Train

Jan 27, 2020

Contents

A library to build and train reinforcement learning agents in OpenAI Gym environments.

Read full documentation here.

CHAPTER 1

Getting Started

An agent has to implement the `act()` method which takes the current `state` as input and returns an action:

```
from train import Agent

class RandomAgent(Agent):

    def act(self, state):
        return self.env.action_space.sample()
```

Create an environment using OpenAI Gym:

```
import gym

env = gym.make('CartPole-v0')
```

Initialize your agent using the environment:

```
agent = RandomAgent(env=env)
```

Now you can start training your agent (in this example, the agent acts randomly always and doesn't learn anything):

```
scores = agent.train(episodes=100)
```

You can also visualize how the training progresses but it will slow down the process:

```
scores = agent.train(episodes=100, render=True)
```

Once you are done with the training, you can test it:

```
scores = agent.test(episodes=10)
```

Alternatively, visualize how it performs:

```
scores = agent.test(episodes=10, render=True)
```

To learn more about how to build an agent that learns see agents documentation.

See examples directory to see implementations of some algorithms (DQN, A3C, PPO etc.) created using TensorFlow, PyTorch and NN libraries.

CHAPTER 2

Installation

Requirements:

- Python >= 3.6

Install from PyPI (recommended):

```
pip install train
```

Alternatively, install from source:

```
git clone https://github.com/marella/train.git
cd train
pip install -e .
```

To run examples and tests, install from source.

Other libraries such as Gym, TensorFlow, PyTorch and NN should be installed separately.

CHAPTER 3

Examples

To run examples, install TensorFlow, PyTorch and install other dependencies:

```
pip install -e .[examples]
```

and run an example in examples directory:

```
cd examples
python PPO.py
```

CHAPTER 4

Testing

To run tests, install dependencies:

```
pip install -e .[tests]
```

and run:

```
pytest tests
```

CHAPTER 5

Agents

All agents should extend the base *Agent* class and implement the *act()* method:

```
from train import Agent

class MyAgent(Agent):

    def act(self, state):
        ...
```

When *train()* or *test()* methods are called, an action is selected by calling the *act()* method and passed to the environment. Then the environment returns a reward and observation. This entire transition (S, A, R, S') is saved in a *Transitions* object which can be accessed using `self.transitions`. When an episode terminates, a new episode is started by resetting the environment and agent.

During training, the following callback methods on agent are called at respective stages:

```
on_step_begin
on_step_end
on_episode_begin
on_episode_end
```

These methods combined with the *Transitions* object in `self.transitions` can be used to implement various algorithms. `on_step_end()` can be used to implement online algorithms such as TD(0) and `on_episode_end()` can be used to implement algorithms such as Monte Carlo methods:

```
class MyAgent(Agent):

    def on_step_end(self):
        # DQN
        S, A, R, Snext, dones = self.transitions.sample(32) # randomly sample␣
↪transitions
        ...

    def on_episode_end(self):
```

(continues on next page)

(continued from previous page)

```
        # REINFORCE
        S, A, R, Snext, dones = self.transitions.get() # get all recent transitions
        self.transitions.reset() # reset transitions for next episode
        ...
```

Note: Transitions are not recorded when running *test()*.

5.1 Agent

class train.**Agent**(*state=0*, *transitions=1*, ***kwargs*)

Base class for all agents.

Parameters

- **state** (*int, State*) – A number representing the number of recent observations to save in state or a custom *State* object.
- **transitions** (*int, Transitions*) – A number representing the number of recent transitions to save in history or a custom *Transitions* object.
- **env** – OpenAI Gym like environment object.
- **gamma** (*float*) – A custom parameter that can be used as discount factor,
- **alpha** (*float*) – A custom parameter that can be used as learning rate ,
- **lambd** (*float*) – A custom parameter that can be used by various algorithms such as TD(lambda),
- **parameters** – List of trainable variables used by agent.

act(*state*)

Select an action by reading the current state.

Parameters **state** (*array_like*) – Current state of agent based on past observations.

Returns An action to take in the environment.

run(*episodes*, *env=None*, *max_steps=-1*, *max_episode_steps=-1*, *render=False*)

Run the agent in environment.

