CS292F Statistical Foundation of Reinforcement Learning **Spring 2021**

Lecture 5: April 12

Lecturer: Yu-Xiang Wang *Scribes: Yi-Lin Tuan*

Note: *LaTeX template courtesy of UC Berkeley EECS dept.*

This lecture's notes illustrate some uses of various LaTeX macros. Take a look at this and imitate.

5.1 Variant / improvements of Sarsa and Q-learning

5.1.1 Expected SARSA

Expected SARSA is a variant of Sarsa. The idea is since we actually have the policy π, we could avoid relying on the realized value function V. The update of Q value is

$$\begin{aligned} Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma E[Q(S_{t+1}, A_{t+1})|S_{t+1}] - Q(S_t, A_t)] \\ &\leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t)] \end{aligned} \tag{5.1}$$

Where we can observe that the expected SARSA only requires SARS'π, where S' denotes the next state (S_{t+1} in the equation). Also, when $\pi = \arg\max Q$, expected SARSA is the same as the original SARSA.

5.1.2 Bias in Q-learning and double Q-learning

SARSA simulates Bellman equation, while Q-learning simulates the Bellman optimality equation whereas you might playing sub-optimal policy. However, Q-learning is expected to mitigate the maximization bias. Let's first see the update of Q-learning.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{5.2}$$

The maximum term can be formed as $\max_a E_{s_{t+1} \sim P(\cdot|S_t, A_t)} Q(S_{t+1}, a)$. Nonetheless, the Bellman optimality equation is actually $E \max_a Q(S_{t+1}, a)$. And by Jensen's inequality, we know that $E \max \geq \max E$. Because of this, doing the standard Q-learning, we will have a maximization bias (also known as survival bias or overfitting in machine learning).

One way to solve the overfitting is to split data into training and validation sets. During training, also look into the validation set to ensure the training does not overfit. With this idea, double Q-learning is proposed by keep track of two Q function estimates.

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha[R_{t+1} + \gamma Q_2(S_{t+1}, \arg\max_a Q_1(S_{t+1}, a)) - Q_1(S_t, A_t)] \tag{5.3}$$

This method will have larger variance due to data splitting but have lower bias.

5.2 The problem of large state-space

When using Q-learning and SARSA, we need to represent and learn SA parameters, where the state space can be infeasible in many cases, such as:

- 9-puzzle, Tic-Tac-Toe: $S = 9! = 362,800$, $S^2 = 1.3*10^{11}$
- PACMAN with 20 by 20 grid: $S = O(2^{400})$, $S^2 = O(2^{800})$

Needs to think of ways to "generalize"/share information across state.

5.3 Linear Value Functions

One solution is to represent each state as features (also describe a Q-state (s,a) using a vector of features). Having the feature representation, we can adopt linear value functions instead of Q-learning and SARSA by writing a Q function (or value function) for any state using a few weights. That is,

$$\begin{aligned} V_w(s) &= w_1 f_1(s) + w_2 f_2(s) + ... + w_n f_n(s) \\ Q_w(s) &= w_1 f_1(s,a) + w_2 f_2(s,a) + ... + w_n f_n(s,a) \end{aligned} \tag{5.4}$$

Each weight is for each dimension of the vector of features f_i, $i = 1, ..., n$.

- Advantage: our experience is summed up in a few powerful numbers.
- Disadvantage: states may share features but actually be very different in value.

Updating a linear value function is similar to gradient descent. Let's say the original Q learning rule predicts Q and tries to reduce the prediction error at s,a:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)] \tag{5.5}$$

Instead, linear value function updates the weights to reduce error at s,a. And since Q_w is linear,

$$\begin{aligned} w_i &\leftarrow w_i + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)] \frac{\partial Q_w(s,a)}{\partial w_i} \\ &= w_i + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)] f_i(s,a) \end{aligned} \tag{5.6}$$

Note that $R(s,a,s')$ is the expected reward but not the observed reward (But we can rewritten the equation to replace expected reward by observed rewards). To qualitative justify the equation, we can see that if we get pleasant surprise ($R(s,a,s') + \gamma \max_{a'} Q(s',a') > Q(s,a)$), the update will increase weights on positive features and decrease negative ones. On the other hand, if we get unpleasant surprise ($R(s,a,s') + \gamma \max_{a'} Q(s',a') < Q(s,a)$), the update will decrease weights on positive features and increase negative ones.

