Online and Reinforcement Learning

2024-2025

Home Assignment 5

Yevgeny Seldin **Christian Igel**
Department of Computer Science
University of Copenhagen

The deadline for this assignment is **12 March 2025, 21:00**. You must submit your *individual* solution electronically via the Absalon home page.

A solution consists of:

- A PDF file with detailed answers to the questions, which may include graphs and tables if needed. Do *not* include your full source code in the PDF file, only selected lines if you are asked to do so.
- A .zip file with all your solution source code with comments about the major steps involved in each question (see below). Source code must be submitted in the original file format, not as PDF. The programming language of the course is Python.

Important Remarks:

- IMPORTANT: Do NOT zip the PDF file, since zipped files cannot be opened in *SpeedGrader*. Zipped PDF submissions will not be graded.
- Your PDF report should be self-sufficient. I.e., it should be possible to grade it without opening the .zip file. We do not guarantee opening the .zip file when grading.
- Your code should be structured such that there is one main file (or one main file per question) that we can run to reproduce all the results presented in your report. This main file can, if you like, call other files with functions, classes, etc.
- Handwritten solutions will not be accepted.

1 Deep Q-Learning `(50 points) [Christian]`

We (re-)consider the *Lunar Lander* environment from the Gymnasium framework, see `https://gymnasium.farama.org/environments/box2d/lunar_lander/` for details of the RL task. You may need to install Gymnasium via `pip install gymnasium[box2d]` and perhaps some other packages.

The notebook `LunarLanderDQN2025Assignment.ipynb` implements a simple deep Q-learning except for one line. If you have problems with the visualization, you can comment out the corresponding lines (try to get it running, it is fun).

1.1 Neural architecture

The policy network definition contains the lines:

```
    def forward(self, x_input):
        x = F.tanh(self.fc1(x_input))
        x = F.tanh(self.fc2(x))
        x = torch.cat((x_input, x), dim=1)
        x = self.output_layer(x)
        return x
```

1.1.1 Structure (5 points)

What is `x = torch.cat((x_input, x), dim=1)` doing? What is this neural network structure concept called?

1.1.2 Activation function (5 points)

Why has the tanh activation function been chosen instead of, say, the standard logistic function?

1.2 Adding the Q-learning (15 points)

Add the missing line to compute the variable `y` setting the actual Q-learning targets. The line should be added to the notebook after the comment line `# Compute targets`.

After that, the notebook should reliably find a policy solving the task with a return larger than 200 (of course, there might be bad trials that do not find a sufficiently good solution).

Report the line in the report and *briefly* explain what it is doing. Describe the equation it implements using the notation introduced in the lecture.

The notebook saves a plot of the learning process in the file `deepQ.pdf`. Show a successful learning process in the report by including the corresponding plot (or a beautified version of it).

1.3 Epsilon (5 points)

Explain what the line

```
    explore_p = explore_stop + (explore_start - explore_stop)*np.exp(-decay_rate*ep)
```

is doing.

1.4 Gather (5 points)

Explain what the line

```
    Q_tensor = torch.gather(output_tensor, 1, actions_tensor.unsqueeze(-1)).squeeze()}
```

is doing and why it is necessary (it is for a data structure/programming reason, not a theoretical one).

1.5 Target network (15 points)

Introduce a delayed target Q update via a target network `targetQN`. Smoothly blend the parameters:

$$\theta_{\text{target}} \leftarrow \tau\theta_{\text{main}} + (1-\tau)\theta_{\text{target}}$$

for $\tau \in]0,1[$ (sometimes referred to as Polyak averaging), where θ_{main} and θ_{target} are the parameters of the networks `mainQN` and `targetQN`, respectively.

The code for updating the target network can be added directly after the gradient-based update of the main network. You can get the parameters of network like this:

```
        sdTargetQN = targetQN.state_dict()
```

You can change the values in this data structure as follows:

```
        for key in sdTargetQN:
            sdTargetQN[key] = 42.
```

You can overwrite the parameters using `load_state_dict`.

Report the code you added in the report and *briefly* explain what it is doing.

Keeping all other hyperparameters from the notebook unchanged and the same for both approaches, empirically compare the performance with and without target network. Do you observe a difference?

1.6 Programming (optional, 0 points)

Any suggestions on how to make programming more elegant/efficient? Send your suggestions to igel@diku.dk.

2 A tighter analysis of the Hedge algorithm `(20 points) [Yevgeny]`

Solve Exercise 5.7 in Yevgeny's lecture notes.

3 Empirical evaluation of algorithms for adversarial environments `(5 points) [Yevgeny]`

Solve Exercise 5.10 in Yevgeny's lecture notes.

4 Empirical comparison of UCB1 and EXP3 algorithms `(25 points) [Yevgeny]`

Solve Exercise 5.11 in Yevgeny's lecture notes.

5 The doubling trick `(0 points) [Yevgeny]` (Optional)

Solve Exercise 5.12 in Yevgeny's lecture notes.

6 Rewards vs. losses `(0 points) [Yevgeny]` (Optional)

Solve Exercise 5.13 in Yevgeny's lecture notes.

7 Regularization by relative entropy and the Gibbs distribution `(0 points) [Yevgeny]` (Optional)

Solve Exercise 5.9 in Yevgeny's lecture notes.