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

Lecture 4: April 7

Lecturer: Yu-Xiang Wang *Scribes: Avinash Nargund*

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

4.1 Advantage function and Performance Difference Lemma

The advantage function $A^\pi(s,a)$ gives the advantage of taking a given action over following the policy and is given by,

$$A^\pi(s,a) \coloneqq Q^\pi(s,a) - V^\pi(s)$$

For the optimal policy π^* the advantage function is always non-positive i.e $A^*(s,a) \coloneqq A^{\pi^*}(s,a) \leq 0$

The advantage function clearly describes the difference in performance of two policies.

Lemma 4.1. *(The performance difference lemma) For all policies π, π' and distributions μ over* $\mathtt{S}$.

$$V^\pi - V^{\pi'} = \frac{1}{1-\gamma}\mathbb{E}_{s'\sim d_\mu^\pi}\mathbb{E}_{a'\sim\pi(\cdot|s')}[A^{\pi'}(s',a')]$$

where

$$d_\mu^\pi(s) = (1-\gamma)\sum_{t=1}^{\infty}\gamma^{t-1}\mathbb{P}^\pi[S_t = s] = (1-\gamma)\nu_\mu^\pi(s)$$

This lemma characterizes the difference in the performance of the policies π, π' under the initial state distribution μ.

d_μ^π is the marginal distribution over entire trajectory rolled out using π of seeing s. The lemma states the difference in performance of these policies is same as inner product of advantage function and state-action occupancy measure, $\langle \mathcal{V}^\pi(s,a), A^{\pi'}(s,a)\rangle$

Proof. Let $Pr^\pi(\tau|s_0 = s)$ denote the probability of observing trajectory τ starting at state s and then

following π.

$$\begin{aligned}
V^{\pi}(s) - V^{\pi'}(s) &= \mathbb{E}_{\tau \sim Pr^{\pi}(\tau|s_0=s)} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right] - V^{\pi'}(s) \\
&= \mathbb{E}_{\tau \sim Pr^{\pi}(\tau|s_0=s)} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) + V^{\pi'}(s_t) - V^{\pi'}(s_t) \right] - V^{\pi'}(s) \\
&\overset{(a)}{=} \mathbb{E}_{\tau \sim Pr^{\pi}(\tau|s_0=s)} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) + V^{\pi'}(s_{t+1}) - V^{\pi'}(s_t) \right] \\
&\overset{(b)}{=} \mathbb{E}_{\tau \sim Pr^{\pi}(\tau|s_0=s)} \left[\sum_{t=0}^{\infty} \gamma^t \left(r(s_t, a_t) + \gamma \mathbb{E} \left[V^{\pi'}(s_{t+1}) \right) | s_t, a_t \right] - V^{\pi'}(s_t)) \right] \\
&\overset{(c)}{=} \mathbb{E}_{\tau \sim Pr^{\pi}(\tau|s_0=s)} \left[\sum_{t=0}^{\infty} \gamma^t \left(Q^{\pi'}(s_t, a_t) - V^{\pi'}(s_t) \right) \right] \\
&= \mathbb{E}_{\tau \sim Pr^{\pi}(\tau|s_0=s)} \left[\sum_{t=0}^{\infty} \gamma^t A^{\pi'}(s_t, a_t) \right] \\
&= \frac{1}{1-\gamma} \mathbb{E}_{s' \sim d_s^{\pi}} \mathbb{E}_{a' \sim \pi(\cdot|s)} \gamma^t A^{\pi'}(s', a)
\end{aligned}$$

where (a) is from rearranging the terms in the telescopic sum, (b) follows from law of iterated expectations and c from the definition of $Q^{\pi'}$ □

4.2 Finite Horizon MDPs

In finite horizon MDPs there is no discount factor. Finite horizon MDP, $M = (\mathcal{S}, \mathcal{A}, P_h, r_h, H, \mu)$ is specified by,

- State space $\mathcal{S}$ which may be infinite or finite.
- Action space $\mathcal{A}$
- Time dependent transition function $P_h : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$
- Reward function $r_h : \mathcal{S} \times \mathcal{A} \to [0, 1]$. $r_h(s, a)$ is the immediate reward associated with taking action a in state s at time step h. $r_h(s, a) = \mathbb{E}[R_h | S_h = s_h, A_h = a_h]$
- H defines the horizon
- $\mu \in \Delta(S)$ is the initial state distribution

The goal of the finite MDP problem is to find the ϵ optimal policy defined by value, $V^{\pi}(\mu) = \mathbb{E}^{\pi} \left[\sum_{t=1}^{H} R_h \right]$.

4.2.1 Bellman equations and optimal policies

A finite horizon MDP is stationary if $P_h(s'|s, a) = P_{h'}(s'|s, a) \ \forall \ 1 \leq h, h' \leq H$. But even if P and r are stationary the V and Q are not stationary.

