National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Faculty of Computing

CS 368: Reinforcement Learning

Lab 05: Model Free Reinforcement Learning

(Monte Carlo and Temporal Difference Learning)

Date: 24 October 2024

Time: 02:00 PM – 5:00 PM

Instructor: Dr. Zuhair Zafar

Lab 05: Model Free Reinforcement Learning

Objectives

Given a simulated environment, evaluate the given policy using Monte Carlo and Temporal Difference Learning algorithms.

Tools/Software Requirement:

Google Colab, Python, Gymnasium Library

Introduction

Model-free reinforcement learning figures out the best way to act without needing a model of the environment. It's like learning to play a game by trying over and over, rather than reading the rule book. Over time, it learns what actions give good results through trial and error. It works well in situations where we can't predict what will happen next or don't know all the rules.

Understanding key concepts is essential in grasping model-free reinforcement learning. Let's dive into the main ideas that shape this approach.

- **Learning from Interaction**: Agents learn from the choices they make and their outcomes without any map of the environment. They try different actions and remember which ones lead to better rewards.
- **Trial and Error**: This method is all about making mistakes and then learning from them, kind of like learning to ride a bike.
- **Rewards are Key**: In model-free reinforcement learning, getting rewards is what it's all about. The agent wants to get as many rewards as possible because that's how it knows it's doing well.
- **No Need for a Model**: Unlike other methods, you don't need to build a fancy world model first. Agents figure things out as they go, which saves time at the start.

Gymnasium Library

Gymnasium is an open-source Python library for developing and comparing reinforcement learning algorithms by providing a standard API to communicate between learning algorithms and environments, as well as a standard set of environments compliant with that API.

Environments

Gymnasium includes the following families of environments along with a wide variety of third-party environments

- **Classic Control** - These are classic reinforcement learning based on real-world problems and physics.
- **Box2D** - These environments all involve toy games based around physics control, using box2d based physics and PyGame-based rendering

- **Toy Text** - These environments are designed to be extremely simple, with small discrete state and action spaces, and hence easy to learn. As a result, they are suitable for debugging implementations of reinforcement learning algorithms.
- **MuJoCo** - A physics engine-based environments with multi-joint control which are more complex than the Box2D environments.
- **Atari** - Emulator of Atari 2600 ROMs simulated that have a high range of complexity for agents to learn.

Standalone Installation on Windows PC

Step: 1 Install Anaconda (How to Install on windows)

Step: 2 Open Anaconda Powershell Prompt (by running as administrator)

Step: 3 Create new conda environment for gymnasium library using the following command

```
conda create -n gym-env
```

Step 4: Activate the gym-env using the following command

```
conda activate gym-env
```

Step 5: Now install python. The Gymnasium library is supported and tested for Python 3.8, 3.9, 3.10, 3.11 and 3.12.

```
conda install python=3.12
```

Step 6: To install the base Gymnasium library, use `pip install gymnasium` but this does not include dependencies for all families of environments (there's a massive number, and some can be problematic to install on certain systems). Use the following command to install all dependencies.

```
pip install "gymnasium[all]"
```

This will throw an error message "ERROR: Could not build wheels for box2d-py, …". This is because it requires the swig library. You first need to install swig using the following command.

```
conda install swig
```

Or you can also download and install swig by visiting the swig download page.

Step 7: Run the following command again

```
pip install "gymnasium[all]"
```

It may throw an error again with the error message "error: Microsoft Visual C++ 14.0 or greater is required…". You can download and install the Visual Studio Build Tools by visiting the Microsoft website. While installing, select an option of Desktop development with C++.

Step 8: Run the following command again. This time it should work.

```
pip install "gymnasium[all]"
```

The Gymnasium Library is installed.

Gymnasium Documentation: https://gymnasium.farama.org/content/basic_usage/#
Gymnasium Library (Github): https://github.com/Farama-Foundation/Gymnasium
Gymnasium Library (Installation Video): https://www.youtube.com/watch?v=gMgj4pSHLww

Monte Carlo Algorithm for Prediction

What is meant by Monte Carlo?

