B9140 Dynamic Programming & Reinforcement Learning Lecture 5 - 09 Oct 2017

Lecture 5

Lecturer: Daniel Russo *Scribe: Sharon Huang, Wenjun Wang, Jalaj Bhandari*

1 Change of notation

We introduce some change of notation with respect to the previous lectures:

- Maximizing reward instead of minimizing costs.
- Let (s_k, a_k, r_k) denote the (State, Action, Reward) at step k
- Work with the value function, $V(\cdot)$, instead of the cost-to-go function $J(\cdot)$

2 Batch methods for Policy Evaluation

Consider the set-up where we fix a policy, μ, and generate data following μ (episodes or otherwise). Given this data, we want to estimate the value function for every state. We introduce two methods for policy evaluation:

1. Look up table: where we store an estimate of the value to go for each individual state. Typically, the amount of data required scales at least linearly with the number of states.
2. Value function approximation: motivated by practical application where the state space is large (think exponentially large), and we don't want to store such large value functions. Typically the amount of data required to estimate e.g. a linearly parameterized value function scales with the dimension of the approximation rather than the number of states.

2.1 Look up table:

Let us consider an episodic MDP with state space: $S \cup \{t\}$, where $\{t\}$ is *terminal* state and is costless, absorbing. We assume the terminal state, t, can be reached with probability 1 under policy μ, implying that $V_\mu(t) = 0$; and that the initial states are drawn from some (unknown) distribution $\alpha(s)$. We have a batch of data organized by episodes $n \in \{1, 2, \ldots, N\}$; for each episode n, we observe: $\big[s_0^{(n)}, r_0^{(n)}, s_1^{(n)}, \ldots, s_{\tau_n}^{(n)}, r_{\tau_n}^{(n)}, t\big]$ with τ_n being the number of periods in episode n. Our goal is to estimate $V_\mu(s)$, the value function under policy μ for any state s.

2.1.1 (First Visit) Monte Carlo Value Prediction:

Suppose that state s is visited in episode n for the first time in period k. Then, by definition of value function, we have:

$$V_\mu(s) = \mathbb{E}\Big[\sum_{i=k}^{\tau_n} r_i^{(n)}\Big]$$

We can use a noisy estimate of this expectation to approximate V_μ. Algorithm (1) provides a summary. We can similarly define an *every visit* Monte Carlo, where we can take into account the accumulated rewards for every visit to state s. However, this approach will be biased.

Algorithm 1 (First visit) Monte Carlo value prediction:

```
1: for $n \in \{1, 2, \ldots, N\}$ do
2:     for every state $s$ visited in episode $n$ do
3:         Let $k$ be the first time state $s$ is visited in episode $n$
4:         $G^n(s) = \sum_{i=k}^{\tau_n} r_i^{(n)}$                ▷ noisy sample of $V_\mu(s)$
5:     end for
6: end for
7: return $\hat{V}_\mu(s) = \frac{1}{N}\sum_{n=1}^{N} G^n(s) \quad \forall\ s$
```

2.1.2 Sutton & Barto: Example 6.4

Suppose we have observed the following 8 episodes:

$$(A, 0, B, 0) \quad (B, 1) \quad (B, 1) \quad (B, 1) \quad (B, 0) \quad (B, 1) \quad (B, 1) \quad (B, 1)$$

We have the Monte Carlo estimates of A and B as: $\hat{V}_{mc}(B) = \frac{6}{8} = \frac{3}{4}$, $\hat{V}_{mc}(A) = 0$. However, since we only visited state A once, it makes sense to have the value function of A to be

$$\hat{V}_{TD}(A) = 0 + \hat{V}(B)$$

if we assume Markovian transitions. To get some intuition, we can think of this estimate in terms of data augmentation/bootstrapping: we expand our data with trajectories we didn't observe but believe have equal probability of occurring. For instance, we can expand the data of the above example to be:

$$\begin{array}{lcl}
(A, 0, B) & \text{followed by} & (B, 0, t) \\
(A, 0, B) & \text{followed by} & (B, 1, t) \\
(A, 0, B) & \text{followed by} & (B, 1, t) \\
(A, 0, B) & \text{followed by} & (B, 1, t) \\
(A, 0, B) & \text{followed by} & (B, 1, t) \\
(A, 0, B) & \text{followed by} & (B, 1, t) \\
(A, 0, B) & \text{followed by} & (B, 1, t) \\
(A, 0, B) & \text{followed by} & (B, 0, t)
\end{array}$$

The Monte Carlo estimate under this bootstrapped dataset matches $\hat{V}_{TD}$.

