CS234 Notes - Lecture 3
Model Free Policy Evaluation: Policy Evaluation Without Knowing How the World Works

Youkow Homma, Emma Brunskill

March 20, 2018

4 Model-Free Policy Evaluation

In the previous lecture, we began by discussing three problem formulations of increasing complexity, which we recap below.

1. A Markov process (MP) is a stochastic process augmented with the Markov property.
2. A Markov reward process (MRP) is a Markov process with rewards at each time step and the accumulation of discounted rewards, called values.
3. A Markov decision process (MDP) is a Markov reward process augmented with choices, or actions, at each state.

In the second half of last lecture, we discussed two methods for evaluating a given policy in an MDP and three methods for finding the optimal policy of an MDP. The two methods for policy evaluation were directly solving via a linear system of equations and dynamic programming. The three methods for control were brute force policy search, policy iteration and value iteration.

Implicit in all of these methods was the assumption that we know both the rewards and probabilities for every transition. However, in many cases, such information is not readily available to us, which necessitates **model-free algorithms**. In this set of lectures notes, we will be discussing **model-free policy evaluation**. That is, we will be given a policy and will attempt to learn the value of that policy without leveraging knowledge of the rewards or transition probabilities. In particular, we will not be discussing how to improve our policies in the model-free case until next lecture.

4.1 Notation Recap

Before diving into some methods for model-free policy evaluation, we'll first recap some of the notation surrounding MDP's from last lecture that we'll need in this lecture.

Recall that we define the return of an MRP as the discounted sum of rewards starting from time step t and ending at horizon H, where H may be infinite. Mathematically, this takes the form

$$G_t = \sum_{i=t}^{H-1} \gamma^{i-t} r_i, \tag{1}$$

for $0 \leq t \leq H-1$, where $0 < \gamma \leq 1$ is the discount factor and r_i is the reward at time step i. For an MDP, the return G_t is defined identically, and the rewards r_i are generated by following policy $\pi(a|s)$.

The state value function $V^\pi(s)$ is the expected return from starting at state s under stationary policy π. Mathematically, we can express this as

$$V^\pi(s) = \mathbb{E}_\pi[G_t|s_t = s] \tag{2}$$

$$= \mathbb{E}_\pi\left[\sum_{i=t}^{H-1} \gamma^{i-t} r_i | s_t = s\right]. \tag{3}$$

The state-action value function $Q^\pi(s,a)$ is the expected return from starting in state s, taking action a, and then following stationary policy π for all transitions thereafter. Mathematically, we can express this as

$$Q^\pi(s,a) = \mathbb{E}_\pi[G_t|s_t = s, a_t = a] \tag{4}$$

$$= \mathbb{E}_\pi\left[\sum_{i=t}^{H-1} \gamma^{i-t} r_i | s_t = s, a_t = a\right]. \tag{5}$$

Throughout this lecture, we will assume an infinite horizon as well as stationary rewards, transition probabilities and policies. This allows us to have time-independent state and state-action value functions, as derived last lecture.

There is one new definition that we will use in this lecture, which is the definition of a history.

Definition 4.1. *The **history** is the ordered tuple of states, actions and rewards that an agent experiences. In episodic domains, we will use the word episode interchangeably with history. When considering many interactions, we will index the histories in the following manner: the jth history is*

$$h_j = (s_{j,1}, a_{j,1}, r_{j,1}, s_{j,2}, a_{j,2}, r_{j,2}, \ldots, s_{j,L_j}),$$

where L_j is the length of the interaction, and $s_{j,t}, a_{j,t}, r_{j,t}$ are the state, action and reward at time step t in history j, respectively.

4.2 Dynamic Programming

Recall also from last lecture the dynamic programming algorithm to calculate the value of an infinite horizon MDP with $\gamma < 1$ under policy π, which we rewrite here for convenience as Algorithm 1.

