Reinforcement Learning **Lecture–4**

Model-Free Prediction

Anubhav Gupta

We have already seen how dynamic programming methods are used for planning in reinforcement learning. These methods rely on the underlying MDP for prediction and control. i.e. they assume complete knowledge of the environment. We will now study a different kind of algorithms, that learn only from experience without the complete knowledge of the environment. These methods are called model-free methods. Specifically, we study following model-free methods for prediction problem:

- Monte-Carlo Learning
- Temporal-Difference Learning
- TD(λ)

The term "Monte Carlo" is often used more broadly for estimations based on significant randomness. In RL, we use it specifically for methods based on **averaging complete returns**.

- MC methods learn directly from episodes from experience
- MC is model-free, i.e. no knowledge of MDP is required
- Learns from complete **episodes** of experience, i.e. it doesn't bootstrap

1 Monte-Carlo (MC) Prediction

The goal in Monte-Carlo prediction methods is to learn the state-value function v_π for a given policy π, from episodes of experience

$$S_1, A_1, R_2, \cdots, S_k \sim \pi$$

Recall that the return at time-step t is the total discounted reward after time-step t, i.e.

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

where T is the length of the episode. Also, recall that the value function $v_\pi(s)$ is the expected return:

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

Each occurence of state s in an episode is called a *visit* to s. Depending on the updates on visits of s, there are two kinds of MC estimates - **first-visit MC method** and **every-visit MC method**. First-visit MC method estimates $v_\pi(s)$ as the average of the returns following first visit to s, whereas every-visit MC method averages the returns following all visits to s.

1.1 First-Visit MC Method

To evaluate value function for state s, steps of first-visit Monte-Carlo Method can be summarized as follows:

- The **first** time-step t that state s is visited in an episode
  1. increment counter $N(s) \leftarrow N(s) + 1$
  2. increment total return $S(s) \leftarrow S(s) + G_t$
- Value is estimated by mean return $V(s) = \frac{S(s)}{N(s)}$

It can be shown, by law of large numbers, that $V(s) \rightarrow v_\pi(s)$ as $N(s) \rightarrow \infty$

1.2 Every-Visit MC Method

Steps for every-visit MC are same as above except that average is taken over returns from every visit to s.

- **Every** time-step t that state s is visited in an episode
    1. increment counter $N(s) \leftarrow N(s) + 1$
    2. increment total return $S(s) \leftarrow S(s) + G_t$
- Value is estimated by mean return $V(s) = \frac{S(s)}{N(s)}$

It can be shown, by law of large numbers, that $V(s) \rightarrow v_\pi(s)$ as $N(s) \rightarrow \infty$

An important fact about MC methods is that estimates for each state are independent, i.e. the estimates for one state do no build upon the estimates of other states. In other words, Monte Carlo methods **do not bootstrap**.

One more thing to note is that computational expense of estimating the value of a single state is independent of the number of states in MDP. This property of Monte Carlo methods makes these very attractive in situations where one needs to evaluate only a single state. In such cases, one can generate several samples/episodes starting from the state of interest.

1.3 Monte-Carlo Estimation of Action Values

When we later consider the problem of control in model-free environment, it becomes essential that we estimate action values rather than state values. This is because, in the absence of model of the environment state values themselves are not enough in suggesting new policies.

To estimate action value $q_\pi(s, a)$ for any state-action pair, we can follow the same approach as of state-values, but instead of averaging returns from the state s, we now average returns over all the state-action pairs (s, a) when action a was taken from state s.

However, there is one complication that there may be many state-action pairs that may never be visited. If π is a deterministic policy, then in following π one will only observe returns for single state-action pair from each state. So, MC will not be able to estimate other actions from this state. To compare alternatives, we need to estimate all the alternative actions from all the states. This is a general problem of **maintaining exploration**. One way to solve it is by specifying that each episode start in a state-action pair and that each pair has a non-zero probability of being selected as start. This is called the assumption of **exploring starts**.

