14 Model-Free Stochastic Control for Unknown Systems

Need Model-Free Methods with Unknown Dynamics

- Recall that, in the Markov decision problem (infinite horizon discounted case), we have a controlled Markov chain $\{\mathbb{P}(u)\}_{u\in\mathcal{U}}$ over a finite state space $S = \{0, 1, \dots, I\}$ with initial distribution $\pi_0 \in [0, 1]^I$. For any given policy $g \in \mathcal{G}_{SM}$, the value of state $x \in S$ under policy g is defined by

$$V^g(x) = E^g\left(\sum_{t=0}^{\infty} \beta^t c(X_t, g(X_t)) \mid X_0 = x\right),$$

which can be computed according to the **Bellman Equation**

$$V^g = c^g + \beta\mathbb{P}^g V^g.$$

Then we can improve policy g_n to g_{n+1} by the following **policy improvement** procedure

$$g_{n+1} = \arg\min_{g\in\mathcal{G}}\{c^g + \beta\mathbb{P}^g V^{g_n}\}.$$

- To implement the policy evaluation and the policy improvement, we need to know the underlying actual dynamic of the system, i.e., transition probability matrices $\{\mathbb{P}(u)\}_{u\in\mathcal{U}}$. However, this model information may be unknown a priori in many problems and one has to *learn* (estimate) the transition probability based on the sample data (simulations).

- Let us first consider the policy evaluation problem. Assume that the system starts from state $x \in S$. Under a given policy $g \in \mathcal{G}_{SM}$, the state trajectory of the system is a stochastic process $\{X_t^g\}$. Therefore, the discounted cost incurred from state x is also a randomly variable

$$C^g(x) = \sum_{t=0}^{\infty} \beta^t c(X_t^g, g(X_t^g)).$$

To evaluate policy g, we essentially want to compute the expected value of the cost random variable from each state $x \in S$, i.e., $V^g(x) = E(C(x))$. This can be done either analytically by knowing $\mathbb{P}^g$, which is the case of Bellman equation, or using a **simulation-based approach** by estimating from a set of samples of $C^g(x)$, say $\bar{c} = (c_1, c_2, \dots, c_n)$. Then we can compute the sample mean $\frac{1}{n}\sum_{i=1}^{n} c_i$. When $n \to \infty$, this estimated value $\bar{c}$ will converge to $E(C^g(x))$ with probability one.

- For each time of sampling $i = 1, \dots, n$, we can just simulate the system from state $x \in S$ following policy g, which gives us an infinite sequence of states

$$\mathbf{x}_i = (x_0, x_1, x_2, \dots), \text{ where } x_0 = x.$$

Then we have $c_i = \sum_{t=0}^{\infty} \beta^t c(x_t, g(x_t))$. In the context of reinforcement learning, each simulated trajectory $\mathbf{x}_i$ is also referred to as an **episode**. Theoretically, it needs to be infinitely long, but a large enough finite horizon is usually sufficient in practice.

- Note that here we assume that the cost structure $c(x, u)$ is completely known. This setting is without loss of generality and we can extend to the case where $c(x, u)$ is unknown or is even a random variable. (Think how? Hint: We can put the unknown information of $c(x, u)$ into the transition probability by extending the state-space.)

From State Values to Action Values

- Note that, even if we estimate the state value $V^{g_n}(x)$ from data, it is still not sufficient for the purpose of policy improvement since $\mathbb{P}^g$ also needs the information of transition probability. To address this issue, for each pair of state $x \in S$ and action $u \in \mathcal{U}$, we investigate the **state-action value** (or simply, **action value**)

$$Q^g(x,u) = E^g\left(\sum_{t=0}^{\infty} \beta^t c(X_t, U_t) \mid X_0 = x, U_0 = u\right).$$

  That is, starting from state $x \in S$, we force the system to first take action $u \in \mathcal{U}$ and then play policy g. Then $Q^g(x,u)$ is the expected discounted cost by doing so.