They satisfy a set of Bellman equations,

$$\begin{aligned} V_t^\pi &= r_t^\pi + P_t^\pi V_{t+1}^\pi \in \mathcal{R}^{|\mathcal{S}|} \\ Q_t^\pi &= r + P_t V_{t+1}^{pi} \\ &= r + \tilde{P}_t^\pi Q_{t+1}^\pi \in \mathcal{R}^{|\mathcal{S}||\mathcal{A}|} \\ &\forall t = 1, 2, \ldots, H \end{aligned}$$

The optimality equations are defined separately for each timestep and are given by,

$$V_t^*(s) = \max_a \left[r_t(s,a) + \sum_{s'} P_t(s'|s,a) V_t^*(s') \right]$$

By the Markovian property it suffices to consider "nonstationary" but "memoryless" policies for finite horizon MDPs and there exists a deterministic/memoryless optimal policy. The advantage function and performance difference lemma is defined for each timestep and are given by,

$$\begin{aligned} V^\pi - V^{\tilde{\pi}} &= \sum_{h=0} {}_{H-1} \mathbb{E}_{s,a \sim \mathbb{P}_h^\pi} \left[A_h^{\tilde{\pi}}(s,a) \right] \\ A_h^\pi(s,a) &= Q_h^\pi(s,a) - V_h^\pi(s) \end{aligned}$$

4.2.2 Two-way reductions between finite and infinite horizon MDPs

Moving from infinite horizon MDP to finite horizon it suffices to consider clipped version of the inifinite horizon MDP with $H = \mathcal{O}(\frac{\log(\frac{1}{\epsilon})}{1-\gamma})$. Also, time-varying rewards have to be defined. $r_h(s,a) = \gamma^{h-1} r(s,a)$.

Reducing finite horizon MDP to infinite horizon MDP can be achieved by appending a trivial absorbing state with self-loops and zero rewards. Further the discount factor γ should be set to 1.

4.3 RL Algorithms

The dynamic programming algorithms, Policy Iteration and Value Iteration, which are used to compute the optimal V and Q functions from MDP parameters are not valid in RL. This is because the MDP parameters such as the reward function and transition kernel are not available in a RL setting. Instead, the agent will have access to the environment and it can take actions and experiment to learn the dynamics/policy rather than performing offline planning.

There are two main RL approaches - model-based and model-free RL algorithms.

4.3.1 Model-based RL

The main idea in this approach is to let the agent run in the environment, collect the experiments data and then estimate the model using this data. The general workflow is -

- Get data by running the agent and exploring the environment. The data collected will be a set of trajectories of the form -
$$\{(s_1, a_1, s_2, r_1), \ldots, (s_N, a_N, s_{N+1}, r_N)\}$$

These data points come from a distribution which depend on the actions taken by the agent and the environment. It should be noted that these data points are not independent.

- Estimate the transition kernel and reward function using the collected data. To estimate $\hat{P}(s'|s,a)$ reliably the transition has to be observed many times for each s, a.
- Then these estimated parameters can be used to create an approximate MDP and it can be solved by using the dynamic programming approaches.

$$V_{k+1}^{\pi}(s) \leftarrow \sum_{a} \pi(a|s) \sum_{s}^{'} \hat{P}(s'|s,a) \left[\hat{r}(s,a,s') + \gamma V_k^{\pi}(s')\right]$$

$$\pi' \leftarrow \operatorname*{argmax}_{a} \sum_{s}^{'} \hat{P}(s'|s,a) \left[\hat{r}(s,a,s') + \gamma V_k^{\pi}(s')\right]$$

$$V_{k+1}(s) \leftarrow \max_{a} \sum_{s}^{'} \hat{P}(s'|s,a) \left[\hat{r}(s,a,s') + \gamma V_k^{\pi}(s')\right]$$

While this method is usable with a generative model usually there are some complications. Often to reach a state the agent would have to take a particular sequence of actions. As the time horizon increases the probability of reaching such states decreases exponentially if the agent is picking actions randomly.

Generally, the model-based method is an algorithm design principle that is independent pf the data collection process. Once the data is collected the main aim is to estimate the dynamics. This can be achieved by using function approximation on P using function classes such as generative neural networks, mixture of Gaussians which provide the desired conditional probability. The simulation lemma still applies for this situation provided that the estimate $\hat{P}$ is a valid probability distribution. However since P might not belong to the chose function classes, the estimation error $(P - \hat{P})$ will be non-zero. Due to this $\hat{\pi}^* = \text{argmax}_{\pi}\, \hat{Q}^{\pi}$ will be not the maximizer for Q^{π}. Also, this approximation error will amplify the sub-optimality of $\hat{\pi}^*$.