1.4 Incremental Updates

Current implementation of Monte Carlo method is offline in the sense that it reads through the entire sample of episodes in order to calculate the mean returns. The calculation of mean can be made incremental to update value estimates after each episode.

Incremental Mean The mean $\mu_1, \mu_2, \cdots$ of a sequence $x_1, x_1, \cdots$ can be computed incrementally as follows:

$$\begin{aligned}
\mu_k &= \frac{1}{k}\sum_{j=1}^{k} x_j \\
&= \frac{1}{k}(x_k + \sum_{j=1}^{k-1} x_j) \\
&= \frac{1}{k}(x_k + (k-1)\mu_{k-1}) \\
&= \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1})
\end{aligned}$$

Now, MC update to estimate $v_\pi(s)$ can be computed incrementally as follows:

- After each episode $S_1, A_1, R_2, \cdots, S_T$, update $V(s)$ incrementally
- For each state S_t with return G_t

$$\begin{aligned}
N(S_t) &\leftarrow N(S_t) + 1 \\
V(S_t) &\leftarrow V(S_t) + \frac{1}{N(S_t)}(G_t - V(S_t))
\end{aligned}$$

- In non-stationary problems, it can be useful to forget old episodes and just remember the running mean.

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

Algorithm-1 gives a complete pseudocode for incremental every-visit Monte-Carlo prediction.

```
Algorithm 1: Incremental on-policy MC prediction for estimating $v_\pi$
Input: The policy $\pi$ to be evaluated
Algorithm Parameters: Step size $\alpha \in (0, 1]$
Initialization: Initialize $V(s)$ arbitrarily for all states
for each episode until convergence do
    Generate an episode $S_0, A_0, R_1, S_1, \cdots, S_{T-1}, A_{T-1}, R_T$ following policy $\pi$
    $G \leftarrow 0$
    for each timestep $t = T-1, T-2, \cdots, 0$ of episode do
        $G \leftarrow \gamma G + R_{t+1}$
        $V(S_t) \leftarrow V(S_t) + \alpha(G - V(S_t))$
    end
end
```

2 Temporal-Difference (TD) Learning

Monte-Carlo learning method discussed so far have an inherent drawback that they learn from complete sequences, i.e. they only work in an episodic environment where all sequences terminate. Temporal-difference (TD) learning methods, on the other hand work very well for non-episodic tasks. TD learning is a combination of Monte Carlo ideas and dynamic programming ideas. TD methods learn from incomplete episodes by bootstrapping: estimate the values based on values of other estimates. Like MC, these are also model-free and learn directly from experience. Whereas incremental MC methods must wait until the end of episode to make update to value function, TD methods need to wait only until the next time-step. The simplest temporal-difference learning algorithm, called **TD(0)**, or **one-step TD** updates value $V(S_t)$ toward estimated return $R_{t+1} + \gamma V(S_{t+1})$

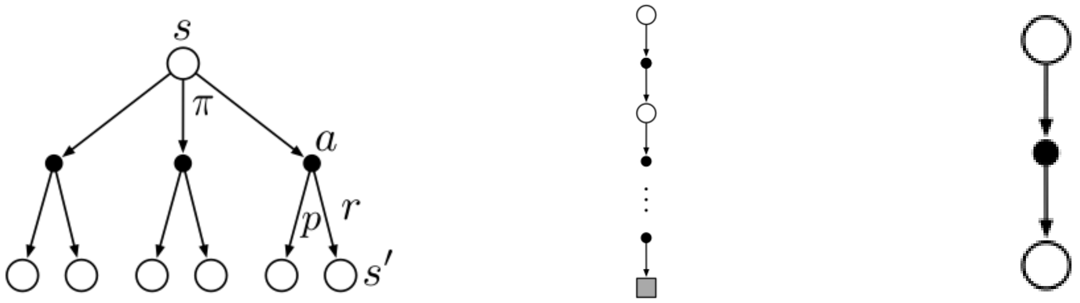
$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

