

Lecture 3: Model-free Prediction and Control

Bolei Zhou

<https://github.com/zhoubolei/introRL>

March 29, 2020

This Week's Plan

① Last time

- ① MDP, policy evaluation, policy iteration and value iteration for solving a **known** MDP

② This time

- ① Model-free prediction: Estimate value function of an **unknown** MDP
- ② Model-free control: Optimize value function of an **unknown** MDP

Review on the control in MDP

- ① When the MDP is known?
 - ① Both R and P are exposed to the agent
 - ② Therefore we can run policy iteration and value iteration
- ② Policy iteration: Given a known MDP, compute the optimal policy and the optimal value function
 - ① Policy evaluation: iteration on the Bellman expectation backup

$$v_i(s) = \sum_{a \in \mathcal{A}} \pi(a|s) (R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) v_{i-1}(s'))$$

- ② Policy improvement: greedy on action-value function q

$$q_{\pi_i}(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) v_{\pi_i}(s')$$

$$\pi_{i+1}(s) = \arg \max_a q_{\pi_i}(s, a)$$

Review on the control in MDP

- ① Value iteration: Given a known MDP, compute the optimal value function
- ② Iteration on the Bellman optimality backup

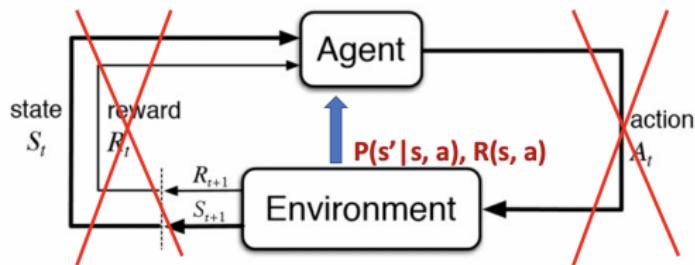
$$v_{i+1}(s) \leftarrow \max_{a \in \mathcal{A}} R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a)v_i(s')$$

- ③ To retrieve the optimal policy after the value iteration:

$$\pi^*(s) \leftarrow \arg \max_a R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a)v_{end}(s') \quad (1)$$

RL with knowing how the world works

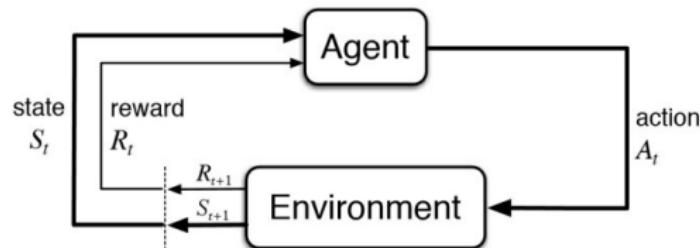
- ① Both of the policy iteration and value iteration assume the direct access to the **dynamics** and **rewards** of the environment



- ② In a lot of real-world problems, MDP model is either unknown or known by too big or too complex to use
 - ① Atari Game, Game of Go, Helicopter, Portfolio management, etc

Model-free RL: Learning by interaction

- ① Model-free RL can solve the problems through interaction with the environment



- ② No more direct access to the known transition dynamics and reward function
- ③ Trajectories/episodes are collected by the agent's interaction with the environment
- ④ Each trajectory/episode contains $\{S_1, A_1, R_1, S_2, A_2, R_2, \dots, S_T, A_T, R_T\}$

Model-free prediction: policy evaluation without the access to the model

- ① Estimating the expected return of a particular policy if we don't have access to the MDP models
 - ① Monte Carlo policy evaluation
 - ② Temporal Difference (TD) learning

Monte-Carlo Policy Evaluation

- ① Return: $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$ under policy π
- ② $v^\pi(s) = \mathbb{E}_{\tau \sim \pi}[G_t | s_t = s]$, thus expectation over trajectories τ generated by following π
- ③ MC simulation: we can simply sample a lot of trajectories, compute the actual returns for all the trajectories, then average them
- ④ MC policy evaluation uses empirical mean return instead of expected return
- ⑤ MC does not require MDP dynamics/rewards, no bootstrapping, and does not assume state is Markov.
- ⑥ Only applied to episodic MDPs (each episode terminates)

Monte-Carlo Policy Evaluation

- ① To evaluate state $v(s)$
 - ① Every time-step t that state s is visited in an episode,
 - ② Increment counter $N(s) \leftarrow N(s) + 1$
 - ③ Increment total return $S(s) \leftarrow S(s) + G_t$
 - ④ Value is estimated by mean return $v(s) = S(s)/N(s)$
- ② By law of large numbers, $v(s) \rightarrow v^\pi(s)$ as $N(s) \rightarrow \infty$

Incremental Mean

Mean from the average of samples x_1, x_2, \dots

$$\begin{aligned}\mu_t &= \frac{1}{t} \sum_{j=1}^t x_j \\ &= \frac{1}{t} \left(x_t + \sum_{j=1}^{t-1} x_j \right) \\ &= \frac{1}{t} (x_t + (t-1)\mu_{t-1}) \\ &= \mu_{t-1} + \frac{1}{t} (x_t - \mu_{t-1})\end{aligned}$$

Incremental MC Updates

- ① Collect one episode $(S_1, A_1, R_1, \dots, S_t)$
- ② For each state s_t with computed return G_t

$$N(S_t) \leftarrow N(S_t) + 1$$
$$\nu(S_t) \leftarrow \nu(S_t) + \frac{1}{N(S_t)}(G_t - \nu(S_t))$$

