CmpE 343: Introduction to Probability and Statistics for Computer Engineers (Fall 2020)

Bonus Homework

Due January 10, 2021 by 11:59pm on Moodle

Note: Please check deliverables in the text. The deadline is strict. To get full points, please show your steps clearly.

Reinforcement Learning

Reinforcement Learning (RL) is a type of Machine Learning, where there is an agent that tries to find suitable actions to achieve a task in an environment.

The agent can be a software that plays the game "GO": `https://www.youtube.com/watch?v=8tq1C8spV_g` or a real robot that acts in the real world: `https://www.youtube.com/watch?v=n2gE7n11h1Y`

In the first case, the action of the agent is putting a stone on the board and in the second case, the action of the robot is the joint angles of its legs.

In these cases, we can not provide true actions to the agent, because they are hard to estimate. The value of a move in GO is dependent on the following moves and can be interpreted only after the game is over. Similarly, it is hard to tell the robot how to move the legs to walk. Therefore, we provide rewards to the agent in reinforcement learning and the agent tries to maximize the reward that it gets by trial and error. For example, in the first case, the program receives a positive reward when it wins and a negative reward when it loses. In the second case, the reward can be the distance that it walks.

You may have noticed that writing an analytical reward function is hard too. Let's say, the agent has a policy such that it will play G13 if the opponent plays G14, etc. Since we don't have the analytical reward function, the agent has to try the move to observe the outcome. That is called "trial and error". Formally, we have a policy distribution denoted with π but we don't know the corresponding reward distribution R. We use Monte-Carlo sampling to estimate the corresponding reward for the policy we select.

Homework

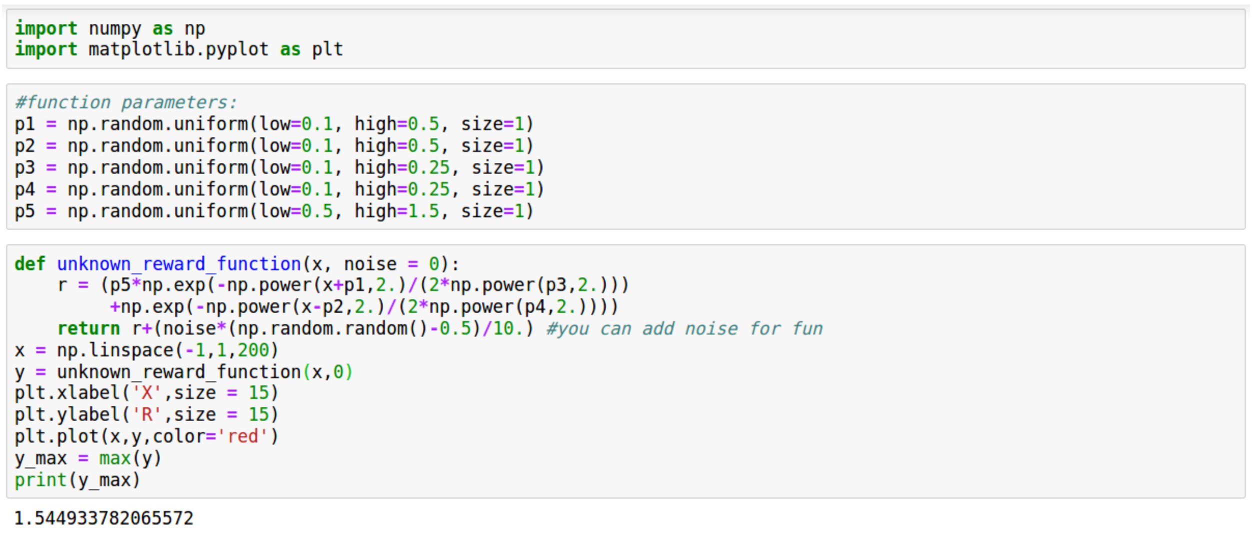
In this homework, your action parameter is denoted with the random variable X. Assume that there is a reward function that maps your action parameters to rewards: $r = g(x)$. **You will only use this function for sampling, i. e. you will put sampled x into the function to read the reward.**

For the function, you will once sample 5 parameters(p_1, p_2, p_3, p_4, p_5) from different uniform distributions at first: $f_1(p_1; 0.1, 0.5)$, $f_2(p_2; 0.1, 0.5)$, $f_3(p_3; 0.1, 0.25)$, $f_4(p_4; 0.1, 0.25)$, $f_5(p_5; 0.5, 1.5)$. Then, the function will be:

$$g(x) = e^{\frac{-(x-p_2)^2}{2(p_4)^2}} + p_5 e^{\frac{-(x+p_1)^2}{2(p_3)^2}}$$

You can check the code of the equation in figure 1.