- $R_{t+1} + \gamma V(S_{t+1}$ is called the TD target
- $\delta_t = R_{t+1} + \gamma V(S_{t+1} - V(S_t))$ is called TD error

A psuedocode for estimating v_π using TD(0) learning is as follows.

```
Algorithm 2: TD(0) for estimating $v_\pi$
Input: The policy $\pi$ to be evaluated
Algorithm Parameters: Step size $\alpha \in (0, 1]$
Initialization: Initialize $V(s)$ arbitrarily for all states. $V(terminal) = 0$
for each episode do
    Initialize $S$
    for each timestep $t$ of episode do
        $A \leftarrow$ action given by $\pi$ for $S$
        Take action $A$, observe $R, S'$
        $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
        $S \leftarrow S'$
    end
    Stop if $S$ is terminal state
end
```

Notice that the TD error at each time t is the error in the estimate made at that time. TD error at time t is not available until $t+1$. Also note that if the array V does not change during the episode (as it does not in



(a) DP (b) MC (c) TD(0)

Figure 1: Backup diagrams for dynamic programming, Monte Carlo and temporal difference methods

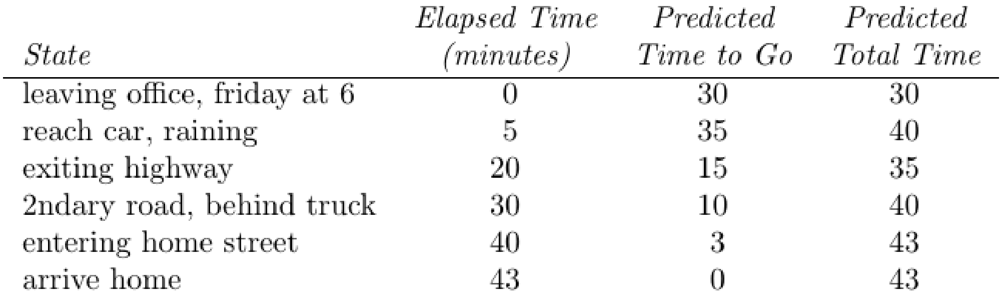
MC updates, where it is changed after the episode is finished), then the MC error can be written as a sum of TD errors.

$$\begin{aligned}
G_t - V(S_t) &= R_{t+1} + \gamma G_{t+1} - V(S_t) \\
&= R_{t+1} + \gamma G_{t+1} - V(S_t) + \gamma V(S_{t+1}) - \gamma V(S_{t+1}) \\
&= \delta_t + \gamma (G_{t+1} - V(S_{t+1})) \\
&= \delta_t + \gamma \delta_{t+1} + \gamma^2 (G_{t+2} - V(S_{t+2})) \\
&= \delta_t + \gamma \delta_{t+1} + \gamma^2 \delta_{t+2} + \cdots + \gamma^{T-t-1} \delta_{T-1} + \gamma^{T-t}(G_T - V(S_T)) \\
&= \delta_t + \gamma \delta_{t+1} + \gamma^2 \delta_{t+2} + \cdots + \gamma^{T-t-1} \delta_{T-1} + \gamma^{T-t}(0 - 0) \\
&= \sum_{k=1}^{T-1} \gamma^{k-t} \delta_k
\end{aligned}$$

2.1 Comparison between TD and MC

Temporal difference methods combine the sampling of Monte Carlo method with the bootstrapping of DP. The difference and similarities between the three kinds of methods DP, MC and TD can be illustrated by their backup diagrams in figure 1.