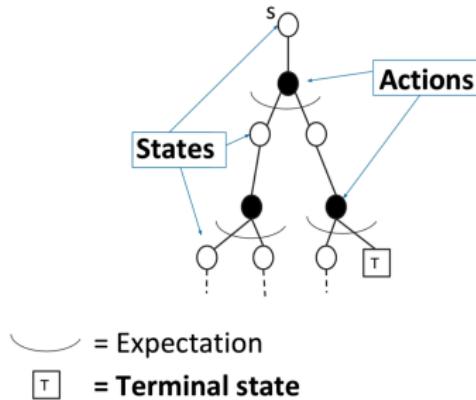
- ③ Or use a running mean (old episodes are forgotten). Good for non-stationary problems.

$$\nu(S_t) \leftarrow \nu(S_t) + \alpha(G_t - \nu(S_t))$$

Difference between DP and MC for policy evaluation

- ① Dynamic Programming (DP) computes v_i by bootstrapping the rest of the expected return by the value estimate v_{i-1}
- ② Iteration on Bellman expectation backup:

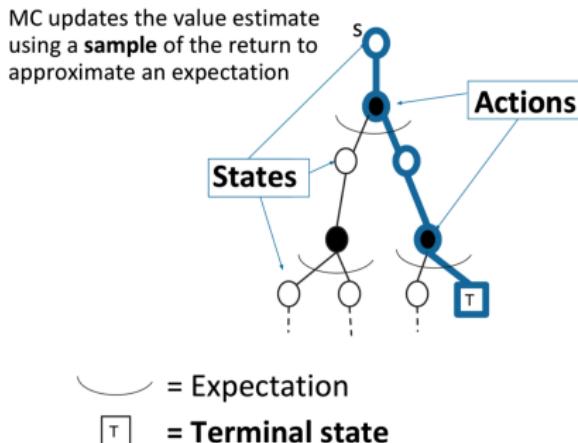
$$v_i(s) \leftarrow \sum_{a \in \mathcal{A}} \pi(a|s) \left(R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) v_{i-1}(s') \right)$$



Difference between DP and MC for policy evaluation

- ① MC updates the empirical mean return with one sampled episode

$$v(S_t) \leftarrow v(S_t) + \alpha(G_{i,t} - v(S_t))$$



Advantages of MC over DP

- ① MC works when the environment is unknown
- ② Working with sample episodes has a huge advantage, even when one has complete knowledge of the environment's dynamics, for example, transition probability is complex to compute
- ③ Cost of estimating a single state's value is independent of the total number of states. So you can sample episodes starting from the states of interest then average returns

Temporal-Difference (TD) Learning

- ① TD methods learn directly from episodes of experience
- ② TD is model-free: no knowledge of MDP transitions/rewards
- ③ TD learns from **incomplete** episodes, by bootstrapping

Temporal-Difference (TD) Learning

- ① Objective: learn v_π online from experience under policy π
- ② Simplest TD algorithm: TD(0)

- ① Update $v(S_t)$ toward estimated return $R_{t+1} + \gamma v(S_{t+1})$

$$v(S_t) \leftarrow v(S_t) + \alpha(R_{t+1} + \gamma v(S_{t+1}) - v(S_t))$$

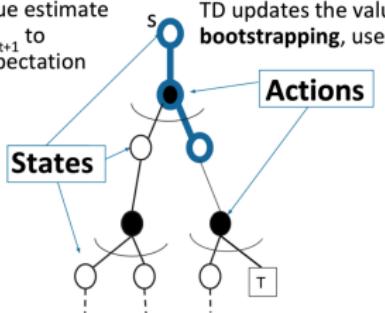
- ③ $R_{t+1} + \gamma v(S_{t+1})$ is called TD target
- ④ $\delta_t = R_{t+1} + \gamma v(S_{t+1}) - v(S_t)$ is called the TD error
- ⑤ Comparison: Incremental Monte-Carlo

- ① Update $v(S_t)$ toward actual return G_t given an episode i

$$v(S_t) \leftarrow v(S_t) + \alpha(G_{i,t} - v(S_t))$$

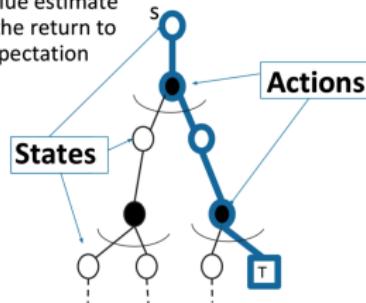
Advantages of TD over MC

TD updates the value estimate using a **sample** of s_{t+1} to approximate an expectation



TD updates the value estimate by **bootstrapping**, uses estimate of $V(s_{t+1})$

MC updates the value estimate using a **sample** of the return to approximate an expectation



