EE675A

Lecture 13: Model-Free RL

01/03/2023

Lecturer: Subrahmanya Swamy Peruru *Scribe: Nitika Singhvi and Shubham Korde*

1 Recap and Overview

Solving finite MDPs using Dynamic programming based methods which include policy iteration and the value iteration. There are two main steps that in these algorithms which are policy evaluation followed by policy improvement. They can be used to compute optimal policies and value function given complete knowledge about the MDP. Policy evaluation involves computation of the value function typically in an iterative manner for a given policy. Policy improvement involves computation of an improved policy given the value function for that policy.
This lecture begins with some more discourse on the dynamic programming based methods for

$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_*$$

Figure 1: E is the Evaluation and I is the Improvement step

solving finite MDPs. We then look at the first learning methods for estimating value functions and discovering optimal policies.

2 Number of Iterations required for Policy evaluation

In case of Policy Evaluation for a policy π we have the Bellman Expectation update equations given by:

$$f_\pi(v) = R_s^\pi + \gamma \sum_{s'} v(s') P_{ss'}^\pi$$
$$v_{k+1} = f(v_k) \tag{1}$$

Here k is the iteration step, v_k is the value function after k iterations, R_s^π is the expected reward given policy π and state s, γ is the discount factor, $P_{ss'}^\pi$ is the probability of going from state s to s' given policy π.
From Contraction Mapping we have,

$$d(f(u), f(v)) \le \gamma d(u, v) \tag{2}$$

Here d is given by,

$$d(x,y) = \max_s \Big| x(s) - y(s) \Big| \tag{3}$$

The following theorem gives an idea of the number of iterations of Policy Evaluation

Theorem 2.1. *If Policy Evaluation is run for $k+1$ iterations such that $k \geq \frac{\log\left[\frac{\epsilon(1-\gamma)}{d(v_1,v_0)}\right]}{\log \gamma}$ then it is guaranteed that v_{k+1} is ϵ close to v_π that is, $d(v_{k+1}, v_\pi) \leq \epsilon$.*

Proof. Since $v_k \to v_\pi$ as $k \to \infty$ we can bound the distance between v_{k+1} and v_π by triangle inequality such that we have,

$$d(v_{k+1}, v_\pi) \leq d(v_{k+1}, v_{k+2}) + d(v_{k+2}, v_{k+3}) + \ldots \tag{4}$$

From (2) we get,

$$d(f_\pi(v_i), f_\pi(v_{i+1})) \leq \gamma d(v_i, v_{i+1})$$

Using (1) we can write the above equation as follows:

$$\begin{aligned} d(v_{i+1}, v_{i+2}) &\leq \gamma d(v_i, v_{i+1}) \\ d(v_{i+1}, v_{i+2}) &\leq \gamma^i d(v_0, v_1) \end{aligned}$$

Using the above observation and (4) we have,

$$\begin{aligned} d(v_{k+1}, v_\pi) &\leq \sum_{i=k}^{\infty} d(v_{i+1}, v_{i+2}) \\ &\leq \sum_{i=k}^{\infty} \gamma^i d(v_0, v_1) \\ &\leq \frac{\gamma^k}{1-\gamma} d(v_0, v_1) \end{aligned}$$

Hence by choosing $k \geq \frac{\log\left[\frac{\epsilon(1-\gamma)}{d(v_1,v_0)}\right]}{\log \gamma}$ and since $\gamma \leq 1$,

$$d(v_{k+1}, v_\pi) \leq \frac{\gamma^k}{1-\gamma} d(v_0, v_1) \leq \epsilon$$ □

3 Variants of Dynamic Programming

If we wish to implement the iterations in policy evaluation as it is we would require two arrays that are to be updated that are : v_{new} and v_{old}. An algorithm would look as follows:

Algorithm 1 Normal DP

```
1: At $k = 0$ we have $v_0$
2: $v_{old} = v_0$
3: for $k = 1$ to $T$ do
4:     for each $s \in S$ do
5:         $v_{new}(s) = R_s^\pi + \gamma \sum_{s'} v_{old}(s^{'}) P_{ss^{'}}^\pi$
6:     end for
7:     $v_{old} = v_{new}$
8: end for
```

Here we can easily see that the major disadvantage of the way in which a normal DP is implemented is that they can turn out to be very expensive in terms of computations in case of a large state set since they involve operations over all the states of the MDP, that is, they require sweeps of the entire state set. Another problem would be having to work with two big arrays of the value function v. We have the following variants of Dynamic Programming to tackle these problems:

3.1 In-place DP

In place Dynamic Programming uses only one array v. As the value function for states get updated we lose value function for other states. Convergence to the required v_π is typically faster as we are using the latest values while updating.