Consider the following example (Example 6.1 from Sutton). Each day you drive home from work, you try to predict your estimated time to reach home. One particular day, the sequence of states, times and predictions looks like this:



State	*Elapsed Time (minutes)*	*Predicted Time to Go*	*Predicted Total Time*
leaving office, friday at 6	0	30	30
reach car, raining	5	35	40
exiting highway	20	15	35
2ndary road, behind truck	30	10	40
entering home street	40	3	43
arrive home	43	0	43

Let's consider the undiscounted case ($\gamma = 1$). The rewards are the elapsed time on each lag of the journey. If MC method is used to estimate values of different states, estimates will only be updated once you reach home, i.e. episode terminates. Whereas, TD methods keep updating estimates after each state. The learning of MC and TD methods on this example can be visualized as follows:

TD methods provides following advantages over other methods such as DP and MC:

- They learn as estimate from other estimates, i.e. they bootstrap
- TD do not require a model of the environment, of its reward and next-state probability distributions
- TD methods are naturally implemented in an online, fully incremental fashion
- TD can learn from incomplete sequences and in continuing environments

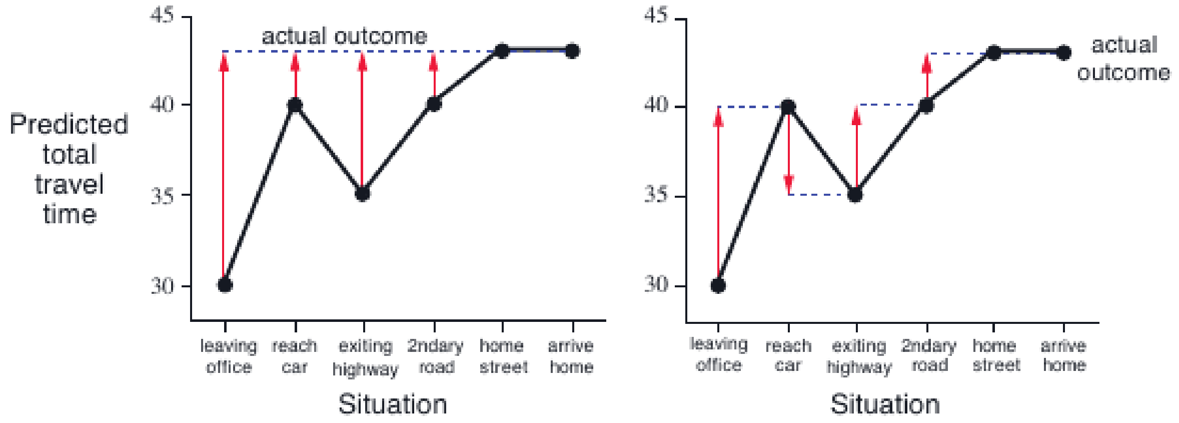


Figure 2: Changes recommended by MC (left) and TD (right) methods

2.2 Bias-Variance Trade-off

- True return $G_t = R_{t+1} + \gamma R_{t+2} + \cdots \gamma^{T-1} R_T$ is unbiased estimate of $v_\pi(S_t)$
- True TD target $R_{t+1} + \gamma v_\pi(S_{t+1})$ is unbiased estimate of $v_\pi(S_t)$
- TD target $R_{t+1} + \gamma V(S_{t+1})$ is a **biased** estimate of $v_\pi(S_t)$
- TD target is much lower variance than the return. This is because return depends on many random variables, such as actions, rewards, transitions, whereas TD target depends on only one such random choice.

Monte Carlo methods have high variance, but zero bias. MC method exhibits good convergence properties. On the other hand temporal difference methods have low variance, but higher bias than MC methods. In practice, TD methods are usually more efficient than MC. TD(0) also have the convergence guarantees that it converges to $v_\pi(s)$.

2.3 Batch MC and TD

Both Monte Carlo and temporal difference methods converge to true value function $V(s) \to v_\pi(s)$ as number of episodes (and hence time steps) reaches infinity. But, what happens when the available experience is limited, say 10 episodes or 100 episodes.

$$\begin{aligned} & s_1^1, a_1^1, r_2^1, \cdots, s_{T_1}^1 \\ & \vdots \\ & s_1^K, a_1^K, r_2^K, \cdots, s_{T_1}^K \end{aligned}$$

One common approach in such situations is to repeatedly sample all K episodes and update estimates of V based on episodes in the sample available using MC or TD. Then all the available experience is processed again with this new value function to produce new overall increment. This process is repeated until the value function converges. This form of updating is called **batch updating** because updates are made only after processing each complete batch of training data.

