19. Key Algorithms in Reinforcement Learning

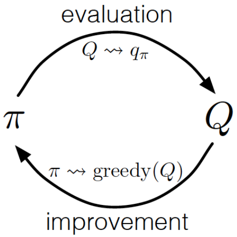
Dongwoo Kim

dongwoo.kim@postech.ac.kr

CSED515 - 2023 Spring

Welcome to Reinforcement Learning

- So far, we've studied planning problem when we know MDP completely
- This lecture focuses on reinforcement learning, in which we need learn MDP from interactions
- Recalling the generalized policy iteration to find optimal policy, we focus on the evaluation of q_π as the greedy improvement will eventually lead us to an optimal policy


evaluation
$Q \rightsquigarrow q_\pi$
π
Q
$\pi \rightsquigarrow \text{greedy}(Q)$
improvement

Outline

Key algorithms in reinforcement learning

1 Monte-Carlo methods

2 Temporal-difference learning

3 Scalable RL with function approximation

Monte Carlo (MC) Method

Basic Idea:

- To compute expectation, we use sample mean, i.e.,

$$\mathbb{E}[X] \approx \frac{1}{n}\sum_{i=1}^{n} x_i$$

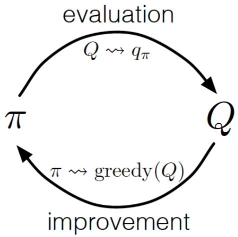
Convergence:

- Law of large number guarantees that the sample mean converges to the true expectation as (i) the number of samples increases if (ii) all samples are independent to each other (and the variance is bounded)

$$\frac{1}{n}\sum_{i=1}^{n} x_i \xrightarrow{p} \mathbb{E}[X] \quad \text{as } n \to \infty$$

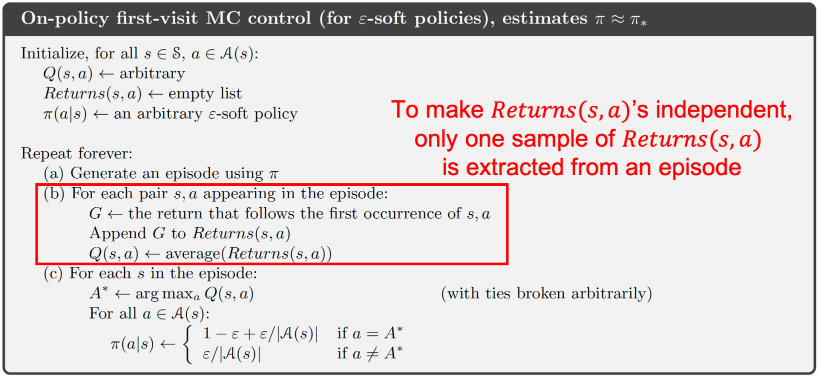
MC for RL

- For the evaluation of given policy π, take the sample mean of q_π (prediction/evaluation)
- Then, using the estimation $Q \approx q_\pi$, improve policy π greedily and repeat
- For simplicity, we focus on episodic MDP in which all episodes eventually terminate in finite time (w.p. 1)


evaluation
$Q \rightsquigarrow q_\pi$
π
Q
$\pi \rightsquigarrow \text{greedy}(Q)$
improvement

Straightforward Algorithm: Evaluation

- Lesson from law of large number: in order to guarantee unbiased estimation, samples need to be independent



On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

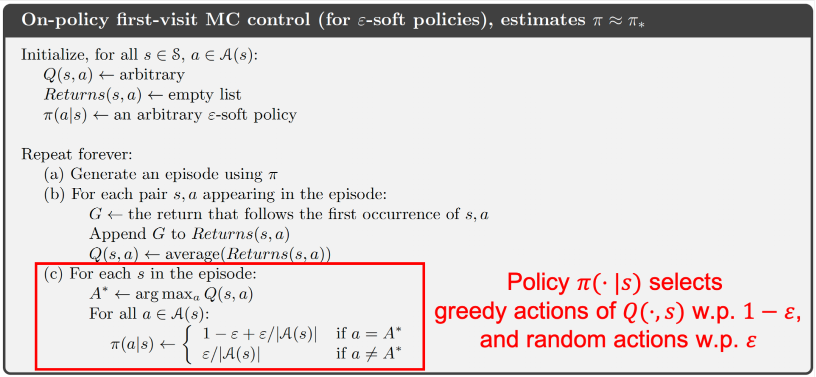
```
Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
    $Q(s,a) \leftarrow$ arbitrary
    $Returns(s,a) \leftarrow$ empty list
    $\pi(a|s) \leftarrow$ an arbitrary $\varepsilon$-soft policy

Repeat forever:
    (a) Generate an episode using $\pi$
    (b) For each pair $s, a$ appearing in the episode:
            $G \leftarrow$ the return that follows the first occurrence of $s, a$
            Append $G$ to $Returns(s,a)$
            $Q(s,a) \leftarrow \text{average}(Returns(s,a))$
    (c) For each $s$ in the episode:
            $A^* \leftarrow \arg\max_a Q(s,a)$                    (with ties broken arbitrarily)
            For all $a \in \mathcal{A}(s)$:
                $\pi(a|s) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(s)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(s)| & \text{if } a \neq A^* \end{cases}$
```