Algorithm 1 Iterative algorithm to calculate MDP value function for a stationary policy π

procedure POLICY EVALUATION(M, π, ϵ)
  For all states $s \in S$, define $R^\pi(s) = \sum_{a\in A} \pi(a|s) R(s,a)$
  For all states $s, s' \in S$, define $P^\pi(s'|s) = \sum_{a\in A} \pi(a|s) P(s'|s,a)$
  For all states $s \in S$, $V_0(s) \leftarrow 0$
  $k \leftarrow 0$
  while $k = 0$ or $||V_k - V_{k-1}||_\infty > \epsilon$ **do**
    $k \leftarrow k + 1$
    For all states $s \in S$, $V_k(s) = R^\pi(s) + \gamma \sum_{s'\in S} P^\pi(s'|s) V_{k-1}(s')$
  return V_k

Written in this form, we can think of V_k in a few ways. First, $V_k(s)$ is the exact value of following policy π for k additional transitions, starting at state s. Second, for large k, or when Algorithm 1 terminates, $V_k(s)$ is an estimate of the true, infinite horizon value $V^\pi(s)$ of state s.

We can additionally express the behavior of this algorithm via a **backup diagram**, which is shown in Figure 1. This backup diagram is read top-down with white circles representing states, black circles

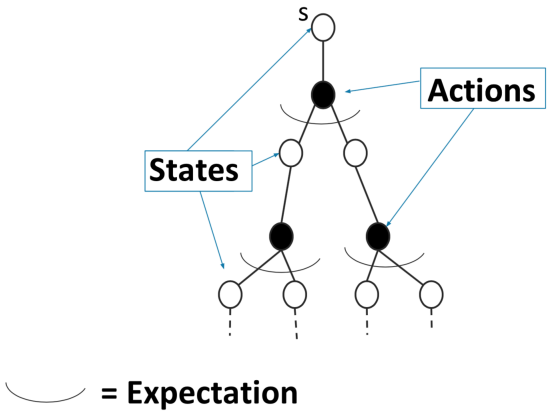

Figure 1: Backup diagram for the dynamic programming policy evaluation algorithm.

representing actions and arcs representing taking the expectation. The diagram shows the branching effect of starting at state s at the top and transitioning two time steps as we move down the diagram. It furthermore shows how after starting at state s and taking an action under policy π, we take the expectation over the value of the next state. In dynamic programming, we **bootstrap**, or estimate, the value of the next state using our current estimate, $V_{k-1}(s')$.

4.3 Monte Carlo On Policy Evaluation

We now describe our first model-free policy evaluation algorithm which uses a popular computational method called the Monte Carlo method. We first walk through an example of the Monte Carlo method outside the context of reinforcement learning, then discuss the method more generally, and finally apply Monte Carlo to reinforcement learning. We emphasize here that this method only works in episodic environments, and we'll see why this is as we examine the algorithm more carefully in this section.

Suppose we want to estimate how long the commute from your house to Stanford's campus will take today. Suppose we also have access to a commute simulator which models our uncertainty of how bad the traffic will be, the weather, construction delays, and other variables, as well as how these variables interact with each other. One way to estimate the expected commute time is to simulate our commute many times on the simulator and then take an average over the simulated commute times. This is called a Monte Carlo estimate of our commute time.

In general, we get the Monte Carlo estimate of some quantity by observing many iterations of how that quantity is generated either in real life or via simulation and then averaging over the observed quantities. By the law of large numbers, this average converges to the expectation of the quantity.

In the context of reinforcement learning, the quantity we want to estimate is $V^\pi(s)$, which is the average of returns G_t under policy π starting at state s. We can thus get a Monte Carlo estimate of $V^\pi(s)$ through three steps:

1. Execute a **rollout** of policy π until termination many times
2. Record the returns G_t that we observe when starting at state s
3. Take an average of the values we get for G_t to estimate $V^\pi(s)$.

The backup diagram for Monte Carlo policy evaluation can be seen in Figure 2. The new blue line indicates that we sample an entire episode until termination starting at state s.