Using batch update, both TD(0) and constant-α methods converge deterministically to a single answer independent of the step-size α, as long is α is chosen sufficiently small. Both methods, however, converge to different answers. Let's see an **example** to illustrate this. Consider an unknown Markov reward process and following observed eight episodes:

A,0,B,0	B,1	B,1
B,1	B,1	B,0
B,1	B,1	

Given this batch of data, what would be the optimal values for states A & B? Since all episodes terminate in state B, optimal value for this state is probably $V(B) = 3/4$. But how do you estimate optimal value of A? There may be two different interpretations in estimating the value of A $V(A)$:

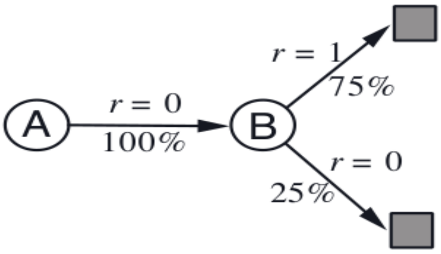


Figure 3: Markov Process for batch TD example

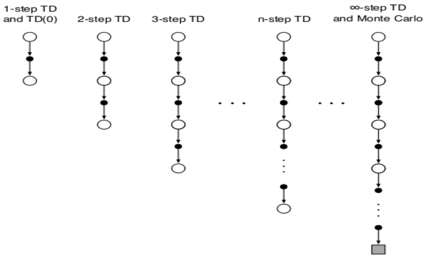


Figure 4: Backup diagrams for n-step methods

- One way is to observe, whenever we are in state A, we transition to state B with probability 1 with immediate reward 0, and so the value of A must be same as the value of B. So $V(A) = 3/4$ as well.
- The other way of estimating A might be that whenever we see state A, the return that followed is always 0, and so $V(A)$ is estimated to be 0.

The above two answers are essentially the estimates given by batch TD(0) and batch MC methods respectively. **Batch MC** methods always find the estimates that minimize mean-squared error on the *training data*

$$\sum_{k=1}^{K}\sum_{t=1}^{T_k}(G_t^k - V(s_t^k))^2$$

Batch TD(0) method, on the other hand always finds the estimates that would be exactly correct for maximum-likelihood model of the Markov process, as illustrated in figure 3. Solution to MDP $\langle \mathcal{S}, \mathcal{A}, \hat{\mathcal{P}}, \hat{\mathcal{R}}, \gamma \rangle$ that best fits the data:

$$\hat{\mathcal{P}}_{s,s'}^{a} = \frac{1}{N(s,a)}\sum_{k=1}^{K}\sum_{t=1}^{T_k}\mathbb{1}(s_t^k, a_t^k, s_{t+1}^k = s, a, s')$$
$$\hat{\mathcal{R}}_{s}^{a} = \frac{1}{N(s,a)}\sum_{k=1}^{K}\sum_{t=1}^{T_k}\mathbb{1}(s_t^k, a_t^k = s, a)r_t^k$$

Given this model, we can compute the estimate of value function that would be exactly correct if the model were exactly correct. This is called the **certainty-equivalence estimate**. In general, batch TD(0) converges to certainty-equivalence estimate.

3 n-step Prediction

One step TD methods use the estimate of value function of state after one time step to estimate the value function of current state. MC methods, on the other hand, perform updates based on entire sequence of observed rewards until the end of episode. n-step TD methods generalizes the ideas of TD(0) and MC methods so that one can shift from one to other smoothly. Instead of one step, we can bootstrap from two steps, three steps, or more as shown in figure 4. Consider the following n-step returns for $n = 1, 2, \cdots, \infty$:

$$\begin{array}{ll} n = 1\ (TD) & G_t^{(1)} = R_{t+1} + \gamma V(S_{t+1}) \\ n = 2 & G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2}) \\ \vdots & \quad \vdots \\ n = \infty\ (MC) & G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_T \end{array}$$