= Expectation

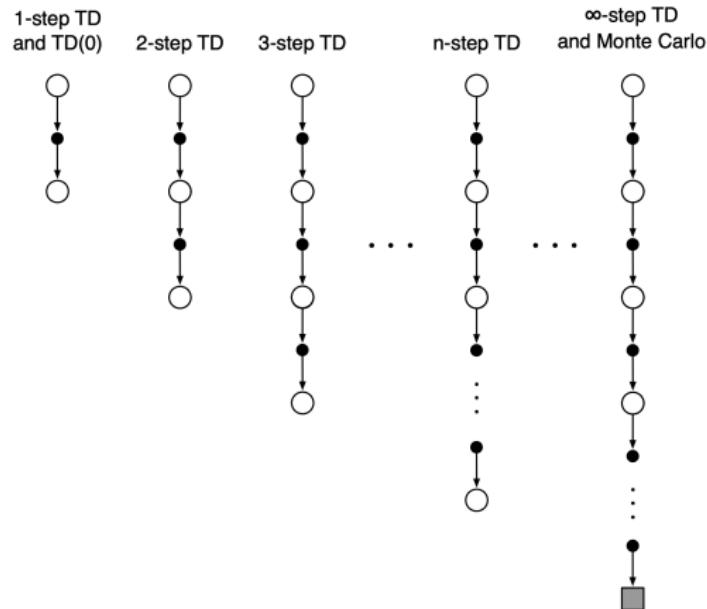
[T] = Terminal state

Comparison of TD and MC

- ① TD can learn online after every step
- ② MC must wait until end of episode before return is known
- ③ TD can learn from incomplete sequences
- ④ MC can only learn from complete sequences
- ⑤ TD works in continuing (non-terminating) environments
- ⑥ MC only works for episodic (terminating) environments
- ⑦ TD exploits Markov property, more efficient in Markov environments
- ⑧ MC does not exploit Markov property, more effective in non-Markov environments

n-step TD

- ➊ n-step TD methods that generalize both one-step TD and MC.
- ➋ We can shift from one to the other smoothly as needed to meet the demands of a particular task.



n-step TD prediction

- ① Consider the following n -step returns for $n = 1, 2, \infty$

$$n = 1(TD) \quad G_t^{(1)} = R_{t+1} + \gamma v(S_{t+1})$$

$$n = 2 \quad G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 v(S_{t+2})$$

⋮

$$n = \infty(MC) \quad G_t^{\infty} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-t-1} R_T$$

- ② Thus the n-step return is defined as

$$G_t^n = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{n-1} R_{t+n} + \gamma^n v(S_{t+n})$$

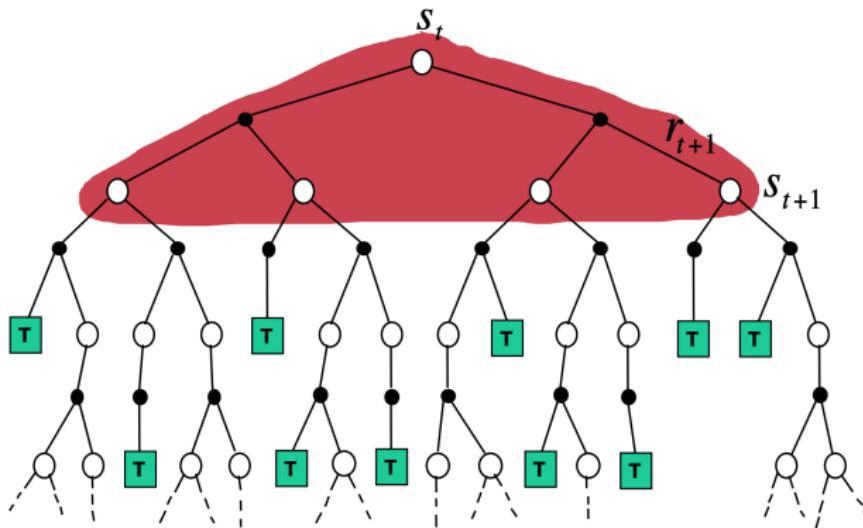
- ③ n -step TD: $v(S_t) \leftarrow v(S_t) + \alpha(G_t^n - v(S_t))$

Bootstrapping and Sampling for DP, MC, and TD

- ① Bootstrapping: update involves an estimate
 - ① MC does not bootstrap
 - ② DP bootstraps
 - ③ TD bootstraps
- ② Sampling: update samples an expectation
 - ① MC samples
 - ② DP does not sample
 - ③ TD samples

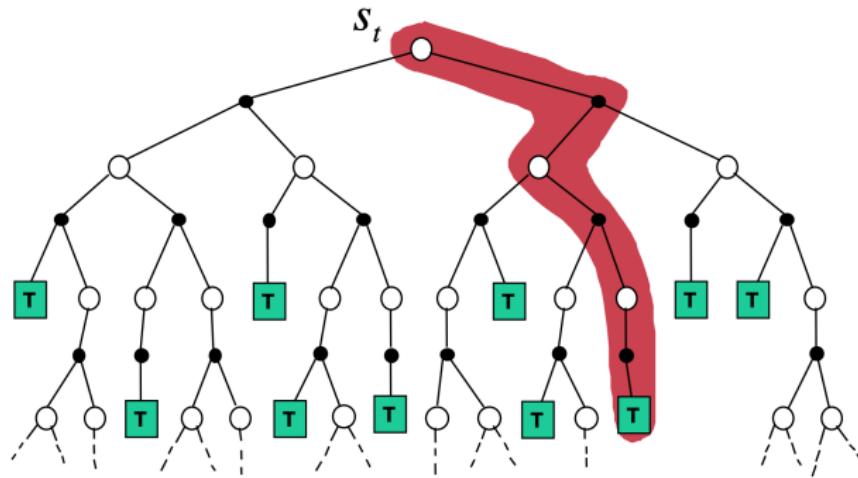
Unified View: Dynamic Programming Backup