There are two forms of Monte Carlo on policy evaluation, which are differentiated by whether we take an average over just the first time we visit a state in each rollout or every time we visit the state in each

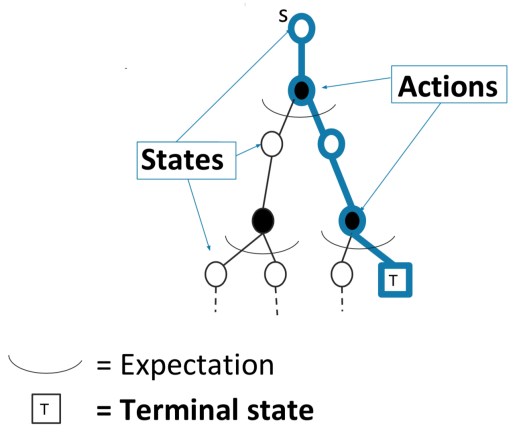

Figure 2: Backup diagram for the Monte Carlo policy evaluation algorithm.

rollout. These are called First-Visit Monte Carlo and Every-Visit Monte Carlo On Policy Evaluation, respectively.

More formally, we describe the First-Visit Monte Carlo in Algorithm 2 and the Every-Visit Monte Carlo in Algorithm 3.

Algorithm 2 First-Visit Monte Carlo Policy Evaluation

procedure FIRST-VISIT-MONTE-CARLO($h_1, \ldots, h_j$)
  For all states s, $N(s) \leftarrow 0$, $S(s) \leftarrow 0$, $V(s) \leftarrow 0$
  for each episode h_j **do**
    for $t = 1, \ldots, L_j$ **do**
      if $s_{j,t} \neq s_{j,u}$ for $u < t$ **then**
        $N(s_{j,t}) \leftarrow N(s_{j,t}) + 1$
        $S(s_{j,t}) \leftarrow S(s_{j,t}) + G_{j,t}$
        $V^{\pi}(s_{j,t}) \leftarrow S(s_{j,t})/N(s_{j,t})$
  return V^{π}

Algorithm 3 Every-Visit Monte Carlo Policy Evaluation

procedure EVERY-VISIT-MONTE-CARLO($h_1, \ldots, h_j$)
  For all states s, $N(s) \leftarrow 0$, $S(s) \leftarrow 0$, $V(s) \leftarrow 0$
  for each episode h_j **do**
    for $t = 1, \ldots, L_j$ **do**
      $N(s_{j,t}) \leftarrow N(s_{j,t}) + 1$
      $S(s_{j,t}) \leftarrow S(s_{j,t}) + G_{j,t}$
      $V^{\pi}(s_{j,t}) \leftarrow S(s_{j,t})/N(s_{j,t})$
  return V^{π}

Note that in the body of the for loop in Algorithms 2 and 3, we can remove vector S and replace the update for $V^{\pi}(s_{j,t})$ with

$$V^{\pi}(s_{j,t}) \leftarrow V^{\pi}(s_{j,t}) + \frac{1}{N(s_{j,t})}(G_{j,t} - V^{\pi}(s_{j,t})). \tag{6}$$

This is because the new average is the average of $N(s_{j,t}) - 1$ of the old values $V^{\pi}(s_{j,t})$ and the new return $G_{j,t}$, giving us

$$\frac{V^{\pi}(s_{j,t}) \times (N(s_{j,t}) - 1) + G_{j,t}}{N(s_{j,t})} = V^{\pi}(s_{j,t}) + \frac{1}{N(s_{j,t})}(G_{j,t} - V^{\pi}(s_{j,t})), \tag{7}$$

which is precisely the new form of the update.

Replacing $\frac{1}{N(s_{j,t})}$ with α in this new update gives us the more general **Incremental Monte Carlo On Policy Evaluation**. Algorithms 4 and 5 detail this procedure in the First-Visit and Every-Visit cases, respectively.

Algorithm 4 Incremental First-Visit Monte Carlo Policy Evaluation

procedure INCREMENTAL-FIRST-VISIT-MONTE-CARLO($\alpha, h_1, \ldots, h_j$)
  For all states s, $N(s) \leftarrow 0$, $V(s) \leftarrow 0$
  for each episode h_j **do**
    for $t = 1, \ldots, terminal$ **do**
      if $s_{j,t} \neq s_{j,u}$ for $u < t$ **then**
        $N(s_{j,t}) \leftarrow N(s_{j,t}) + 1$
        $V^\pi(s_{j,t}) \leftarrow V^\pi(s) + \alpha(G_{j,t} - V^\pi(s))$
  return V^π

