Programming Project: Reinforcement Learning

Work in groups of 2 or 3 students. Exceptions have to be arranged with the instructor.

1 Environment

Solve the **mountain car** task (`https://gym.openai.com/envs/MountainCar-v0/`). The goal is to drive up the mountain on the right; however, the car's engine is not strong enough to scale the mountain in a single pass. Therefore, the only way to succeed is to drive back and forth to build up momentum.

State: The state consists of the the position and velocity of the car.

Action: One out of three possible actions can be taken: push left, no push, push right

Reward: -1 for each time step, until the goal position of 0.5 is reached.

View further descriptions at `https://github.com/openai/gym/wiki/MountainCar-v0`.

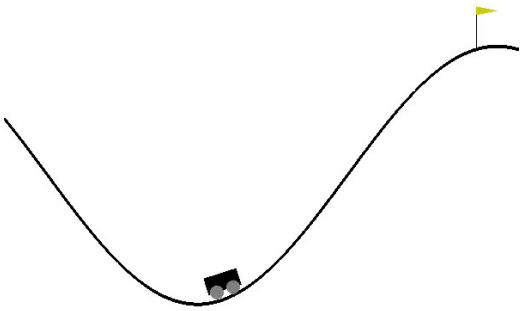

Figure 1: Mountain car

After installing *gym*[1], the mountain car environment can be set up using the following snippet:

```
import gym
# Create the gym environment
env = gym.make("MountainCar-v0")
env.seed(3333) # Set a seed for reproducability
```

Refer to the code examples shown in the lecture for the detailed steps.

2 Value Function Approximation

Try to solve the mountain car task using linear value-function approximation with temporal difference weight updates. The tasks are:

(i) Use the states and actions directly as inputs to the action-value function approximation $\hat{q}(s, a, \mathbf{w})$ for learning the weights $\mathbf{w}$. Use appropriate hyperparameters for training.

(ii) Convert the states and actions into a polynomial feature vector $\Phi(s, a)$ (choose a suitable value for the polynomial order n) which can be used to compute $\hat{q}(s, a, \mathbf{w}) = \Phi(s, a)^T \mathbf{w}$. Repeat the training process to learn the weights $\mathbf{w}$. Use the same hyperparameters as (i). In addition to polynomial features, you can also try out other feature construction methods discussed in the lecture. Mention in your report which type of features you used.

(iii) Perform multiple runs for (i) and (ii). For each run you will get a list of average rewards per episode (=total reward/number of steps). Plotting these lists directly can lead to noisy graphs. Hence average the lists for (i) and for (ii) and then compare the results by plotting a graph. Note down your observations and the reason for the difference in performance, if any.

[1] Installation instructions can be found at `https://gym.openai.com/docs/#installation`. We only need the *Classic control* environments. So `pip install gym` is enough.

Task (i)

	Average reward per episode				
	Ep1	Ep2	Ep3	Ep4	...
Run 1					
Run 2					
Run 3					
⋮					
Avg. of all runs					

Task (ii)

	Average reward per episode				
	Ep1	Ep2	Ep3	Ep4	...
Run 1					
Run 2					
Run 3					
⋮					
Avg. of all runs					

Figure 2: Plot the average results from multiple runs for (i) and (ii) in a single graph. Tables shown here are only for demonstration, do not use them in the report.

(iv) Additional task for groups with 3 members: Repeat (ii) by decaying the value of ϵ gradually instead of using a fixed value throughout (more exploration in initial episodes). Perform multiple runs and graphically compare the performance with (i) and (ii). Write down your observations and insights, and include relevant plots in the report.

3 Submission Instructions

Prepare a report of a few pages with the following content:

- Names of all members of the group. If the members' contributions differ substantially, explain their respective contributions.
- Properly labeled and captioned plots, and answers for the questions in Sec. 2.
- Explain any deviations from the methods discussed in class.
- Discuss additional insights or observations if any.

One member of the group should submit the report in PDF format as well as the complete source code as indicated on the course web page.