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

**Faculty of Computing**

# CS 368: Reinforcement Learning

## Lab 06: Model Free Reinforcement Learning - Prediction

## (n-Step TD and TD ($\lambda$))

**Date: 26 October 2024**

**Time: 02:00 PM – 5:00 PM**

**Instructor: Dr. Zuhair Zafar**

# Lab 06: n-Step TD and TD ($\lambda$)

## Objectives

Given a simulated environment, evaluate the given policy using n-Step Temporal Difference Learning and TD ($\lambda$) algorithms.

## Tools/Software Requirement:

Google Colab, Python, Gymnasium Library

## Introduction

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.

**Disadvantage of Monte Carlo and TD (0)**

TD (0) method allow us to learn online – at the same time we interact with an environment – and are based on the notion of bootstrapping. This means that we use our current approximation for the value of a state (which might be wrong) to update our estimated value for another state.

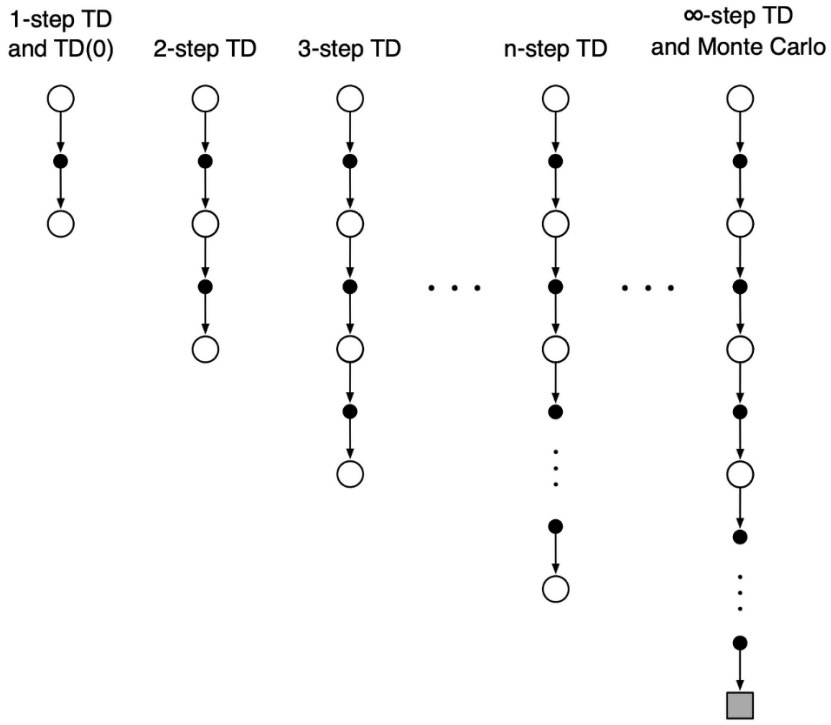
Monte Carlo and Temporal Difference Learning are similar in the sense that they both use real-world experience to evaluate a given policy, however, Monte Carlo methods wait until the return following the visit is known which is after the episode ends is available to update the value of the state, whereas 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.

Everything goes well as long as all of the approximations get better with time. This method is called TD (0), and is **biased**, while having reduced **variance**. A Monte-Carlo estimation method is not biased but has a lot of **variance** since it uses the outcome of a complete episode to perform an update. The variance comes from the fact that, at each interaction, there's randomness involved in picking an action – in the case of a stochastic policy – and in the fact that the environment's dynamics are also random (remember, we have a distribution over the possible next states, that depends on the current state and the action taken).

One problem with TD (0) is that it uses information from only one step to perform an update. Typically, the action that caused a reward to be seen might have happened several timesteps in the past. This means that using only the most recent information can lead to slow convergence.

## The n-Step TD Method

To solve above disadvantages, we can use more than one step to perform an update. In this way, we can balance the biasness and variance trade-off, which affects Monte Carlo and TD (0). Moreover, by taking multiple steps it can lead to comparatively faster convergence than TD (0). The simplest way to think about what methods are between MC and TD is the following image:




Consider estimating the value function Vπ from some sample episodes. 1 step TD methods will perform an update for each state using the next reward and the estimated value of the next state as an approximation for the following rewards.