- The state value and the action value have the following relation

$$Q^g(x,u) = c(x,u) + \beta \sum_{x' \in S} P(x' \mid x,u) V^g(x').$$

  Recall that the policy improvement procedure in terms of state value is

$$g_{n+1}(x) = \arg\min_{u \in \mathcal{U}} \{c(x,u) + \beta \sum_{x' \in S} P(x' \mid x,u) V^{g_n}(x')\}.$$

  Therefore, the policy improvement can be expressed directly in terms of action value by

$$g_{n+1}(x) = \arg\min_{u \in \mathcal{U}} Q^g(x,u).$$

- Therefore, instead of estimating the state value $V^g(x)$ from the sample trajectories, we can estimate the action value $Q^g(x,u)$ as follows
  - Simulate the system from state x by first taking action u and then following policy g. This gives us an episode $\mathbf{x}_i = (x_0, x_1, x_2, \ldots, x_T, \ldots)$, which is an infinite sequence of states with cost $c_i = \sum_{t=0}^{\infty} \beta^t c(x_t, g(x_t))$
  - Repeat the above procedure for K episodes $(\mathbf{x}_1, \ldots, \mathbf{x}_K)$. Then we can estimate the action value by $Q^g(x,u) \approx \frac{1}{n}\sum_{i=1}^{n} c_i$.

- The above discussion serves as the basis for all model-free decision making algorithms. The basic idea is very simple: if we do not have model information to precisely compute values, then we can use data (simulated sample trajectories) to estimate them. This gives the most basic form of **model-free policy iteration** algorithm.

Naive Model-Free Policy Iteration Algorithm (Monte Carlo Based)

Step 1 Start with arbitrary stationary Markov policy $g_0 \in \mathcal{G}_{SM}$

Step 2 For each state-action pair $(x,u) \in S \times \mathcal{U}$, estimate the action value $Q^{g_n}(x,u)$ by simulating the system from (x,u) for many episodes.

Step 3 Improve policy g_n to a new policy g_{n+1} by $g_{n+1}(x) = \arg\min_{u \in \mathcal{U}} Q^{g_n}(x,u)$.

Step 4 Repeat **Steps 2 and 3** until the policy cannot be improved.

Challenges in Model-Free Policy Iteration

- The above naive model-free policy iteration algorithm is, in fact, very inefficient and even not practical. Here, we discuss some of the major challenges and how to overcome them.

- **Waste of Information in Data**: One major inefficiency of the above procedure is that it does not make fully use of the simulated data

$$\mathbf{x}_i = (x_0 \xrightarrow{u_0} x_1 \xrightarrow{u_1} x_2 \xrightarrow{u_2} \cdots \xrightarrow{u_{T-1}} x_T \xrightarrow{u_T} \cdots)$$

  in each episode. Previously, we only use $\mathbf{x}_i$ to estimate the value for the *first* action $Q^g(x_0, u_0)$. However, this episode data also provides us the information of $Q^g(x_1, u_1)$, $Q^g(x_2, u_2), \ldots$, but we do not make use of these information at all!

  Note that a state-action pair (x, u) may appear multiple times in each episode $\mathbf{x}_i$, for example $(x, u) = (x_{i_1}, u_{i_1}) = (x_{i_2}, u_{i_2}) = \cdots$. Depending on whether we estimate $Q^g(x, u)$ for only once or for multiple times, we have two different approaches to make better use of the information in data:

  - **First-Visit Method**: for each (x, u), we only put trajectory $(x_{i_1} \xrightarrow{u_{i_1}} x_{i_1+1} \xrightarrow{u_{i_1+1}} \cdots)$ to the estimation data for $Q^g(x, u)$ once.
  - **Every-Visit Method**: for each (x, u), we put all trajectories starting from (x, u), i.e., $(x_{i_1} \xrightarrow{u_{i_1}} x_{i_1+1} \xrightarrow{u_{i_1+1}} \cdots)$, $(x_{i_2} \xrightarrow{u_{i_2}} x_{i_2+1} \xrightarrow{u_{i_2+1}} \cdots)$, and so forth, to the estimation data for $Q^g(x, u)$.

  The main difference between the above two methods is that (i) first-visit estimator is unbiased, while (ii) every-visit estimator is asymptotically unbiased.

- **Waste of Time for Update**: in the naive model-free algorithm, we need to wait for a long time to update the policy because

  - To perform estimations for each action value, we need to generate many episodes.
  - To obtain each episode, we need to generate the entire (very long) sample trajectory.

  To address the first issue, our approach is to use the so called **generalized policy iteration**, i.e., we will improve the policy based on the value of each single sample trajectory rather than the mean of all sample trajectories. To address the second issue, we need to make the estimation procedure **recursive or incremental**, which leads to the well-known **temporal difference learning**.