To make $Returns(s,a)$'s independent, only one sample of $Returns(s,a)$ is extracted from an episode

Straightforward Algorithm: Improvement

- Lesson from law of large number: the empirical mean gets close to the true mean when the number of samples increases
- The small noise ε in ε-greedy ensures positive probability to play each action a at each state s i.e., the number of samples for each $Q(s, a)$ can go infinite



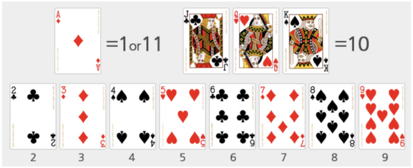
On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

```
Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
    $Q(s, a) \leftarrow$ arbitrary
    $Returns(s, a) \leftarrow$ empty list
    $\pi(a|s) \leftarrow$ an arbitrary $\varepsilon$-soft policy

Repeat forever:
    (a) Generate an episode using $\pi$
    (b) For each pair $s, a$ appearing in the episode:
            $G \leftarrow$ the return that follows the first occurrence of $s, a$
            Append $G$ to $Returns(s, a)$
            $Q(s, a) \leftarrow \text{average}(Returns(s, a))$
    (c) For each $s$ in the episode:
            $A^* \leftarrow \arg\max_a Q(s, a)$
            For all $a \in \mathcal{A}(s)$:
                $\pi(a|s) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(s)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(s)| & \text{if } a \neq A^* \end{cases}$
```

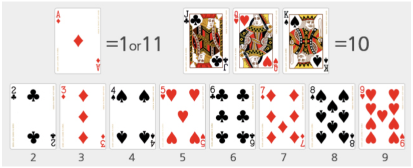
Policy $\pi(\cdot\,|s)$ selects greedy actions of $Q(\cdot, s)$ w.p. $1 - \varepsilon$, and random actions w.p. ε

Example: Blackjack



- **Object**: without exceeding 21, (i) having your card sum be greater than the dealer's (ii) making dealer's sum greater than 21 (bust), where the dealer keeps taking cards until the sum is less than 17
- **Reward**: $+1$ for winning, -1 for losing, 0 for drawing
  - No discount, i.e., $\gamma = 1$
- **Action**: hit (receiving another card) or stay (stop receiving cards)

State in Blackjack



State:

- option1 (1,000+ states): [cards in your hand; dealer's open card]
    - Do $[9, 3; 10]$ and $[6, 6; 10]$ are different?
- option2 (200 states): [current sum, useable ace, dealer's open card]
    - Cards in your hand: current sum (12-21) and indicator of useable ace (0 or 1)
    - The card that the dealer is showing (Ace - 10)

First-visit MC Control in Blackjack

Suppose the following episode is observed:

$$
\begin{aligned}
s_0 &= [12 = 9+3, \text{no ace}; 10], & a_0 &= \text{hit}, & r_1 &= 0 \\
s_1 &= [14 = 9+3+2, \text{no ace}; 10], & a_1 &= \text{hit}, & r_2 &= 0 \\
s_2 &= [21 = 9+3+2+7, \text{no ace}; 10], & a_2 &= \text{stay}, & r_3 &= 1
\end{aligned}
$$

option1 (1,000+ states): [cards in your hand; dealer's open card]

- Update in $Q([9,3;10], \text{hit})$, $Q([9,3,2;10], \text{hit})$, $Q([9,3,2,7;10], \text{stay})$
- The experience on the case starting with $9,3$ in my hand cannot generalized to the cases starting with $10,2$, $8,4$, $7,5$ and $6,6$ in my hand, i.e., sample inefficient

option2 (200 states): [current sum, useable ace, dealer's open card]

- Update in $Q([12, \text{no ace}; 10], \text{hit})$, $Q([14, \text{no ace}; 10], \text{hit})$, $Q([21, \text{no ace}; 10], \text{stay})$
- The experience on the case starting with $9+3$ in my hand is generalized to the cases starting with $10+2$, $8+4$, $7+5$ and $6+6$ in my hand, i.e., sample efficient

Drawback of Monte-Carlo Methods

- Long delay
  - We need to wait the end of episode for an update
  - What if there is no terminal state, i.e., continuing MDP?
- High variance
  - The sample mean $\bar{G} = \frac{1}{n}\sum_{i=1}^{n} G^i$ is an unbiased estimation of $\mathbb{E}[G]$, i.e., $\mathbb{E}[\bar{G}] = \mathbb{E}[G]$
  - However, its variance is high when the length of episode is long as return G includes more randomness in longer episode

$$\mathrm{Var}[\bar{G}] = \frac{1}{n}\mathrm{Var}[G]$$

Outline

Key algorithms in reinforcement learning

1 Monte-Carlo methods

2 Temporal-difference learning

3 Scalable RL with function approximation

Rewriting Monte Carlo (MC) Method

Basic Idea:

- To compute expectation, we use the sample mean (a.k.a. empirical mean), i.e.,

$$
\begin{aligned}
\mathbb{E}[X] \approx S_n &:= \frac{1}{n}\sum_{i=1}^{n} x_i = \frac{1}{n}\sum_{i=1}^{n-1} x_i + \frac{1}{n}x_n \\
&= \left(\frac{n}{n(n-1)} - \frac{1}{n(n-1)}\right)\sum_{i=1}^{n-1} x_i + \frac{1}{n}x_n \\
&= \frac{1}{n-1}\sum_{i=1}^{n-1} x_i + \frac{1}{n}\left[x_n - \frac{1}{n-1}\sum_{i=1}^{n-1} x_i\right] \\
&= S_{n-1} + \frac{1}{n}\left[x_n - S_{n-1}\right]
\end{aligned}
$$

Rewriting Monte Carlo Prediction

- For every episode and time t, if (S_t, A_t) is the first visit, then update the estimated action-value Q as:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[G_t - Q(S_t, A_t)\right]$$

- The update aims at $Q(s, a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a]$, i.e., stochastic approximation

- α is a step-size parameter, e.g., if we set $\alpha = 1/n_{S_t, A_t}$ where $n_{s,a}$ is the number of episodes visiting (s, a) so far, then the above just computes the empirical mean of value

- Constant α is useful for non-stationary environment, c.f., π is changing

Bellman Equation for Q-function

$$\begin{aligned}
q_\pi(s,a) &= \mathbb{E}_\pi\left[\sum_{k=1}^{T-t}\gamma^{k-1}R_{t+k} \mid S_t = s, A_t = a\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma \sum_{k=1}^{T-(t+1)}\gamma^{k-1}R_{(t+1)+k} \mid S_t = s, A_t = a\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma \sum_{k=1}^{T-(t+1)}\gamma^{k-1}R_{(t+1)+k} \mid S_t = s, A_t = a\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a\right] \\
&= \sum_{s',r} p(s',r|s,a)\left[r + \gamma \sum_{a'} \pi(a' \mid s') q_\pi(s',a')\right]
\end{aligned}$$

Temporal Difference Learning for Control

- Bellman equation for Q-function:

$$q_\pi(s,a) = \mathbb{E}_\pi\left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a\right]$$

- This leads us the following stochastic approximation:

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

  where the blue part is often called temporal difference error as it quantifies the difference between estimations of $Q(S_t, A_t)$ at two different times t and $t+1$:

  - $Q(S_t, A_t)$: estimate right after S_t, A_t
  - $R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$: estimate right after $R_{t+1}, S_{t+1}, A_{t+1}$

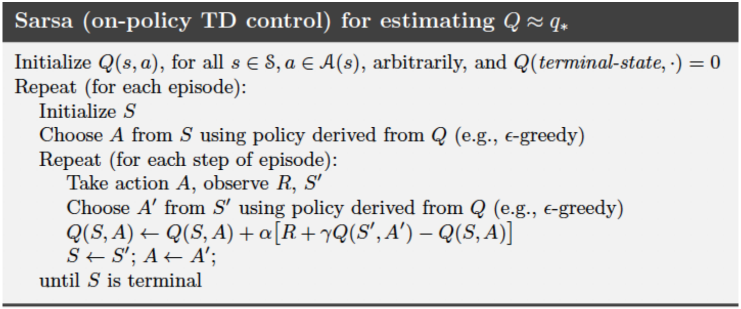
- This is an bootstrapping method as it updates an estimate $Q(S_t, A_t)$ using some other estimate $Q(S_{t+1}, A_{t+1})$

SARSA Algorothm

The iterative algorithm,

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \;,$$

is called SARSA algorithm



Sarsa (on-policy TD control) for estimating $Q \approx q_*$

```
Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
   Initialize $S$
   Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
   Repeat (for each step of episode):
      Take action $A$, observe $R$, $S'$
      Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
      $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
      $S \leftarrow S'$; $A \leftarrow A'$;
   until $S$ is terminal
```

Understanding Temporal Difference (TD)

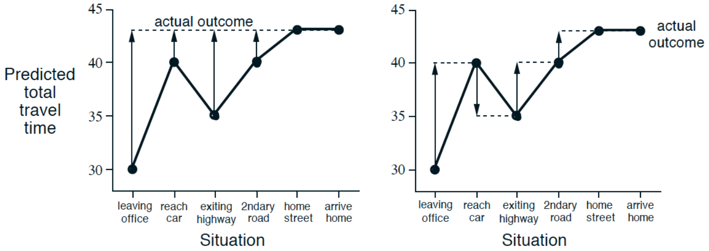
Consider predicting time to arrival when driving home:

State	*Elapsed Time (minutes)*	*Predicted Time to Go*	*Predicted Total Time*
leaving office, friday at 6	0	30	30
reach car, raining	5	35	40
exiting highway	20	15	35
2ndary road, behind truck	30	10	40
entering home street	40	3	43
arrive home	43	0	43

- As state evolves, we can do better predictions

Advantage of TD

MC vs. TD:



- TD method provides prompt update of predictions at each time
- Such a prompt update of prediction may be useful to find optimal policy as we need to keep evaluating policies, although hasty learning based on guess can be somewhat biased if the other guess is not estimated well

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

MC vs. TD

- Monte Carlo method
  - Pros: robustness (no bootstrapping)
  - Cons: long delay
- Temporal-difference method
  - Pros: short delay (online)
  - Cons: (possibly) biased estimation

In practice, TD methods have usually been found to converge faster than constant-α MC method on stochastic tasks. Example : Random walk.