Now, you will define your policy as a distribution. You can choose any distribution or change it in different iterations as you like, but mostly, Gaussian distribution is used in literature. **Find the desired parameters of the distribution so that sampled actions x result in high rewards around the global maximum with 0.01 threshold. The purpose is to use as few samples as possible because trials with real robots take time and resources. You are free to try any method that you want except using information that did not come from**



```python
import numpy as np
import matplotlib.pyplot as plt
```

```python
#function parameters:
p1 = np.random.uniform(low=0.1, high=0.5, size=1)
p2 = np.random.uniform(low=0.1, high=0.5, size=1)
p3 = np.random.uniform(low=0.1, high=0.25, size=1)
p4 = np.random.uniform(low=0.1, high=0.25, size=1)
p5 = np.random.uniform(low=0.5, high=1.5, size=1)
```

```python
def unknown_reward_function(x, noise = 0):
    r = (p5*np.exp(-np.power(x+p1,2.)/(2*np.power(p3,2.)))
        +np.exp(-np.power(x-p2,2.)/(2*np.power(p4,2.))))
    return r+(noise*(np.random.random()-0.5)/10.) #you can add noise for fun
x = np.linspace(-1,1,200)
y = unknown_reward_function(x,0)
plt.xlabel('X',size = 15)
plt.ylabel('R',size = 15)
plt.plot(x,y,color='red')
y_max = max(y)
print(y_max)
```

```
1.544933782065572
```

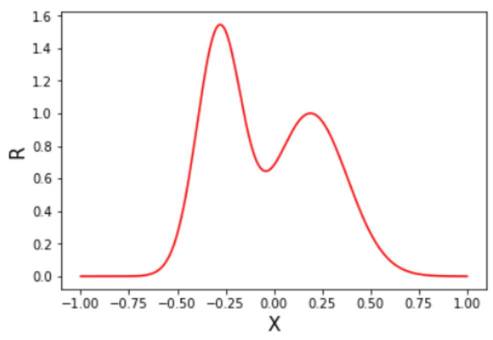


Figure 1: Sample code.

samples (checking the plot of the function, etc.). Allowed libraries: numpy, matplotlib; Deliverables: Your code as a separate file, your PDF report including the explanation of your method, your iteration number, sample number at each iteration and figures like figure 2 for five different environments (different p_1, p_2, p_3, p_4, p_5). Naming of your files should include your name and number.

Although there are many ways to solve this problem, there are 2 popular approaches in RL:

1. Gradient-based approaches: You sample N actions and their rewards. Then, you check how the changes (value increase or decrease) of your actions affect the obtained reward. Afterward, you change policy parameters a little so that you make good actions more likely and you will sample more rewarding actions at the next iteration. This became the most popular approach after the break-through of neural networks.

2. Evolutionary approaches: You sample n actions and their rewards. Then, you select the best m actions ($m < n$), disregard the others and fit your distribution to these m actions. At the next iteration, you will select actions from the new distribution. Generally, these algorithms can not handle high dimensional spaces, but there are recent promising studies.

I used the latter and my iterations are illustrated in figure 2.