2.1.3 Temporal difference and Fitted Value Iteration:

For the Temporal difference method, we split our dataset $\left[s_0^{(n)}, r_0^{(n)}, s_1^{(n)}, \ldots, s_{\tau_n}^{(n)}, r_{\tau_n}^{(n)}, t\right]$ into tuples: $\Big[(s_0^{(n)}, r_0^{(n)}, s_1^{(n)})$, $(s_1^{(n)}, r_1^{(n)}, s_2^{(n)}), \ldots,\ (s_{\tau_n}^{(n)}, r_{\tau_n}^{(n)}, t)\Big]$. Let H be the set of all tuples and H_s be the set of tuples originating from s:

$$\begin{aligned}
H &= \left\{(s_k^{(n)}, r_k^{(n)}, s_{(k+1)}^{(n)}) \mid n \le N,\ k \le \tau_n\right\} \\
H_s &= \left\{(s, r_k^{(n)}, s_{(k+1)}^{(n)}) \mid n \le N,\ k \le \tau_n\right\}
\end{aligned}$$

We solve an empirical Bellman Equation by letting:

$$V(s) = \begin{cases} 0 & \text{if} \quad s = t \\ \frac{1}{|H_s|}\sum_{(s,r,s')\in H_s}(r + V(s')) & \forall\ s \ne t \end{cases}$$

It's worth thinking about cases when Temporal difference (TD) method would be useful: TD method uses the Markovian assumption and can therefore leverage on past experiences when we see a completely new state. In that sense it is a lot more data efficient and can help to reduce variance. Below, we look at three examples where this is the case.

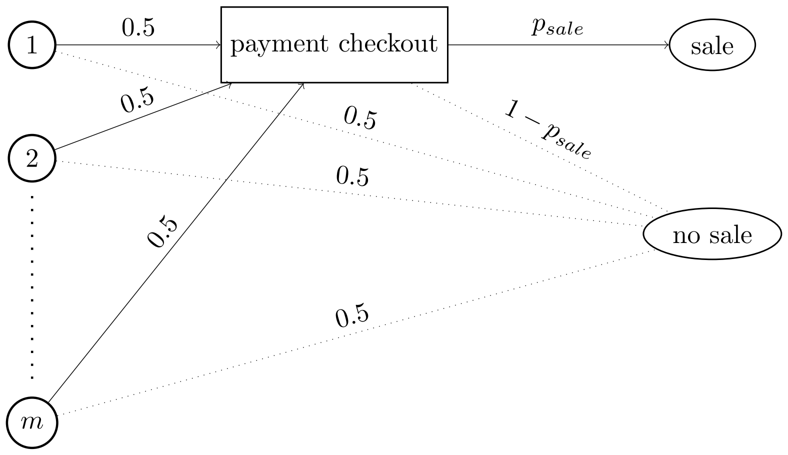

Figure 1: Display advertisement set-up

Driving home - Excercise 6.2 of Sutton&Barto: Suppose you have lots of experience driving home from work. Then you move to a new building and a new parking lot (but you still enter the highway at the same place). Now you are starting to learn predictions for the new building. Can you see why TD updates are likely to be much better, at least initially, in this case?

Basketball: Suppose the Warriors (a basketball team) are trying to evaluate a new play designed to lead to a Steph Curry 3-point shot. For simplicity, let us only consider outcomes where every play ends with a Steph Curry 3-pointer; so there are two outcomes: he either make the shot or misses it. Suppose that we know from the start all the formations that each team is planning to run and we want to estimate the value of this play. Also, we observe an intermediate state – the position of Steph Curry and the defenders right before the shot is taken– along with the outcome (reward): whether he makes or misses the shot.

There are two estimators one can use to evaluate this new play i) a Monte Carlo estimator where we run this play many times and compute the average number of points scored or ii) a TD estimator which leverages a huge volume of available data on the on past 3-point shots from diffent positions (by Steph Curry and others) to estimate the odds of a successful shot as a function of the intermediate state. This is likely to be a lower variance estimator than the Monte Carlo one.

Display Advertising: In the previous two examples, TD is more data efficient because it is able to leverage historical data. These examples are not entirely satisfying, however, since the both motivating stories involve using data that was generated by following different policies (e.g. different routes home, or different basketball plays). This raises the question: does TD have advantages when all is generated by following the policy being evaluated. The following example shows it can.

Consider a display advertisement set-up where we have n users and m display ads; with an intermediate state, "payment checkout", and two terminal states, "sale" and "no sale". This is illustrated in Figure 1. We assume that the users are randomly shown an ad, and every user clicks on an ad uniformly with probability $p = 0.5$, in which case they are taken to a checkout page, or the episode ends with no sale (with probability 0.5). From the checkout page, the episode terminates in a sale with a very small probability p_{sale} and otherwise ends in no sale.

