Reinforcement Learning: From Foundations to Deep Approaches

Summer Semester 2024, Homework 2

Prof. Georgia Chalvatzaki, Dr. Davide Tateo

Total points: 103

Deadline: 23:59, Friday, 5th July 2024

Submission: Use the provided Python scripts to solve the problems. We provided you with a zip file for the homework project that includes the Python scripts (to solve), bash scripts (to run experiments), and a yaml file (to create a conda environment). The structure of the project is similar to MushroomRL. Provide a report with all the requested plots mentioning the respective question number. In addition, submit your solutions in the provided Python scripts. For the submission, upload a zip file that contains the Python scripts and the report.

We encourage you to use Conda to create an environment that contains all packages necessary for solving the homework. We provide you with a yaml file (environment.yaml) that you can use to set up the conda environment as follows:

```
conda env create --file=environment.yaml
```

Each submission is individual!

Late submissions: We will accept late submissions, but you will lose 10% of the total reachable points for every day of delay. Note that even 15-minute delays will be counted as being one day late! Submissions three days later than the designated deadline will not be considered. If you are not able to make the deadline, e.g.due to medical reasons, you need to contact us before the deadline. We might waive the late penalty in such a case.

2.1 Feature Construction and Function Approximation [40 Points]

Solve the CartPole environment (Fig. 1) using Online Sarsa Lambda using various types of features. CartPole is a continuous environment with a two-dimensional state containing the angle $\alpha \in [-\pi, \pi]$ and angular velocity $\dot{\alpha} \in [-3\pi, 3\pi]$ of the pole. The action space is discrete, allowing the agent to move the cart left, right, or remain stationary. At each state, the agent receives at each step a reward signal defined as:

$$r = \begin{cases} -10 & \text{if } |\alpha| \geq \frac{\pi}{2} \text{ or } |\dot{\alpha}| \geq 3\pi \\ 0.1 \times \left(\frac{\pi}{8} - |\alpha|\right) + 0.05 \times \left(\frac{\pi}{8} - |\dot{\alpha}|\right) & \text{otherwise} \end{cases}$$

Please refer to [1] and the mushroom implementation for more details.

Follow the *cart_pole_func_approx.py* template filling in the spots annotated by ”[YOUR CODE!]”. The task is composed of the following components:

1. **Create an appropriate set of features to solve the CartPole environment. The experiment file should support using Gaussian radial basis functions, Fourier basis, Rectangular Tiles, and raw state (no feature construction). [20 Points]**

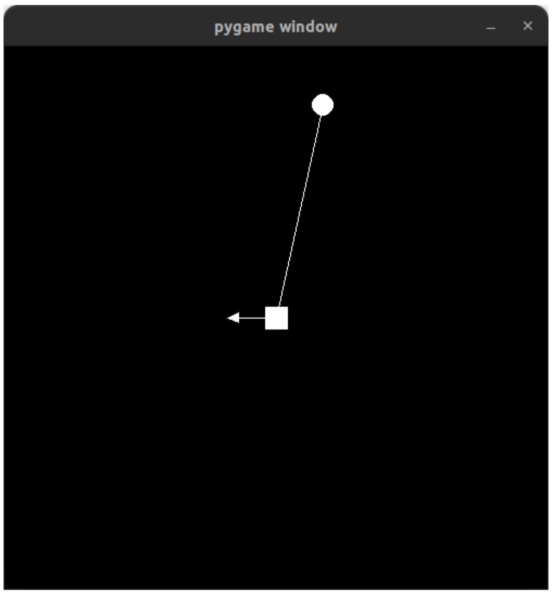

Fig. 1: A visual representation of the cartpole environment where the line and circle represent the pole/inverted pendulum, the square represents the cart controlled by the agent, and the arrow represents the action taken by the agent.

2. **Create a Sarsa-λ agent with a linear approximator. Write down the training loop. Use functions from MushroomRL to evaluate the agent and log the average undiscounted return, average discounted return, and average episode length while training. [10 Points]**

3. **Tune the parameters of each feature annotated by "[TUNE PARAMETER!]". For the best parameter values you find, run every feature type for 30 runs for the final results. Remember to provide three plots with the assignment submission, comparing the average return (discounted and undiscounted) and episode length for every feature type with the mean and confidence interval across all the 30 runs. [10 Points]**

2.2 Deep Q-Learning [63 Points]

DQN [2] is one of the earliest attempts to merge deep learning with Reinforcement Learning. In this exercise, we will study the performance of different DQN variants on the Space Invaders (Fig. 2) game from [3]. Instead of using a linear approximator as in 2.1, you will build your first ConvNet to approximate the Q function. Please use the hints in the Python script provided to implement the network.

Follow the *minatar_dqn_variants.py* template filling in the spots annotated by "[YOUR CODE!]". Some comments are included in the script to help you solve the task. Please use the values of the experiment's parameters specified in the script without changing them. The default values of all the parameters used are in the argparser.py file.