Algorithm 5 Incremental Every-Visit Monte Carlo Policy Evaluation

procedure INCREMENTAL-EVERY-VISIT-MONTE-CARLO($\alpha, h_1, \ldots, h_j$)
  For all states s, $N(s) \leftarrow 0$, $V(s) \leftarrow 0$
  for each episode h_j **do**
    for $t = 1, \ldots, terminal$ **do**
      $N(s_{j,t}) \leftarrow N(s_{j,t}) + 1$
      $V^\pi(s_{j,t}) \leftarrow V^\pi(s) + \alpha(G_{j,t} - V^\pi(s))$
  return V^π

Setting $\alpha = \frac{1}{N(s_{j,t})}$ recovers the original Monte Carlo On Policy Evaluation algorithms given in Algorithms 2 and 3, while setting $\alpha > \frac{1}{N(s)}$ gives a higher weight to newer data, which can help learning in non-stationary domains. If we are in a truly Markovian-domain, Every-Visit Monte Carlo will be more data efficient because we update our average return for a state every time we visit the state.

Exercise 4.1. *Recall our Mars Rover MDP from last lecture, shown in Figure 3 below. Suppose that our estimate for the value of each state is currently 0. If we experience the history*

$$h = (S3, TL, +0, S2, TL, +0, S1, TL, +1, terminal),$$

then:

1. *What is the first-visit Monte Carlo estimate of V at each state?*
2. *What is the every-visit Monte Carlo estimate of each state?*
3. *What is the incremental first-visit Monte Carlo estimate of V with $\alpha = \frac{2}{3}$?*
4. *What is the incremental every-visit Monte Carlo estimate of V with $\alpha = \frac{2}{3}$?*

4.4 Monte Carlo Off Policy Evaluation

In the section above, we discussed the case where we are able to obtain many realizations of G_t under the policy π that we want to evaluate. However, in many costly or high stakes situations, we aren't able to obtain rollouts of G_t under the policy that we wish to evaluate. For example, we may have data associated with one medical policy, but want to determine the value of a different medical policy. In this section, we describe Monte Carlo off policy evaluation, which is a method for using data taken from one policy to evaluate a different policy.

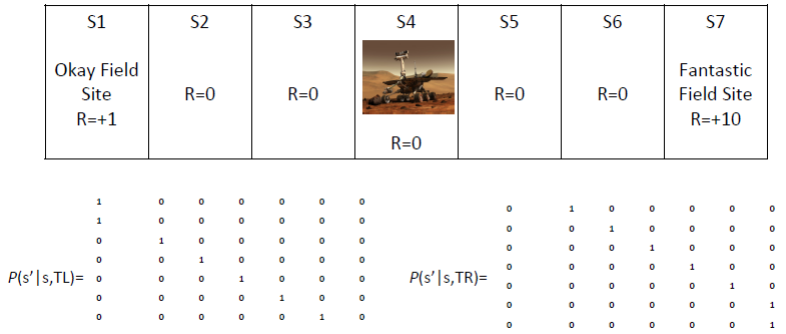

Figure 3: Mars Rover Markov Decision Process with actions Try Left (TL) and Try Right (TR)

4.4.1 Importance Sampling

The key ingredient of off policy evaluation is a method called importance sampling. The goal of importance sampling is to estimate the expected value of a function $f(x)$ when x is drawn from distribution q using only the data $f(x_1), \ldots, f(x_n)$, where x_i are drawn from a different distribution p. In summary, given $q(x_i), p(x_i), f(x_i)$ for $1 \leq x_i, \leq n$, we would like an estimate for $\mathbb{E}_{x\sim q}[f(x)]$. We can do this via the following approximation:

$$\mathbb{E}_{x\sim q}[f(x)] = \int_x q(x)f(x)dx \tag{8}$$

$$= \int_x p(x)\left[\frac{q(x)}{p(x)}f(x)\right]dx \tag{9}$$

$$= \mathbb{E}_{x\sim p}\left[\frac{q(x)}{p(x)}f(x)\right] \tag{10}$$

$$\approx \sum_{i=1}^{n}\left[\frac{q(x_i)}{p(x_i)}f(x_i)\right]. \tag{11}$$

The last equation gives us the **importance sampling estimate** of f under distribution q using samples of f under distribution p. Note that the first step only holds if $q(x)f(x) > 0$ implies $p(x) > 0$ for all x.

