COMP-579: Reinforcement Learning - Assignment 2

Posted Wednesday January 28, 2026
Due Wednesday, February 11, 2026

The assignment can be undertaken individually or in teams of two. Alongside this document, you have been provided with a partially completed .IPYNB (Jupyter Notebook) file. Your task is to complete the .IPYNB file by adding the necessary code, displaying results, and providing explanations for questions. **For the theory part, all written answers must be typed (computer-written); handwritten submissions will not be accepted.**

Following the notation and interfaces of the predefined classes and functions is mandatory; however, you may adapt variable names and intermediate notations in your own code to better align with your thought process and coding style. For coding questions, ensure that you only submit the completed. IPYNB file for evaluation.

You must submit two separate files: one PDF containing all theory answers, and one completed `.ipynb` file containing all code, results, and code-related explanations. **You must also include an author contribution statement that explains the contributions of the various team members in detail. Additionally, you are required to include a detailed statement on LLM usage for every aspect of usage in the assignment.**

If you have any questions or require clarification, feel free to reach out for assistance on the Ed. We will try to answer your questions within 24 hours. Good luck with your assignment!

Theoretical questions[30 points]

1. **[5 points] Question 1**

   Consider a discounted Markov Decision Process (MDP) where the rewards are bounded:

   $$|R(s,a)| \leq R_{\max} \quad \text{for all states } s \in S \text{ and actions } a \in A.$$

   The value of state s for the policy π is defined as:

   $$v_\pi(s) = \mathbb{E}_\pi \left[\sum_{k=t}^{\infty} \gamma^{k-t} R_{k+1} \;\middle|\; S_t = s \right].$$

   What is the maximum value of $v_\pi(s)$ across all states and policies? Provide an upper bound on $|v_\pi(s)|$ in terms of $R_{\max}$ and γ.

2. **[10 points] Question 2**

   Define the λ-return G_t^λ and the corresponding Bellman operator $\mathcal{T}^\lambda$. Prove that $\mathcal{T}^\lambda$ is a contraction mapping in the L_∞ norm. In other words, show that:

   $$\|\mathcal{T}^\lambda v - \mathcal{T}^\lambda u\|_\infty \leq \eta \|v - u\|_\infty$$

   for some $\eta < 1$. Briefly explain the rate of contraction as a function of λ and γ.

3. **[15 points] Question 3**

   (a) Explain the bias–variance trade-off between first-visit Monte Carlo and every-visit Monte Carlo algorithms. What happens to the bias asymptotically?

   (b) Discuss bias-variance trade-off between first-visit Monte Carlo and TD(0) learning algorithms.

   (c) Consider state aggregation for value function approximation as discussed in the class. Explain the bias-variance trade-off as a function of the number of aggregates to estimate the value function. How does the quality of estimates affect when you just have a handful of returns to approximate the value function.

Coding Questions [70 points]

1. **[30 Points] Dynamic Programming Policy Evaluation**

   We will use Gymnasium, which is a commonly-used python library in Reinforcement Learning. You can check the Gymnasium Documentation to familiarize yourself with the library.

   In this part, you will evaluate a fixed stochastic policy in the gymnasium pre-built Frozen Lake environment using Dynamic Programming (DP). Frozen Lake is a finite tabular environment. The goal is to go from the starting point to the goal without falling in the holes. The environment is stochastic due to the slippery ice. This means that, given a certain action, there is a probability that the agent may move perpendicular to the intended direction. Take some time to go over the Frozen Lake documentation to familiarize yourself with it before you proceed.

   For this question, use a value of γ = 0.9 unless otherwise specified

   (a) **[4 points] Implement the fixed policy:** You will evaluate a fixed stochastic policy $\pi(a \mid s)$ defined as follows:

   - For all non-terminal states (start tile and frozen tiles), the policy prefers the action "Move right".
     - If "Move right" is a valid action in the current state, the policy selects it with probability 0.7. The remaining probability is distributed uniformly among the other valid actions in that state.
     - If "Move right" is not a valid action, the policy selects uniformly at random from all valid actions.

     To determine the set of actions that are valid at a particular state, the function `get_valid_actions` has been provided for you.
   - Complete the `get_terminal_states` function to identify all terminal states based on the transition model. Normally, when calling `env.step`, the `done` flag indicates whether the state we stepped into is terminal. For this implementation, we will only

use the transition model `P`. Assume that at terminal states, the `done` flag is `True` for all actions in that state. You can also verify your terminal states by visualizing the grid using `env.unwrapped.desc`.

- Once terminal states are identified, implement the policy logic in the `right_favoring_policy` method according to the specifications above.

(b) **[4 points] Solving for v_π analytically** Implement the function `policy_evaluation_analytical` to derive the value function v_π for the policy by solving the Bellman equations using matrix inversion.

(c) **[8 points] Synchronous Dynamic Programming Policy Evaluation.** Implement a synchronous Dynamic Programming (DP) algorithm to compute the value function v_π for the given policy.

Convergence is reached when

$$\|V_{k+1} - V_k\|_\infty < 10^{-8}.$$

At each iteration k, record:

- the *Bellman residual* $\|V_{k+1} - V_k\|_\infty$,

Generate a plot showing the evolution of the Bellman residual over iterations in order to visualize the convergence behavior.

(d) **[4 points] Value Function Visualization** Upon convergence, visualize the computed state-value function V_π as a heatmap.

- Map the 1D value vector back to the grid dimensions.
- Use `env.unwrapped.desc` to overlay the environment tile types (S, F, H, G) onto the corresponding heatmap cells.
- Annotate each cell with both the numerical value and the tile type.