The term "Monte Carlo" is often used more broadly for any estimation method whose operation involves a significant random component. In reinforcement learning, Monte Carlo methods require only experience — sample sequences of states, actions, and rewards from actual or simulated interaction with an environment. Learning from actual experience is striking because it requires no prior knowledge of the environment's dynamics, yet one can still attain optimal behavior.

It is a method for estimating Value-action (Value|State, Action) or Value function (Value|State) using some sample runs from the environment for which we are estimating the Value function. The Monte Carlo method for reinforcement learning learns directly from episodes of experience without any prior knowledge of MDP transitions. Here, the random component is the return or reward.

One caveat is that it can only be applied to episodic MDPs. It's fair to ask why, at this point. The reason is that the episode has to terminate before we can calculate any returns. Here, we don't do an update after every action, but rather after every episode. It uses the simplest idea – the value is the mean return of all sample trajectories for each state. Similar to dynamic programming, there is a policy evaluation (finding the value function for a given random policy).

Monte Carlo Policy Evaluation

The goal here, again, is to learn the value function v_π(s) from episodes of experience under a policy pi. Recall that the return is the total discounted reward:

$$S_1, A_1, R_1, \ldots . S_t \sim \pi$$

Also recall that the value function is the expected return:

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s]$$

We know that we can estimate any expected value simply by adding up samples and dividing by the total number of samples:

$$\bar{V}_\pi(s) = \frac{1}{N} \sum_{i=1}^{N} G_{i,s}$$

- i – Episode index
- s – Index of state

The question is how do we get these sample returns? For that, we need to play a bunch of episodes and generate them. For every episode we play, we'll have a sequence of states and rewards. And from these rewards, we can calculate the return by definition, which is just the sum of all future rewards.

First Visit Monte Carlo

Average returns only for first time s is visited in an episode.

Here's a step-by-step view of how the algorithm works:

1. Initialize the policy, state-value function
2. Start by generating an episode according to the current policy
    1. Keep track of the states encountered through that episode
3. Select a state in 2.1
    1. Add to a list the return received after first occurrence of this state
    2. Average over all returns
    3. Set the value of the state as that computed average
4. Repeat step 3
5. Repeat 2-4 until satisfied

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated
Initialize:
    $V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$
    $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):
    Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:
            Append G to $Returns(S_t)$
            $V(S_t) \leftarrow$ average$(Returns(S_t))$

Every visit Monte Carlo

Average returns for every time 's' is visited in an episode.

Here's a step-by-step view of how the algorithm works:

1. Initialize the policy, state-value function
2. Start by generating an episode according to the current policy
    1. Keep track of the states encountered through that episode
3. Select a state in 2.1
    1. Add to a list the return received after every occurrence of this state

2. Average over all returns
3. Set the value of the state as that computed average

4. Repeat step 3
5. Repeat 2-4 until satisfied

Incremental Mean

It is convenient to convert the mean return into an incremental update so that the mean can be updated with each episode, and we can understand the progress made with each episode. We already learnt this when solving the multi-armed bandit problem.

We update v(s) incrementally after episodes. For each state S_t, with return G_t:

$$N(S_t) \leftarrow N(S_t) + 1$$
$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}(G_t - V(S_t))$$

In non-stationary problems, it can be useful to track a running mean, i.e., forget old episodes:

$$V(S_t) \leftarrow V(S_t) + \alpha\,(G_t - V(S_t))$$

Temporal Difference Learning for Prediction

Temporal Difference Learning (TD Learning) is one of the central ideas in reinforcement learning, as it lies between Monte Carlo methods and Dynamic Programming in a spectrum of different Reinforcement Learning methods.

TD (0)

As we saw for Monte Carlo methods, Prediction refers to the problem of estimating the values of states, a value of a state is an indication of how good is that state for an agent in the given environment, the higher the value of the state the better it is to be in that state.

Monte Carlo and Temporal Difference Learning are similar in the sense that they both use real-world experience to evaluate a given policy, however, Monte Carlo methods wait until the return following the visit is known which is after the episode ends is available to update the value of the state, whereas TD methods update the state value in the next time step, at the next time step t+1 they immediately form a target and make a useful update using the observed reward.