$$v(S_t) \leftarrow \mathbb{E}_\pi[R_{t+1} + \gamma v(S_{t+1})]$$



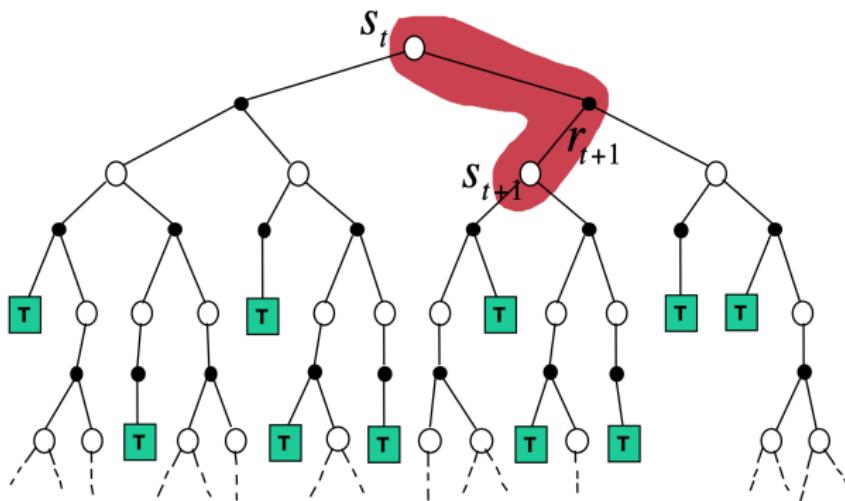
Unified View: Monte-Carlo Backup

$$v(S_t) \leftarrow v(S_t) + \alpha(G_t - v(S_t))$$

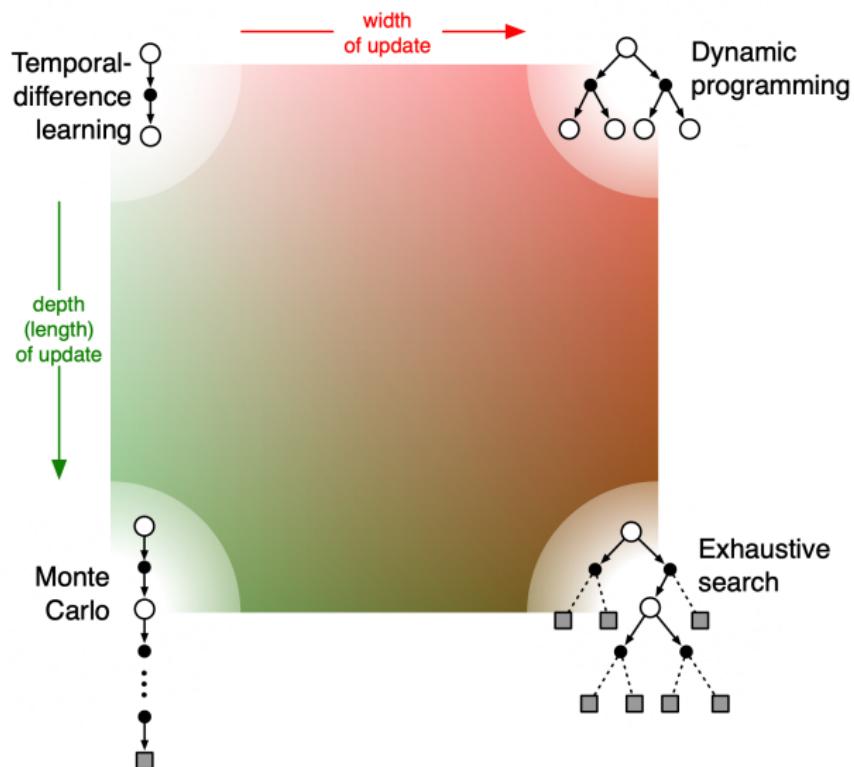


Unified View: Temporal-Difference Backup

$$TD(0) : v(S_t) \leftarrow v(S_t) + \alpha(R_{t+1} + \gamma v(s_{t+1}) - v(S_t))$$



Unified View of Reinforcement Learning



Summary

① Model-free prediction

- ① Evaluate the state value without knowing the MDP model, by only interacting with the environment

Model-free Control for MDP

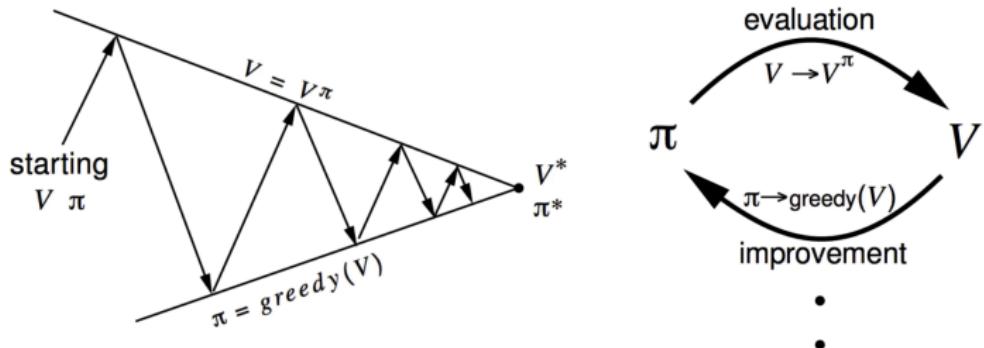
- ① Model-free control: Optimize the value function of an **unknown** MDP
- ② Generalized Policy Iteration (GPI) with MC and TD

Policy Iteration

① Iterate through the two steps:

- ① Evaluate the policy π (computing v given current π)
- ② Improve the policy by acting greedily with respect to v_π

$$\pi' = \text{greedy}(v_\pi) \quad (2)$$



Policy Iteration for a Known MDP

- ① compute the state-action value of a policy π :

$$q_{\pi_i}(s, a) = R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) v_{\pi_i}(s')$$