Define the n-step return as follows:

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$$

All n-step returns can be considered approximations to full return, truncated after n steps, then corrected for remaining missing term by $V(S_{t+n})$. This is n-step Temporal Difference algorithm, and its update equation is given by:

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t^{(n)} - V(S_t))$$

Note that no changes are made during the first $n - 1$ steps of each episode. To make up for that, an equal number of additional updates are made at the end of each episode. Complete pseudocode for n-step TD methods is given by algorithm 3.

```
Algorithm 3: n-step TD for estimating $V \approx v_\pi$
Input: The policy $\pi$ to be evaluated
Algorithm Parameters: Step size $\alpha \in (0, 1]$, positive integer $n$
Initialization: Initialize $V(s)$ arbitrarily for all states
for each episode do
    Initialize and store $S_0 \neq$ terminal
    $T \leftarrow \infty$
    for each timestep $t = 0, 1, 2, \cdots$ until $t' = T - 1$ do
        if $t < T$ then
            Take action according to $\pi(\cdot|S_t)$
            Observe and store $R_{t+1}$ and $S_{t+1}$
            If $S_{t+1}$ is terminal, then $T \leftarrow t + 1$
        end
        $t' \leftarrow t - n + 1$ (time whose state's estimate is being updated)
        if $t' > 0$ then
            $G \leftarrow \sum_{i=t'+1}^{min(t'+n,T)} \gamma^{i-t'-1} R_i$
            If $t' + n < T$, then $G \leftarrow G + \gamma^n V(S_{t'+n})$
            $V(S_{t'}) \leftarrow V(S_{t'}) + \alpha[G - V(S_{t'})]$
        end
    end
end
```

4 Eligibility Traces and TD(λ)

We now have an algorithm which considers exact returns for n steps and bootstraps for returns after n time steps. Now the question is how to decide which value of n should be used. One way is to consider all values of n at once, and average all n-step returns. But again, how do we efficiently combine information from all time-steps?

The algorithm that does exactly this is called TD(λ). Here λ refers to the use of eligibility traces. Eligibility traces provide a way to implement MC methods online as well as for continuing problems.

4.1 λ-Return

The λ-return combines all the n-step returns, by taking a weighted average of all n-step updates. The λ-return is defined as:

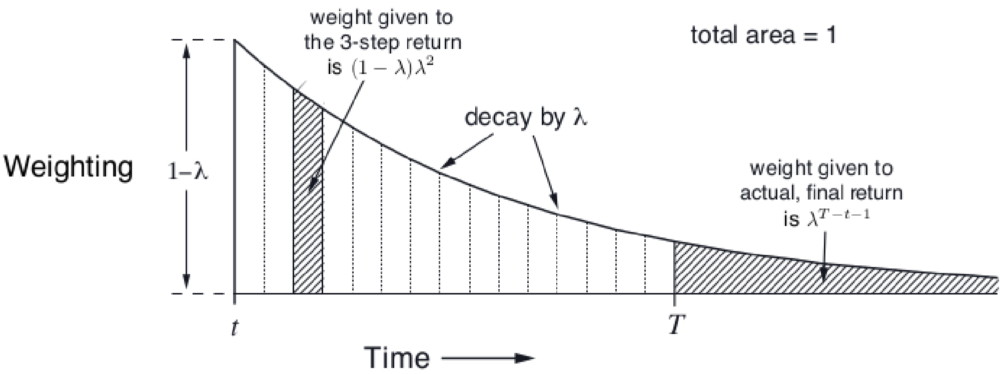


Figure 5: Weighting given by TD(λ) to each of n step returns

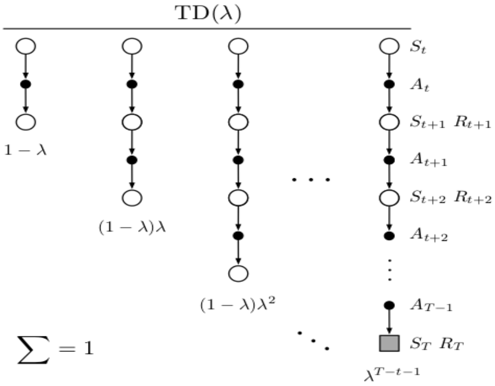


Figure 6: Backup diagram for TD(λ)