4.4.2 Importance Sampling for Off Policy Evaluation

We now apply the general result of importance sampling estimates to reinforcement learning. In this instance, we want to approximate the value of state s under policy π_1, given by $V^{\pi_1}(s) = \mathbb{E}[G_t|s_t = s]$, using n histories $h_1, \ldots, h_n$ generated under policy π_2. Using the importance sampling estimate result gives us that

$$V^{\pi_1}(s) \approx \frac{1}{n}\sum_{j=1}^{n}\frac{p(h_j|\pi_1, s)}{p(h_j|\pi_2, s)}G(h_j), \tag{12}$$

where $G(h_j) = \sum_{t=1}^{L_j-1}\gamma^{t-1}r_{j,t}$ is the total discounted sum of rewards for history h_j.

Now, for a general policy π, we have that the probability of experiencing history h_j under policy π is

$$p(h_j|\pi, s = s_{j,1}) = \prod_{t=1}^{L_j-1} p(a_{j,t}|s_{j,t})p(r_{j,t}|s_{j,t}, a_{j,t})p(s_{j,t+1}|s_{j,t}, a_{j,t}) \tag{13}$$

$$= \prod_{t=1}^{L_j-1} \pi(a_{j,t}|s_{j,t})p(r_{j,t}|s_{j,t}, a_{j,t})p(s_{j,t+1}|s_{j,t}, a_{j,t}), \tag{14}$$

where L_j is the length of the jth episode. The first line follows from looking at the three components of each transition. The components are:

1. $p(a_{j,t}|s_{j,t})$ - probability we take action $a_{j,t}$ at state $s_{j,t}$
2. $p(r_{j,t}|s_{j,t}, a_{j,t})$ - probability we experience reward $r_{j,t}$ after taking action $a_{j,t}$ in state $s_{j,t}$
3. $p(s_{j,t+1}|s_{j,t}, a_{j,t})$ - probability we transition to state $s_{j,t+1}$ after taking action $a_{j,t}$ in state $s_{j,t}$

Now, combining our importance sampling estimate for $V^{\pi_1}(s)$ with our decomposition of the history probabilities, $p(h_j|\pi, s = s_{j,1})$, we get that

$$V^{\pi_1}(s) \approx \frac{1}{n}\sum_{j=1}^{n} \frac{p(h_j|\pi_1, s)}{p(h_j|\pi_2, s)} G(h_j) \tag{15}$$

$$= \frac{1}{n}\sum_{j=1}^{n} \frac{\prod_{t=1}^{L_j-1} \pi_1(a_{j,t}|s_{j,t})p(r_{j,t}|s_{j,t}, a_{j,t})p(s_{j,t+1}|s_{j,t}, a_{j,t})}{\prod_{t=1}^{L_j-1} \pi_2(a_{j,t}|s_{j,t})p(r_{j,t}|s_{j,t}, a_{j,t})p(s_{j,t+1}|s_{j,t}, a_{j,t})} G(h_j) \tag{16}$$

$$= \frac{1}{n}\sum_{j=1}^{n} G(h_j) \prod_{t=1}^{L_j-1} \frac{\pi_1(a_{j,t}|s_{j,t})}{\pi_2(a_{j,t}|s_{j,t})}. \tag{17}$$

Notice we can now explicitly evaluate the expression without the transition probabilities or rewards since all of the terms involving model dynamics canceled out in the second step of the equation. In particular, we are given the histories h_j, so we can calculate $G(h_j) = \sum_{t=1}^{L_j-1} \gamma^{t-1} r_{j,t}$, and we know the two policies π_1 and π_2, so we can also evaluate the second term.

4.5 Temporal Difference (TD) Learning

So far, we have two methods for policy evaluation: dynamic programming and Monte Carlo. Dynamic programming leverages bootstrapping to help us get value estimates with only one backup. On the other hand, Monte Carlo samples many histories for many trajectories which frees us from using a model. Now, we introduce a new algorithm that combines bootstrapping with sampling to give us a second model-free policy evaluation algorithm.