- **State Initialization**: in the naive model-free algorithm, to estimate $Q^g(x, u)$, we have to *reset* the system from state x each time when we want to generate an episode. This may be very inefficient or even infeasible in many cases. For example, we may need to physically move a robot to a location each time. An ideal approach is that, if we can keep running an infinite sequence in which each pair (x, u) occurs infinite number of times, then we will still obtain sufficient statistic information for each $Q^g(x, u)$.

  This issue can be resolved by using the so called **ϵ-greedy randomized policy**. Specifically, suppose that we have the estimated action value $Q(x, u)$ for all state-action pairs. Previously, the action value induces a greedy policy by $g(x) = \arg\min_{u \in \mathcal{U}} Q(x, u)$. Here we relax this by allowing $g(x)$ to be probability distribution over all actions $\mathcal{U}$, i.e., $\sum_{u \in \mathcal{U}} g(u \mid x) = 1, \forall x \in S$, where we allow a very small probability $\epsilon > 0$ for those non-greedy actions for the purpose of getting data.

Generalized Policy Iteration without Re-start

- In policy iteration, one needs to first evaluate the policy and then improve it. In terms of the simulation-based approach, the policy evaluation is done by estimating the action-value $V^g(x,u)$ from **a set of episodes** from (x,u). One may ask "can we improve the policy immediately after each episode". Then by alternating between policy evaluation and improvement more frequently, one may have better convergence performance.

- The above idea is actually widely used in the stochastic optimization problem

$$\min_w J(w) = E(f(w, X)),$$

where X is a random variable. When the distribution of X is known and computable, we can use the gradient descent (GD) method to seek for the optimal value w^* iteratively by

$$w_{k+1} = w_k - \alpha_k \nabla_w E(f(w_k, X)) = w_k - \alpha_k E(\nabla_w f(w_k, X)).$$

When the distribution of X is unknown or very difficult to compute, we can take n i.i.d. samples $(x_1, \dots, x_n)$ of random variable X and use the estimated gradient. This leads to the bath gradient descent (BGD) method

$$w_{k+1} = w_k - \alpha_k \left(\frac{1}{n} \sum_{i=1}^{n} \nabla_w E(f(w_k, x_i)) \right).$$

The **stochastic gradient descent (SGD)** method is to simply update the optimal value immediately after each sample x of X

$$w_{k+1} = w_k - \alpha_k \nabla_w E(f(w_k, x_i)).$$

Conceptually, the above algorithm is very simple, but one may ask whether it works because $\nabla_w E(f(w_k, x_i))$ may be not even closed to the actual gradient since it is just one sample. However, the nice property of SDG is that it will still converge to the optimal point when the iteration goes to infinite.

- We can apply the above technique to simulation-based policy improvement as follows.
  - We use $\mathbb{Q}(x,u)$ as a set to store all sampled values of $Q(x,u)$
  - Starting from (x,u), we use the current policy to generate an episode $\mathbf{x}$ which will contribute estimation to different (x,u)
  - We update the estimation of $Q(x,u)$ by taking $\texttt{average}[\mathbb{Q}(x,u)]$
  - Then we update the policy by $g(x) = \arg\min_{u \in \mathcal{U}} Q(x,u)$

  To ensure the correctness of the above procedure, we need to make sure that all state-action pairs initialized can be selected infinitely often. Otherwise, $Q(x,u)$ will not converge to the optimal value. This method is usually referred to as the **exploring start** method.

- An alternative approach is to use *every visit method* to collect estimations for each $Q(x,u)$, but we need to ensure that each (x,u) appears infinitely many times in a single episode $\mathbf{x}$. This can be done by changing $g(x) = \arg\min_{u \in \mathcal{U}} Q(x,u)$ to the **ϵ-greedy policy**. Then this method avoids the initialization for each episode since we can collect infinite data (because we use every visit method) for all actions (because we use randomized policy).

Recursive Estimation of Expected Value

- The above approach helps us to perform update after each episode without waiting for the completion of all episodes. However, this still needs to wait for the completion of a single episode, which might still be very long. A natural question is that "can we update the policy immediately whenever we move a single step in each episode". This is actually the idea of temporal difference learning. Before this, let us recall how we do incremental estimation for the expected value of a random variable.

