CS486/686 Spring 2023 Assignment 4: Reinforcement Learning

Out: July 10, 2023 Due: July 21, 2023 (11.59 pm)

Submit an electronic copy of your assignment via LEARN. Late submissions incur a 2% penalty for every rounded up hour past the deadline. For example, an assignment submitted 5 hours and 15 min late will receive a penalty of ceiling(5.25) * 2% = 12%. Assignments submitted more than 50 hours late will not be marked.

This assignment is divided into three parts. The first part will be about Dynamic Programming methods, the second part will be about tabular Q-learning and the third part will be about Deep Q-learning.

1 Dynamic Programming Methods

You will program value iteration and policy iteration for Markov Decision processes in Python in the first part. More specifically, fill in the functions in the skeleton code of the file MDP.py. The file TestMDP.py contains the simple MDP example from Lecture 16 Slides 21-22. You can verify that your code compiles properly with TestMDP.py by running "python TestMDP.py". Add print statements to this file to verify that the output of each function makes sense.

Installs: Numpy (make sure to install using 'pip install numpy'). Python 3.6 is recommended, though the code should work with the later versions of python as well.

Submit a report containing the following:

1. Your Python code (worth 10%).
2. Test your code with the maze problem described in TestMDP.py.
   (a) Report the policy, value function and number of iterations needed by value iteration when using a tolerance of 0.01 and starting from a value function set to 0 for all states (worth 10%).
   (b) Report the policy, value function and number of iterations needed by policy iteration to find an optimal policy when starting from the policy that chooses action 0 in all states (worth 10%).

2 Tabular Q-learning

You will program the Q-learning algorithm in Python. More specifically, fill in the functions in the skeleton code of the file RL.py. This file requires the file MDP.py that you programmed for Part 1, so make sure to include it in the same directory. The file TestRL.py contains a simple RL problem to test your functions (i.e. the output of each function will be printed to the screen). You can verify that your code compiles properly by running "python TestRL.py".

Installs: Same as Part 1.
Submit a report containing the following:

1. Your Python code (worth 10%).

2. Test your code with the maze problem described in TestRL.py (same maze problem as in Part 1). Produce a graph where the x-axis indicates the episode (from 0 to 200) and the y-axis indicates the average (based on 100 trials) of the cumulative discounted rewards per episode (100 steps). The graph should contain 4 curves corresponding to the exploration probability epsilon=0.05, 0.1, 0.3 and 0.5. The initial state is 0 and the initial Q-function is 0 for all state-action pairs. Explain the impact of the exploration probability epsilon on the cumulative discounted rewards per episode earned during training as well as the resulting Q-values and policy (worth 20%).

3 Deep Q-learning

In this part, you will train a deep Q-network to solve the CartPole problem from Open AI Gym. This problem has a large state space that prevents the use of a tabular representation. Instead, you will use a neural network to represent the Q-function. Follow these steps to get started:

1. Get familiar with the CartPole problem. Read a brief description of the CartPole problem from Open AI Gym.

2. For this part of the assignment, you will use a PyTorch implementation of DQN. See the link for an explanation of different parts of the code.

3. Run the code in the file "main.py" to solve the CartPole problem with a Deep Q-Network.

Installs: Instructions available as comments in main.py file.
Submit a report containing the following:

1. Your python code along with the random seeds for all the experiments in this part.

2. Modify and run the code for CartPole DQN to produce a graph where the y-axis is the running average of 100 episodes for the cumulative rewards

obtained at each episode and the x-axis is the number of episodes up to a minimum of 20000 episodes. The graph should contain 4 curves corresponding to updating the target network every 1, 10 (default), 30, and 100 training step(s). To reduce stochasticity in the results, report curves that are the average of atleast 3 runs of the given code (with different random seeds). Based on the results, explain the impact of the target network and relate the target network to value iteration (worth 20%).

3. Modify and run the code for CartPole DQN to produce a graph where the y-axis is the running average of 100 episodes for the cumulative rewards obtained at each episode and the x-axis is the number of episodes up to a minimum of 20000 episodes. The graph should contain 4 curves corresponding to sampling mini-batches of 1, 16 (default), 30, and 200 experience(s) from the replay buffer. To reduce stochasticity in the results, report curves that are the average of atleast 3 runs of the given code (with different random seeds). Based on the results, explain the impact of the replay buffer and explain the difference between using the replay buffer and exact gradient descent (worth 20%).