To see how to combine sampling with bootstrapping, let's go back to our incremental Monte Carlo update:

$$V^{\pi}(s_t) \leftarrow V^{\pi}(s_t) + \alpha(G_t - V^{\pi}(s_t)). \tag{18}$$

Recall that G_t is the return after rolling out the policy from time step t to termination starting at state s_t. Let's now replace G_t with a Bellman backup like in dynamic programming. That is, let's replace G_t with $r_t + \gamma V^{\pi}(s_{t+1})$, where r_t is a sample of the reward at time step t and $V^{\pi}(s_{t+1})$ is our current estimate of the value at the next state. Making this substitution gives us the TD-learning update

$$V^{\pi}(s_t) \leftarrow V^{\pi}(s_t) + \alpha(r_t + \gamma V^{\pi}(s_{t+1}) - V^{\pi}(s_t)). \tag{19}$$

The difference

$$\delta_t = r_t + \gamma V^{\pi}(s_{t+1}) - V^{\pi}(s_t) \tag{20}$$

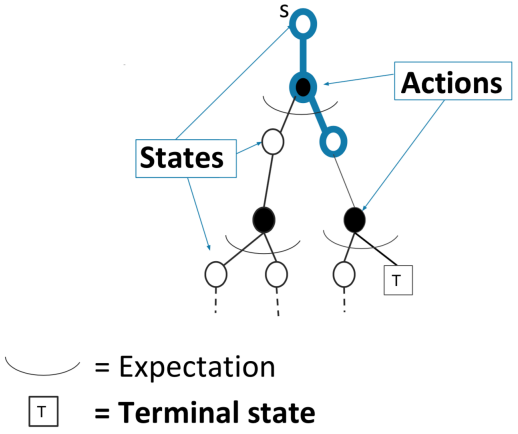

Figure 4: Backup diagram for the TD Learning policy evaluation algorithm.

is commonly referred to as the **TD error**, and the sampled reward combined with the bootstrap estimate of the next state value,

$$r_t + \gamma V^\pi(s_{t+1}), \tag{21}$$

is referred to as the **TD target**. The full TD learning algorithm is given in Algorithm 6. We can see that using this method, we update our value for $V^\pi(s_t)$ directly after witnessing the transition (s_t, a_t, r_t, s_{t+1}). In particular, we don't need to wait for the episode to terminate like in Monte Carlo.

Algorithm 6 TD Learning to evaluate policy π

procedure TDLEARNING(step size α, number of trajectories n)
  For all states s, $V^\pi(s) \leftarrow 0$
  while $n > 0$ **do**
    Begin episode E at state s
    while $n > 0$ and episode E has not terminated **do**
      $a \leftarrow$ action at state s under policy π
      Take action a in E and observe reward r, next state s'
      $V^\pi(s) \leftarrow V^\pi(s) + \alpha(R + \gamma V^\pi(s') - V^\pi(s))$
      $s \leftarrow s'$
  return V^π

We can again examine this algorithm via a backup diagram as shown in Figure 4. Here, we see via the blue line that we sample one transition starting at s, then we estimate the value of the next state via our current estimate of the next state to construct a full Bellman backup estimate.

There is actually an entire spectrum of ways we can blend Monte Carlo and dynamic programming using a method called TD(λ). When $\lambda = 0$, we get the TD-learning formulation above, hence giving us the alias TD(0). When $\lambda = 1$, we recover Monte Carlo policy evaluation, depending on the formulation used. When $0 < \lambda < 1$, we get a blend of these two methods. For a more thorough treatment of TD(λ), we refer the interested reader to Sections 7.1 and 12.1-12.5 of Sutton and Barto [1] which detail n-step TD learning and TD(λ)/eligibility traces, respectively.

Exercise 4.2. *Consider again the Mars Rover example in Figure 3. Suppose that our estimate for the value of each state is currently 0. If we experience the history*

$$h = (S3, TL, +0, S2, TL, +0, S2, TL, +0, S1, TL, +1, terminal),$$

then:

1. *What is the TD(0) estimate of V with $\alpha = 1$?*
2. *What is the TD(0) estimate of V with $\alpha = \frac{2}{3}$?*

4.6 Summary of Methods Discussed

In this lecture, we re-examined policy evaluation using dynamic programming from last lecture, and we introduced two new methods for policy evaluation, namely Monte Carlo evaluation and Temporal Difference (TD) learning.

