CS4375 Practical Assignment: Reinforcement Learning

October 6, 2022

Contents

1 Intro

In this practical assignment, you will develop some hands-on experience with implementing and applying reinforcement learning techniques. The assignment consists of two parts: one focusing on regular Q-learning, and one focusing on Q-learning with a *deep Q network* [1].

Coding Framework

In the zip file you can find the following files:

1. For setting up the coding environment:
   (a) docker/Dockerfile, contains the specifications of the environment and is used to build the coding environment locally.
   (b) Makefile, contains targets that can be used to manage the coding environment and run the python code.
   (c) [OPTIONAL] docker/ait-rl-env.tar.gz, docker image of the coding environment that can be imported if needed. This file can be downloaded separately.
2. For the Q-learning part:
   (a) simple_grid.py, the environment(s) we will be using, see Section 3 for a description.
   (b) q_learning_main.py, contains a main loop you can use to run your experiments.
   (c) q_learning_skeleton.py, file for the Q-learning agent, to be coded up.
3. For the deep Q-learning part:
   (a) deep_q_learning_main.py, contains a main loop you can use to run your experiments.
   (b) 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.

Deliverables

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 The set up

For this assignment you will need an environment with python3 with OpenAI gym installed. To make the setup easy and consistent across all operating systems, we have provided with this assignment additional files and in the following sections, instructions on how to set up the coding environment you will be using.

2.1 Installing Docker

You can skip this section if you already have docker installed on your system.

2.1.1 Windows

On windows systems, first install the Windows Subsystem for Linux Version 2 (WSL2) [1]. In the following section, you will be required to install docker which relies on the WSL2 kernel for containerization. Once you have a working WSL2 installation, go ahead and install Docker [2]. You might need to restart your system after the installation is done.

2.1.2 MacOS and Linux based OSes

Go ahead and install docker[2] by following the installation instructions for your OS. Please note that depending on your specific Linux based OS you might need to manually create the docker group and add your user to it [3]. You may also need to restart your system after the installation is done.

2.1.3 Verifying the installation

You can check if docker is working correctly by running the following command in a UNIX terminal (WSL2 bash shell on Windows):

```
docker run hello-world
```

Also ensure that `make` [4] command is installed and available in your terminal. `make` should be available by default on most UNIX based OSes. You can check if `make` is installed by running

```
make
```

in your terminal, which will output something like this

```
make: *** No targets specified and no makefile found.  Stop.
```

If you do not have `make` installed, please go ahead and install it.

2.2 Readying the coding environment[5]

This section will go over how to build or import the provided docker image which hosts the dependencies that are needed to run the give code. There are two possible ways to do this.

2.2.1 Building the Docker image [RECOMMENDED]

In a UNIX terminal (or WSL2 bash shell on Windows), from the root of the working directory you can run

```
make build
```

which will start building the image, called `ait-rl-env`, according to the `Dockerfile` present in `docker/` folder.

2.2.2 Importing the Docker image

You also have the option of importing the image archive directly. You can download the image archive, called `ait-rl-env.tar.gz`, from brightspace to the `docker/` folder, then run

```
make import_image
```

[1] https://learn.microsoft.com/en-us/windows/wsl/install
[2] https://www.docker.com/
[3] https://docs.docker.com/engine/install/linux-postinstall/
[4] https://www.gnu.org/software/make/
[5] Please note that the word environment in this context refers to the software dependencies for this assignment. In the next section it will refer to the environments an RL agent will operate on.

2.2.3 Verifying the image

Once the image has been successfully built or imported you can verify it exists by running

`docker images`

which will list all the available docker images in your system, one of which should be `ait-rl-env`.

2.3 Running the code

We have provided you with a `Makefile` to run the assignment code that abstracts away the long commands and arguments that are used by docker. The important commands to interact with and run your code inside the coding environment are.

- `make run_ql` :- will run `q_learning_main.py`
- `make run_dqn` :- will run `deep_q_learning_main.py`

Please be informed that upon executing `deep_q_learning_main.py`, the code will create a folder which contains the recordings of your agent's episodes. These files will be overwritten every time `deep_q_learning_main.py` is executed. You can have a look at the `Makefile` for other targets that have been defined.

3 RL Environments

In this assignment you will work with two environments. The first will be used for the tabular Q-learning part of the assignment, and will be used to develop an understanding of the difficulty of exploration. The second, Lunar Lander, will be used for the deep Q-learning part.