- Suppose that there is a random variable X and we want to estimate its expectation $E(X)$. A direct approach is to get n i.i.d. samples $\mathbf{x} = (x_1, \ldots, x_n)$ and take

$$E(X) \approx \hat{x}_n := \frac{1}{n}\sum_{i=1}^{n} x_i.$$

Actually, the estimation $\hat{x}_n$ for n samples can be computed based on the estimation $\hat{x}_{n-1}$ for the first $n-1$ samples and the last sample x_n by

$$\underbrace{\hat{x}_n}_{\text{new est.}} = \frac{1}{n}\left(\sum_{i=1}^{n-1} x_i + x_n\right) = \frac{1}{n}\left((n-1)\hat{x}_{n-1} + x_n\right) = \underbrace{\hat{x}_{n-1}}_{\text{current est.}} + \frac{1}{n}\underbrace{(x_n - \hat{x}_{n-1})}_{\text{est. error}}.$$

Clearly, the last term $x_n - \hat{x}_{n-1}$ is the error between the observed value and the estimated value. Here, weight $\frac{1}{n}$ for the error is decreasing over time because each single observation should have less and less effect when we have sufficient previous data.

- The above derivation can actually be generalized by changing weight $1/n$ to an arbitrary time-varying sequence of weights $\{\alpha_n\}$, and we write

$$\hat{x}_n = \hat{x}_{n-1} + \alpha_n(x_n - \hat{x}_{n-1}).$$

It is known that, if weight sequence $\{\alpha_n\}$ is selected such that

1. $\sum_{i=1}^{\infty} \alpha_i^2 < \infty$; and
2. $\sum_{i=1}^{\infty} \alpha_i = \infty$,

then $\hat{x}_n$ will still converge to the expected value $E(X)$ when n goes to infinity.

- Intuitively, the first condition says that α_n will converge to zero. This ensures that the estimated value will be stable in the end since

$$\hat{x}_n - \hat{x}_{n-1} = \alpha_n(x_n - \hat{x}_{n-1}) \to 0.$$

The second condition says that α_n will not converge to zero too fast. To see this, we write

$$\hat{x}_\infty - \hat{x}_1 = \sum_{i=1}^{\infty} \alpha_i(\hat{x}_{i-1} - x_i).$$

Therefore, if $\sum_{i=1}^{\infty} \alpha_i < \infty$, then $\hat{x}_\infty - \hat{x}_1$ may be bounded, which means that the estimation process may not be able to compensate the initial error! The proof of the above procedure can be considered as a special case of the Robbins-Monro algorithm.

Incremental Estimation of State Values

- Before we go to the incremental estimation of action value, let us first consider how to estimate state value $V^g(x)$ under a given policy g. Under policy g, we can simulate the system to generate an episode, which is an infinite sequence

$$\mathbf{x} = (x_0 \xrightarrow{u_0} x_1 \xrightarrow{u_1} x_2 \xrightarrow{u_2} \cdots \xrightarrow{u_{T-1}} x_T \xrightarrow{u_T} \cdots).$$

We define $c_i = c(x_{i-1}, u_{i-1})$. Note that, our purpose is not to estimate $V^g(x)$ after the entire $\mathbf{x}$. Instead, we want to do this at each time instant $t = 0, 1, 2, \dots$. Now suppose that $V_t^g(x)$ is our estimate for state value of x at instant t. Then for the next instant, by applying an action, we have a new cost c_{t+1} and go to a new state x_{t+1}. Using this information, we can update the state value of $x_t = x$ as follows

$$\underbrace{V_{t+1}(x_t)}_{\text{new estimate}} = \underbrace{V_t(x_t)}_{\text{current estimate}} + \alpha_t(x_t)\left(\underbrace{\overbrace{(c_{t+1} + \beta V_t(x_{t+1}))}^{\text{TD target}} - V_t(x_t)}_{\text{TD error}}\right).$$

For those $x \neq x_t$, we will keep the estimation unchanged, i.e., $V_{t+1}(x) = V_t(x)$.

- The reason why we call $\bar{v}_t := c_{t+1} + \beta V_t(x_{t+1})$ *target* can be seen by $V_{t+1}(x_t) - \bar{v}_t = (1 - \alpha_t)(V_t(x_t) - \bar{v}_t)$. Since $1 - \alpha_t < 1$, $V_{t+1}(x_t)$ is more closer to $\bar{v}_t$ than $V_t(x_t)$. Also, we call $\bar{v}_t - V_t(x_t)$ the TD error, since the part is zero when $V_t = V^g$.

