ECE 598: Interplay between Control and Machine Learning **Fall 2020**

Lecture 11

Temporal Difference Learning

Lecturer: Bin Hu, Date:10/08/2020-10/15/2020

In this lecture, we discuss how to evaluate a given policy when the model is unknown. The main method here is the temporal difference learning.

11.1 Discrete Space Case

Recall that a MDP is defined by a tuple $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$ where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, P is the transition kernel, R is the reward, and γ is the discount factor. Given a policy π, we want to analyze the associated value function:

$$V^\pi(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R(s_k, a_k) \big| a_k \sim \pi(\cdot|s_k), s_0 = s\right].$$

Suppose both $\mathcal{S}$ and $\mathcal{A}$ are finite. Let V^π be the value function of π. If we use the following vector notation:

$$V^\pi = \begin{bmatrix} V^\pi(1) \\ V^\pi(2) \\ \vdots \\ V^\pi(n) \end{bmatrix}, \ \bar{R}^\pi = \begin{bmatrix} \bar{R}^\pi(1) \\ \bar{R}^\pi(2) \\ \vdots \\ \bar{R}^\pi(n) \end{bmatrix}, \ P^\pi = \begin{bmatrix} P_{11}^{\pi(1)} & \cdots & P_{1n}^{\pi(1)} \\ \vdots & \ddots & \vdots \\ P_{n1}^{\pi(n)} & \cdots & P_{nn}^{\pi(n)} \end{bmatrix}$$

then we can rewrite the Bellman equation as

$$V^\pi = \bar{R}^\pi + \gamma P^\pi V^\pi. \tag{11.1}$$

How to solve the above equation when we do not know P^π? Here is the basic idea of TD learning. Suppose we are using an iterative method and estimate the value function as V_k^π at step k. We sample the trajectory of the underlying Markov chain as $\{s_k\}$. Based on the Bellman equation, $V^\pi(s_k)$ can be estimated in two ways: either $V_k^\pi(s_k)$ or $r(s_k, \pi(s_k)) + \gamma V_k^\pi(s_{k+1})$ (this is called TD target). One reasonable way to do things is to minimize the difference of these two things: $(V_k^\pi(s_k) - r(s_k, \pi(s_k)) - \gamma V_k^\pi(s_{k+1}))^2$. How to make this difference small? Averaging!

$$\begin{aligned} V_{k+1}^\pi(s_k) &= (1-\varepsilon) V_k^\pi(s_k) + \varepsilon (r(s_k, \pi(s_k)) + \gamma V_k^\pi(s_{k+1})) \\ &= V_k^\pi(s_k) + \varepsilon \left(r(s_k, \pi(s_k)) + \gamma V_k^\pi(s_{k+1}) - V_k^\pi(s_k)\right) \end{aligned}$$

Denote $\delta_k = r(s_k, \pi(s_k)) + \gamma V_k^\pi(s_{k+1}) - V_k^\pi(s_k)$. The term δ_k is the so-called TD error. The above algorithm is called TD(0). Usually ε is small, and this means that we do not want to make too much change for a given random sample point.

TD(λ). An extension which interpolates TD(0) and Monte Carlo method is TD(λ). We briefly review the idea here. TD(0) just interpolates $V_k^\pi(s_k)$ and the one-step expansion $r(s_k, \pi(s_k)) + \gamma V_k^\pi(s_{k+1})$. The generalized m-step TD learning interpolates $V_k^\pi(s_k)$ with the m-step expansion $G_k^{(m)} := r(s_k, \pi(s_k)) + \gamma r(s_{k+1}, \pi(s_{k+1})) + \gamma^2 r(s_{k+2}, \pi(s_{k+2})) + \ldots + \gamma^m V_k^\pi(s_{k+m})$. Monte-Carlo simulation can be viewed as $G_k^{(\infty)}$. One issue is how to choose m. The idea of TD(λ) is that we can avoid choosing m and simply interpolates $V_k^\pi(s_k)$ with a weighted sum of $G_k^{(m)}$ for all m:

$$V_{k+1}^\pi(s_k) = (1-\varepsilon)V_k^\pi(s_k) + \varepsilon(1-\lambda)\sum_{m=1}^{\infty} \lambda^{m-1} G_k^{(m)}$$

This is a forward update which is hard to implement. Hence we need to convert it to a backward scheme. In the backward setup, instead of waiting for what is going to happen in the future, we will remember what happened in the past and use all past information together. Therefore, we interpolate $V_k^\pi(s_k)$ with something that can capture all the past information. To do this, we introduce the eligibility traces $e_k(s) = \gamma\lambda e_{k-1}(s) + \mathbf{1}(s_k = s)$ and TD(λ) updates as $V_{k+1}^\pi(s) = V_k^\pi(s) + \varepsilon\delta_k e_k(s)$. The backward view and the forward view are equivalent, providing different intuitions of TD(λ).

