IN4010 Practical Assignment: Reinforcement Learning 1

December 18, 2019

Contents

1 Intro **2**

2 Set up **2**

3 Environments **2**

4 Deep Q-Learning **3**

1 Intro

In the previous assignment we got acquainted with Q-learning. One limitation is that it doesn't scale well to larger problems. In this practical assignment, we will get more familiar with deep reinforcement learning through deep Q-learning. The paper introducing this method can be found here.

In the zip with code you can find the following files:

1. deep_q_learning_main.py, contains a main loop you can use to run your experiments.
2. deep_q_learning_skeleton.py, file for the deep Q-learning agent, a part of the code is already provided, part still needs to be implemented.

Deliverable

For this assignment you are required to upload a zip-file containing:

1. Code files with your solutions to the coding exercises
2. A pdf with answers to the questions

2 Set up

You will need to get a working python3 installation and install open AI gym.

Set up your python environment

For managing python environments, we highly recommend virtualenv and virualenvwrapper.

See: https://virtualenvwrapper.readthedocs.io/en/latest/

These will allow you to create project-specific python environments. For instance, after installing virtualenvwrapper, setting up a project workspace is as simple as

```
mkvirtualenv -p /usr/bin/python3 AIT
workon AIT
pip install ipython #optional if you want ipython
```

Install open AI gym

Now we can install gym:

```
pip install gym[box2d]
```

(For some reason, the box2d environments are not installed by default and hence need to be specified explicitly: https://github.com/openai/gym/issues/1603)

Optional: install pytorch

If you want to use the available neural network code

```
pip install torch
```

3 Environments

We will use the LunarLander-v2 from Gym. Check here for a short description and here for the source code.

4 Deep Q-Learning

In regular Q-learning, we had a table to look up the Q-value for each state-action pair. In Deep reinforcement learning we instead use function approximation. We define the Q-value as $Q(s, a; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ are the parameters of the function approximation, in this case a neural network.

In deep_q_learning_skeleton.py a basic version of deep Q-learning has already been implemented.

Todo 1. Familiarize yourself with the code in deep_q_learning_skeleton.py.

Question 1. Run deep_q_learning_main.py a couple of times. What behavior from the agent do you observe? Does it learn to land safely between the flags?

Coding Exercise 1. Complete the class ReplayMemory in deep_q_learning_skeleton.py. Change QLearner so that it uses the experience replay, that is:

1. store_experience should be called in the function process_experience
2. In process_experience sample a batch of "self.batch_size" from the replay memory and update the network using this experience.

Question 2. Again run deep_q_learning_main.py a couple of times. What behavior from the agent do you observe? Does it learn to land safely between the flags? Did the agent improve compared to Question 1?

Coding Exercise 2. Now we will use an additional target network, $Q(s, a; \boldsymbol{\theta}^-)$. Initially our Q-network and the target network will have the same parameters. We will use the target network to provide the estimated future values. That is, we change our target from

$$r + \gamma \max_{a'} Q(s', a', \boldsymbol{\theta}) - Q(s, a, \boldsymbol{\theta})$$

to

$$r + \gamma \max_{a'} Q(s', a', \boldsymbol{\theta}^-) - Q(s, a, \boldsymbol{\theta})$$

and at the end of every episode we set $\boldsymbol{\theta}^- = \boldsymbol{\theta}$.

1. In deep_q_learning_main.py add a target network to the initialization of the QLearner.
2. At the end of every episode set $\boldsymbol{\theta}^- = \boldsymbol{\theta}$,

   ```
   self.target_network.load_state_dict(self.Q.state_dict())
   ```

3. Change single_Q_update (and batch_Q_update) to use the Q value estimation for the next state from the target network.

Question 3. Again run deep_q_learning_main.py a couple of times. What behavior from the agent do you observe? Does it learn to land safely between the flags? Did the agent improve compared to Question 2?