- By the TD algorithm, we have V_t converges to V^g as $t \to \infty$ if $\sum_{t=1}^{\infty} \alpha_t(x) = \infty$ and $\sum_{t=1}^{\infty} \alpha_t^2(x) < \infty$ for all $x \in S$, where $\alpha_t(x) = 0$ when x is not visited at instant t.

SARSA Algorithm

- As we have discussed before, only estimating the state value is not sufficient for the purpose of policy improvement. To do so, we need to estimate the action value $Q(x, u)$. This can be done similarly. Suppose that the data in the entire episode is

$$\mathbf{x} = (x_0, u_0, c_1, x_1, u_1, c_2, x_2, u_2, \dots).$$

Then at each instant t, we use the one-step data $(x_t, u_t, c_{t+1}, x_{t+1}, u_{t+1})$ to estimate the action value $Q(x_t, u_t)$ by:

$$Q_{t+1}(x_t, u_t) = Q_t(x_t, u_t) + \alpha_t(x_t, u_t)\left[(c_{t+1} + \beta Q_t(x_{t+1}, u_{t+1})) - Q_t(x_t, u_t)\right].$$

The convergence condition is similar to the previous case for state value.

- Then using the above idea with the ϵ-greedy technique, we can provide an algorithm called **SARSA** because we use the information of State-Action-Reward-State-Action.
  - Simulate the system under policy g_t and obtain a new data $(x_t, u_t, c_{t+1}, x_{t+1}, u_{t+1})$.
  - Update $Q_t(x_t, u_t)$ to $Q_{t+1}(x_t, u_t)$ based on the above update equation.
  - Update policy g_t to g_{t+1} by

$$g_t(u \mid x_t) = \begin{cases} 1 - \frac{\epsilon(|\mathcal{U}|-1)}{|\mathcal{U}|} & \text{if } u = \arg\min_u Q_{t+1}(x_t, u) \\ \frac{\epsilon}{|\mathcal{U}|} & \text{otherwise} \end{cases}.$$

Q-Learning Algorithm

- The SARSA algorithm is essentially based on the (action value version) Bellman equation such that we first estimate the value of a policy and then improve it. One may think "can we estimate the *optimal value* among all policies directly from data". This essentially requires us to solve the Bellman optimality equation from sample data. Specifically, we can write the Bellman optimality equation in terms of action value by

$$Q(x,u) = c(x,u) + \beta \sum_{x' \in S} P(x' \mid x,u) \min_{u} Q(x',u).$$

- Now, still suppose that the data in the entire episode is

$$\mathbf{x} = (x_0, u_0, c_1, x_1, u_1, c_2, x_2, u_2, \dots).$$

  Then at each instant t, we use the one-step data $(x_t, u_t, c_{t+1}, x_{t+1})$ (without u_{t+1}) to estimate the action value $Q(x_t, u_t)$ by:

$$Q_{t+1}(x_t,u_t) = Q_t(x_t,u_t) + \alpha_t(x_t,u_t)\left[(c_{t+1} + \beta \min_{u \in \mathcal{U}} Q_t(x_{t+1},u)) - Q_t(x_t,u_t)\right].$$

- Then by replacing the update equation in SARSA by the above equation, we get the **on-policy** version of the **Q-learning** algorithm. The reason why we call this on-policy is because we also use the target policy (the optimal one we are looking for) to generate behavior $\mathbf{x}$. This is also why we need to use ϵ-greedy policy; otherwise, the target policy cannot be used as a behavior policy (the actual applied one to generate data).

- Compared with SARSA, a major advantage of Q-learning is that it also supports the case of **off-policy** execution. Particularly, in SARSA, given (x_t, u_t), we note that c_{t+1} and x_{t+1} are independent of the underlying behavior policy, but u_{t+1} is determined by the behavior policy. However, in Q-learning, we do not actually use the information of u_{t+1}. Therefore, the episode data $\mathbf{x}$ can be generated by an arbitrary behavior policy and we can use this data to learn $Q_t(x,u)$, which corresponds to a different target policy. For this case, there is no need to use ϵ-greedy policy for exploration because data $\mathbf{x}$ is generated differently. Instead, we can just focus on incrementally updating the action value $Q(x,u)$ along the episode data. When the entire estimation is done, we can decode a deterministic policy for $Q(x,u)$ directly by $g(x) = \arg\min_{u \in \mathcal{U}} Q(x,u)$.