☰

CS885 Spring 2018 - Reinforcement Learning

There will be three assignments, each worth 10% of the final grade. Assignments are done individually (i.e., no team). Each assignment will have a programming part to be done in Python. Some assignments will make use of TensorFlow and OpenAI Gym. For GPU acceleration, feel free to use Google's Colaboratory environment. This is a free cloud service where you can run Python code (including TensorFlow, which is pre-installed) with GPU acceleration. A virtual machine with two CPUs and one Nvidia K80 GPU will run up to 12 hours after which it must be restarted. The following steps are recommended:

- Create a Python notebook in Google Colab
- Click on "edit", then "notebook settings" and select "None" (CPU) or "GPU" for hardware acceleration.

The approximate out and due dates are:

- A1: out May 23, due June 5 (11:59 pm)
- A2:
- A3:

On the due date of an assignment, the work done to date should be submitted electronically on the LEARN website; further material may be submitted with 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.

Assignment 1: out May 23, **due June 5 (11:59 pm)**

This assignment has three parts.

Part I

In the first part, you will program value iteration, policy iteration and modified policy iteration for Markov decision processes in Python. More specifically, fill in the functions in the skeleton code of the file MDP.py. The file TestMDP.py contains the simple MDP example that we saw Lecture 2a Slides 13-14. 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.

- Skeleton code: MDP.py
- Simple MDP from Lecture 2a Slides 13-14 to test your code: TestMDP.py

Submit the following material via LEARN:

- Your Python code.
- Test your code with the maze problem described in TestMDPmaze.py.
    - 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.
    - 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.
    - Report the number of iterations needed by modified policy iteration to converge when varying the number of iterations in partial policy evaluation from 1 to 10. Use a tolerance of 0.01, start with the policy that chooses action 0 in all states and start with the value function that assigns 0 to all states. Discuss the impact of the number of iterations in partial policy evaluation on the results and relate the results to value iteration and policy iteration.

Part II

In the second part, 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 I 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".

- Skeleton code: RL.py (requires MDP.py from part I)
- Simple RL problem to test your code: TestRL.py

Submit the following material via LEARN:

- Your Python code.
- Test your code with the maze problem described in TestRLmaze.py (same maze problem as in Part I). 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 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 rewards per episode earned during training as well as the resulting Q-values and policy.

Part III

In the third part, you will train a deep Q-network in OpenAI Gym to solve problems with continuous states that prevent the use of a tabular representation. Instead, you will use a neural network to represent the Q-function. Follow these steps to get started:

- Go to gym.openai.com and install the Gym package by following the instructions in the documentation.

- Follow the instructions in the documentation to run a simple agent that executes actions at random in the CartPole environment.
- Replace the default random agent by a Deep Q Network (DQN) agent. More precisely, clone or download the OpenAI baseline algorithms from the following GitHub repository: github.com/openai/baselines. Train a cartPole agent by running the file baselines/baselines/deepq/experiments/custom_cartpole.py.
- If you would like to run the code with GPU acceleration, use Google Colab: colab.research.google.com. This is a free cloud service where you can run Python code (including OpenAI gym and TensorFlow) with GPU acceleration. A virtual machine with two CPUs and one Nvidia K80 GPU will run up to 12 hours after which it must be restarted. To use Google Colab, create a Python notebook. For GPU acceleration, clid on edit, then notebook settings and select GPU.

Submit the following material via LEARN:

- Modify and run the cartPole agent described in baselines/baselines/deepq/experiments/custom_cartpole.py to produce a graph where the y-axis is the cumulative reward of each episode and the x-axis is the # of episodes up to a maximum of 1000 episodes. The graph should contain 4 curves corresponding to updating the target network every 1000 (default), 250, 50, 1 step(s). Based on the results, explain the impact of the target network and relate the target network to value iteration.
- Modify and run the cartPole agent described in baselines/baselines/deepq/experiments/custom_cartpole.py to produce a graph where the y-axis is the cumulative reward of each episode and the x-axis is the # of episodes up to a maximum of 1000 episodes. The graph should contain 4 curves corresponding to sampling mini-batches of 32 (default), 15, 5 and 1 experience(s) from the replay buffer. Based on the results, explain the impact of the replay buffer and relate the replay buffer to exact gradient descent.