Parameters

- **episodes** (*int*) – Maximum number of episodes to run.
- **env** – OpenAI Gym like environment object.
- **max_steps** (*int*) – Maximum number of total steps to run.
- **max_episode_steps** (*int*) – Maximum number steps to run in each episode.
- **render** (*bool*) – Visualize interaction of agent in environment.

Returns List of cumulative rewards in each episode.

Return type list

test(**args*, ***kwargs*)

Run the agent in test mode by setting `self.training = False`.

See: *run()*

train(**args*, ***kwargs*)
Run the agent in training mode by setting `self.training = True`.

See: *run()*

CHAPTER 6

State

State objects can be used to represent the agent's state. They can be used to save the recent observations seen by agent and process them before passing to the *act()* method. The following example saves last 2 observations (images) after transforming them (crop, scale etc.) and computes the difference between them which can be useful for tracking motion:

```
from train import State

class MyState(State):

    def __init__(self, **kwargs):
        super(MyState, self).__init__(length=2, **kwargs)

    def process_observation(self, observation):
        x = observation
        x = x[35:-15, :, :] # crop
        x = np.dot(x, [.299, .587, .114]) # grayscale
        x = x / 255 # scale
        return x

    def process_state(self, state):
        prev, current = state
        diff = current - prev
        return diff.reshape(diff.shape + (1, ))
```

Custom state objects can be passed to agent during initialization:

```
state = MyState()
agent = MyAgent(state=state, env=env)
```

6.1 State

class train.**State**(*length=0*, *zeros=None*)
Core class to represent agent's state. Saves recent observations seen by agent.

Parameters

- **length** (*int*) – Number of recent observations to save.
- **zeros** (*array_like*) – Array of zeros with same shape as each observation that will be used to pad initial states when number of recent observations is smaller than length of state.

get(*asarray=True*, *dtype='float32'*)
Get the current state.

Parameters

- **asarray** (*bool*) – If `True` returns an `ndarray`.
- **dtype** (*dtype*) – Data type of the returned value.

Returns Processed state.

Return type (array_like, list)

process_observation(*observation*)
Process observation before saving it.

Parameters **observation** (*array_like*) – Observation returned by environment.

Returns Processed observation.

Return type array_like

process_state(*state*)
Process state before passing it to *act()*.

Parameters **state** (*array_like, list*) – List of recent observations.

Returns Processed state.

Return type (array_like, list)

reset()
Reset current state.

update(*observation*)
Update the current state based on new observation.

Parameters **observation** (*array_like*) – Observation returned by environment.

6.2 Transitions

class train.**Transitions**(*maxlen*)
Queue like data structure to save recent transitions observed by agent. Can be used as a replay buffer for algorithms like DQN.

Parameters **maxlen** (*int*) – Number of recent transitions to save. When negative, there is no limit on the number of transitions saved.

get(***kwargs*)
Get all transitions.

Returns List of transitions or a Transition object containing lists of values.

Return type (list, *Transition*)

last()
Return last transition.

Returns Last transition.

Return type *Transition*

Raises IndexError – When it is empty.

reset()
Reset transitions.

sample(*batch_size*, ***kwargs*)
Randomly sample transitions.

Parameters **batch_size** (*int*) – Number of transitions to sample.

Returns List of transitions or a Transition object containing lists of values.

Return type (list, *Transition*)

6.3 Transition

class train.**Transition**(*state*, *action*, *reward*, *next_state*, *done*)

CHAPTER 7

Utils

train.utils.**check_shape**(*a*, *b*)

Check if the shapes of given values match.

Parameters

- **a** (*array_like, tuple*) – An object with shape attribute or a tuple representing shape.
- **b** (*array_like, tuple*) – An object with shape attribute or a tuple representing shape.

Raises Exception – When shapes don't match.

train.utils.**zeros_like**(*a*, *dtype='float32'*)

Return an array of zeros with same shape as given array.

Parameters **a** (*array_like, iterable*) – An object with shape attribute or an iterable.

Returns Array of zeros with the same shape as a.

Return type (array_like, list)

Python Module Index

t

Index