MC methods on the other hand, will perform an update using the entire sequence of rewards that it observed from the starting state until the end of the episode.

So, it's quite easy to see what's in between: for example, 2 step TD will perform an update based on the next 2 rewards, and the estimated value of the corresponding state (2 steps ahead).

Note that these n step methods are still TD methods because they alter the previous estimate value based on other its difference from the next estimated value.

More formally, the return of MC method:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$$

Where T is the last time step.

The return in 1 step TD method also known as TD (0):

$$G_{t:t+1} \doteq R_{t+1} + \gamma V_t(S_{t+1})$$

The return in 2 step TD method:

$$G_{t:t+2} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 V_{t+1}(S_{t+2})$$

Lastly, the return for n step TD method:

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})$$

**How it works**

TD Learning operates by taking actions according to some policy, observing the rewards at each step and the next state, and then updating the value of the current state based on the observed rewards and the estimated value of the n[th] state. The update is done using the formula:

$$V(S_t) = V(S_t) + \alpha * [G_{t:t+n} - V(S_t)]$$

where:

- $V(S_t)$ is the current estimate of the state's value
- α is the learning rate
- $G_{t:t+n}$ is the n-Step TD return
- γ is the discount factor
- $V(S_{t+n})$ is the estimated value of the n[th] state

**$n$-step TD for estimating $V \approx v_\pi$**

Input: a policy $\pi$
Algorithm parameters: step size $\alpha \in (0, 1]$, a positive integer $n$
Initialize $V(s)$ arbitrarily, for all $s \in \mathcal{S}$
All store and access operations (for $S_t$ and $R_t$) can take their index mod $n+1$

Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$ :
  | If $t < T$, then:
  |  Take an action according to $\pi(\cdot|S_t)$
  |  Observe and store the next reward as $R_{t+1}$ and the next state as $S_{t+1}$
  |  If $S_{t+1}$ is terminal, then $T \leftarrow t+1$
  | $\tau \leftarrow t - n + 1$  ($\tau$ is the time whose state's estimate is being updated)
  | If $\tau \geq 0$:
  |  $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
  |  If $\tau + n < T$, then: $G \leftarrow G + \gamma^n V(S_{\tau+n})$  $(G_{\tau:\tau+n})$
  |  $V(S_\tau) \leftarrow V(S_\tau) + \alpha [G - V(S_\tau)]$
  Until $\tau = T - 1$

## Backward View of TD ($\lambda$)

TD (λ) can be thought of as a combination of TD and MC learning, so as to avoid to choose one method or the other and to take advantage of both approaches.

More precisely, TD (λ) is temporal-difference learning with a λ-return, which is defined as an average of all n-step returns, for all n, where an n-step return is the **target** used to update the estimate of the value function that contains nn future rewards (plus an estimate of the value function of the state nn steps in the future).

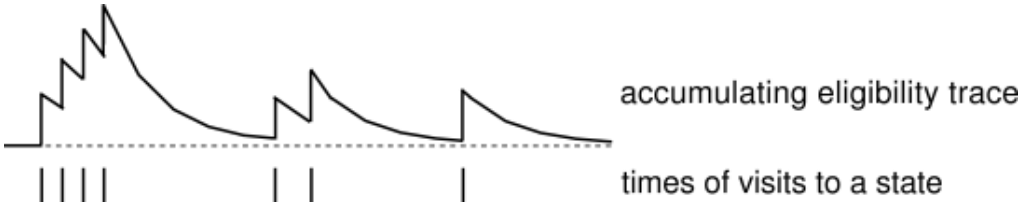
The forward or theoretical view of the tabular TD (λ) algorithm is a way of mixing backups that parametrically shifts from a TD method to a Monte Carlo method.