3.2 Asynchronous DP

Asynchronous DP Algorithms do not require systematic sweeps over the entire state set. In this method not all states are updated in each iteration. One state is randomly updated in each iteration.

3.3 Prioritized DP

This method also updates one state in each iteration but that state is selected such that it has the largest Bellman error, that is,

$$s^* = \arg\max_s \Big| v_k(s) - v_{k-1}(s) \Big|$$

4 Model Free Setting

Till now, we are not interacting with the environment. We assume that the complete MDP dynamics representing the environment are already known to us. We have formulated the problem as an optimization problem for a given MDP. For Example, in the bandit setting, we knew the reward distribution of each of the arms. However, in real-life settings, we do not know about the environment completely.

The model-free setting is one in which we do not have complete knowledge of the environment. They typically require only experience given in RL terminologies by sample sequences of states, actions, and rewards from actual or simulated interaction with an environment. It is useful to work with this problem for the following reasons:

- In case of an actual environment, no prior knowledge of the environment is required.
- In the case of a simulated environment, a model is required. Still, the model is not required to generate the complete probability distributions of all possible transitions as required for dynamic programming (DP) but only sample transitions.

Here like in the algorithms discussed before we have the prediction and the control step. We will look into the methods for prediction.

5 Monte Carlo Method for v_π prediction

We have to interact with the environment to find out the value function of a state. We know that,

$$V_\pi(s) = E[G_t|S_t = s]$$

Here we do not know the expression of the probability distribution of G_t. However, we are allowed to interact with the environment. So we run the experiment to get samples of G_t. This could be done by starting with some state s_t and taking action A_t according to policy $\pi(a|s_t)$ resulting in reward R_{t+1} and the agent reaching a state s_{t+1}. We repeat this process till we reach the end of the episode. To get a sample of G_t we can sum up all the rewards.

$$G_t = \sum_{i=t+1}^{T} R_i \gamma^{i-t-1}$$

This way, we can get multiple samples of G_t by following the policy π many times, starting from state s. To estimate the value function, we can find the average of total returns across different trials.

$$v_\pi(s) = \frac{\sum_{i=1}^{N} G_{t,i}}{N}$$

By the Weak Law of Large Numbers, we can prove that the estimate will converge to the actual value of $v_\pi(s)$ for large values of N.
A given trajectory starting from state s can not only be used to estimate the value of $v_\pi(s)$ but also to estimate the value of $v_\pi(s')$, where s' is some intermediate state in the trajectory. Similarly, if the state s is encountered multiple times in the trajectory. In that case, the cumulative rewards from those points onward can be used as multiple samples for estimating the value of $v_\pi(s)$.
If we only use a single sample for all the states in the given trajectory, the method is known as **Monte Carlo First Visit**. If we only use multiple samples for all the states in the given trajectory, the method is known as **Monte Carlo Every Visit**.

Note that it is easier to argue that Monte Carlo first visit method converges because the returns are iid random variables, whereas, in the case of Monte Carlo every visit method, the returns are correlated.
Even when the model is available, if finding the transition probabilities p^{π} is complex, Monte Carlo methods are useful. Monte Carlo methods differ from dynamic programming in two ways. Firstly, Monte Carlo methods are sample-based methods. Secondly, Monte Carlo methods are not bootstrapped. **Bootstrapping** refers to estimating $v(s)$ based on the other state estimates.

6 Temporal Difference Method for v_{π} prediction

The temporal difference method is also a sample-based method which does not require a model. It is an online model; therefore, it does not have to wait till the end of the episode to update the v_{π} estimates. Also, it is Bootstrapped.

7 Update Equations

The Monte Carlo update can be written as,

$$v_k(s) = v_{k-1}(s) + \frac{1}{k} \times [G_k - v_{k-1}(s)]$$

where, G_k is the return obtained in the k^{th} episode. This could also be written as,

$$v_k(s) = v_{k-1}(s) + \alpha \times [G_k - v_{k-1}(s)]$$

Where α is a constant in some non-stationary environment. The term $G_k - v_{k-1}(s)$ is called the Monte-Carlo error. G_k is called the Monte-Carlo target and $v_{k-1}(s)$ is the previous estimate.
The update step will remain the same in the temporal difference method, except we will have a new target.
Assume that the agent is in state s_t, took some action A_t based on $\pi\ (a|s_t)$, got some reward R_{t+1} and went to state s_{t+1}. TD methods will not wait till the end of the episode to update $v(s_t)$.

$$v_{new}(s_t) = v_{old}(s_t) + \alpha \times [R_{t+1} + \gamma v_{old}(s_{t+1}) - v_{old}(s)]$$

A key thing to note is that the samples in the Monte Carlo method are unbiased, whereas in the TD method, they are biased. *The material is primarily based on the content from [1]*

References

[1] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 2020.