Consider the limit $n, m \to \infty$ such that $\frac{m}{n} \to 0$ (i.e. the number of users are much larger than the number of ads) and assume that $p_{sale} > 0$ is extremely small. We want to estimate the value of an initial state, i.e. the value of showing an ad to a user. Here the TD estimator is intuitively better: for Monte Carlo estimator, we have to (implicitly) estimate the conversion probability p_{sale} separately for every ad ($\mathcal{O}(n/2m)$ samples while for the TD method we pool data for all users that reach the checkout page to estimate the conversion probability ($\mathcal{O}(n/2)$ samples). For any state $s \in \{1, 2, \ldots, m\}$, the Monte Carlo estimator of the reward is

$\hat{J}(s) = \frac{\#\text{of sales}}{\#\text{of trials}}$, with the variance of the estimator scaling as

$$\sqrt{\frac{n}{m}}\left(\frac{\hat{J}(s) - p/2}{p/2}\right) \approx \mathcal{N}(0, \frac{2}{p}) \tag{1}$$

whereas for the TD estimator, the variance scales as:

$$\sqrt{\frac{n}{m}}\left(\frac{\hat{J}(s) - p/2}{p/2}\right) \approx \mathcal{N}(0, 1) \tag{2}$$

where $p = p_{sale}$. Thus we see that for this simple example, the TD estimator is analytically better. Generally, even for simple cases, it is hard to evaluate whether TD is better than MC because it is hard to analyze the TD estimator. The TD estimator involves computing $\hat{V}_\mu$ which solves the fixed point $V = \hat{T}V$ where $\hat{T}$ is the empirical Bellman operator. It is also not clear whether using a plug-in estimator for the Bellman operator is the right thing to do. All we can say is that the TD approach is biased but data efficient. One interesting research question would be to do an asymptotic analysis of the TD estimator or to have simple examples where we can compute the variance of the TD estimator and compare that with the Monte Carlo one.

2.2 Value Function Approximation

For working in MDPs with a large state space (for example: think of continuous state space), we need some approximation of the state value function $V(s)$. Consider a parameterized class of value functions, parameterized by a parameter θ such the the true value function $V_\mu(s)$, under some policy μ, is well approximated by $V_\theta(s)$. One example we discussed in the previous lecture is that of a linear model:

$$V_\theta(s) = \phi(s)^\top \theta, \quad \phi(\cdot),\, \theta \in \mathbb{R}^d \tag{3}$$

where $\phi(s)$ is the feature vector of a state and is assumed to be known and θ is the unknown parameter, which is shared across all states, that we want to learn. Note two things: i) θ, depends on the current policy under evaluation, and is sometimes also referred to as the "policy code" and ii) typically some domain knowledge of the underlying problem is required to construct these feature vectors, for example in the tetris game we discussed in an earlier class, we had 20 features: the heights of the 10 columns, 9 inter-column height differences and the maximum height among all the columns.

Notation: Let $V_\mu \in \mathbb{R}^{|S|}$ be some (exponentially) large value function vector that we don't want to store/work with. We want to find a point, V_θ, within the span of some low dimensional features that well approximates V_μ:

$$V_\mu \approx V_\theta = \begin{bmatrix} \phi(s_1)^\top \\ \vdots \\ \phi_k(s_n)^\top \end{bmatrix} \theta = \Phi\theta$$

Some of the very recent successful methods which make up "Deep Reinforcement Learning" use neural networks as function approximators: $V_\theta(s) = f_\theta(s)$, where $f(\cdot)$ is a neural network parameterized by θ. One advantage of linear models are convergence results when used with Monte Carlo or TD methods; such results do not yet exist of neural network based models and is an area of active research. Below, we outline Monte Carlo and TD method with linear function approximation.