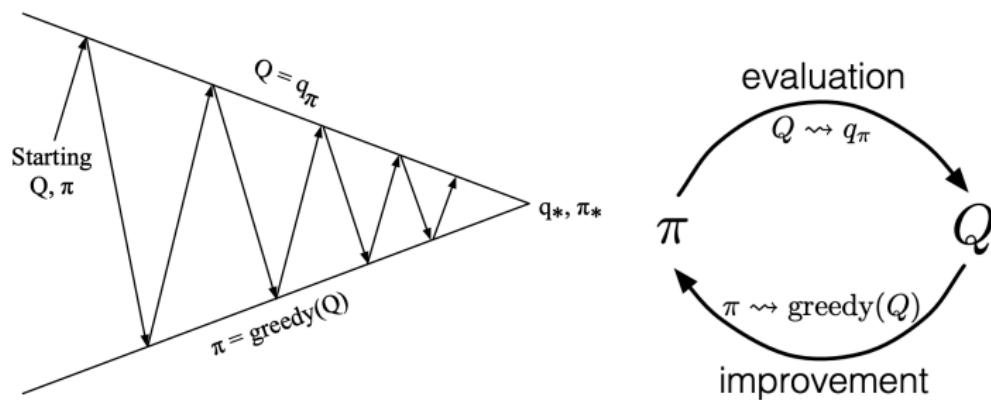
- ② Compute new policy π_{i+1} for all $s \in \mathcal{S}$ following

$$\pi_{i+1}(s) = \arg \max_a q_{\pi_i}(s, a) \quad (3)$$

- ③ Problem: What to do if there is neither $R(s, a)$ nor $P(s'|s, a)$ known/available?

Generalized Policy Iteration with Action-Value Function

Monte Carlo version of policy iteration



- ① Policy evaluation: Monte-Carlo policy evaluation $Q = q_\pi$
- ② Policy improvement: Greedy policy improvement?

$$\pi(s) = \arg \max_a q(s, a)$$

Monte Carlo with Exploring Starts

- ① One assumption to obtain the guarantee of convergence in PI:
Episode has exploring starts
- ② Exploring starts can ensure all actions are selected infinitely often

```
Monte Carlo ES (Exploring Starts), for estimating  $\pi \approx \pi_*$ 

Initialize:
 $\pi(s) \in \mathcal{A}(s)$  (arbitrarily), for all  $s \in \mathcal{S}$ 
 $Q(s, a) \in \mathbb{R}$  (arbitrarily), for all  $s \in \mathcal{S}, a \in \mathcal{A}(s)$ 
>Returns(s, a)  $\leftarrow$  empty list, for all  $s \in \mathcal{S}, a \in \mathcal{A}(s)$ 

Loop forever (for each episode):
    Choose  $S_0 \in \mathcal{S}, A_0 \in \mathcal{A}(S_0)$  randomly such that all pairs have probability  $> 0$ 
    Generate an episode from  $S_0, A_0$ , following  $\pi$ :  $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$ 
     $G \leftarrow 0$ 
    Loop for each step of episode,  $t = T-1, T-2, \dots, 0$ :
         $G \leftarrow \gamma G + R_{t+1}$ 
        Unless the pair  $S_t, A_t$  appears in  $S_0, A_0, S_1, A_1, \dots, S_{t-1}, A_{t-1}$ :
            Append  $G$  to  $Returns(S_t, A_t)$ 
             $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$ 
             $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$ 
```

Monte Carlo with ϵ -Greedy Exploration

- ① Trade-off between exploration and exploitation (we will talk about this in later lecture)
- ② ϵ -Greedy Exploration: Ensuring continual exploration
 - ① All actions are tried with non-zero probability
 - ② With probability $1 - \epsilon$ choose the greedy action
 - ③ With probability ϵ choose an action at random

$$\pi(a|s) = \begin{cases} \epsilon/|\mathcal{A}| + 1 - \epsilon & \text{if } a^* = \arg \max_{a \in \mathcal{A}} Q(s, a) \\ \epsilon/|\mathcal{A}| & \text{otherwise} \end{cases}$$

Monte Carlo with ϵ -Greedy Exploration

- ① Policy improvement theorem: For any ϵ -greedy policy π , the ϵ -greedy policy π' with respect to q_π is an improvement, $v_{\pi'}(s) \geq v_\pi(s)$

$$\begin{aligned} q_\pi(s, \pi'(s)) &= \sum_{a \in \mathcal{A}} \pi'(a|s) q_\pi(s, a) \\ &= \frac{\epsilon}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} q_\pi(s, a) + (1 - \epsilon) \max_a q_\pi(s, a) \\ &\geq \frac{\epsilon}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} q_\pi(s, a) + (1 - \epsilon) \sum_{a \in \mathcal{A}} \frac{\pi(a|s) - \frac{\epsilon}{|\mathcal{A}|}}{1 - \epsilon} q_\pi(s, a) \\ &= \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s, a) = v_\pi(s) \end{aligned}$$

Therefore, $v_{\pi'}(s) \geq v_\pi(s)$ from the policy improvement theorem

Monte Carlo with ϵ -Greedy Exploration

Algorithm 1

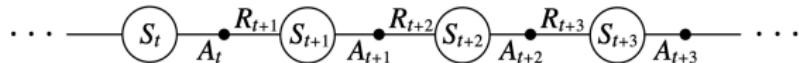
- 1: Initialize $Q(S, A) = 0, N(S, A) = 0, \epsilon = 1, k = 1$
- 2: $\pi_k = \epsilon\text{-greedy}(Q)$
- 3: **loop**
- 4: Sample k -th episode $(S_1, A_1, R_2, \dots, S_T) \sim \pi_k$
- 5: **for** each state S_t and action A_t in the episode **do**
- 6: $N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$
- 7: $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)}(G_t - Q(S_t, A_t))$
- 8: **end for**
- 9: $k \leftarrow k + 1, \epsilon \leftarrow 1/k$
- 10: $\pi_k = \epsilon\text{-greedy}(Q)$
- 11: **end loop**

