# Lecture 9: Learning Value Functions

## 1 Introduction

**Review: Value Functions**

- What is $Q(s,a)$ and $V(s)$? How do these quantities depend on the policy?

$$Q^\pi(s,a) = \mathbb{E}_\pi \left[ \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 = s, a_0 = a \right] \tag{1}$$

$$V^\pi(s) = \mathbb{E}_\pi \left[ \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 = s \right]. \tag{2}$$

- Bellman equations.

$$Q^\pi(s,a) = r(s,a) + \gamma \mathbb{E}_{p(s'|s,a)\pi(a'|s')}[Q^\pi(s',a')] \qquad \text{(Bellman expectation eq.)}$$

$$Q^\star(s,a) = r(s,a) + \gamma \mathbb{E}_{p(s'|s,a)}[\max_{a'} Q^\star(s',a')]. \qquad \text{(Bellman optimality eq.)}$$

- What is value iteration and policy iteration. Note that both require a model

The fact that policy and value iteration require a model makes them challenging to apply in practice, especially to high-dimensional problems. E.g., if you have continuous states, how can you do the step of value iteration where you enumerate over all the states?

In today's class, we'll see how we can lift this assumption of the model. We will replace the model with data. Because we're not using a model, today's methods are all known as **model-free** methods.

**Learning objectives for today** At the end of this lecture, you will be able to

1. learn Q-functions and value functions using MC regression
2. improve the sample efficiency of these methods by using SARSA.
3. estimate the Q-values for a different policy using expected SARSA.

### 1.1 Assumptions for today.

1. **Tabular states and actions.** Today's class is all going to be about estimating value functions. For today, we'll restrict ourselves to settings with a relatively small number of states and actions, so that we can represent the value functions as big table. $V(s)$ will be a big vector with length $|\mathcal{S}|$, and $Q(s,a)$ is a matrix with shape $|\mathcal{S}| \times |\mathcal{A}|$:

$$Q = \begin{pmatrix} \ddots & & \\ & Q(s=i, a=j) & \\ & & \ddots \end{pmatrix}. \tag{3}$$

2. No model. This lifts the limitation of value iteration

## 2 Monte Carlo Estimation of Value Functions

The goal for today is to design an algorithm that takes trajectories as input and fills in the entries of the Q-value table. Recall that each entry of the Q-value table is the expected return, given that you started at state $s$ and took action $a$.

**Thought experiment: if you had a simulator.** If you had access to a simulator, one way to estimate $Q(s,a)$ would be to reset the environment to state $s$, take action $a$, and then see how much reward policy $\pi$ got in the future. Repeating this a handful of times, you could average over the results and then fill in entry $Q(s,a)$ in your table with this average.

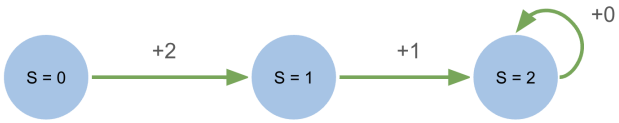
Today, we'll see how we can do this without access to a simulator.

### 2.1 Monte Carlo Estimation

So we're given as input $\{(s_0, a_0, r_0, s_1, a_1, r_1, \cdots)\}$ and want to fill in the entries in our Q value table.

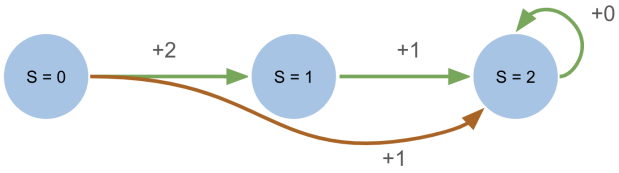
One approach to doing this, typically known as the **Monte Carlo** approach, is to look at the state-action pairs that you have visited, and see what the future returns were afterwards.

As an example, let's say that you had the trajectory below, where there is a single action $a = 0$:




To compute the Q-value $Q(s = 0, a = 0)$, we'd simply count up the future rewards after that state-action pair: $\gamma^0 2 + \gamma^1 1 + \gamma^2 0 + \gamma^3 0 + \cdots = 2 + \gamma$. Similarly, we'd get $Q(s = 1, a = 0) = \gamma$ and $Q(s = 2, a = 0) = 0$.

As another example, let's say that we have two trajectories. They both start with the same $(s = 0, a = 0)$ pair, but because the dynamics are stochastic the next state can be different:




The Q-values are:

$$Q(s = 2, a = 0) = 0 \tag{4}$$

$$Q(s = 1, a = 0) = \gamma^0 1 + \gamma^1 0 + \cdots = 1 \tag{5}$$

$$Q(s = 0, a = 0) = \frac{1}{2}\left(\gamma^0 2 + \gamma^1 1 + \gamma^2 0 + \cdots\right) + \frac{1}{2}\left(\gamma^0 1 + \gamma^1 0 + \cdots\right) \tag{6}$$

$$= \frac{1}{2}(2 + \gamma + 1) = \frac{3+\gamma}{2}. \tag{7}$$

**Limitations.** The main challenge with these MC approaches is that getting a reliable estimate of the Q-values requires that you have good coverage over all *trajectories*. However, the number of trajectories can grow exponentially with the number of states. For example, if you went on a sightseeing walk in Manhattan (roughly a $12 \times 150$ grid) and wanted to traverse it north to south picking one of the 12 avenues for each of the 149 street-to-street steps, the number of different paths you could take would be $12^{149} \simeq 6.28 \times 10^{160}$, which is orders of magnitude larger than the number of atoms in the universe (around $10^{80}$).