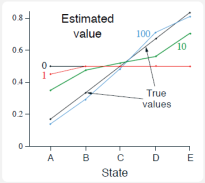
Example 6.2 Random Walk

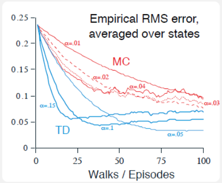
In this example we empirically compare the prediction abilities of TD(0) and constant-α MC when applied to the following Markov reward process:

$$\begin{tikzcd} \blacksquare & \mathsf{A} \arrow[l, "0"'] \arrow[r, leftrightarrow, "0"] & \mathsf{B} \arrow[r, leftrightarrow, "0"] & \mathsf{C} \arrow[r, leftrightarrow, "0"] & \mathsf{D} \arrow[r, leftrightarrow, "0"] & \mathsf{E} \arrow[r, "1"] & \blacksquare \end{tikzcd}$$

start

A *Markov reward process*, or MRP, is a Markov decision process without actions. We will often use MRPs when focusing on the prediction problem, in which there is no need to distinguish the dynamics due to the environment from those due to the agent. In this MRP, all episodes start in the center state, C, then proceed either left or right by one state on each step, with equal probability. Episodes terminate either on the extreme left or the extreme right. When an episode terminates on the right, a reward of $+1$ occurs; all other rewards are zero. For example, a typical episode might consist of the following state-and-reward sequence: $\mathsf{C}, 0, \mathsf{B}, 0, \mathsf{C}, 0, \mathsf{D}, 0, \mathsf{E}, 1$. Because this task is undiscounted, the true value of each state is the probability of terminating on the right if starting from that state. Thus, the true value of the center state is $v_\pi(\mathsf{C}) = 0.5$. The true values of all the states, A through E, are $\frac{1}{6}, \frac{2}{6}, \frac{3}{6}, \frac{4}{6}$, and $\frac{5}{6}$.





The left graph above shows the values learned after various numbers of episodes on a single run of TD(0). The estimates after 100 episodes are about as close as they ever come to the true values—with a constant step-size parameter ($\alpha = 0.1$ in this example), the values fluctuate indefinitely in response to the outcomes of the most recent episodes. The right graph shows learning curves for the two methods for various values of α. The performance measure shown is the root mean-squared (RMS) error between the value function learned and the true value function, averaged over the five states, then averaged over 100 runs. In all cases the approximate value function was initialized to the intermediate value $V(s) = 0.5$, for all s. The TD method was consistently better than the MC method on this task.

A Variant of SARSA: Expected SARSA

- SARSA

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

from

$$\begin{aligned} q_\pi(s, a) &= \mathbb{E}_\pi \left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a\right] \\ &= \mathbb{E}_\pi \left[R_{t+1} + \gamma \mathbb{E}_\pi[q_\pi(S_{t+1}, A_{t+1}) \mid S_{t+1}] \mid S_t = s, A_t = a\right] \end{aligned}$$

- Expected SARSA

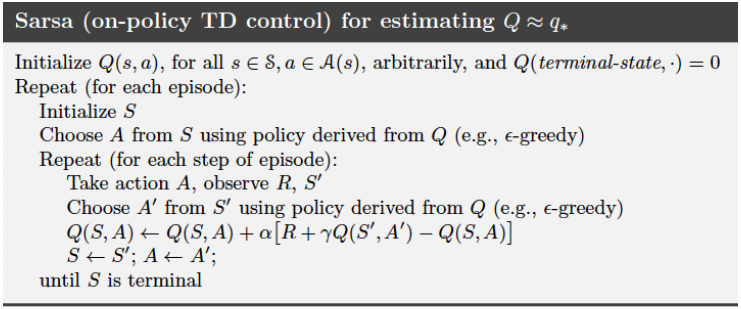
$$\begin{aligned} Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \mathbb{E}_\pi[Q(S_{t+1}, A_{t+1}) | S_{t+1}] - Q(S_t, A_t)] \\ &= Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \sum_a \pi(a \mid S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t)] \end{aligned}$$

which reduces the variance from random selection of A_{t+1}

On-policy vs. Off-policy

We've focused on on-policy setting: behavior policy = target policy

- *Behavior policy* is the one *generating* samples
- *Target policy* is the one *being evaluated*
- e.g.,



Sarsa (on-policy TD control) for estimating $Q \approx q_*$

```
Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
   Initialize $S$
   Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
   Repeat (for each step of episode):
      Take action $A$, observe $R$, $S'$
      Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
      $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$
      $S \leftarrow S'$; $A \leftarrow A'$;
   until $S$ is terminal
```

On-policy vs. Off-policy

We've focused on on-policy setting: behavior policy $b =$ target policy π

- *Behavior policy* is the one *generating* samples
- *Target policy* is the one *being evaluated*