4.3.2 Model-free RL

Model-free RL is an algorithm design principle which advocates for learning the Q function directly. The motivation behind this principle is that Q has $|\mathcal{S}||\mathcal{A}|$ parameters while the transition kernel P has $|\mathcal{S}|^2|\mathcal{A}|$ parameters. The estimate $\hat{Q}$ can be generated using function classes such as neural networks. In this case the output of the function class is not a probability distribution but a prediction of the value of the state-action pair, i.e, $\hat{Q} \in \mathcal{H} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ where $\mathcal{H}$ is the function class. Each element of $\mathcal{H}$ induce a policy class which could be given by,

$$\Pi_{\mathcal{H}} \coloneqq \left\{ \operatorname*{argmax}_{a,s} h(a,s) | \forall h \in \mathcal{H} \right\}$$

4.3.2.1 Monte Carlo Policy Evaluation (Prediction)

The performance of given policy π without the knowledge of transition dynamics can estimated by running policy π repeatedly. The first-visit MC algorithm can be used to estimate the value function $V^{\pi}(s)$. In this algorithm the average returns following the first visit to state s is collected in every episode. Then these values are averaged to obtain the value function. The discounted factor can also be incorporated during the collection of reward after each episode.

4.3.2.2 Monte Carlo Policy Optimization (Control)

For policy improvement just estimating the value function V^π is not sufficient. Alternatively, the $Q^\pi(s,a)$ can be estimated using the first visit (s,a) pair and keeping tracking of the rewards after the first visit. Once $Q^\pi(s,a)$ is estimated then policy improvement can be performed by defining a new policy as -

$$\pi'(s) = \underset{a}{\operatorname{argmax}}\, Q^\pi(s,a)$$

The new policy is then used in the policy evaluation step.
There are two issues with this method -

1. There will be exploration problems as the greedy policy will not explore all the actions. So ϵ-greedy policy or some reward bonus design might be needed.
2. This method also requires many independent episodes for the estimated value function to be accurate.

The Bellman equations can be used to improve the Monte-Carlo Q-function (empirical) estimate given by

$$\widehat{Q^\pi}(s,a) = \widehat{r^\pi}(s,a) + \gamma \widehat{\mathbb{E}}_{s' \sim P(s'|s,a)} \left[\widehat{V^\pi}(s') \right]$$

4.3.2.3 Online averaging representation of MC

Instead of waiting for the episodes to complete and then computing the estimate it possible to output an improved estimator after each episode. The online averaging update is given by -

$$V(S_t) \leftarrow V(S_t) + \alpha \left[G_t - V(S_t) \right]$$

where $\alpha = \frac{1}{N_{S_t}}$ with N_{S_t} is the number of times state S_t is encountered.

4.3.2.4 Temporal Difference Learning

This methodology is aimed at improving the Monte-Carlo approach and making it more practical. MC method needs multiple trajectories and a long exploration time to obtain a good estimate of the value function. Also, it is based on the first principle definition of V^π and doesn't exploit the recursive structure of the MDP. Hence, MC methods have poor statistical efficiency.

The Monte-Carlo approach is combined with dynamic programming approach to form an incremental update algorithm called Temporal Difference Learning.

In the MC approach G_t can be obtained only after the entire episode. The main idea of TD learning is to estimate this using Bellman equations and the recursive definition. The expected value of G_t is then given by,

$$\mathbb{E}_\pi \left[G_t \right] = \mathbb{E}_\pi \left[R_t | S_t \right] + \gamma V^\pi(S_{t+1})$$

With this only step is needed before estimating G_t. However, the true value function V^π is not available. To overcome this the current estimate of V^π can be used and the use bootstrapping to update the estimate. The TD policy evaluation is then given by,

$$V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right]$$

4.3.2.5 TD policy optimization

There are two different approaches to policy optimization in TD.

1. SARSA (On-policy) In this approach the Q function is updated by bootstrapping the Bellman equation and the update is given by,

$$Q(S, A) \leftarrow Q(S, A) + \alpha \left[R + \gamma Q(S', A') - Q(S, A)\right]$$

   The policy which used to decide the next action is always changing. The next action is usually chosen according to some function of Q e.g. greedy, ϵ-greedy.

2. Q-Learning (Off-policy) In this case the Q function is updated by bootstrapping the Bellman optimality equation. The update is given by,

$$Q(S, A) \leftarrow Q(S, A) + \alpha \left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$$

   The next action A' need not depend on Q and can be based on arbitrary policy.

These methods are proven to converge asymptotically and are much more data-efficient than MC methods. The main advantage of TD over Monte-Carlo is that given a trajectory of T steps MC updates the Q function is updated only once while TD updates it T times.