How it works

TD Learning operates by taking actions according to some policy, observing the reward and the next state, and then updating the value of the current state based on the observed reward and the estimated value of the next state. The update is done using the formula:

$$V(S_t) = V(S_t) + \alpha * [R_{t+1} + \gamma * V(S_{t+1}) - V(S_t)]$$

where:

- $V(S_t)$ is the current estimate of the state's value
- α is the learning rate
- R_{t+1} is the reward observed after taking the action
- γ is the discount factor
- $V(S_{t+1})$ is the estimated value of the next state

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    $A \leftarrow$ action given by π for S
    Take action A, observe R, S'
    $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
    $S \leftarrow S'$
  until S is terminal

Lab Tasks

- Learning rate, alpha, α = 0.05
- Discount factor, gamma, γ = 0.95
- Number of Episodes for Cliff walking = 500000 and Frozen Lake = 500000

1. **You are initially provided with the following deterministic policy in Cliff Walking environment. Implement the Monte Carlo First Visit algorithm to determine the values of each state for the given policy. Report the final converged values of each state.**

```
Policy (UP = 0, RIGHT = 1, DOWN = 2, LEFT = 3, N/A = -1):
[[ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  2.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  2.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  2.]
 [ 0. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.]]
```

2. **For the given policy in Task 1 for the cliff walking environment, implement the Monte Carlo Every Visit algorithm to evaluate the given policy. Report the final converged values of each state.**

3. **Apply the Temporal Difference Learning, TD (0) algorithm to determine the values of each state for the given policy in Task 1 for the Cliff walking environment. Report the final converged values of each state.**

4. **You are initially provided with the following deterministic policy in Frozen Lake (4x4) environment. Implement the Monte Carlo First Visit algorithm to determine the values of each state for the given policy. Report the final converged values of each state.**

```
Policy (UP = 3, RIGHT = 2, DOWN = 1, LEFT = 0, N/A = -1):
[[ 1  1  1  0]
 [ 1 -1  1 -1]
 [ 1  1  1 -1]
 [-1  2  2 -1]]
```

Use the following command to initialize the frozen lake environment.

```
env=gym.make('FrozenLake-v1', desc=None, map_name="4x4", is_slippery=True)
```

5. **For the given policy in Task 4 for the Frozen Lake environment, implement the Monte Carlo Every Visit algorithm to evaluate the given policy. Report the final converged values of each state.**

6. **Implement the TD (0) algorithm to determine the values of each state for the given policy in Task 4 for the Frozen Lake environment. Report the final converged values of each state.**

Deliverable:

Please submit your notebook on LMS before the deadline **(28th October 2024, 11:59pm).**

Lab Rubrics

Assessment	Does not meet expectation (1/2 marks)	Meets expectation (3/4 marks)	Exceeds expectation (5 marks)
Software Problem Realization **(CLO1, PLO1)**	The student struggles to formulate the problem as RL and does not apply RL prediction algorithms to solve it. There is a lack of understanding of the problem's requirements and no attempt to evaluate the given policy effectively.	The student formulates the problem as RL with some guidance, applies Monte Carlo and TD (0) algorithms with hints, and shows it's working. However, the approach might not be fully optimized or lacks a thorough justification.	The student independently formulates the given problem as RL, applies Monte Carlo and TD (0) algorithms without guidance, and effectively evaluates the given policy. The approach is fully optimized and can be applied to different similar problems.
Software Tool Usage **(CLO4, PLO5)**	Code has syntax errors, and the implementation of the Monte Carlo / TD (0) algorithm is incorrect or incomplete. The code is not modular and lacks comments for readability and reuse. The student shows limited ability to use gymnasium library functions where required.	The student has implemented the Monte Carlo and TD (0) algorithms correctly for the given problem with minor mistakes. The code is mostly correct in terms of syntax and functionality but might not be optimized or well-structured for reuse. Some documentation is provided. The student also shows working knowledge of gymnasium library where required.	The student has implemented the Monte Carlo and TD (0) algorithms efficiently and correctly. The code is clean, modular, well-documented, and follows best practices for reproducibility and reuse. The student demonstrates full command of the gymnasium library and its functions.