However, we often need to consider off-policy setting: behavior policy $b \neq$ target policy π

- We may want to use data generated previously
- We cannot control systems to generate samples

Importance Sampling

- Suppose we want to compute $\mathbb{E}_\pi[X]$, while we can sample X from only distribution $b \neq \pi$
- Note that

$$\mathbb{E}_{\pi}[X] = \sum_x \pi(x)x = \sum_x b(x)\frac{\pi(x)}{b(x)}x = \mathbb{E}_{b}\left[\frac{\pi(X)}{b(X)}X\right]$$

where $\frac{\pi(X)}{b(X)}$ is often called importance sampling ratio

- This provides a translation of sample mean for π from samples x_i's are from μ

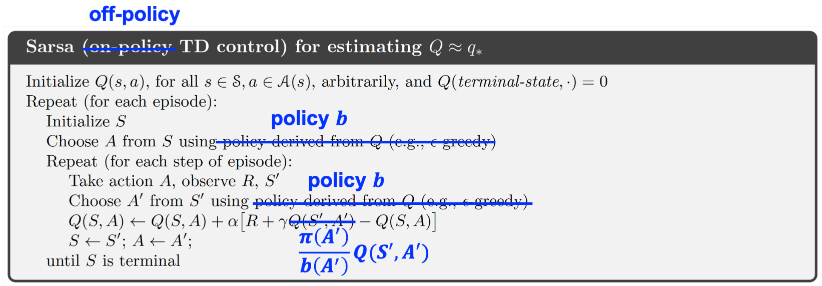
$$\mathbb{E}_\pi[X] \approx \frac{1}{n}\sum_{i=1}^{n}\frac{\pi(x_i)}{b(x_i)}x_i$$

Off-policy SARSA

Rewrite Bellman equation as follows:

$$
\begin{aligned}
q_\pi(s,a) &= \mathbb{E}_\pi\left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a\right] \\
&= \mathbb{E}_b\left[R_{t+1} + \gamma \frac{\pi(A_{t+1})}{b(A_{t+1})} q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a\right]
\end{aligned}
$$

Then, off-policy SARSA is



off-policy

Sarsa ~~(on-policy~~ TD control) for estimating $Q \approx q_*$

```
Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(terminal\text{-}state, \cdot) = 0$
Repeat (for each episode):
   Initialize $S$
   Choose $A$ from $S$ using ~~policy derived from $Q$ (e.g., $\epsilon$-greedy)~~ policy $b$
   Repeat (for each step of episode):
      Take action $A$, observe $R$, $S'$
      Choose $A'$ from $S'$ using ~~policy derived from $Q$ (e.g., $\epsilon$-greedy)~~ policy $b$
      $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \cancel{Q(S',A')} \frac{\pi(A')}{b(A')} Q(S',A') - Q(S,A)\big]$
      $S \leftarrow S'$; $A \leftarrow A'$;
   until $S$ is terminal
```

Optimal Bellman Equation

What if we don't know behavior policy b?

- If a policy π is optimal, then

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}_\pi\left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a\right] \\ &= \mathbb{E}_\pi\left[R_{t+1} + \gamma \max_{a'} q_\pi(S_{t+1}, a') \mid S_t = s, A_t = a\right] \\ &= \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_\pi(S_{t+1}, a') \mid S_t = s, A_t = a\right] \end{aligned}$$

- Hence, the optimal Q-function, denoted by q_*, must verify

$$q_*(s,a) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_\pi(S_{t+1}, a') \mid S_t = s, A_t = a\right]$$

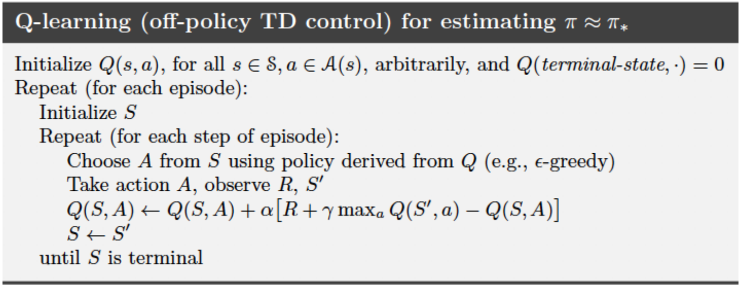
which is the optimal Bellman equation

Q-learning: Off-policy TD Control

The optimal Bellman equation provides an off-policy TD control:

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

which is called Q-learning, and can use any arbitrary behavior policy generating episodes



Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

```
Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize $S$
    Repeat (for each step of episode):
        Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
        Take action $A$, observe $R$, $S'$
        $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
        $S \leftarrow S'$
    until $S$ is terminal
```

c.f., on-policy TD control: SARSA

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

Remark on Off-policy vs. On-policy

Off-policy setting: behavior policy $\neq$ target policy

- Pros) can be used even when interaction with environment is limited
- Cons) can be unstable: samples generated by behavior policy need to be translated into target policy, e.g., max operator in Q-learning or importance sampling in off-policy SARSA

On-policy setting: behavior policy $=$ target policy

- Pros) may be stable: no translation of samples is required
- Cons) can be used only when we can freely interact with environment, e.g., virtual reality, games...