Function approximation. Now we talk about the function approximation case. Suppose n is too large and we do not want to learn an n-dimensional vector for V^π. Instead, we estimate $V^\pi(s) \approx \theta^\mathsf{T}\phi(s)$ where ϕ is the feature vector. Here, after we get a sample trajectory, what shall we do? We can try to minimize $\frac{1}{2}\|\theta^\mathsf{T}\phi(s_k) - r(s_k, \pi(s_k)) - \gamma\theta_k^\mathsf{T}\phi(s_{k+1})\|^2$. However, we do not want to completely solve this problem since we do not want to trust one sample point too much. Instead, we can perform one step gradient descent as

$$\theta_{k+1} = \theta_k + \varepsilon\left(r(s_k, \pi(s_k)) + \gamma\theta_k^\mathsf{T}\phi(s_{k+1}) - \theta_k^\mathsf{T}\phi(s_k)\right)\phi(s_k)$$

Again, we have TD error $\delta_k = r(s_k, \pi(s_k)) + \gamma\theta_k^\mathsf{T}\phi(s_{k+1}) - \theta_k^\mathsf{T}\phi(s_k)$. The derivations of TD($\lambda$) with linear function approximation is introduced in Section 2.4.1 in [1]. Again, we will use the backward view and the update rule is given as $z_k = \lambda\gamma z_{k-1} + \phi(s_k)$ and $\theta_{k+1} = \theta_k + \varepsilon\delta_k z_k$. This is equivalent o

$$\theta_{k+1} = \theta_k + \varepsilon_k\delta_k\sum_{t=0}^{k}(\lambda\gamma)^t\phi(s_{k-t})$$

We can clearly see that all the past information is somehow weighted.

Asymptotic analysis v.s Finite sample analysis. Several theoretical guarantees of TD learning can be proved. For illustrative purposes, let's look at TD(0). Notice TD(0) can be modeled as a linear stochastic approximation scheme $\theta_{k+1} = \theta_k + \varepsilon(A_{i_k}\theta_k + b_{i_k})$ where i_k is the augmented vector of (s_{k+1}, s_k). As $\varepsilon \to 0$, we will have an ODE to model its

asymptotic dynamics: $\dot{\theta} = \bar{A}\theta + \bar{b}$ where $\bar{A}$ and $\bar{b}$ are some averaged quantities. Based on the so-called ODE approach, the asymptotic convergence of TD(0) with diminishing step size can be guaranteed by the stability of this ODE and several extra technical conditions. More recently, the finite sample bound is studied. This is a more technical topic, and there are still a lot of research activities going on here.

Off policy vs. on policy. The above method is on policy since the data has to be sampled using the policy π. Sometimes we do not use to directly use π to gather data since that may be dangerous. Then we will use off-policy methods. In off-policy TD learning, the data is sampled from a behavior policy μ. Then importance sampling trick is used to evaluate the value function of the target policy π using the data generated by the behavior policy μ. What is importance sampling? Originally important sampling is used in rare event simulations to sample "important but rare" events. Just imagine $X \sim \mathcal{N}(0,1)$ and one wants to estimate $P(X > 10) = \mathbb{E}\mathbf{1}_{X>10}$ using Monte Carlo simulations. Since it is extremely unlikely to get a sample with $X > 10$, the Monte Carlo simulation may just return 0. Suppose $f(x)$ is the density function for $\mathcal{N}(0,1)$ and $g(x)$ is the density function for $\mathcal{N}(10,1)$. Then importance sampling uses the following reformulation:

$$\mathbb{E}\mathbf{1}_{X>10} = \int_{-\infty}^{\infty} \mathbf{1}_{X>10} f(x)dx = \int_{-\infty}^{\infty} \mathbf{1}_{X>10} \frac{f(x)}{g(x)} g(x)dx$$

Now we can sample X from $\mathcal{N}(10,1)$ and average the quantity $\mathbf{1}_{X>10}\frac{f(x)}{g(x)}$. Clearly by doing this, we see the event $X > 10$ much more often, and this is exactly the idea of importance sampling. Here the ratio of the two densities is called importance sampling ratio. Off-policy TD learning uses a similar idea, and requires using the importance sampling ratio between π and μ. To study some details of off-policy TD learning, see Sections 2-4 in [2].

