Lecture 3： Model-free Prediction & Control

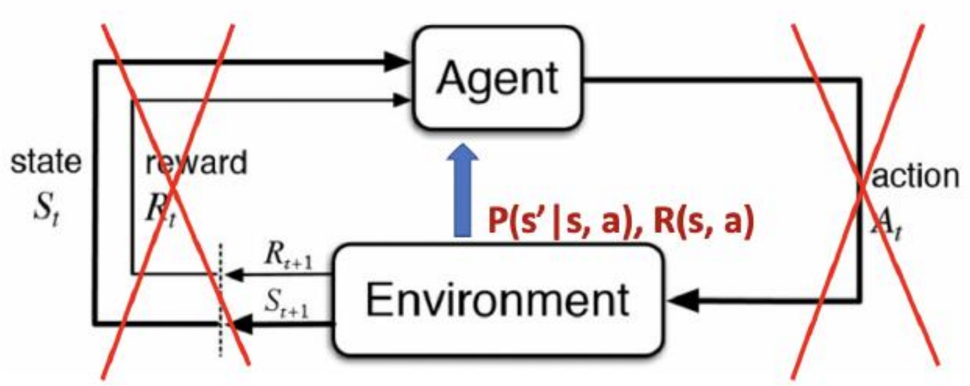
Recap

- Last lecture:
  - MDP
  - policy evaluation
  - policy iteration and value iteration for solving a known MDP

- This lecture:
  - Model-free prediction: Estimate value function of an unknown MDP
  - Model-free control: Optimize value function of an unknown MDP

RL with knowing how the world works

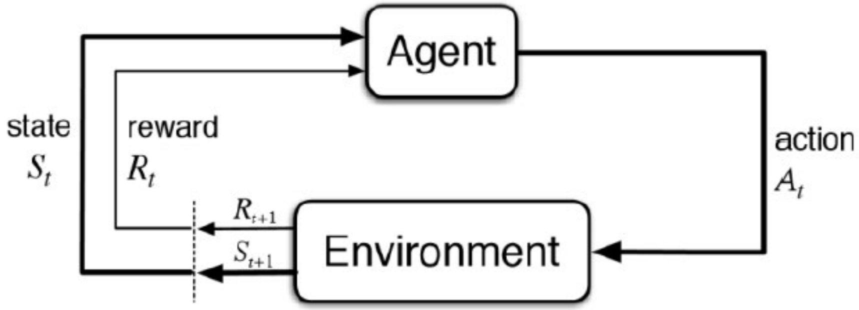
- Both of the policy iteration and value iteration assume the direct access to the dynamics and rewards of the environment



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

Model-free RL: Learning by interaction

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



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

Model-free prediction

- Model-free prediction: policy evaluation without the access to the model
- Estimating the expected return of a particular policy if we don't have access to the MDP models
  - Monte Carlo policy evaluation
  - Temporal Difference (TD) learning

Monte-Carlo Policy Evaluation

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

Monte-Carlo Policy Evaluation

- 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 $v(s) = S(s)/N(s)$

- By law of large numbers, $v(s) \rightarrow v^{\pi}(s)$ as $N(s) \rightarrow \infty$

Incremental MC Updates

- Mean from the average of samples $x_1, x_2, \ldots$

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

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

$$
\begin{aligned}
N(S_t) &\leftarrow N(S_t) + 1 \\
v(S_t) &\leftarrow v(S_t) + \frac{1}{N(S_t)}\left(G_t - v(S_t)\right)
\end{aligned}
$$

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

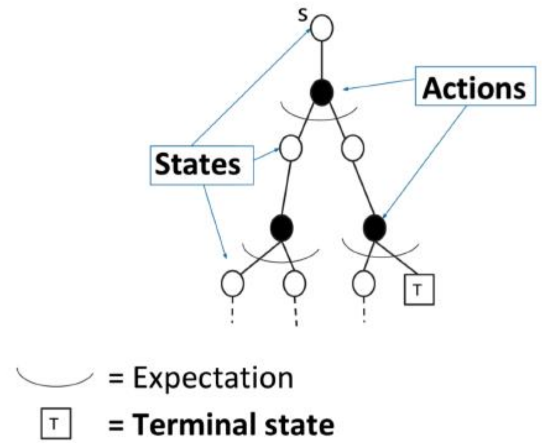
$$
v(S_t) \leftarrow v(S_t) + \alpha\left(G_t - v(S_t)\right)
$$

Difference between DP and MC

- Dynamic Programming (DP) computes v_i by bootstrapping the rest of the expected return by the value estimate v_{i-1}

- Iteration on Bellman expectation backup:

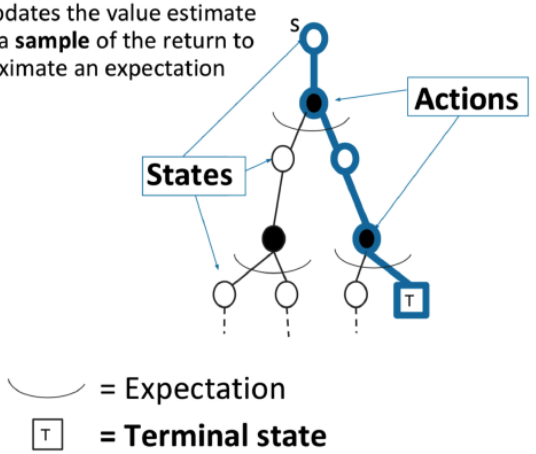
$$v_i(s) \leftarrow \sum_{a \in \mathcal{A}} \pi(a|s)\Big(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) v_{i-1}(s')\Big)$$



- MC updates the empirical mean return with one sampled episode

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

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



Advantages of MC over DP

- MC works when the environment is unknown
- 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
- 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

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

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

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

$$v(S_t) \leftarrow v(S_t) + \alpha(\underbrace{\boxed{R_{t+1} + \gamma v(S_{t+1})}}_{\text{TD target}} - v(S_t))$$

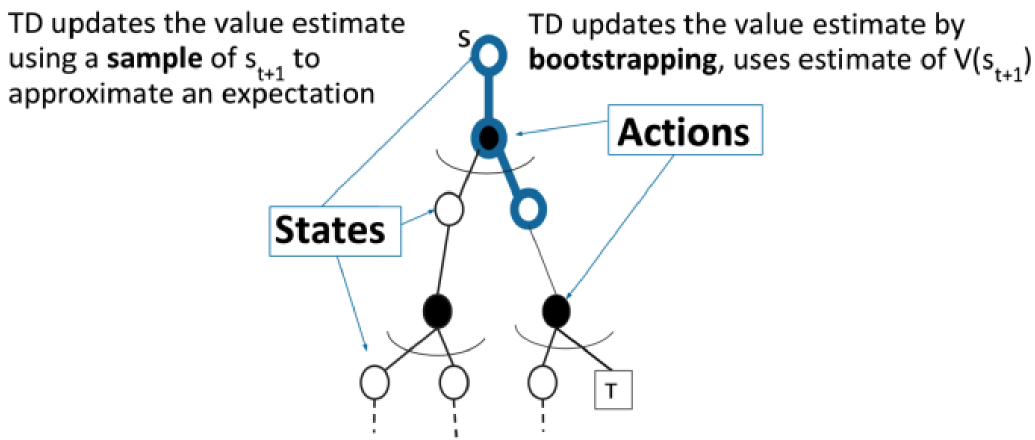
$\delta_t = R_{t+1} + \gamma v(S_{t+1}) - v(S_t)$ ⇨ TD error

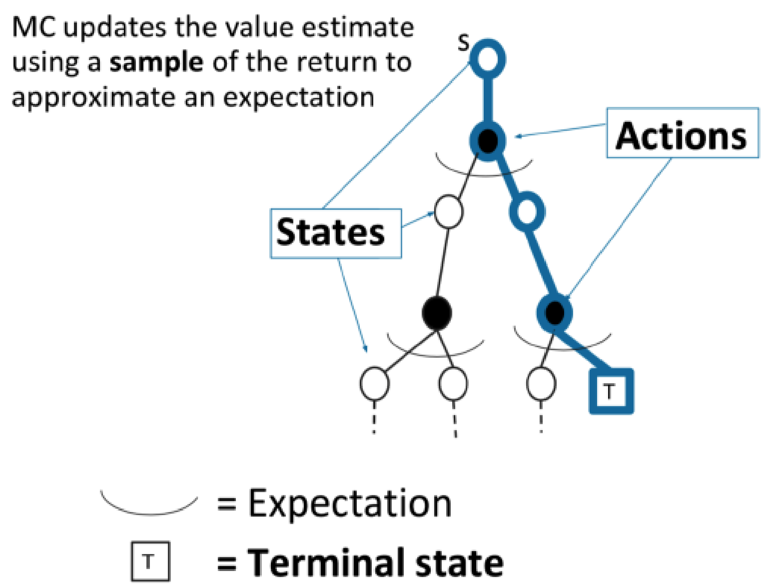
- Comparison: Incremental Monte-Carlo

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





Comparison of TD and MC

- TD can learn online after every step
- MC must wait until end of episode before return is known