The backward view instead defines TD (λ) mechanistically. The mechanistic, or backward view is useful because it is simple conceptually and computationally. In particular, the forward view itself is not directly implementable because it is acausal, using at each step knowledge of what will happen many steps later. The backward view provides a causal, incremental mechanism for approximating the forward view and, in the off-line case, for achieving it exactly.

In the backward view of TD (λ), there is an additional memory variable associated with each state, its *eligibility trace*. The eligibility trace for state $s$ at time $t$ is denoted $e_t(s) \in \Re^+$. On each step, the eligibility traces for all states decay by $\gamma\lambda$, and the eligibility trace for the one state visited on the step is incremented by $1$.

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & \text{if } s \neq s_t; \\ \gamma\lambda e_{t-1}(s) + 1 & \text{if } s = s_t, \end{cases}$$

for all nonterminal states $s$, where $\gamma$ is the discount rate and $\lambda$ is the parameter introduced in the previous section. Henceforth we refer to $\lambda$ as the *trace-decay parameter*. This kind of eligibility trace is called an *accumulating* trace because it accumulates each time the state is visited, then fades away gradually when the state is not visited, as illustrated below:




At any time, the traces record which states have recently been visited, where "recently" is defined in terms of $\gamma\lambda$. The traces are said to indicate the degree to which each state is *eligible* for undergoing learning changes should a reinforcing event occur. The reinforcing events we are concerned with are the moment-by-moment one-step TD errors. For example, the TD error for state-value prediction is:

$$\delta_t \;=\; r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t).$$

In the backward view of TD (λ), the global TD error signal triggers proportional updates to all recently visited states, as signaled by their nonzero traces:

$$\Delta V_t(s) = \alpha \delta_t e_t(s), \qquad \text{for all } s \in \mathcal{S}.$$

As always, these increments could be done on each step to form an on-line algorithm or saved until the end of the episode to produce an off-line algorithm.

Initialize $V(s)$ arbitrarily and $e(s) = 0$, for all $s \in \mathcal{S}$
Repeat (for each episode):
  Initialize $s$
  Repeat (for each step of episode):
    $a \leftarrow$ action given by $\pi$ for $s$
    Take action $a$, observe reward, $r$, and next state, $s'$
    $\delta \leftarrow r + \gamma V(s') - V(s)$
    $e(s) \leftarrow e(s) + 1$
    For all $s$:
      $V(s) \leftarrow V(s) + \alpha \delta e(s)$
      $e(s) \leftarrow \gamma \lambda e(s)$
    $s \leftarrow s'$
  until $s$ is terminal

## Lab Tasks

- Learning rate, alpha, α = 0.05
- Discount factor, gamma, γ = 0.95
- Lambda, $\lambda$, = 0.85
- 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 n-step TD algorithm to determine the values of each state for the given policy. Report the final converged values of each state for n = 2 and n = 3.**

```
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. **Apply the Backward view of Temporal Difference Learning, TD (λ) 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.**

3. **You are initially provided with the following deterministic policy in Frozen Lake (4x4) environment. Implement the n-step TD algorithm to determine the values of each state for the given policy. Report the final converged values of each state for n = 2 and n = 3.**

```
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)
```

4. **For the given policy in Task 3 for the Frozen Lake environment, apply the Backward view of Temporal Difference Learning, TD (λ) algorithm to determine the values of each state. Report the final converged values of each state.**

**Deliverable:**

Please submit your notebook on LMS before the deadline **(31st October 2024, 11:59pm).**

# Lab Rubrics

| Assessment | Does not meet expectation<br><br>(1/2 marks) | Meets expectation<br><br>(3/4 marks) | Exceeds expectation<br><br>(5 marks) |
|---|---|---|---|
| **Software Problem Realization**<br><br>**(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 TD 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 TD 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**<br><br>**(CLO4, PLO5)** | Code has syntax errors, and the implementation of the n-step TD / TD ($\lambda$) 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 n-step TD and TD ($\lambda$) 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 n-step TD and TD ($\lambda$) 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. |