Outline

Key algorithms in reinforcement learning

1 Monte-Carlo methods

2 Temporal-difference learning

3 Scalable RL with function approximation

Toward Scalable Reinforcement Learning

- Methods that we've learned so far require tables of system size
  - Transition probability $p(s' \mid s, a)$
  - Mean reward $r(s, a)$
  - State-value function $V(s)$
  - Action-value function $Q(s, a)$

- Such tabular representations are often infeasible but also inefficient for large system in practice, e.g., system size of video game/driving car
  - We can't maintain tables of infinite size
  - We can't experience every state and action, while an experience can be generalized to similar ones

- Any solution? Function approximation (FA)!

RL with FA

We approximate action-value function instead, i.e.,

$$\hat{q}(s, a; \boldsymbol{w}) \approx q_*(s, a)$$

- e.g.1, linear model

$$\hat{q}(s, a; \boldsymbol{w}) = \sum_{i=1}^{K} w_i f_i(s, a) = \boldsymbol{w}^\top \boldsymbol{f}(s, a)$$

  where $\boldsymbol{f}(s, a)$ is a feature vector of (s, a), e.g.,
  $f_1(s, \text{throttle}) = \mathbb{1}[\text{throttle} \in [0\%, 20\%]]$,
  $f_2(s, \text{throttle}) = \mathbb{1}[\text{throttle} \in [20\%, 40\%]]$, ...

- e.g.2, neural network model

$$\hat{q}(s, \cdot; \boldsymbol{w}) = \text{FC}_{\boldsymbol{w}_{\text{FC}}}(\text{CNN}_{\boldsymbol{w}_{\text{CNN}}}(s)) \in \mathbb{R}^{\mathcal{A}}$$

On-policy RL with FA

- We approximate action-value function instead, i.e.,

$$\hat{q}(s, a; \boldsymbol{w}) \approx q_\pi(s, a)$$

- To find optimal action-value function, one may interleave the followings:
    - Approximate policy evaluation using value function approximation, i.e., minimizing

    $$\overline{VE}(\boldsymbol{w}) = \mathbb{E}_\pi[(q_\pi(s, a) - \hat{q}(s, a; \boldsymbol{w}))^2]$$

    which is an on-policy objective
    - Perform ε-greedy policy improvement, i.e., π is ε-greedy w.r.t. $\hat{q}(\cdot, \cdot; \boldsymbol{w})$

Semi-gradient Method

$$\text{minimize}_{\boldsymbol{w}} \quad \overline{VE}(\boldsymbol{w}) = \mathbb{E}_\pi[(q_\pi(s,a) - \hat{q}(s,a;\boldsymbol{w}))^2]$$

To use the gradient descent method, we compute

$$\begin{aligned} -\frac{1}{2}\nabla_{\boldsymbol{w}}\overline{VE}(\boldsymbol{w}) &= -\frac{1}{2}\nabla_{\boldsymbol{w}}\,\mathbb{E}_\pi[(q_\pi(s,a) - \hat{q}(s,a;\boldsymbol{w}))^2] \\ &= \mathbb{E}_\pi[(q_\pi(s,a) - \hat{q}(s,a;\boldsymbol{w}))]\nabla_{\boldsymbol{w}}\,\mathbb{E}_\pi[\hat{q}(s,a;\boldsymbol{w})] \\ &= \mathbb{E}_\pi[(q_\pi(s,a) - \hat{q}(s,a;\boldsymbol{w}))]\,\mathbb{E}_\pi[\nabla_{\boldsymbol{w}}\hat{q}(s,a;\boldsymbol{w})] \end{aligned}$$

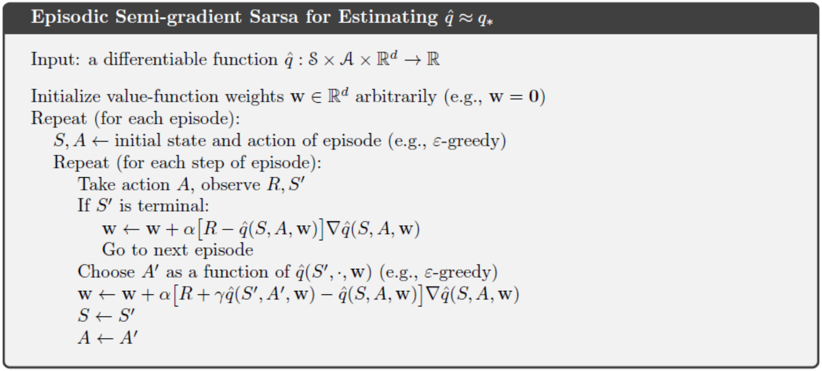
As we don't know q_π, we sample or bootstrap it (semi-gradient)

- Monte-Carlo:

$$\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t + \alpha(G_t - \hat{q}(S_t, A_t;\boldsymbol{w}))\nabla_{\boldsymbol{w}}\hat{q}(S_t, A_t;\boldsymbol{w})$$