$$G_t^{\lambda} = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_t^{(n)}$$

with weight parameter $\lambda \in [0,1]$. The weighting to different steps is further illustrated by figure 5. The algorithm which uses this n-step weighted return to update the estimate of value functions is called **TD(λ)**. The update equation for **forward-view TD(λ)** is given by:

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t^{\lambda} - V(S_t))$$

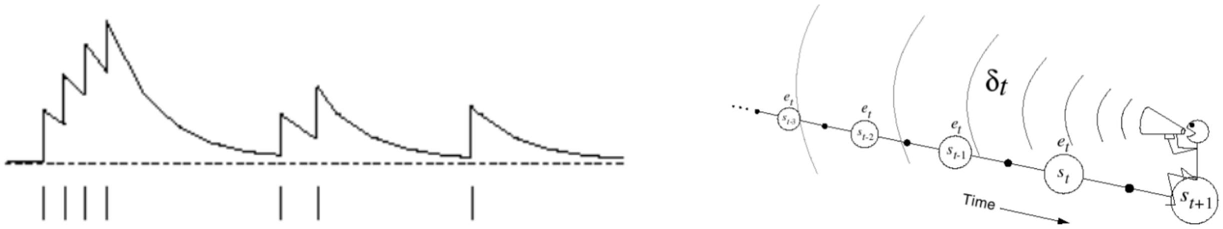
Note that depending on the weight parameter, TD(λ) behaves as TD(0) (if $\lambda = 0$) as well as Monte Carlo (if $\lambda = 1$). In episodic problems with T time-steps long episode, the return is calculated by assigning remaining weight λ^{T-t-1} to the actual final return, as shown in figure 6.

The TD(λ) algorithm discussed so far is the forward view of the algorithm. That is for each visited state, we look forward in time to get all the future rewards and take their weighted average. This algorithm is also called λ-return algorithm. So, this forward-view also suffers from the same problem that MC has. It can only be updated from completed episodes.

The view of TD(λ) actually used is called backward-view TD(λ). It provides a way to learn online as well as from incomplete sequences through the use of eligibility traces. Eligibility traces provide a way of assigning credit scores for reward to each of the state. Eligibility traces are based on two heuristics:

- **Frequency heuristic:** assign credit to most frequent states
- **Recency heuristic:** assign credit to most recent states

$$\begin{aligned} E_0(s) &= 0 \\ E_t(s) &= \gamma\lambda E_{t-1}(s) + \mathbb{1}(S_t = s) \end{aligned}$$



(a) Graphical view of eligibility traces

(b) Backward view of eligibility traces

Figure 7: Eligibility traces - Graphical and Backward views

Then, using this eligibility trace, whenever a non-zero reward is available value functions for all relevant states can be updated in proportion to their eligibility traces $E_t(s)$ and TD-error δ_t. This backward view of eligibility traces is also illustrated in figure 7b.

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$
$$V(s) \leftarrow V(s) + \alpha\delta_t E_t(s)$$

Theorem: The sum of offline updates is identical for forward-view and backward-view TD(λ).

$$\sum_{t=1}^{T} \alpha\delta_t E_t(s) = \sum_{t=1}^{T} \alpha(G_t^\lambda - V(S_t))\mathbb{1}(S_t = s)$$

4.2 Offline vs Online TD(λ)

For offline TD(λ):

- Updates are accumulated within episode
- Applied in batch at the end of episode

For online TD(λ):

- TD(λ) updates are applied online at each step within episode
- Forward and backward-view TD(λ) are slightly different. They use slightly different form of eligibility trace

Table 1 summarizes different TD(λ) methods depending on value of λ.