- TD can learn from incomplete sequences
- MC can only learn from complete sequences

- TD works in continuing (non-terminating) environments
- MC only works for episodic (terminating) environments

- TD exploits Markov property, more efficient in Markov environments
- MC does not exploit Markov property, more effective in non-Markov environments

Bias/Variance Trade-Off

- Return $G_t = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-1} R_T$ is *unbiased* estimate of $v_\pi(S_t)$
- True TD target $R_{t+1} + \gamma v_\pi(S_{t+1})$ is *unbiased* estimate of $v_\pi(S_t)$
- TD target $R_{t+1} + \gamma V(S_{t+1})$ is *biased* estimate of $v_\pi(S_t)$
- TD target is much lower variance than the return:
  - Return depends on *many* random actions, transitions, rewards
  - TD target depends on *one* random action, transition, reward

Comparison of TD and MC

- MC has high variance, zero bias
  - Good convergence properties
  - (even with function approximation)
  - Not very sensitive to initial value
  - Very simple to understand and use

- TD has low variance, some bias
  - Usually more efficient than MC
  - TD(0) converges to $V_\pi(s)$
  - (but not always with function approximation)
  - More sensitive to initial value

Batch MC and TD

- MC and TD converge: $V(s) \to v_\pi(s)$ as experience $\to \infty$
- But what about batch solution for finite experience?

$$
\begin{gathered}
s_1^1, a_1^1, r_2^1, \ldots, s_{T_1}^1 \\
\vdots \\
s_1^K, a_1^K, r_2^K, \ldots, s_{T_K}^K
\end{gathered}
$$

  - e.g. Repeatedly sample episode $k \in [1, K]$
  - Apply MC or TD(0) to episode k

Certainty Equivalence

- MC converges to solution with minimum mean-squared error
  - Best fit to the observed returns

$$\sum_{k=1}^{K} \sum_{t=1}^{T_k} \left(G_t^k - V(s_t^k)\right)^2$$

- TD(0) converges to solution of max likelihood Markov model
  - Solution to the MDP $\langle \mathcal{S}, \mathcal{A}, \hat{\mathcal{P}}, \hat{\mathcal{R}}, \gamma \rangle$ that best fits the data

$$\hat{\mathcal{P}}_{s,s'}^a = \frac{1}{N(s,a)} \sum_{k=1}^{K} \sum_{t=1}^{T_k} \mathbf{1}(s_t^k, a_t^k, s_{t+1}^k = s, a, s')$$

$$\hat{\mathcal{R}}_s^a = \frac{1}{N(s,a)} \sum_{k=1}^{K} \sum_{t=1}^{T_k} \mathbf{1}(s_t^k, a_t^k = s, a) r_t^k$$

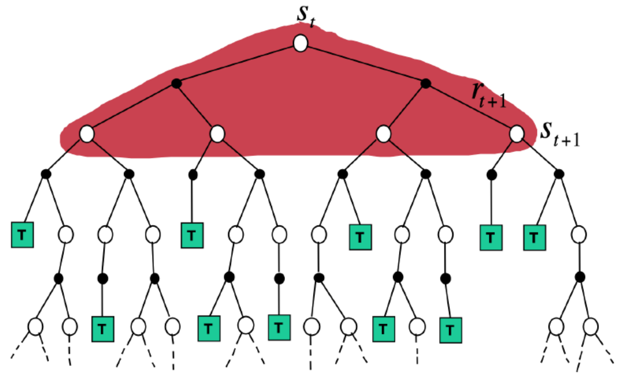
Bootstrapping and Sampling for DP, MC, and TD

- Bootstrapping: update involves an estimate
  - MC does not bootstrap
  - DP bootstraps
  - TD bootstraps

- Sampling: update samples an expectation
  - MC samples
  - DP does not sample
  - TD samples