(e) **[10 points] Non-Stationary Scenario**

We now define a uniform random policy as follows:

$$\pi(a \mid s) = \begin{cases} \dfrac{1}{|\mathcal{A}(s)|}, & \text{if } a \in \mathcal{A}(s) \\ 0, & \text{otherwise or if } s \text{ is terminal} \end{cases}$$

where $\mathcal{A}(s) \subseteq \mathcal{A}$ is the set of *valid actions* from state s.

That is, the policy selects uniformly at random among all valid actions in the current state, and assigns zero probability to invalid actions or when the state is terminal.

- Using the above definition, implement the uniform random policy.
- Modify the evaluation process to simulate a non-stationary case by changing the policy π at fixed intervals ($\pi_0 \to \pi_1 \to \pi_0 \to \pi_1 \cdots$). Start with the "right favoring policy" and switch to the "uniform random policy" every 4 iterations. Run the process for 40 iterations. Plot how the Bellman residual changes over iterations.

2. **2D Continuous Random Walk [30 points]**

In this question you will be estimating the value function of the policy using two method: (1) linear approximation using tile coding to generate features; (b) directly estimating the value using neural network as the function approximator. For tile coding, use the "tile coding method" provided to you in the starter code to convert the continuous states into a binary feature vector. Use 8 tilings with each tiling comprising of 16 tiles in total (4×4), and an index hash table of size 4096.

For all the sub-questions below, **report the mean over 30 runs along with confidence intervals computed using the standard deviation across runs.** For evaluation **the estimated value function is compared against the true value function using the Mean Squared Value Error (MSVE)** over a grid of 21×21 evenly spaced states in the interval $[0, 1] \times [0, 1]$ (increments of 0.05 in both x and y). **You will implement this inside the `compute_msve` method.**

Task description: In this task, the agent performs a random walk in a two-dimensional continuous state space, and its goal is to estimate the value function for its policy. Continuous random walk has a continuous state space between $[0, 1] \times [0, 1]$, and a continuous action space in $[-0.2, 0.2] \times [-0.2, 0.2]$. At each time step, the agent selects a random action according to uniform probability from its action space. As a result of its action, the next state is updated as:

$$S_{t+1} = S_t + A_t,$$

where $A_t = (\Delta x_t, \Delta y_t)$ is the action chosen by the agent.

The task is episodic: the episode terminates when either coordinates of the state falls outside the interval $[0, 1] \times [0, 1]$. At the beginning of each episode, the agent is at the center of the state space, $s_0 = (0.5, 0.5)$.

The reward, R_t, is 0 everywhere except for the terminal state. When the task terminates, the reward is equal to the sum of the coordinates of the terminal state. For example, if agent terminates at $(-0.05, 0.7)$, the terminal reward is $-0.05 + 0.7 = 0.65$. We set the discount rate, γ, to 1.0.

Questions:

(a) **[5 points] True value function:** In order to measure the effectiveness of various algorithms you will implement in the subsequent questions, you will first show that the true value of a state (x, y) is $V(x, y) = x + y$. **The estimates will be compared against this true value, so make sure you get this question right as all subsequent questions make use of it**. *Hint: Use properties of expectation and the symmetry of the random walk to get the desired result.*

(b) **[7 points] Monte Carlo with tile coding:** Implement every-visit Monte Carlo (MC) prediction algorithm in `MCAgentTileCoding` class, to estimate the value function. Initialize all weights to zero.

- For each episode, compute the return from each visited state and update the value function estimate accordingly, by completing `run_mc_experiment` in the starter code.
- Train your algorithm for 1000 episodes on various learning rate values from {0.1, 0.03, 0.01, 0.003, 0.001}.

- Plot the MSVE for various learning rates, which is computed using the `compute_msve` method, as a function of episodes.

(c) **[10 points] TD(λ) with Accumulating Traces and tile coding:** Implement TD(λ) with accumulating eligibility traces in `TDAgentLambda` class, with weights initialized to zero.

- Complete function `run_td_lambda_experiment` to run experiments across different values of the trace parameter (λ) and learning rate (α), where $\lambda \in \{0, 0.25, 0.5, 0.75, 0.9\}$ and $\alpha \in \{0.001, 0.003, 0.01, 0.03, 0.08\}$.
- Use the same MSVE function that you implemented in the previous question, and plot MSVE as a function of the learning rate α, **similar to Figure 12.6 in Sutton and Barto**. Also, include a separate MSVE curve as a function of the trace parameter λ. Plot a heatmap of mean MSVE over training for the (λ, α) pairs and find the best combination of λ and α.
- Explain how the choice of (λ) interpolates between Monte Carlo and TD methods.
- Compare Monte Carlo and TD(λ) in terms of convergence speed and stability.

(d) **[8 points] TD(0) with Neural Networks:** Train a fully connected feedforward two-layer neural network using TD(0) learning to estimate the value function. The network should take the continuous state as input and output the value estimates for that state.

- Train your algorithm for 1000 episodes on various learning rate values from {0.03, 0.01, 0.003, 0.001}.
- Plot the MSVE for various learning rates, which is computed using the `compute_msve` method, as a function of episodes.
- Compare TD(0) trained with tile coding with TD(0) trained using a neural network.

3. **[10 points] Environment Design**: Using the Linear Chain environment provided in the starter code and your implementations for MC and TD(0) (from Q2), design two versions of the environment:

   (a) One where the mean MSVE over episodes for TD(0) is higher than that for MC

   (b) Another where the mean MSVE over episodes for MC is higher than that for TD(0)

   You can only vary the `reward_noise` argument. Use a learning rate = 0.003 and episode number of 10,000. Explain your choice of parameter value and support it by plotting the MSVE over episodes for both TD(0) and MC on the newly designed environment.