First, we motivated the introduction of Monte Carlo and TD-Learning by noting that dynamic programming relied on a model of the world. That is, we needed to feed our dynamic programming policy evaluation algorithm with the rewards and transition probabilities of the domain. Monte Carlo and TD-Learning are both free from this constraint, making them model-free methods.

In Monte Carlo policy evaluation, we generate many histories and then average the returns over the states that we encounter. In order for us to generate these histories in a finite amount of time, we require the domain to be episodic - that is, we need to ensure that each history that we observe terminates. In both dynamic programming and temporal difference learning, we only backup over one transition (we only look one step ahead in the future), so termination of histories is not a concern, and we can apply these algorithms to non-episodic domains.

On the flip side, the reason we are able to backup over just one transition in dynamic programming and TD learning is because we leverage the Markovian assumption of the domain. Furthermore, Incremental Monte Carlo policy evaluation, described in Algorithms 4 and 5 can be utilized in non-Markovian domains.

In all three methods, we converge to the true value function. Last lecture, we proved this result for dynamic programming by using the fact that the Bellman backup operator is a contraction. We saw in today's lecture that Monte Carlo policy evaluation converges to the policy's value function due to the law of large numbers. TD(0) converges to the true value as well, which we will look at more closely in the next section on batch learning.

Because we are taking an average over the true distribution of returns in Monte Carlo, we obtain an unbiased estimator of the value at each state. On the other hand, in TD learning, we bootstrap the next state's value estimate to get the current state's value estimate, so the estimate is biased by the estimated value of the next state. Further discussions on this can be found in section 6.2 of Sutton and Barto [1].

The variance of Monte Carlo evaluation is relatively higher than TD learning because in Monte Carlo evaluation, we consider many transitions in each episode with each transition contributing variance to our estimate. On the other hand, TD learning only considers one transition per update, so we do not accumulate variance as quickly.

Finally, Monte Carlo is generally more data efficient than TD(0). In Monte Carlo, we update the value of a state based on the returns of the entire episode, so if there are highly positive or negative rewards many trajectories in the future, these rewards will still be immediately incorporated into our update of the value of the state. On the other hand in TD(0), we update the value of a state using only the reward in the current step and some previous estimate of the value at the next state. This means that if there are highly positive or negative rewards many trajectories in the future, we will only incorporate these into the current state's value update when that reward has been used to update the bootstrap estimate of the next state's value. This means that if a highly rewarding episode has length L, then we may need to experience that episode L times for the information of the highly rewarding episode to travel all the way back to the starting state.

In Table 1, we summarize the strengths and limitations of each method discussed here.

	Dynamic Programming	Monte Carlo	Temporal Difference
Model Free?	No	Yes	Yes
Non-episodic domains?	Yes	No	Yes
Non-Markovian domains?	No	Yes	No
Converges to true value	Yes	Yes	Yes
Unbiased Estimate	N/A	Yes	No
Variance	N/A	High	Low

Table 1: Summary of methods in this lecture.

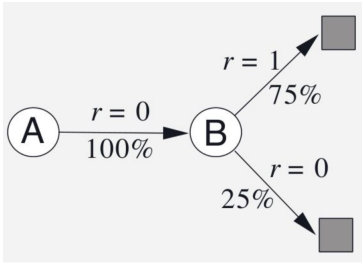

Figure 5: Example 6.4 from Sutton and Barto [1].