In addition, use the *run_experiments.sh* bash script to run your experiments based on the algorithm's name passed to the script. For example, to run a DQN experiment, you can run the following command line:

```
sh run_experiment.sh DQN dqn
```

The script can also be used on the Lichtenberg cluster to submit jobs as shown in practical session 7. Please do not forget to activate your environment before running the bash script. If you want to run DQN experiments on the cluster, you can run the following in your cluster workspace:

```
sbatch run_experiment.sh DQN dqn
```

The task is composed of the following components:

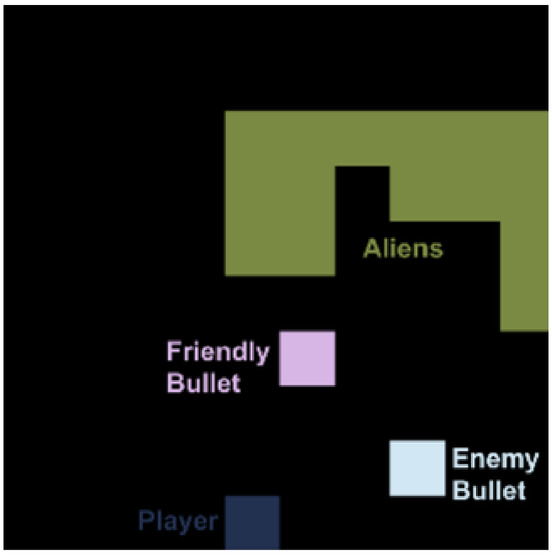

Fig. 2: Colour indicates active channel at each spatial location, but note that the representation provided by the environment consists of binary values for each channel and not RGB values.

1. **Use MushroomRL to implement the training and evaluation loop. You will need to log the average discounted return, value estimate at the initial states, and episode length metrics of the learning algorithms. [5 Points]**

2. **Implement a convolutional neural network according to the specifications given in the python script. Complete the required code and ese MushroomRL to run each of the following algorithms for 5 runs on the Space Invaders game: DQN [2], DoubleDQN [4], MaxminDQN [5], and AveragedDQN [6] and report the plot. [20 Points]**

3. **Implement a plotting script in plot.py to visualise the data generated by the learning loop described in the previous question. Plots should show the mean and confidence interval of each algorithm computed from the different seeds. [5 Points]**

4. **Implement a DQN algorithm without a replay buffer. This variant is the on-policy version of the original DQN algorithm. Train and evaluate the algorithm for 5 seeds and report the plot. [15 Points]**

5. **In MaxminDQN, we use n models for each the online and target approximators. The estimation of Q is based on the minimum value across all the n models. Estimating the Q_{next}, for the TD target, is similar to the DQN algorithm by taking the maximum of the Q values in the next state. In this question, we ask you to modify the MaxminDQN algorithm implemented in MushroomRL by creating a new algorithm called DoubleMaxminDQN where the Q_{next} is estimated based on the DoubleDQN algorithm. Train and evaluate the algorithm for 5 seeds and report the plot. [15 Points]**

6. **(BONUS!) Implement a Naive DQN algorithm without all the tricks introduced in the lecture. In addition to removing the replay buffer as in question 4, you need to estimate the Q_{next} using only the online approximation and ignore the clipping of reward. Train and evaluate the algorithm for 5 seeds and report the plot. [3 Points]**

Hint! Please report three plots for average return, value estimate at the initial states, and episode length for all the learned algorithms.

Reference

[1] Stefan Friedrich, Michael Schreibauer und Martin Buss. “Least-squares policy iteration algorithms for robotics: Online, continuous, and automatic”. In: *Engineering Applications of Artificial Intelligence* 83 (Aug. 2019), S. 72–84. DOI: 10.1016/j.engappai.2019.04.001.

[2] Volodymyr Mnih u. a. “Playing Atari with Deep Reinforcement Learning”. In: *CoRR* abs/1312.5602 (2013). arXiv: 1312.5602. URL: http://arxiv.org/abs/1312.5602.

[3] Kenny Young und Tian Tian. “MinAtar: An Atari-inspired Testbed for More Efficient Reinforcement Learning Experiments”. In: *CoRR* abs/1903.03176 (2019). arXiv: 1903.03176. URL: http://arxiv.org/abs/1903.03176.

[4] Hado van Hasselt, Arthur Guez und David Silver. “Deep Reinforcement Learning with Double Q-learning”. In: *CoRR* abs/1509.06461 (2015). arXiv: 1509.06461. URL: http://arxiv.org/abs/1509.06461.

[5] Qingfeng Lan u. a. “Maxmin Q-learning: Controlling the Estimation Bias of Q-learning”. In: *CoRR* abs/2002.06487 (2020). arXiv: 2002.06487. URL: https://arxiv.org/abs/2002.06487.

[6] Oron Anschel, Nir Baram und Nahum Shimkin. “Deep Reinforcement Learning with Averaged Target DQN”. In: *CoRR* abs/1611.01929 (2016). arXiv: 1611.01929. URL: http://arxiv.org/abs/1611.01929.