- Temporal-Difference:

$$\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t + \alpha(R_t + \gamma\hat{q}(S_{t+1}, A_{t+1};\boldsymbol{w}) - \hat{q}(S_t, A_t;\boldsymbol{w}))\nabla_{\boldsymbol{w}}\hat{q}(S_t, A_t;\boldsymbol{w})$$



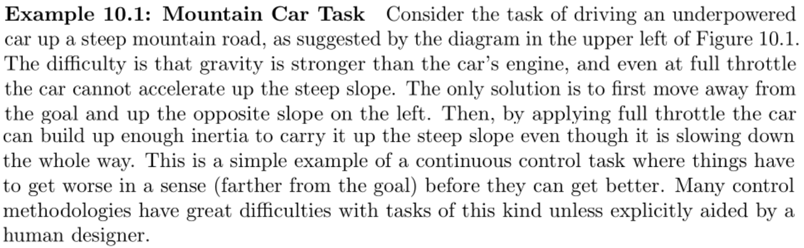
Episodic Semi-gradient Sarsa for Estimating $\hat{q} \approx q_*$

```
Input: a differentiable function $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$

Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)
Repeat (for each episode):
  $S, A \leftarrow$ initial state and action of episode (e.g., $\varepsilon$-greedy)
  Repeat (for each step of episode):
    Take action $A$, observe $R, S'$
    If $S'$ is terminal:
      $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
      Go to next episode
    Choose $A'$ as a function of $\hat{q}(S', \cdot, \mathbf{w})$ (e.g., $\varepsilon$-greedy)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
    $S \leftarrow S'$
    $A \leftarrow A'$
```

[from Sutton and Barto]

Example: Mountain Car Task



Example 10.1: Mountain Car Task Consider the task of driving an underpowered car up a steep mountain road, as suggested by the diagram in the upper left of Figure 10.1. The difficulty is that gravity is stronger than the car's engine, and even at full throttle the car cannot accelerate up the steep slope. The only solution is to first move away from the goal and up the opposite slope on the left. Then, by applying full throttle the car can build up enough inertia to carry it up the steep slope even though it is slowing down the whole way. This is a simple example of a continuous control task where things have to get worse in a sense (farther from the goal) before they can get better. Many control methodologies have great difficulties with tasks of this kind unless explicitly aided by a human designer.

The reward in this problem is -1 on all time steps until the car moves past its goal position at the top of the mountain, which ends the episode. There are three possible actions: full throttle forward $(+1)$, full throttle reverse (-1), and zero throttle (0). The car moves according to a simplified physics. Its position, x_t, and velocity, $\dot{x}_t$, are updated by

$$x_{t+1} \doteq bound\big[x_t + \dot{x}_{t+1}\big]$$

$$\dot{x}_{t+1} \doteq bound\big[\dot{x}_t + 0.001 A_t - 0.0025\cos(3x_t)\big],$$

where the *bound* operation enforces $-1.2 \leq x_{t+1} \leq 0.5$ and $-0.07 \leq \dot{x}_{t+1} \leq 0.07$. In addition, when x_{t+1} reached the left bound, $\dot{x}_{t+1}$ was reset to zero. When it reached the right bound, the goal was reached and the episode was terminated. Each episode started from a random position $x_t \in [-0.6, -0.4)$ and zero velocity. To convert the two continuous state variables to binary features, we used grid-tilings as in Figure 9.9. We used 8 tilings, with each tile covering 1/8th of the bounded distance in each dimension, and asymmetrical offsets as described in Section 9.5.4.[1] The feature vectors $\mathbf{x}(s,a)$ created by tile coding were then combined linearly with the parameter vector to approximate the action-value function:

$$\hat{q}(s,a,\mathbf{w}) \doteq \mathbf{w}^\top \mathbf{x}(s,a) = \sum_{i=1}^{d} w_i \cdot x_i(s,a), \tag{10.3}$$

for each pair of state, s, and action, a.

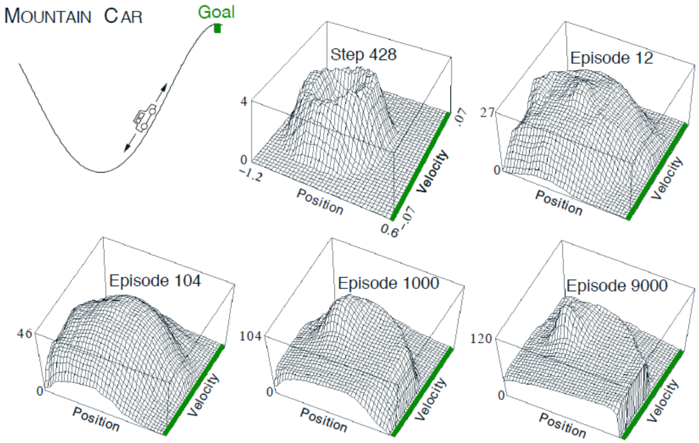


Figure 10.1: The Mountain Car task (upper left panel) and the cost-to-go function ($-\max_a \hat{q}(s, a, \mathbf{w})$) learned during one run.