Online vs. Off-line: Least square methods. TD(0) and TD(λ) are online methods. When a new data point s_k is observed, the weight θ_k is updated and then the data point s_k is completely thrown away after this update. How to make more efficient use of data? If all the data are available, then we can use off-line methods. For example, when the linear approximation is used, we can apply the least square method to fit θ directly. If we look at the recursive formula for TD(0), eventually the method will converge if the term $\left(r(s_k, \pi(s_k)) + \gamma\theta_k^{\mathsf{T}}\phi(s_{k+1}) - \theta_k^{\mathsf{T}}\phi(s_k)\right)\phi(s_k)$ is roughly 0. Therefore, we want to choose θ to have

$$\phi(s_k)\left(\phi(s_k) - \gamma\phi(s_{k+1})\right)\theta \approx \phi(s_k) r(s_k, \pi(r_k)),\ \forall k$$

This becomes a least square problem. We can just choose $A = \sum_k \phi(s_k)\left(\phi(s_k) - \gamma\phi(s_{k+1})\right)$ and $b = \sum_k \phi(s_k) r(s_k, \pi(r_k))$. Then we fit θ as $\theta = A^{-1}b$. When A is not invertible, we can use pseudo inverse. We can add regularization or eligibility trace into the algorithm. Details are omitted.

Neural network as the function approximators. For practical problems, typically we will use neural networks to approximate the value function, i.e. $V^\pi(s) = W^{(1)}\sigma(W^{(0)}\phi(s))$. For simplicity, we augment all the weights as θ, and use the notation $V^\pi(s) = f_\theta(s)$. Again, the goal here is to find θ. At every step of TD(0), we can try to minimize the following function.

$$\frac{1}{2}\|f_\theta(s_k) - r(s_k, \pi(s_k)) - \gamma f_{\theta_k}(s_{k+1})\|^2$$

Again, we do not want to trust one data point too much and hence just do one step gradient descent as

$$\theta_{k+1} = \theta_k + \varepsilon \left(r(s_k, \pi(s_k)) + \gamma f_{\theta_k}(s_{k+1}) - f_{\theta_k}(s_k)\right) \nabla_\theta f_{\theta_k}(s_k)$$

We denote the TD error as $\delta_k = r(s_k, \pi(s_k)) + \gamma f_{\theta_k}(s_{k+1}) - f_{\theta_k}(s_k)$ and then the TD(0) just iterates as $\theta_{k+1} = \theta_k - \varepsilon\delta_k \nabla_\theta f_{\theta_k}(s_k)$. Again, such an online method is not efficient in making use of all the data. In addition, the use of the neural network may introduce some stability issue for training. A better way to fit θ is to use a supervised learning framework. The idea is similar to fitted Q-iteration. Now the data collection is decoupled from the algorithm iteration. Suppose $\{s_k\}$ has been sampled. At every step l, we solve the following finite-sum optimization

$$\theta_{l+1} = \arg\min_\theta \sum_k \frac{1}{2}\|f_\theta(s_k) - r(s_k, \pi(s_k)) - \gamma f_{\theta_l}(s_{k+1})\|^2$$

For simplicity, we denote $y_k^{(l)} = r(s_k, \pi(s_k)) + \gamma f_{\theta_n}(s_{k+1})$. Then at every l, we are exactly solving a supervised learning problem and try to minimize the finite-sum function

$$\sum_k \frac{1}{2}\|W_n^{(1)}\sigma(W_n^{(0)}\phi(s_k)) - y_k^{(l)}\|^2.$$

This problem can be efficiently solved by SGD/ADAM and standard routines in TensorFlow or PyTorch.