	$\lambda = 0$	$\lambda \in (0, 1)$	$\lambda = 1$
Backward View	TD(0)	TD(λ)	TD(1)
Forward View	TD(0)	Forward TD(λ)	MC
Exact Online	TD(0)	Exact TD(λ)	Exact TD(1)

Table 1: Categorization of TD methods

5 Off-Policy Learning

All learning (control) methods face a dilemma that they all need to find optimal policies, which requires taking actions based on subsequent optimal behavior. But at the same time, they also need to behave non-optimally in order to explore more actions. One way to achieve this is by using two different policies: one that is being learned about which converges to optimal policy, and the other more exploratory policy. The policy being learned is called the **target policy** and the policy which is used to generate behavior is called **bahavior policy**. This process of learning from two policies is called off-policy learning.

So, the goal of off-policy learning is to evaluate a target policy $\pi(a|s)$ to compute $v_\pi(s)$ or $q_\pi(s, a)$, while following bahavior policy $\mu(a|s)$

$$\{S_1, A_1, R_2, \cdots, S_T\} \sim \mu$$

To perform off-policy learning, assumption of coverage must be satisfied. It states that every action taken under π must have a non-zero probability of being chosen under the behavior policy μ. That is $\pi(a|s) > 0$ implies $\mu(a|s) > 0$.

Importance sampling is a general technique which is used in most off-policy learning methods. The idea of importance sampling is to estimate expectations under one distribution given samples from another.

$$\begin{aligned}\mathbb{E}_{X\sim P}[f(X)] &= \sum P(X)f(X) \\ &= \sum Q(X)\frac{P(X)}{Q(X)}f(X) \\ &= \mathbb{E}_{X\sim Q}\left[\frac{P(X)}{Q(X)}f(X)\right]\end{aligned}$$

5.1 Importance sampling for off-policy MC

To apply importance sampling to off-policy learning, we weight returns according to relative probabilities of their trajectories occurring under different policies, called **importance-sampling ratio**. Given starting state S_t, probability of subsequent state-action trajectory $A_t, S_{t+1}, \cdots, S_T$ under any policy π is

$$\begin{aligned}P(A_t, S_{t+1}, \cdots, S_T | S_t, A_{t:T-1} \sim \pi) &= \pi(A_t|S_t)p(S_{t+1}|S_t, A_t)\pi(A_{t+1}|S_{t+1})\cdots p(S_T|S_{T-1}, A_{T-1}) \\ &= \prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k)\end{aligned}$$

where p is the state transition probability function.
Weighted return G_t according to the similarity between policies π and μ is then

$$\begin{aligned}G_t^{\mu/\pi} &= \frac{\prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k)}{\prod_{k=t}^{T-1} \mu(A_k|S_k)p(S_{k+1}|S_k, A_k)}G_t \\ &= \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{\mu(A_k|S_k)}G_t\end{aligned}$$

Monte Carlo methods now use this corrected return to make the update:

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t^{\mu/\pi} - V(S_t)\right)$$

5.2 Importance sampling for off-policy TD

To use importance sampling for temporal difference learning, weight TD target $R + \gamma V(S')$ by importance sampling ratio for next state (one step correction). For TD(0), policies only need to be similar over a single time step. TD(0) update using importance sampling is given as follows:

$$V(S_t) \leftarrow V(S_t) + \alpha\left(\frac{\pi(A_t|S_t)}{\mu(A_t|S_t)}(R_{t+1} + \gamma V(S_{t+1})) - V(S_t)\right)$$

One thing to keep in mind is that importance sampling can dramatically increase variance of estimated value functions. This is specially true for Monte Carlo methods where variance of estimates is significantly larger than TD methods.

References

[1] Reinforcement Learning: An Introduction (2nd Edition), by Richard S. Sutton, Andrew G. Barto.

[2] David Silver's course "Introduction to Reinforcement Learning" available at `https://www.davidsilver.uk/teaching/`