MC vs. TD for Prediction and Control

- ① Temporal-difference (TD) learning has several advantages over Monte-Carlo (MC)
 - ① Lower variance
 - ② Online
 - ③ Incomplete sequences
- ② So we can use TD instead of MC in our control loop
 - ① Apply TD to $Q(S, A)$
 - ② Use ϵ -greedy policy improvement
 - ③ Update every time-step rather than at the end of one episode

Recall: TD Prediction

- ① An episode consists of an alternating sequence of states and state-action pairs:



- ② TD(0) method for estimating the value function $V(S)$

$A_t \leftarrow$ action given by π for S

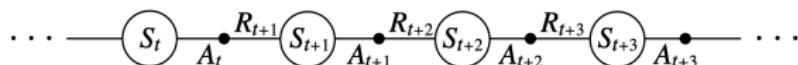
Take action A_t , observe R_{t+1} and S_{t+1}

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

- ③ How about estimating action value function $Q(S, A)$?

Sarsa: On-Policy TD Control

- ① An episode consists of an alternating sequence of states and state-action pairs:



- ② ϵ -greedy policy for one step, then bootstrap the action value function:

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

- ③ The update is done after every transition from a nonterminal state S_t
- ④ TD target $\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$

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 $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

 Initialize S

 Choose A from S using policy derived from Q (e.g., ϵ -greedy)

 Repeat (for each step of episode):

 Take action A , observe R, S'

 Choose A' from S' using policy derived from Q (e.g., ϵ -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

n-step Sarsa

- ① Consider the following *n*-step Q-returns for $n = 1, 2, \infty$

$$n=1(\text{Sarsa}) q_t^{(1)} = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$$

$$n=2 \quad q_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 Q(S_{t+2}, A_{t+2})$$

$$\vdots$$

$$n=\infty(MC) \quad q_t^{\infty} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-t-1} R_T$$

- ② Thus the n-step Q-return is defined as

$$q_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{n-1} R_{t+n} + \gamma^n Q(S_{t+n}, A_{t+n})$$

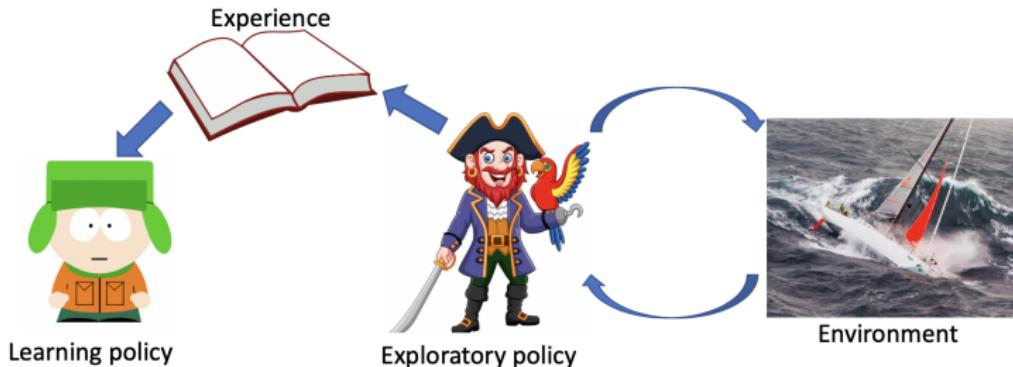
- ③ *n*-step Sarsa updates $Q(s,a)$ towards the *n*-step Q-return:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(q_t^{(n)} - Q(S_t, A_t) \right)$$

On-policy Learning vs. Off-policy Learning

- ① On-policy learning: Learn about policy π from the experience collected from π
 - ① Behave non-optimally in order to explore all actions, then reduce the exploration. e.g., ϵ -greedy
- ② Another important approach is **off-policy learning** which essentially uses **two different policies**:
 - ① the one which is being learned about and becomes the optimal policy
 - ② the other one which is more exploratory and is used to generate trajectories
- ③ Off-policy learning: Learn about policy π from the experience sampled from another policy μ
 - ① π : target policy
 - ② μ : behavior policy

Off-policy Learning



- ① Following behaviour policy $\mu(a|s)$ to collect data

$$S_1, A_1, R_2, \dots, S_T \sim \mu$$

Update π using $S_1, A_1, R_2, \dots, S_T$

- ② It leads to many benefits:

- ① Learn about optimal policy while following exploratory policy
- ② Learn from observing humans or other agents
- ③ Re-use experience generated from old policies $\pi_1, \pi_2, \dots, \pi_{t-1}$

Off-Policy Control with Q Learning

- ① Off-policy learning of action values $Q(s, a)$
- ② No importance sampling is needed
- ③ Next action in TD target is selected from the alternative action $A' \sim \pi(\cdot | S_t)$
- ④ update $Q(S_t, A_t)$ towards value of alternative action