5.3.1 Deriving the TD via incremental optimization that minimizes Bellman errors

How we should understand Temporal Difference learning (TD) from both the tabular setting and the function approximation setting.

If the objective function is Mean Square Error, then we are minimizing w to reduce the value prediction mean square error by consider policy evaluation.

$$\min_{w \in R^d} \sum_s \mu(s)(V^\pi(s) - \hat{V}^\pi(s;w))^2 \approx \min_w \frac{1}{N} \sum_i \sum_s \mu(s)(G_i^s - \hat{V}^\pi(s;w))^2 \tag{5.7}$$

where G_i^s is from Monte Carlo sampling and we approximate $V^\pi(s) = 1/N \sum_i G_i^s$. The w is then updated as:

$$\begin{aligned} w^+ &= w - \alpha \nabla_w (G_i^s - V^\pi(s;w))^2 \\ &= w + \alpha(-V^\pi(s;w) + G_i^s) \nabla V^\pi(s;w) \end{aligned} \tag{5.8}$$

where $-V^\pi(s;w) + G_i^s$ is the Monte Carlo TD estimate.

If we want to use bootstrapping, we can derive from the Mean square bellman error and adopt sampling to collect a series of data points $S_1, A_1, R_2, S_2, A_2, R_3, ...$ Then we get:

$$\begin{aligned} &\min_{w \in R^d} \sum_s \mu(s)(r^\pi(s) + \gamma E_{s' \sim P^\pi(s)} \hat{V}^\pi(s') - \hat{V}^\pi(s;w))^2 \\ &\approx min_{w \in R^d} \frac{1}{T} \sum_{t=1}^{T} (R_{t+1} + \gamma \hat{V}(S_{t+1};w) - \hat{V}(S_t;w))^2 \end{aligned} \tag{5.9}$$

The w is then updated as:

$$w^+ = w - \alpha \nabla_w (R_{t+1} + \gamma \hat{V}(S_{t+1};w) - \hat{V}(s_t;w)^2 \text{ semi gradient.} \tag{5.10}$$

Moreover, in TD learning, the w is not updated as online gradient descent or stochastic gradient descent, but as *semi-gradient method*. It is because in TD learning the term $\hat{V}(S_{t+1};w)$ is fair to be considered as fixed yet not a function of w. So the update of w can be rewritten as:

$$\begin{aligned} w^+ &= w + \alpha(R_{t+1} + \gamma \hat{V}(S_{t+1};w) - \hat{V}(s_t;w)) \nabla V(s_t;w) \\ &= w + \alpha(R_{t+1} + \gamma \hat{V}(S_{t+1};w) - \hat{V}(s_t;w)) f(s_t) \end{aligned} \tag{5.11}$$

5.3.2 How do we make sense of the semi-gradient method?

First, we define $x_t := f(S_t) \in R^d$ and $x_{t+1} = f(S_{t+1}) \in R^d$ and simplify the notations:

$$w^+ = w + \alpha[R_{t+1} + \gamma w_t^T x_{t+1} - w_t^T x_t] x_t \tag{5.12}$$

This is the semi-gradient update rule. We first restructure the equation to find the incremental value of w.

$$\frac{w^+ - w}{\alpha} = R_{t+1} x_t + x_t(\gamma x_{t+1} - x_t)^T w_t \tag{5.13}$$

Then we consider a very small α (take limitation to zero) and denote the derivative of w by w'.

$$w' = R_{t+1} x_t + x_t(\gamma x_{t+1} - x_t)^T w_t \tag{5.14}$$

Then we take expectation on both sides:

$$E[w'|w_t] = E[R_{t+1} x_t] + E[x_t(\gamma x_{t+1} - x_t)^T] w_t \tag{5.15}$$

The expectation is over the data points we collect in order to construct the TD update rule. We can observe from the equation that when α goes to zero, the equation keeps simulate everything.