11.2 Continuous space case

Now we discuss the continuous control case. Again, we start from the LQR problem. Obviously we can use the linear function approximation for the value function in this case.

Let's consider a 2D example. Suppose $x = \begin{bmatrix} a \\ b \end{bmatrix}$ and u_t is a scalar. The feature can be calculated as

$$\phi(x) = \begin{bmatrix} a^2 \\ ab \\ b^2 \\ 1 \end{bmatrix}$$

The state value function is parameterized as

$$V(x) = \theta^\top \phi(x) = \begin{bmatrix} p_1 & p_2 & p_3 & r \end{bmatrix} \begin{bmatrix} a^2 \\ ab \\ b^2 \\ 1 \end{bmatrix} = \begin{bmatrix} a \\ b \end{bmatrix}^\top \begin{bmatrix} p_1 & \frac{1}{2}p_2 \\ \frac{1}{2}p_2 & p_3 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} + r$$

Now we want to fit $\{p_1, p_2, p_3, r\}$ from sampled data. We just generate a trajectory of $\{x_t\}_{t=0}^T$ using $x_{t+1} = (A - BK)x_t + w_t$. We fit θ to minimize the target difference error.

TD(0). Again, we can apply the update $\theta_{t+1} = \theta_t + \varepsilon \left(c(x_t) + \gamma \theta_t^\mathsf{T} \phi(x_{t+1}) - \theta_t^\mathsf{T} \phi(x_t)\right) \phi(x_t)$ where $c(x_t) = x_t^\top (Q + K^\top R K) x_t$. Unfortunately, such an online method does not work well.

LSTD. LSTD fits θ by enforcing

$$\left(\sum_{t=0}^{T-1} \phi(x_t)(\phi(x_t) - \gamma \phi(x_{t+1}))^\top \right) \theta \approx \sum_{t=0}^{T-1} c(x_t)\phi(x_t)$$

So we just estimate θ as

$$\theta \approx \left(\sum_{t=0}^{T-1} \phi(x_t)(\phi(x_t) - \gamma \phi(x_{t+1}))^\top \right)^{-1} \left(\sum_{t=0}^{T-1} c(x_t)\phi(x_t) \right)$$

Sometimes the inverse does not exist and we can replace it with pseudo inverse. In `Matlab`, you can use `pinv` to calculate pseudo inverse. The above implementation requires the noise w_t to provide sufficient exploration. If the noise is too small, it will require a huge T to make the above implementation work. In this case, you can use another implementation. Generate x_t randomly for all t using some distribution which explores the state space thoroughly. For example, use a uniform distribution over $[-1000, 1000]$. For all t, generate x'_t as $x'_t = (A - BK)x_t + w_t$. Now estimate θ as

$$\theta \approx \left(\sum_{t=0}^{T} \phi(x_t)(\phi(x_t) - \gamma \phi(x'_t))^\top \right)^{-1} \left(\sum_{t=0}^{T} c(x_t)\phi(x_t) \right)$$

You can try different distribution to generate x_t to see which one explores best.

Nonlinear case. For general nonlinear problems, one can use neural networks to parameterize the value function, and then apply the supervised learning framework introduced in the last section to fit the weights.

11.3 Summary and Plan

One can see that the basic idea of TD learning is to fit the value function by enforcing the left and right sides of the Bellman equation to be roughly equal to each other on the observed data. A similar idea can be used to find the optimal Q function. One can try to fit the optimal value function by enforcing the left and right sides of the optimal Bellman equation to be roughly equal to each other on the observed data. This is the idea behind value-based reinforcement learning methods. The issue here is how to generate the data. We will talk about this in next lecture.

Another big class of RL methods are the so-called policy-based methods. Intuitively, one can directly parameterize the policy and do gradient descent on the policy. The issue here is how to estimate the policy gradient from data when the model is not known. We will also discuss this in future lectures.

Bibliography

[1] C. Dann, G. Neumann, and J. Peters. Policy evaluation with temporal differences: A survey and comparison. *Journal of Machine Learning Research*, 15(1):809–883, 2014.

[2] R. S. Sutton, A. R. Mahmood, and M. White. An emphatic approach to the problem of off-policy temporal-difference learning. *The Journal of Machine Learning Research*, 17(1):2603–2631, 2016.