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A') - Q(S_t, A_t))$$

Off-Policy Control with Q-Learning

- ① We allow both behavior and target policies to improve
- ② The target policy π is **greedy** on $Q(s, a)$

$$\pi(S_{t+1}) = \arg \max_{a'} Q(S_{t+1}, a')$$

- ③ The behavior policy μ could be totally random, but we let it improve following **ϵ -greedy** on $Q(s, a)$
- ④ Thus Q-learning target:

$$\begin{aligned} R_{t+1} + \gamma Q(S_{t+1}, A') &= R_{t+1} + \gamma Q(S_{t+1}, \arg \max Q(S_{t+1}, a')) \\ &= R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') \end{aligned}$$

- ⑤ Thus the Q-Learning update

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

Q-learning algorithm

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 $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

 Initialize S

 Repeat (for each step of episode):

 Choose A from S using policy derived from Q (e.g., ϵ -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

Comparison of Sarsa and Q-Learning

① Sarsa: On-Policy TD control

Choose action A_t from S_t using policy derived from Q with ϵ -greedy

Take action A_t , observe R_{t+1} and S_{t+1}

Choose action A_{t+1} from S_{t+1} using policy derived from Q with ϵ -greedy

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

② Q-Learning: Off-Policy TD control

Choose action A_t from S_t using policy derived from Q with ϵ -greedy

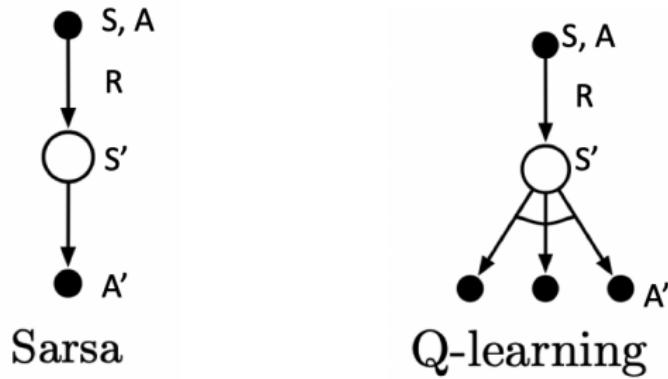
Take action A_t , observe R_{t+1} and S_{t+1}

Then ‘imagine’ A_{t+1} as $\arg \max Q(S_{t+1}, a')$ in the update target

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

Comparison of Sarsa and Q-Learning

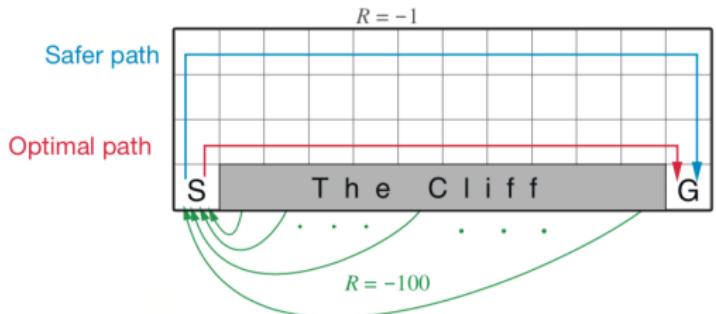
- ① Backup diagram for Sarsa and Q-learning



- ② In Sarsa, A and A' are sampled from the same policy so it is on-policy
- ③ In Q Learning, A and A' are from different policies, with A being more exploratory and A' determined directly by the max operator

Example on Cliff Walk (Example 6.6 from Textbook)

<https://github.com/cuhkrlcourse/RLexample/blob/master/modelfree/cliffwalk.R>

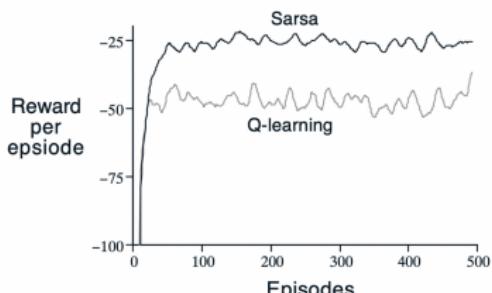


0 0 0 0 R R R R R R R R R R
R R R R R 0 0 0 0 0 0 0 0 0 R
R 0 0 0 0 0 0 0 0 0 0 0 0 0 R
R * * * * * * * * * * * * * G

Result of Sarsa

0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
R R R R R R R R R R R R R R R
R * * * * * * * * * * * * * G