3.1 Q-Learning

For the first part, we will experiment with 'drunken walk environments' that are located in simple_grid.py. It is a simple grid environment, completely based on the code of 'FrozenLake', credits to the original authors.

The story here is as follows: You are finding your way home (G) after a great party which was happening at (S). Unfortunately, due to recreational intoxication you find yourself only moving into the intended direction 80% of the time, and perpendicular to that the other 20%. To make matters worse, the local community has been cutting the budgets for pavement (.) maintenance, which means that the way to home is full of potholes (H), which are very likely to make you trip. If you fall, you are obviously magically transported back to the party, without getting some of that hard-earned sleep.

There are different maps available, an example is the "walkInThePark" which has the following shape:

```
"walkInThePark": [
"S.......",
".....H..",
"........",
"......H.",
"........",
"...H...G"
]
```

Another example is the "theAlley" map:

```
"theAlley": [
"S...H...H...G"
]
```

An episode ends when the agent trips or when it reaches the goal. Reaching the goal gives a reward of +10. When moving to a pothole tile there is a 20% chance to trip[6], tripping results in a broken leg penalty.

For the specifics of the implementation of this environment, take a look at the source (simple_grid.py).

3.2 Deep Q-Learning

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

4 Part I: Q-Learning

The Q-Learning algorithm allows us to estimate the optimal Q function using only trajectories from the MDP obtained by following some policy.

Q-learning with ϵ-greedy exploration acts in the following way at a timestep t:

1. In the current state, s, take action a such that a is random with probability ϵ and the greedy action ($a = \max_{a \in A} Q(s, a)$) with probability $1 - \epsilon$;
2. Observe the reward and the next state, r and s'.
3. Update the Q-value as follows:

$$Q^{\text{new}}(s, a) = (1 - \alpha)Q^{\text{old}}(s, a) + \alpha[r + \gamma \max_{a' \in A} Q^{\text{old}}(s', a')]$$

Note that when the episode terminates in s', the update is as follows:

$$Q^{\text{new}}(s, a) = (1 - \alpha)Q^{\text{old}}(s, a) + \alpha r$$

Coding Exercise 1. Implement Q-learning with ϵ-greedy action selection, complete the class given in q_learning_skeleton.py.

Question 1. Which environment, "walkInThePark" or "theAlley", is more difficult to learn in? Why?

Walk in the park

We'll start using the "walkInThePark" map.

Question 2. For the "walkInThePark" map, run some experiments for 1000 episodes with the following settings: $\epsilon = 0.05$, $\gamma = 0.9$, $\alpha = 0.1$. Does the agent learn an optimal policy? Why (not)? Report the (greedy) policy that the agent learned.

[6] This replaces the probability to move perpendicular.

The alley

Now we will use the "theAlley" map. First you will compute the optimal Q values, Q^*, this can be done by using value iteration.

Question 3. If you were to apply value iteration, how can you deal with the terminal states?

Question 4. Calculate (or compute) Q^*, the optimal Q-values, for the "theAlley" map with $\gamma = 0.9$, BROKEN_LEG_PENALTY $= -10$.

Question 5. Run some experiments for 1000 episodes with the following settings: $\epsilon = 0.05$, $\gamma = 0.9$, $\alpha = 0.1$, BROKEN_LEG_PENALTY $= -10$. Does the agent learn an optimal policy? Why (not)?

Question 6. Now calculate (or compute) Q^*, the optimal Q-values, for the "theAlley" map with $\gamma = 0.9$, BROKEN_LEG_PENALTY $= -5$.

Question 7. Run some experiments for 1000 episodes with the following settings: $\epsilon = 0.05$, $\gamma = 0.9$, $\alpha = 0.1$, BROKEN_LEG_PENALTY $= -5$. Does the agent always learn an optimal policy? Why (not)?

Coding Exercise 2. Try to change the exploration strategy of the agent in a way that allows it to find the optimal solution more often (and quicker).

Question 8. Describe and explain your new exploration strategy. Does it help the agent in learn the optimal policy more often/quicker?

5 Part II: 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 9. Run deep_q_learning_main.py [7] a couple of times. What behavior from the agent do you observe? Does it learn to land safely between the flags?

Coding Exercise 3. 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 10. 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 8?

Coding Exercise 4. Now we will use an additional target network, $Q(s, a; \boldsymbol{\theta}^-)$. Initially our Q-network and

[7] refer to section 2.3 for instructions on how to run the code

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 11. 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 9?

References

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 02 2015.