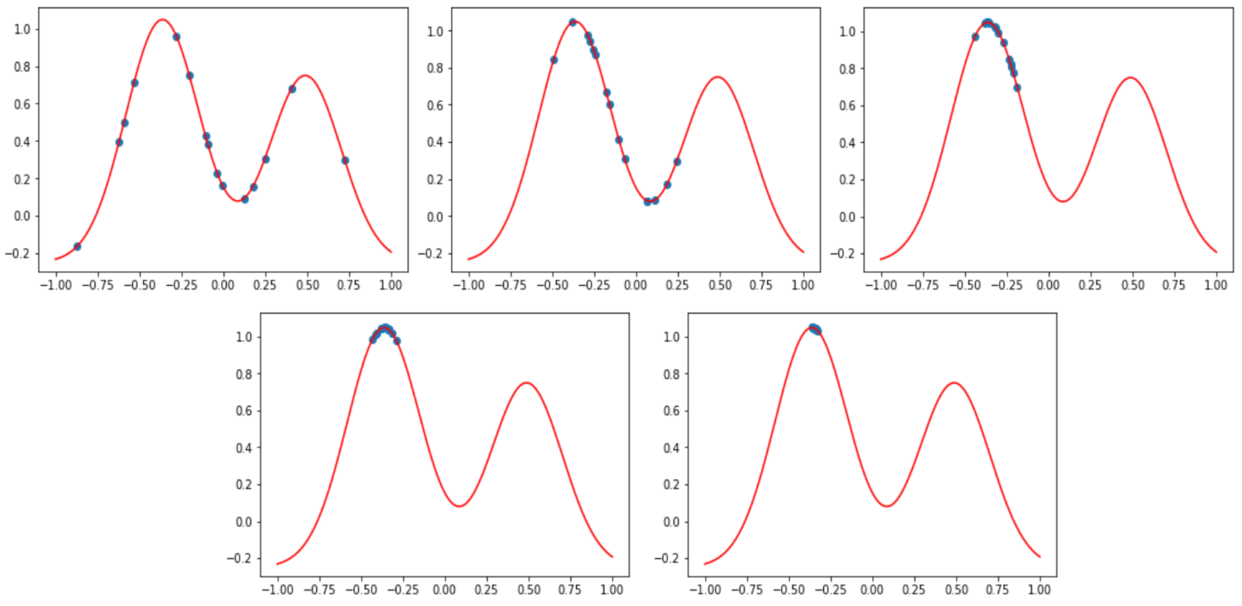


Figure 2: Sample solution.

Exploration - Exploitation dilemma:

Exploration: Try something new and see what you get. Exploitation: Continue with what you know and focus on getting high rewards. There is a trade-off between these two and finding the perfect balance is an ongoing research topic.

Exploration is critical in reinforcement learning because it is possible to get stuck in local optima if the agent tries to exploit with little experience. Check another testing in figure 3. **This will not be regarded as a solution, our aim is to take samples around the global maximum.** You can play with your distributions as you like to avoid the local maximum.

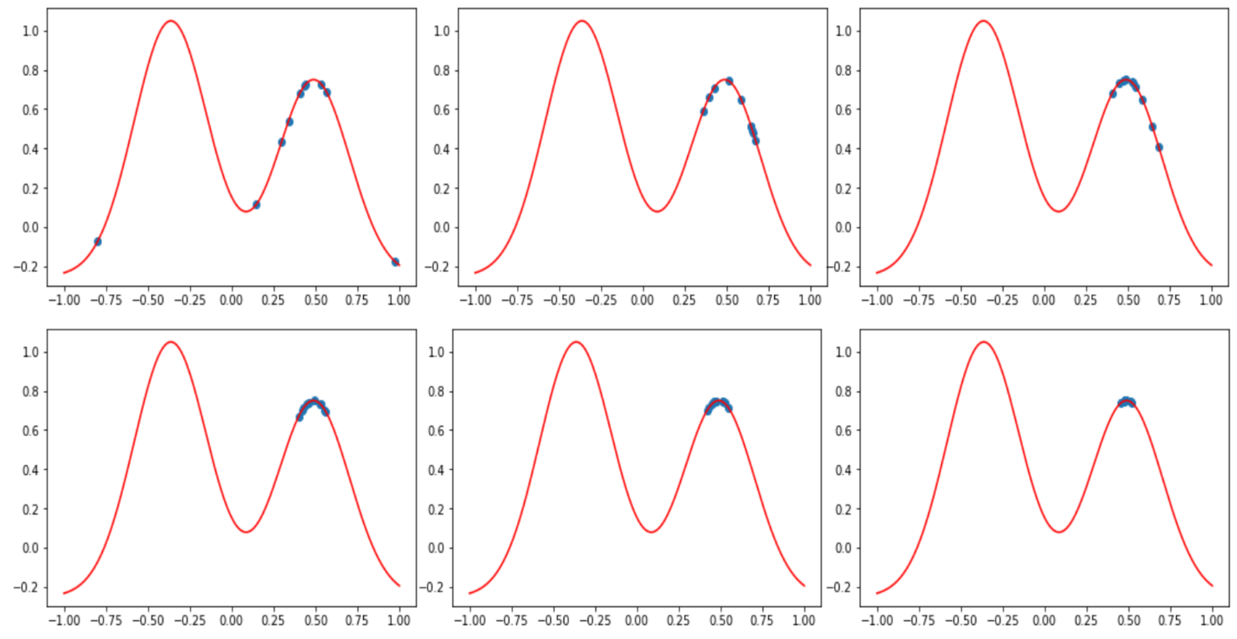


Figure 3: Sample tryout.