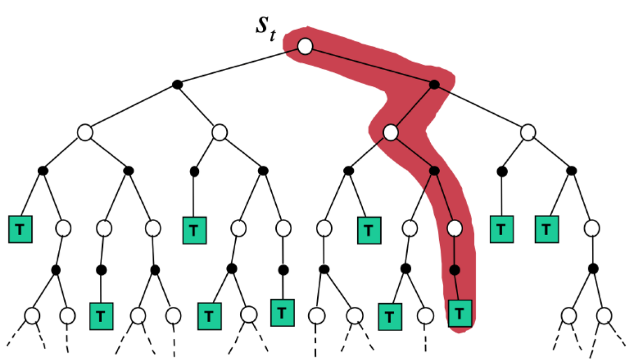
Unified View

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



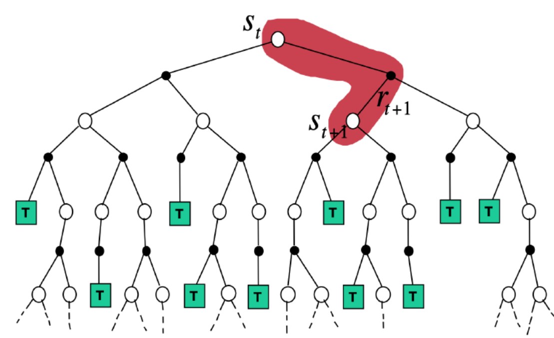
Dynamic Programming Backup

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



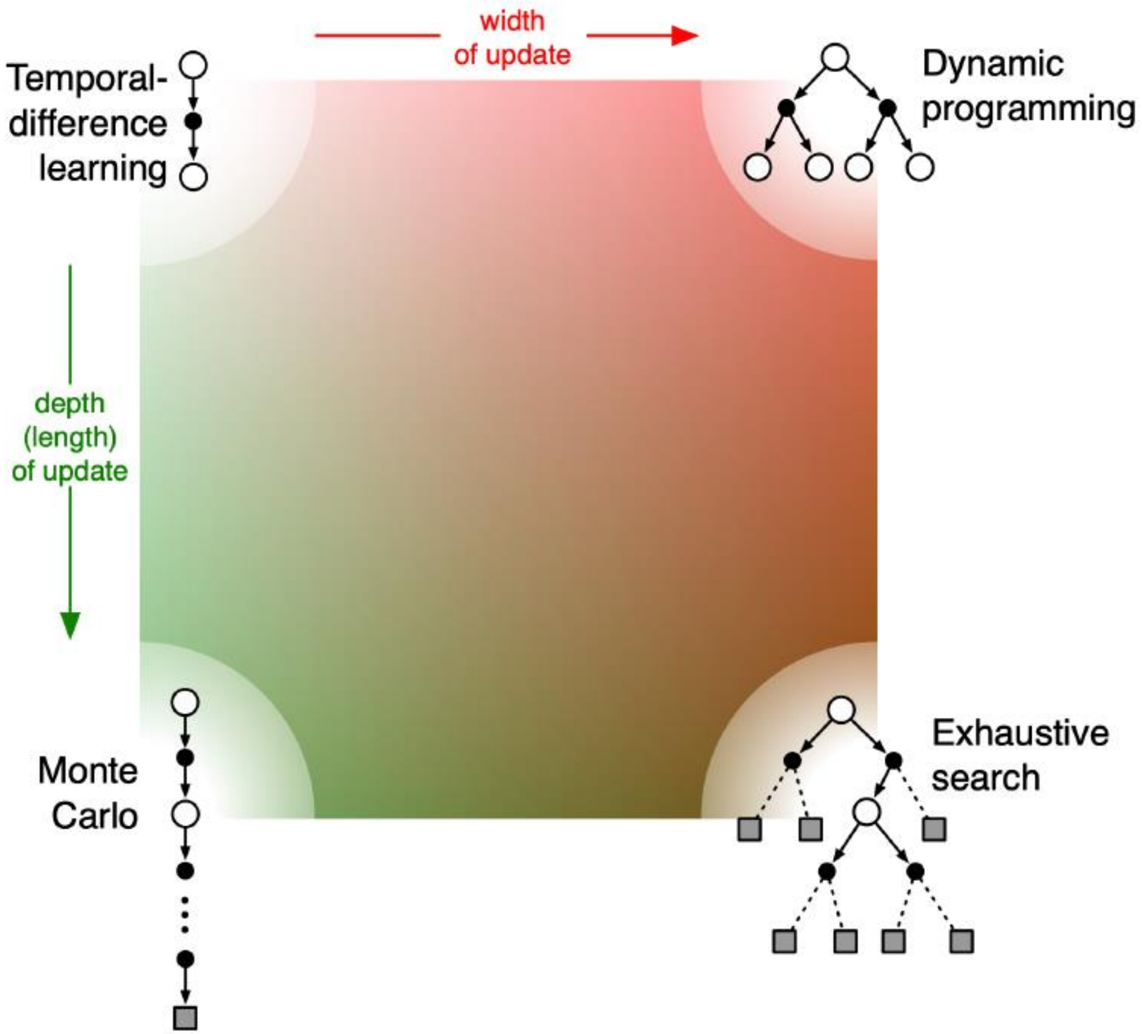
Monte-Carlo Backup

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



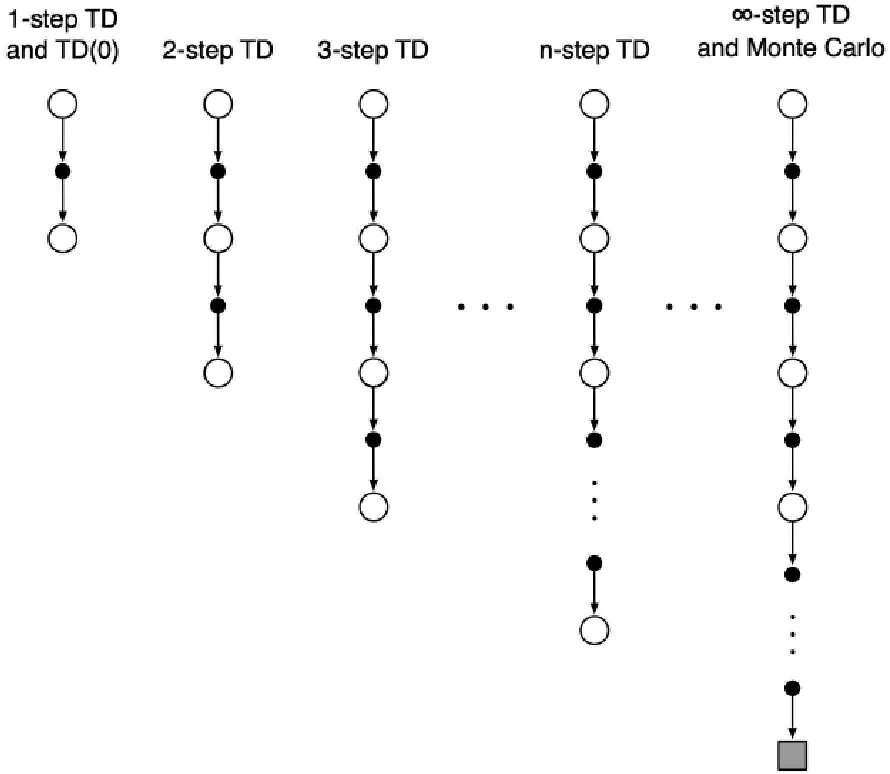
Temporal-Difference Backup

Unified View of RL



n-step TD

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



n-step TD prediction

- Consider the following n-step returns for n = 1,2,∞