2.2.1 (Every Visit) Monte Carlo with Function Approximation:

We can easily extend the Monte Carlo method with function approximation as follows:

Algorithm 2 (Every visit) Monte Carlo value prediction:

```
1: for $n \in \{1, 2, \dots, N\}$ do
2:     Observe $(s_0^{(n)}, r_0^{(n)}, s_1^{(n)}, \dots, s_{\tau_n}^{(n)}, r_{\tau_n}^{(n)}, t)$
3:     for $k = 0, \dots, \tau_n$ do
4:         $G_k^{(n)} = \sum_{i=k}^{\tau_n} r_i^{(n)}$        $\triangleright$ $G_k^{(n)}$ is an unbiased estimator of $V_\mu(s_k^{(n)})$
5:         Append $(s_k^{(n)}, G_k^{(n)})$ to $D$
6:     end for
7: end for
8: return $\hat{\theta} = \arg\min_\theta \frac{1}{|D|} \sum_{(s,G)\in D} \Big(V_\theta(s) - G\Big)^2$        $\triangleright$ Minimize the mean square error
```

Note: For the Monte Carlo approach, the *linear* architecture for approximation is not crucial. As $G_k^{(n)}$ is a noisy estimate of $V_\mu(s_k^{(n)})$, the above algorithm can be reinterpreted as:

$$\hat{\theta} = \arg\min_\theta \ \frac{1}{|D|} \sum_{(s,\cdot)\in D} \Big(V_\theta(s) - V_\mu(s) + \text{noise}\Big)^2 \tag{4}$$

Define $\pi(s) = \mathbb{E}(\sum_{k=0}^{\tau} \mathbb{I}_{\{s_k = s\}})/\mathbb{E}(\tau)$, then as $N \to \infty$, the distribution of states in the dataset D is approximately equal to the long run state visitation frequencies $\pi(s)$ and $\hat{\theta} \to \theta$:

$$\begin{aligned}\hat{\theta} \to \theta &= \arg\min_\theta \ \mathbb{E}_{s\sim\pi(s)} \Big(V_\theta(s) - V_\mu(s)\Big)^2 \\ &= \arg\min_\theta \ \|V_\theta - V_\mu\|_\pi^2\end{aligned}$$

where $\|V\|_\pi := \sqrt{\sum_s \pi(s) V(s)^2}$. For the class of linear function approximators, we can say a bit more: define Π to be the projection operator, i.e. $\Pi V_\mu = \underset{V\in\text{span}(\Phi)}{\arg\min} \ \| V - V_\mu \|_\pi^2$. We can say that the Monte Carlo approximation $\hat{V}_{MC} = \Phi\hat{\theta}$ converges to ΠV_μ as $N \to \infty$.

2.2.2 Fitted Value Iteration:

Let's consider the discounted case as the analysis turns out to be simpler in this case. Suppose we observe a single stream of data $\Big[s_0, r_0, s_1, \dots, r_N, s_{N+1}\Big]$ or equivalently the set of all tuples $H = \Big[(s_n, r_n, s_{n+1}) \mid n = 1, 2, \dots, N\Big]$. Assume that the Markov chain is ergodic under the policy μ, say $\pi(s) = \lim_{n\to\infty} (\sum_{k=0}^{n} \mathbb{I}_{\{s_k = s\}})/n$. Realistically, we need $P(s_n = s, s_{n+i} = s') \approx \pi(s) \times \pi(s')$ for n, i large enough. We can find the estimate of θ by starting with some initial θ_0 and iterating over:

$$\theta_{k+1} = \arg\min_\theta \ \frac{1}{|H|} \sum_{(s,r,s')\in H} \Big(V_\theta(s) - (r + \gamma V_{\theta_k}(s'))\Big)^2$$

until convergence. But is the following map a contraction (to reason about the convergence of this algorithm)?

$$V \mapsto \underset{V'\in\text{span}(\Phi)}{\arg\min} \frac{1}{|H|} \sum_{(s,r,s')\in H} \Big(V'(s) - (r + \gamma V(s'))\Big)^2$$

It's unclear for finite sample cases, but we can claim the following as $N \to \infty$:

$$\begin{aligned}
\arg\min_{V' \in \mathrm{span}(\Phi)} \frac{1}{|H|} \sum_{(s,r,s') \in H} \Big(V'(s) - (r + \gamma V(s')) \Big)^2 &\approx \arg\min_{V' \in \mathrm{span}(\Phi)} \frac{1}{|H|} \sum_{(s,\cdot,\cdot) \in H} \Big(V'(s) - T_\mu V(s) + \text{noise} \Big)^2 \\
&\approx \arg\min_{V' \in \mathrm{span}(\Phi)} \mathbb{E}_{s \sim \pi(s)} \Big(V'(s) - T_\mu V(s) \Big)^2 \\
&= \arg\min_{V' \in \mathrm{span}(\Phi)} \| V' - T_\mu V \|_\pi^2 \\
&= \Pi T_\mu V
\end{aligned}$$

where Π is the projection operator. Hence, the algorithm approximately does the following: start with some initial θ_0 and $V_0 = \Phi\theta_0$ and repeat $V_{k+1} = \Pi T_\mu V_k$. Next time, we will prove the convergence of the this algorithm, which is sometimes called projected Value iteration.