CSC 498: Assignment 2

Matthew Zhang, Claas Voelcker, Liquan Wang, Prof. Animesh Garg

Released: Fri 10/22/2021 – Due: Thur 11/04/2021

Name, First Name: ____________________

Student number: ____________________

Total points: 60 points

To complete the exercise, you can use the tex template provided on the website. Insert your answers into the solution space below each question. In case you are unfamiliar with Latex, you may also submit handwritten solutions, but make sure they are clean and legible.

Submit the exercise before 23:59 pm on the due date on quercus. To submit, please bundle your completed exercise sheet, your jupyter notebook and any material for the bonus task into one zip file. Name the zip file `studentnumber_lastname_firstname.zip` and upload it on quercus.

Each student will have 4 grace days throughout the semester for late assignment submissions. Late submissions that exceed those grace days will lose 33% of their value for every late day beyond the allotted grace days. Late submissions that exceed 3 days after the grace days have been used will unfortunately not be accepted. The official policy of the Registrar's Office at UTM regarding missed exams can be found here `https://www.utm.utoronto.ca/registrar/current-students/examinations`. If you have a compelling reason for missing the deadline, please contact the course staff as soon as possible to discuss hand in.

For assignment questions, please use Piazza and the office hours, but refrain from posting complete or partial solutions.

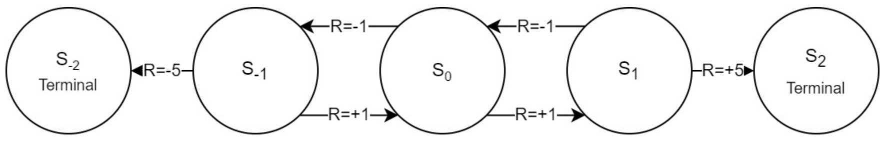

Figure 1: Example MDP

Time	**State**	**Action**	**Reward**
1	S_0	A_1	-1
2	S_{-1}	A_1	$+1$
3	S_0	A_1	$+1$
4	S_1	A_1	$+5$
5	S_2	N/A	N/A

Table 1: Observed sequence of states, rewards and actions

I Q-Learning

For the coding questions, provide your solutions by filling and uploading the jupyter notebook for the exercise. Make sure to only change code where we have marked the notebook with ??? and to only provide additional comments within the provided cells. Note that the environments are randomly generated; if you feel like your performance does not match with expectations, you can freely generate more environments for testing.

1. *Hand Derived Q-Learning* (3)

   Consider the same MDP shown in Figure 1, and the same data observed in Table 1. Perform the Q-learning algorithm with $\alpha = 1$, $\gamma = 1$, for the data observed in the table, if the initial values are $Q(s, a) = 0$ for all s, a.

   Solution:

2. Write down the square Bellman error (3). Denote which parameters are updated in DQN and which are frozen (1). Explain why this is called a "bootstrap target" (1)? [1] (5)

   Solution:

3. Explain the main issue of DQN that Double DQN aims to solve (2). How does Double DQN solve it (3)? (5)

   Solution:

[1] If you are unfamiliar with the saying, research the phrase "Pulling yourself up by your own bootstraps."

4. *Implement Q-Learning* (10)

   To start, download all necessary code from the webpage. Set up your Python environment and make sure you can run jupyter. You should be able to reuse the material from assignment 1.

   Run the first section of the jupyter notebook assignment2.ipynb (This requires you to run all cells within Assignment 2, Task 2).

   Task 2 contains scaffolding code for an (exact) Q-Learning agent. In the code cell provided in the notebook, please replace the sections marked ??? with your own code. Your agent should be able to find a solution that is roughly optimal. You are free to chose parameters such as the learning rate, number of episodes, if you find this helps performance.

   Solution:

5. *ϵ-Greedy Exploration* (7)

   Often in reinforcement learning, we want to encourage our agent to take non-optimal actions in order to obtain more information about the environment. This is referred to as "exploration". One scheme is ϵ-Greedy exploration, which chooses a random action ($a \sim Unif(A)$) with probability $1 - \epsilon$, and with probability ϵ chooses the action ($a = \arg\min Q(s, a)$) with the highest Q-value (ties can be broken arbitrarily). You are free to chose parameters such as the learning rate, number of episodes, if you find this helps performance.

   Add a parameter ϵ into your agent, which controls the level of ϵ-Greedy exploration. Run your algorithm for $\epsilon = 1.0, 0.9, 0.8, 0.5$, and report the performances (4). Write a small discussion about when we might prefer to use a higher ϵ (i.e. ϵ close to 1), and when might we prefer a lower ϵ (i.e. ϵ close to 0) (3)?

   Solution:

II TD-Learning

1. *Implement TD(0) Learning* (10)

   Run the second section of the jupyter notebook assignment2.ipynb.

   Task 3 contains scaffolding code for a simple TD(0) agent. In the code cell provided in the notebook, please replace the sections marked ??? with your own code. Your agent should be able to find a solution that is roughly optimal.

2. *TD(n) Learning* (10)

It is possible to do TD learning using not just the next state, but n observations into the future, with the following update rule:

$$\hat{V}_(s_t) \leftarrow \hat{V}(s_t) + \alpha \left(\gamma^{n+1}\hat{V}(s_{t+n}) - \hat{V}s_t + \sum_{i=0}^{n} \gamma^i r(s_{t+i}, a_{t+i}) \right) \tag{1}$$

Please introduce an additional parameter N_{steps} into your algorithm, which will implement this algorithm with $n = N_{steps}$. Then, run this algorithm for $n = 0, 1, 2, 5$ and report the performances (5). Write a small discussion about what the benefits/drawbacks of this approach might be (5).

III Bonus challenge

The assignment will include a bonus question. These are meant as additional challenges for highly motivated students and require either prior knowledge or some independent learning. You will be able to get full points in all exercises without these questions, but we strongly encourage you to at least try to complete them. The bonus points will improve your final exercise score in the grade calculations.

For each of the bonus questions, we will only provide minimal guidance and a high level task description. This means you are strongly encouraged to play around, think about different strategies and discuss your findings in your submission. Upload a description of your solution and relevant code alongside your submission.

1. *Mountain-Car Q-Learning* (20)

   In the bonus task, you will tackle a more complex problem using value iteration and policy iteration. You will be using the OpenAI gym environment "MountainCar-v0" (not to be confused with MontaincarContinuous-v0, which is similar but unsuitable for Q-learning).

   In the first step, you need to train a model of the environment using 50,000 samples. To obtain these, you should execute random actions in the environment and reset once the done signal is returned. You may use sklearn or torch for this, you do not need to implement your own ML model. The model should predict the next timestep and reward given the last observation.

   Next, you need to discretize the action space to use a Q-learning approach. You are free to use any strategies here, there are no bounds on your creativity (except your hardware limitations). We do suggest to start simple though. You can also implement approximate Q learning with function approximation.

   Finally, evaluate your agent using at least 16 independent runs of the original environment. Does the final reward align with the estimated value function of your agent? Are there failure cases and can you explain these? We expect the whole code to run in under 15 minutes.

To obtain full points, we expect clean code, a small written report containing a short discussion of your choice of discretization scheme or function approximation and a graph of reward over time steps showing the mean and standard deviation over all your runs. In addition, please add a small discussion of the final results. Please provide your code and all written parts together in form of a single jupyter notebook.

Solution: