract memory class

append(*experience*)
Add the experience to the memory

sample(*batch_size*)
Get a sample from the memory

Parameters **batch_size** (*int*) – size of the batch

Returns A *Batch* object

class rl.memory.**SimpleMemory**(*env*, *limit*)
A simple memory directly storing experiences in a circular buffer

Data is stored directly as an array of *Experience*

dump()
Get the memory content as a single array

classmethod **from_file**(*env*, *limit*, *file_path*)
Create a memory from a pickle file

get_idxs(*idxs*, *batch_size*)
Get a non-contiguous series of indexes

save(*file*)
Dump the memory into a pickle file

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

r

Index