Result of Q-Learning

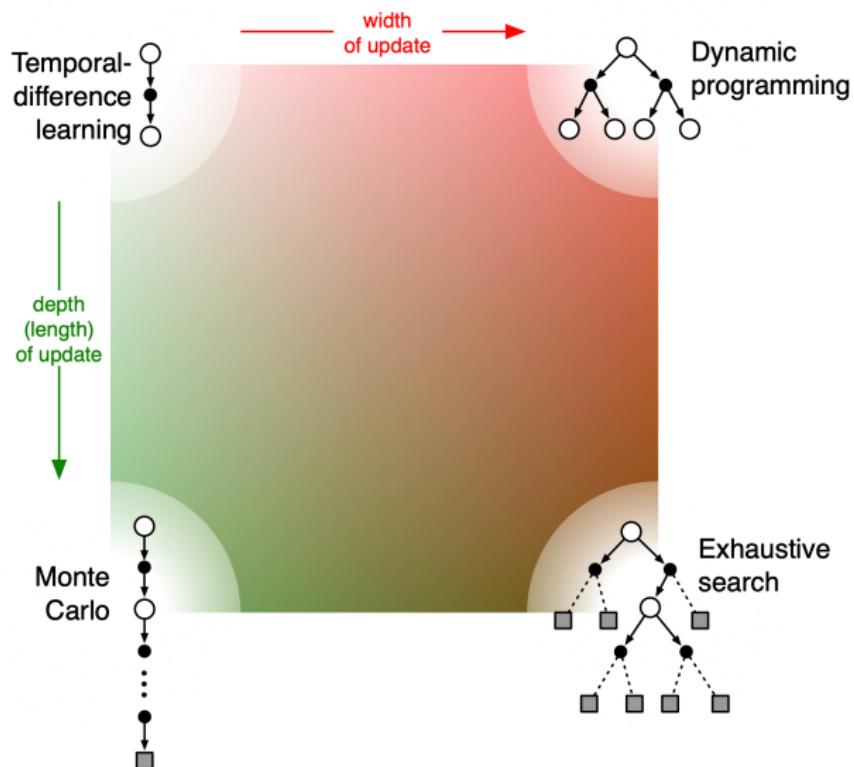


On-line performance of Q-learning is worse than that of Sarsa

Summary of DP and TD

Expected Update (DP)	Sample Update (TD)
Iterative Policy Evaluation $V(s) \leftarrow \mathbb{E}[R + \gamma V(S') s]$	TD Learning $V(S) \leftarrow^{\alpha} R + \gamma V(S')$
Q-Policy Iteration $Q(S, A) \leftarrow \mathbb{E}[R + \gamma Q(S', A') s, a]$	Sarsa $Q(S, A) \leftarrow^{\alpha} R + \gamma Q(S', A')$
Q-Value Iteration $Q(S, A) \leftarrow \mathbb{E}[R + \gamma \max_{a' \in \mathcal{A}} Q(S', A') s, a]$	Q-Learning $Q(S, A) \leftarrow^{\alpha} R + \gamma \max_{a' \in \mathcal{A}} Q(S', a')$
where $x \leftarrow^{\alpha} y$ is defined as $x \leftarrow x + \alpha(y - x)$	

Unified View of Reinforcement Learning



Sarsa and Q-Learning Example

<https://github.com/cuhkrlcourse/RExample/tree/master/modelfree>

Summary

- ① Model-free prediction in MDP
- ② Model-free control in MDP

End

- ① The following backup slides are on importance sampling

Importance Sampling

- ① Estimate the expectation of a function

$$E_{x \sim P}[f(x)] = \int f(x)P(x)dx \approx \frac{1}{n} \sum_i f(x_i)$$

- ② But sometimes it is difficult to sample x from $P(x)$, then we can sample x from another distribution $Q(x)$, then correct the weight

$$\begin{aligned}\mathbb{E}_{x \sim P}[f(x)] &= \int P(x)f(x)dx \\ &= \int Q(x)\frac{P(x)}{Q(x)}f(x)dx \\ &= \mathbb{E}_{x \sim Q}\left[\frac{P(x)}{Q(x)}f(x)\right] \approx \frac{1}{n} \sum_i \frac{P(x_i)}{Q(x_i)}f(x_i)\end{aligned}$$

Importance Sampling for Off-Policy RL

- ① Estimate the expectation of return using trajectories sampled from another policy (behavior policy)

$$\begin{aligned}\mathbb{E}_{T \sim \pi}[g(T)] &= \int P(T)g(T)dT \\ &= \int Q(T)\frac{P(T)}{Q(T)}g(T)dT \\ &= \mathbb{E}_{T \sim \mu}\left[\frac{P(T)}{Q(T)}g(T)\right] \\ &\approx \frac{1}{n} \sum_i \frac{P(T_i)}{Q(T_i)}g(T_i)\end{aligned}$$

Importance Sampling for Off-Policy Monte Carlo

- ① Generate episode from behavior policy μ and compute the generated return G_t

$$S_1, A_1, R_2, \dots, S_T \sim \mu$$

- ② Weight return G_t according to similarity between policies
 - ① Multiply importance sampling corrections along whole episode

$$G_t^{\pi/\mu} = \frac{\pi(A_t|S_t)}{\mu(A_t|S_t)} \frac{\pi(A_{t+1}|S_{t+1})}{\mu(A_{t+1}|S_{t+1})} \cdots \frac{\pi(A_T|S_T)}{\mu(A_T|S_T)} G_t$$

- ③ Update value towards correct return

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t^{\pi/\mu} - V(S_t))$$

Importance Sampling for Off-Policy TD

- ➊ Use TD targets generated from μ to evaluate π
- ➋ Weight TD target $R + \lambda V(S')$ by importance sampling
- ➌ Only need a single importance sampling correction

$$V(S_t) \leftarrow V(S_t) + \alpha \left(\frac{\pi(A_t|S_t)}{\mu(A_t|S_t)} (R_{t+1} + \lambda V(S_{t+1})) - V(S_t) \right)$$

- ➍ Policies only need to be similar over a single step

Why don't use importance sampling on Q-Learning?

- ① Off-policy TD

$$V(S_t) \leftarrow V(S_t) + \alpha \left(\frac{\pi(A_t|S_t)}{\mu(A_t|S_t)} (R_{t+1} + \lambda V(S_{t+1})) - V(S_t) \right) \quad (4)$$

- ② Short answer: Because Q-learning does not make expected value estimates over the policy distribution. For the full answer click [here](#)
- ③ Remember bellman optimality backup from value iteration

$$Q(s, a) = R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) \max_{a'} Q(s', a') \quad (5)$$

- ④ Q-learning can be considered as sample-based version of value iteration, except instead of using the expected value over the transition dynamics, we use the sample collected from the environment

$$Q(s, a) = r + \gamma \max_{a'} Q(s', a') \quad (6)$$

Q-learning is over the transition distribution, not over policy distribution thus no need to correct different policy distributions