To find the fixed point, we can compute compute $0 = E[w'|w]$. And we define $A := -x_t(\gamma x_{t+1} - x_t)^T \in R^{d\times d}$ and $b := R_{t+1}x_t$. So we will get:

$$\begin{aligned} 0 &= b - Aw \\ w &= A^{-1}b \end{aligned} \tag{5.16}$$

So this method can converge to a close form solution that is simulating the function approximated version of the bellman equation. Also whenever A is invertible (can be shown in weak condition), this solution is also a unique minimizer. Because when setting the gradient to zero, it is the only point where the expectation of the semi-gradient method will converge.

This can give us insights of how to develop better algorithm with lower variance, thus bring to the Least-Square Temporal Difference method (LSTD).

5.4 LSTD: why doing incremental optimization when we can...

LSTD is more stable since we no longer suffer from high variance due to state transition and immediate reward observation. The idea of LSTD no longer uses iterative updates but directly solves the linear system given by:

$$w^* = A^{-1}b \tag{5.17}$$

Then try to replace A with $\hat{A}$ and b with $\hat{b}$ as:

$$\begin{aligned} \hat{A}_t &= \sum_{k=0}^{t-1} x_k(x_k - \gamma x_{k+1})^T + \epsilon I \\ \hat{b}_t &= \sum_{k=0}^{t-1} R_{k+1}x_k \\ w_t &= \hat{A}_t^{-1}\hat{b}_t \end{aligned} \tag{5.18}$$

So the space is in d^2 order while the computation is in d^3 order in the worst case. However, there is a *Rank 1 update* trick that can facilitate the computation. So the rank 1 update of $\hat{A}_{t-1}^{-1} \Rightarrow \hat{A}_t^{-1}$ using Sherman-Morrison-Woodbury identity can derive

$$\begin{aligned} \hat{A}_t^{-1} &= (\hat{A}_{t-1} + x_{t-1}(x_{t-1} - \gamma x_t)^T)^{-1} \\ &= \hat{A}_t^{-1} - \hat{A}_{t-1}^{-1}(x_t(x_{t-1} - \gamma x_t)^T)^T \hat{A}_{t-1}^{-1} \end{aligned} \tag{5.19}$$

Then everything can be computed in d^2 time.

5.5 Policy gradient

So far we talked about value function approximation and and induced policy class. For example the estimate value function and Q function parameterized by w.

$$\hat{V}_w^\pi \approx V^\pi$$

$$\hat{Q}_w^* \approx Q^*$$

In particular, once we have estimated Q function, it induces a corresponding greedy policy class:

$$\Pi_w = \{\pi_w = \arg\max \hat{Q}_w\}$$

On the other hand, we can directly parameterize a policy class θ.

$$\Pi_\theta : \{\pi_\theta : S \leftarrow \delta \times V\}$$

5.5.1 Examples

1. Tabular setting: we can define a "softmax" policy class with $\theta \in R^{S\times A}$: $\pi_\theta(a|s) = \frac{exp(\theta_{s,a})}{\sum_{a'} exp(\theta_{s,a'})}$.
2. linear feature setting: we can consider the "log-linear policy class": $\pi_\theta(a|s) = \frac{exp(\theta^T f(s,a))}{\sum_{a'} exp(\theta^T f(s,a'))}$.
3. non-linear function class: we can consider a "neural policy class": $\pi_\theta(a|s) = \frac{exp(f_\theta(s,a))}{\sum_{a'} exp(f_\theta(s,a'))}$, where f_θ is differentiable in θ.

5.5.2 Optimize policy using SGD

The idea is to optimize policy to maximized the expected value.

$$\begin{aligned}
&\min_\theta -V^{\pi_\theta}(\mu) \\
&\iff \min_\theta -E^{\pi_t heta}_{s_0\sim\mu_0}[\sum_{t=0}^{\infty} R_{t+1}] \text{ (stochastic optimization)} \\
&\approx \min_\theta -\frac{1}{N}\sum_{i=1}^{N}\sum R^{(i)}_{t+1} \text{ (rolling out, can also use other approximation with } \theta)
\end{aligned} \tag{5.20}$$

Then the θ can be updated by

$$\theta^+ = \theta + \alpha\hat{\nabla}_\theta V^{\pi_\theta}(\mu) \tag{5.21}$$

where $E[\hat{\nabla}_\theta V^{\pi_\theta}(\mu)] = \nabla_\theta V^{\pi}(\mu)$