4.7 Batch Monte Carlo and Temporal Difference

We now look at the batch versions of the algorithms in today's lecture, where we have a set of histories that we use to make updates many times. Before looking at the batch cases in generality, let's first look at Example 6.4 from Sutton and Barto [1] to more closely examine the difference between Monte Carlo and TD(0). Suppose $\gamma = 1$ and we have eight histories generated by policy π, take action $act1$ in all states:

$$h_1 = (A, act1, +0, B, act1, +0, terminal)$$
$$h_j = (B, act1, +1, terminal) \text{ for } j = 2, \ldots, 7$$
$$h_8 = (B, act1, +0, terminal).$$

Then, using either batch Monte Carlo or TD(0) with $\alpha = 1$, we see that $V(B) = 0.75$. However, using Monte Carlo, we get that $V(A) = 0$ since only the first episode visits state A and has return 0. On the other hand, TD(0) gives us $V(A) = 0.75$ because we perform the update $V(A) \leftarrow r_{1,1} + \gamma V(B)$. Under a Markovian domain like the one shown in Figure 5, the estimate given by TD(0) makes more sense.

In this section, we consider the batch cases of Monte Carlo and TD(0). In the batch case, we are given a batch, or set of histories $h_1, \ldots, h_n$, which we then feed through Monte Carlo or TD(0) many times. The only difference from our formulations before is that we only update the value function after each time we process the entire batch. Thus in TD(0), the bootstrap estimate is updated only after each pass through the batch.

In the Monte Carlo batch setting, the value at each state converges to the value that minimizes the mean squared error with the observed returns. This follows directly from the fact that in Monte Carlo, we take an average over returns at each state, and in general, the MSE minimizer of samples is precisely the average of the samples. That is, given samples $y_1, \ldots, y_n$, the value $\sum_{i=1}^{n}(y_i - \hat{y})^2$ is minimized for $\hat{y} = \sum_{i=1}^{n} y_i$. We can also see this in the example at the beginning of the section. We get that $V(A) = 0$ for Monte Carlo because this is the only history visiting state A.

In the TD(0) batch setting, we do not converge to the same result as in Monte Carlo. In this case, we

converge to the value V^π that is the value of policy π on the maximum likelihood MDP model where

$$\hat{P}(s'|s,a) = \frac{1}{N(s,a)} \sum_{j=1}^{n} \sum_{t=1}^{L_j - 1} \mathbb{1}(s_{j,t} = s, a_{j,t}, s_{j,t+1} = s') \tag{22}$$

$$\hat{r}(s,a) = \frac{1}{N(s,a)} \sum_{j=1}^{n} \sum_{t=1}^{L_j - 1} \mathbb{1}(s_{j,t} = s, a_{j,t}) r_{j,t}. \tag{23}$$

In other words, the maximum likelihood MDP model is the most naive model we can create based on the batch - the transition probability $\hat{P}(s'|s,a)$ is the fraction of times that we see the transition (s,a,s') after we take action a at state s in the batch, and the reward $\hat{r}(s,a)$ is the average reward experienced after taking action a at state s in the batch.

We also see this result in the example from the beginning of the section. In this case, our maximum likelihood model is

$$\hat{P}(B|A, act1) = 1 \tag{24}$$

$$\hat{P}(terminal|B, act1) = 1 \tag{25}$$

$$\hat{r}(A, act1) = 0 \tag{26}$$

$$\hat{r}(B, act1) = 0.75. \tag{27}$$

This gives us $V^\pi(A) = 0.75$, like we stated before.

The value function derived from the maximum likelihood MDP model is known as the **certainty equivalence estimate**. Using this relationship, we have another method for evaluating the policy. We can first compute the maximum likelihood MDP model using the batch. Then we can compute V^π using this model and the model-based policy evaluation methods discussed in last lecture. This method is highly data efficient but is computationally expensive because it involves solving the MDP which takes time $O(|S|^3)$ analytically and $(|S|^2|A|)$ via dynamic programming.

Exercise 4.3. *Consider again the Mars Rover example in Figure 3. Suppose that our estimate for the value of each state is currently 0. If our batch consists of two histories*

$$h_1 = (S3, TL, +0, S2, TL, +0, S1, TL, +1, terminal)$$
$$h_2 = (S3, TL, +0, S2, TL, +0, S2, TL, +0, S1, TL, +1, terminal)$$

and our policy is TL, then what is the certainty equivalence estimate?

References

[1] Sutton, Richard S. and Andrew G. Barto. *Introduction to Reinforcement Learning*. 2nd ed., MIT Press, 2017. Draft. http://incompleteideas.net/book/the-book-2nd.html.