Reinforcement Learning 2021
Practical Assignment 2: Function Approximation

1 Introduction

In the previous assignment, you have worked with search algorithms on a simple board game called Hex. Although we have seen various advanced search techniques, they still suffer from the near-exponential scale-up in the depth of the search. In this assignment, we will therefore start to look at the use of *learning*, or *function approximation*, to help overcome this problem. Learning allows 1) for a compact representation of the solution (usually in the form of a policy or value function, which can with learning be stored in memory in approximate form) and 2) generalization, where we automatically share information between almost similar states. Moreover, function approximation is also a practical solution for tasks with continuous action spaces, like frequently encountered in robotics.

The environment that we will use in this environment is Cartpole as shown in Figure 1. In this environment, there is a cart (black box) that moves along a horizontal axis. The goal is to move the cart in such a way that the pole is upright. Note that you do not have to implement this environment, as OpenAI has already implemented it in Gym (see source below figure, or click on highlighted Cartpole text).

Figure 1: Illustration of the Cartpole environment. Source: [1].

You will implement three function approximation algorithms (see below) that attempt to learn a Q-value for every state-action pair $Q(s, a)$ that models the expected total outcome after performing action a in state s.

2 Tabular Q-learning

First, you will try to use a technique designed for discrete state spaces, namely tabular Q-learning. As its name implies, this method keeps a table of Q-values for every state-action pair. Through experience, the entries in this table are updated to reflect the actual expected outcomes of actions in states.

Since Cartpole is continuous, and not discrete, you will first have to discretize the environment. Think carefully about the best way to do this, and make sure to explain your choices in the report! Also think about how you want to initialize the Q-table.

3 Deep Q-learning

To side-step having to discretize continuous environments in order to apply Q-learning, we may replace the Q-table by a (deep) neural network. This Deep Q-Network (DQN) will receive as input a state s and outputs for every action a the Q-value $Q(s, a)$. This way, the network simply acts as a regular table: you "look up" the Q-value using the state as key (input) and observing the produced values (outputs) for every action.

Implement the network using whatever you are comfortable with (e.g. tensorflow, pytorch, vanilla python), but make sure that we are able to reproduce your results via one single command (e.g. "sh run_task3.sh" or "python task3.py" on a university machine). Think about the architecture of your network: do you need convolutions? How many layers? How many nodes per hidden layer? You may want to experiment with various architectures, and report on these results to explain your final choice.

How does this technique compare to tabular Q-learning, e.g., in terms of learning speed, performance, usability? Is there any advantage to using DQN over tabular q-learning? Try to explain observed differences (if any) in performance and learning speed.

4 Monte-Carlo policy gradient

Another way to find a good parameterization of the policy network is through *policy gradients*. In particular, we will focus on the *Monte Carlo policy gradient*. Denote a trace by $h_t = \{s_t, a_t, r_t, s_{t+1}, .., a_{t+n}, r_{t+n}, s_{t+n+1}\}$, where n is the maximum trace length (in practice we often use $n = \infty$, to indicate that we run a trace until the environment terminates). The total reward in this trace is denoted by $R(h_t) = \sum_{i=0}^{n} \gamma^i r_{t+i}$, and our policy by $p_\theta(a|s)$, parametrized by θ. Our objective is to maximize $J(\theta)$, the expected return from the start state:

$$J(\theta) = \mathbb{E}_{h_0 \sim p_\theta(h_0)} \Big[R(h_0) \Big].$$

An unbiased estimate of the gradient of the above objective is given by:

$$\nabla_\theta J(\theta) = \mathbb{E}_{h_0 \sim p_\theta(h_0)} \Big[\sum_{t=0}^{n} \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot R(h_t) \Big] \tag{1}$$

This is known as the policy gradient. See the lecture material and course notes for further details.

Implement the Monte Carlo policy gradient algorithm. Investigate the learning behavior of the Monte-Carlo policy gradient technique and compare it to the previous two methods (tabular and deep Q-learning). What can you say about the training stability? How does this differ from (deep) Q-learning? What are some disadvantages of this technique?

5 Submission

Make sure to nicely document everything that you do. Your final submission consists of:

- Source code with instructions (e.g., README) that allows us to **easily** (single command per experiment / sub task) rerun your experiments on a university machine booted into Linux (DSLab or computer lab).
- A self-contained pdf report of at least 6 to 8 pages if not even more with figures etc. The page amount we expect of you might vary depending on your layout. This report contains an explanation of the techniques, your experimental design, results (performance statistics, other measurements,...), and overall conclusions, in which you briefly summarize the goal of your experiments, what you have done, and what you have observed/learned.

If you have any questions about this assignment, please visit our lab sessions on Friday where we can help you out. In case you cannot make it, you can post questions about the contents of the course on the Brightspace discussion forums, where other students can also read and reply to your questions. Personal questions (not about the content of the course!) can be sent to our email address: rl@liacs.leidenuniv.nl.

The deadline for this assignment is the **2nd of April 2021 at 23:59 CET**. For each full 24 hours late, one full point will be deducted (e.g., if your work is graded with a 7, but you are two days too late, you get a 5).

Good luck and have fun! :-)

6 Useful resources

For this assignment you may find the following resources useful:

- Chapter 6 of Learning to Play [2]
- OpenAI's Gym website is here. It provides a wealth of reinforcement learning examples. Browse to example Environments, and through algorithms.
- Definitely go to OpenAIs Spinning Up, and the code repo, which is here.
- Many blog posts on Gym exist. Such as here and here. A more advanced intro, using Keras, is here. Google will find many more intro's for you if you need them.
- A nice blog about the Rainbow paper is here.
- Baseline implementations of DQN, PPO, and many other algorithms are here. But first go to Spinning Up.
- Andrej Karpathy's blog is very insightful here.

References

[1] *OpenAI Gym Cartpole-v1*. `https://gym.openai.com/envs/CartPole-v1/`.

[2] Aske Plaat. *Learning to play—reinforcement learning and games*. 2020.