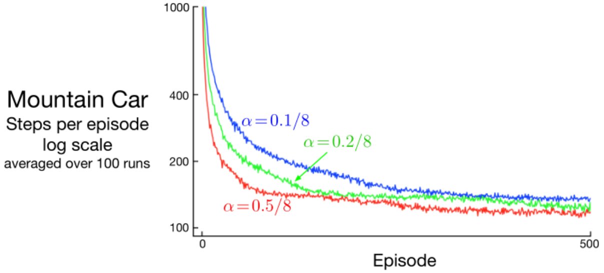


Figure 10.2: Mountain Car learning curves for the semi-gradient Sarsa method with tile-coding function approximation and ε-greedy action selection. ■

Off-policy RL wtih FA (1)

To find optimal action-value function, one may interleave the followings:

- Approximate policy evaluation using value function approximation, i.e., minimizing

$$\overline{VE}(\boldsymbol{w}) = \mathbb{E}_{b}[(q_{\pi}(s,a) - \hat{q}(s,a;\boldsymbol{w}))^2]$$

- Perform ε-greedy policy improvement, i.e., π is ε-greedy w.r.t. $\hat{q}(\cdot,\cdot;\boldsymbol{w})$

Off-policy SARSA with importance sampling and FA

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha \left(R + \gamma \frac{\pi(A')}{b(A')} \hat{q}(S',A';\boldsymbol{w}) - \hat{q}(S,A;\boldsymbol{w}) \right) \nabla_{\boldsymbol{w}} \hat{q}(S,A;\boldsymbol{w})$$

where S, A, R, S', A' are from behavior policy b

Off-policy RL wtih FA (2)

To find optimal action-value function, one may directly minimize

$$\overline{VE}(\boldsymbol{w}) = \mathbb{E}_b[(q_*(s,a) - \hat{q}(s,a;\boldsymbol{w}))^2]$$

Q-learning with FA

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha \left(R + \gamma \max_{a'} \hat{q}(S',a';\boldsymbol{w}) - \hat{q}(S,A;\boldsymbol{w})\right) \nabla_{\boldsymbol{w}} \hat{q}(S,A;\boldsymbol{w})$$

where S, A, R, S', A' are from behavior policy b

Q-learning with FA and target-behavior separation (for stability)

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha \left(R + \gamma \max_{a'} \hat{q}(S',a';\boldsymbol{w}_-) - \hat{q}(S,A;\boldsymbol{w})\right) \nabla_{\boldsymbol{w}} \hat{q}(S,A;\boldsymbol{w})$$

where $\boldsymbol{w}_-$ is set to $\boldsymbol{w}$ at every C ($\gg 1$) steps

Summary: RL with FA

Basic update rule:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha \Delta \boldsymbol{w}$$

- On-policy Monte Carlo: G_t, S_t, A_t sampled from ε-greedy π

$$\Delta \boldsymbol{w} = (G_t - \hat{q}(S_t, A_t; \boldsymbol{w})) \nabla_{\boldsymbol{w}} \hat{q}(S_t, A_t; \boldsymbol{w})$$

- On-policy TD: $S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}$ sampled from ε-greedy π

$$\Delta \boldsymbol{w} = (R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}; \boldsymbol{w}) - \hat{q}(S_t, A_t; \boldsymbol{w})) \nabla_{\boldsymbol{w}} \hat{q}(S_t, A_t; \boldsymbol{w})$$

- Off-policy TD: $S_t, A_t, R_{t+1}, S_{t+1}$ sampled from behavior policy b

$$\Delta \boldsymbol{w} = (R_t + \gamma \max_{a'} \hat{q}(S_t, a'; \boldsymbol{w}) - \hat{q}(S_t, A_t; \boldsymbol{w})) \nabla_{\boldsymbol{w}} \hat{q}(S_t, A_t; \boldsymbol{w})$$

Remark: Convergence

- There have been long line of work on convergence of aforementioned algorithms, where as necessary assumption, the function approximation has linearity or smoothness.
- Recent deep RL algorithms follow the form of semi-gradient method, while a set of techniques to provide stable and fast convergence have been studied extensively,
  - e.g., experience replay, prioritized experience replay, trust region, ...

Remark: Policy-gradient

- So far, we've relied on greedy improvement w.r.t. Q-function
- However, $\arg\max_{a \in \mathcal{A}(s)} Q(s, a)$ can be intractable when the number of actions is large or infinite
- Policy-gradient methods approximating policy $\pi_\theta(a \mid s)$
    - e.g., $\pi(a|s;\theta) = \mathcal{N}(a|\mu(s;\theta), \sigma^2(s;\theta))$ (differentiable w.r.t. θ)
    - Goal: $\text{maximize}_\theta \, v_{\pi_\theta}(s_0)$
    - Update: $\theta \leftarrow \theta + \nabla_\theta v_{\pi_\theta}(s_0)$
- Take CSED627/AIGS627 Reinforcement Learning!

Summary

We have studied

1. Monte-Carlo methods
2. Temporal-difference learning
3. Scalable RL with function approximation

Remark: CSED627/AIGS627 Reinforcement Learning