Because coverage is typically poor, these MC approaches typically have high *variance*: if you collected data again and came up with another estimate, it'd likely differ quite a bit from your original estimate. This is especially true in settings where your policy or dynamics are stochastic.

## 3 SARSA [2]

SARSA[1] is a method for estimating the Q values directly from data. It is a form of policy evaluation (as is value iteration). Like MC regression, SARSA will estimate the Q values of the policy that collected the data, so it will return $Q^\pi$, not $Q^\star$.

Given $(s, a, r, s', a')$ collected from $\pi(a \mid s)$, find $Q^\pi$. Ideally, we want to perform the update

$$Q(s,a) \leftarrow r(s,a) + \gamma \mathbb{E}_{p(s'|s,a)\pi(a'|s')}[Q(s',a')]. \qquad \text{(idealized update)}$$

We can approximate the expectation with a single sample estimate, which would give us the following:

$$Q(s,a) \leftarrow r(s,a) + \gamma Q(s',a'). \qquad \text{(doesn't quite work.)}$$

However, these updates won't converge on their own. For example, let's say that you have one transition $(s, a, r = 0, s' = 0, a')$ where $Q(s' = 0, a') = 0$ and another transition $(s, a, r = 0, s' = 1, a')$ where $Q(s' = 1, a') = 1$. If you did the updates above, you'd oscillate between $Q(s,a) = 0$ and $Q(s,a) = 1$. Since we'd like to compute the *expectation* over the next state $s'$, we need to somehow average between these values. Thus, we will do a "gradual" update to the Q values, taking an average of the current value and the updated value:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \tag{8}$$

$$\alpha\left(r(s,a) + \gamma Q(s',a')\right). \tag{9}$$

The "label" that we've used above is sometimes called the *TD target*. One subtle thing to note is that, even thought the label itself depends on $Q$, it is treated as a constant in this optimization problem. There are several explanations for why we do this. One explanation is that we really want to to an *assignment*, updating our new Q values with our old Q values (and not vice-versa). See [1].

**Gradient descent perspective.** One way of interpreting this is as doing gradient descent where the prediction is $Q(s,a)$, the label is $y = r(s,a) + \gamma Q(s',a')$, and the learning rate is $\alpha$:

$$\mathcal{L} = \frac{1}{2}(Q(s,a) - y)^2 \qquad \text{where} \quad y = r(s,a) + \gamma Q(s',a') \tag{10}$$

$$\frac{d\mathcal{L}}{dQ(s,a)} = Q(s,a) - y \tag{11}$$

$$Q(s,a) - \alpha\frac{d\mathcal{L}}{dQ(s,a)} = Q(s,a) - \alpha(Q(s,a) - y) \tag{12}$$

$$= (1-\alpha)Q(s,a) - \alpha y \tag{13}$$

$$= (1-\alpha)Q(s,a) - \alpha r(s,a) + \gamma Q(s',a'). \tag{14}$$

**Temporal Difference Learning** SARSA is an example of *temporal difference* (TD) learning. The key idea is that you can use predictions at one time step to update your predictions at another time step. While by far the most common use-case of TD learning today is for estimating value functions, the original TD learning paper [3] was mostly a paper about prediction. What other prediction problems might be solved via TD learning?

## 4 Expected SARSA

Expected SARSA is another method for policy evaluation, which will address some of the limitations that we've seen so far. Like MC regression and SARSA, it will estiamte the Q values for a particular policy (not necessarily the optimal policy). Like SARSA, it will just rely on transitions, rather than on entire trajectories (like MC regression). Compared with SARSA, expected SARSA will have two key advantages:

[1] The term "SARSA" was coined 2 years later in footnote. [4]

(a) Example 1 (b) Example 2

Figure 1: **SARSA enables stitching.** Temporal difference methods can be more sample efficient than Monte Carlo methods because they can recombine pieces of data, effectively inflating the size of the training data. *(Left)* SARSA can stitch examples like this, where trajectories share a state-action pair (middle segment. *(Right)* Expected SARSA can additionally stitch examples like this, where trajectories share a state but need not share a state-action pair.

1. More sample efficient because we can average over future actions (lower variance).
2. The fact that we're now sampling the actions at the next time step means that we can estimate the values of a policy that is different from that which collected the data. This is called *off-policy evaluation*.

**Algorithm.** The algorithm for expected SARSA is almost identical to SARSA, with two key differences:

1. Inputs: the inputs to the algorithm are $(s, a, r, s')$ pairs, along with a policy $\pi(a \mid s)$. This policy can be different from the one that collected the data.
2. TD target: For computing the TD target, we will use $a' \sim \pi(a' \mid s')$, rather than using the action observed in the dataset.

