Homework #1

CSC2547 – Introduction to Reinforcement Learning
(Spring 2021)

- **Deadline:** Monday, March 8, 2021 at **16:59**.
- **Submission:** You need to submit two files through MarkUs. One is a PDF file including all your answers and plots. The other is a source file that reproduces your answers. You can produce the file however you like (e.g. LaTeX, Microsoft Word, etc) as long as it is readable. Points will be deducted if we have a hard time reading your solutions or understanding the structure of your code. If the code does not run, you may lose most/all of your points for that question.
- **Late Submission:** 10% of the marks will be deducted for each day late, up to a maximum of 3 days. After that, no submissions will be accepted.
- **Collaboration:** Homework assignments must be done individually, and you cannot collaborate with others.

1 Derivations and Conceptual Questions [30 pts]

These questions, except the last one, are all exercises in the lecture notes, and should be easy to solve. Their main purpose is for you to review the material.

(a) **[5pt]** Write down the Bellman equation for V^π for a deterministic dynamical system $x_{t+1} = f(x_t, a_t)$ (see Example 1.2) and deterministic policy $\pi : \mathcal{X} \to \mathcal{A}$.

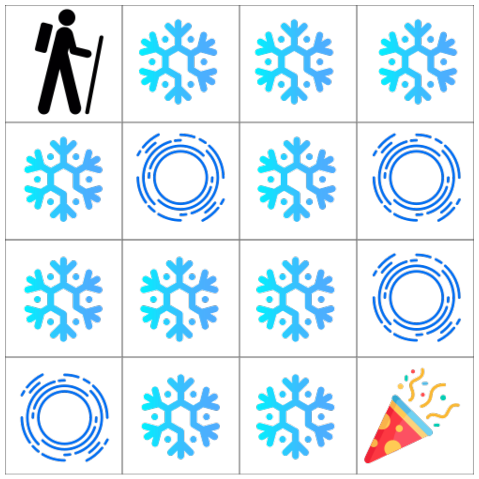
(b) **[5pt]** Prove that the Bellman operators $T^\pi : \mathcal{B}(\mathcal{X} \times \mathcal{A}) \to \mathcal{B}(\mathcal{X} \times \mathcal{A})$ applied on Q satisfy the monotonicity property.

(c) **[5pt]** Prove that for any $V \in \mathcal{B}(\mathcal{X})$ and any $\pi \in \Pi$, we have

$$\|V - V^\pi\|_\infty \leq \frac{\|V - T^\pi V\|_\infty}{1-\gamma}.$$

(d) **[5pt]** Consider a finite state-action MDP. Write down VI for the computation of V^π and Q^π (in explicit form, including the summation notation).

(e) **[10pt]** Describe any real-world application that can be formulated as an MDP. We encourage you to choose a problem close to your application domain (if your research is related to an application), but feel free to pick any other problem too. Describe what the state space, action space, transition model, and reward are. You do not need to be precise in the description of the transition model and reward (no formula is needed). Qualitative description is enough.



Winter is here.[1] You and your friends were tossing around a frisbee at the park when you made a wild throw that left the frisbee out in the middle of the lake. The water is mostly frozen, but there are a few holes where the ice has melted. If you step into one of those holes, you'll fall into the freezing water. At this time, there's an international frisbee shortage, so it's absolutely imperative that you navigate across the lake and retrieve the disc. However, the ice is slippery, so you won't always move in the direction you intend. You receive a reward of 1 if you reach the goal, and zero otherwise.

We will solve this problem by formulating it as a discounted MDP, with $\gamma = 0.9$. The provided code notebook has some starter code for familiarizing yourself with the environment and the OpenAI Gym interface. You can run it on Google Colab: `https://colab.research.google.com/` or on your own computer. You will probably need to install Gym to run locally. This should be relatively straightforward and can be done by following instructions at `https://github.com/openai/gym`.

2 Introductory Questions [10pts]

(a) **[5pt]** Take a look at the source code. What is the episode length of the Frozen-Lake environment (if it exists)?

(b) **[5pt]** Let's run some simulations. Executing a random policy is a good way to familiarize yourself with a new Gym environment. Compute the average discounted return and episode length of a random policy, over 100 episodes. Is this what you expect?

[1]Environment details and description are from: `https://github.com/openai/gym/blob/master/gym/envs/toy_text/frozen_lake.py`

3 Value Iteration [20pts]

We will implement value iteration in this exercise. There are some suggestions on how to do this in the notebook.

(a) **[6pt]** Make a plot of the value estimate $V^{\pi_k}(x_0)$ of the initial state versus iterations k.

(b) **[6pt]** What is the final policy that value iteration converges to? We have provided a helper function to print the policy.

(c) **[8pt]** In lecture, we saw a synchronous version of value iteration, where the value function for all the states is updated simultaneously.
Suppose that we instead make asynchronous updates in our updates to the value function, meaning that we use a single array (list) to store our values and update them live. What might be the advantage of using asynchronous updates? Does the order in which you update the states in policy evaluation matter i.e. affect how quickly the algorithm converges? Empirically verify your answer.

4 Policy Iteration [20pts]

We will implement policy iteration in this exercise.

(a) **[7pt]** Randomly generate a few policies by generating a random action for each state. Run your policy evaluation code and report your results.

(b) **[7pt]** Next, implement the policy improvement step. Once you have code to do policy evaluation and improvement, you can implement policy iteration by iteratively calling the two functions. Run the code with two different initializations: 1) a policy that takes the action "up" at every step and 2) a policy that that takes the action "right" at every step. Again, make a plot of the value estimate $V^{\pi_k}(x_0)$ at the initial state versus iterations k.

(c) **[2pt]** Are the policies found by value iteration and policy iteration the same?

(d) **[2pt]** We will focus on the initial policy that takes the action "right" at every step. Plot the value function V^{π_k} on the 4x4 grid at $k = 0$, $k = 3$, and when the algorithm converges.

(e) **[2pt]** Repeat part (d) with a discount factor $\gamma = 0.5$ and $\gamma = 0.99$. Are the results what you expect?

5 Q-Learning [20pts]

For this last question, we will use a reinforcement learning algorithm and assume no knowledge of the underlying transition and reward model. We will be using the Q-Learning algorithm as described at the end of Lecture 1.

For your implementation of Q-learning, we will perform action selection with the ϵ-greedy policy. If multiple actions have the same value, you should select one randomly (note that np.argmax selects the first action).

(a) **[7pt]** Visualize the value function after 100 episodes and 1000 episodes. You are free to choose your hyperparameters for the learning rate α and ϵ here.

(b) **[7pt]** Plot the performance of the greedy policy according to Q_k until convergence (you should use your policy evaluation code from the previous question).

(c) **[6pt]** Manipulate one other hyperparameter and report your results. For instance, you might want to change the initial value of ϵ or α. You could also implement a decay on the learning rate or ϵ (so that the policy is progressively more greedy over time).