We can thus write expected SARSA as follows:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\left(r(s,a) + \gamma Q(s',a')\right) \tag{15}$$

$$\text{where} \quad y = r(s,a) + \gamma Q(s', a' \sim \pi(a' \mid s')). \tag{16}$$

**Off-policy evaluation.** Note the dependence on the policy: because we sample $a' \sim \pi(a' \mid s')$, we end up learning $Q^\pi$. If we used a different policy for sampling the next actions, we'd get the Q values for a different policy. This is a really important and frankly surprising property. This means that you can estimate how good one policy is *without actually collecting data from that policy*. All you need is to sample some actions from that policy.

One specific form of off-policy evaluation is when you estimate the values of $\pi^\star$. If you had $\pi^\star$, you could run expected SARSA to get $Q^\star$. However, if you already had $\pi^\star$, it's unclear why you would want to compute $Q^\star$. Next class we'll see how you can modify expected SARSA to compute $Q^\star$ when you don't know what $\pi^\star$ is.

**Expected SARSA enables stronger stitching.** Like SARSA, one of the key benefits of expected SARSA over MC regression is that it can stitch together pieces of data. The stitching performed by expected SARSA is slightly stronger than that performed by SARSA. For example, consider the data shown in Fig. 1 (right). Regular SARSA would be unable to stitch together these three trajectories because they do not share a common $(s, a)$ pair. However, if using expected SARSA, we only need that they share a common state.

## 5 (Time Permitting) Monte Carlo Regression

Throughout today's class, we've been assuming that the state and actions are finite, so that we can represent the Q values as as big table. However, for many practical problems, the states and/or actions are continuous or very large (e.g., chess), so we can't represent them in a big table. There's a second, related challenge in this setting: we may often want to know the Q-value for a state-action pair that we haven't exactly been to before. How can

we compare a new state-action pair to ones we've seen before to make a good guess for what the future value is? In the remainder of today's class, we'll start to talk about how we can deal with this. We'll focus on an extension of the Monte Carlo approach above; future classes will look at ways of extending the SARSA-based approaches.

The main idea for handling non-tabular states and actions is to use machine learning, aiming to find patterns in our mapping from state-action pairs to values. Precisely, we will learn a neural network $Q_\theta(s, a)$ that takes as input a state-action pair (e.g., concatenate them) and outputs a single scalar value.

When doing machine learning, we need to think about the data and the loss function. The data will be $\{(x = (s, a), y = \sum_t \gamma^t r_t)\}$ generated in the same way as the Monte Carlo method: for each state-action pair that we've visited, we'll treat the label as the actually-observed discounted future rewards. The loss function will be the MSE:

$$\min_Q \frac{1}{|\mathcal{D}|} \sum_{(s,a,R)} (Q(s, a) - R)^2 \tag{17}$$

$$\tag{18}$$

Consistency of the MSE estimator:

$$\frac{d}{dQ(s, a)} = Q(s, a) - \mathbb{E}_{p(R|s,a)}[R] = 0 \tag{19}$$

$$\implies Q(s, a) = \mathbb{E}[R]. \tag{20}$$

Note that the Monte Carlo estimator that we saw above is effectively a special case of this, where the function $Q(s, a)$ is implemented as a big lookup table.

**Takeaways.** By learning a neural network that makes state-action pairs to future returns, we are able to (1) handle higher-dimensional states and actions, and *(2)* make predictions for state-action pairs which haven't been seen in our training data. It's worth reflecting on what's happening "under the hood": like other applications of deep learning, we are automatically identifying patterns that are correlated with the future rewards. In the same way that an image classifier might learn to identify features like "whiskers" as being predictive of certain types of animals, so our Q-function will identify features of the state that are predictive of the future rewards.

## 6 Conclusion

Today's class was about learning value functions – algorithms (i.e., update rules) that we could use to estimate the value functions. The key idea behind the SARSA method was to exploit the Bellman expectation equations, which we saw last time. The very end of lecture started to allude to how we can learn value functions in settings with high-dimensional states and actions; this is a story that we'll continue in the coming lectures.

Broadly, this course started by focusing on policies (imitation learning, policy gradient), but we've shifted our attention to value functions as a "helper" that will enable us to learn better policies. As we saw last time with policy iteration, in simple settings you can directly use the value function to produce the optimal policy. Next time we'll see how we can do the same trick in settings where we don't know the model, and the following lecture will lift the tabular assumption. Then, after the midterm, we'll put all the pieces together to see how we can solve high-dimensional tasks with continuous states and actions without requiring a model. These will be the algorithms that are most commonly used today (e.g., PPO, TD3).

## References

[1] Baird, L. et al. (1995). Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the twelfth international conference on machine learning*, pages 30–37.

[2] Rummery, G. A. and Niranjan, M. (1994). *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, UK.

[3] Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine learning*, 3:9–44.

[4] Sutton, R. S. (1995). Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in neural information processing systems*, 8.