$$
\begin{aligned}
n = 1(TD) \quad & G_t^{(1)} = R_{t+1} + \gamma v(S_{t+1}) \\
n = 2 \quad & G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 v(S_{t+2}) \\
& \vdots \\
n = \infty(MC) \quad & G_t^{\infty} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-t-1} R_T
\end{aligned}
$$

- Thus the n-step return is defined as

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

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

$\lambda - reutrn$

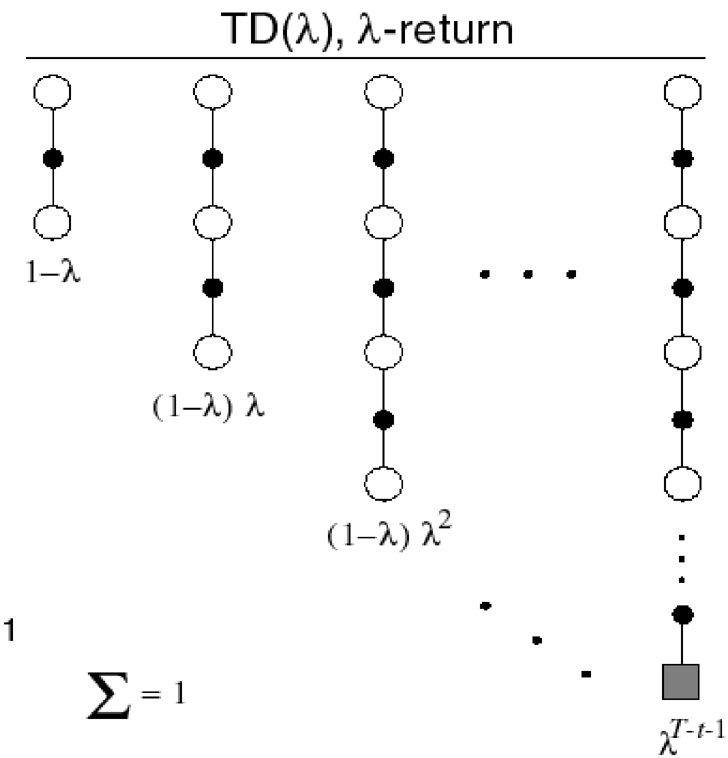
- The $\lambda - return$ G_t^λ combines all n-step returns G_t^n

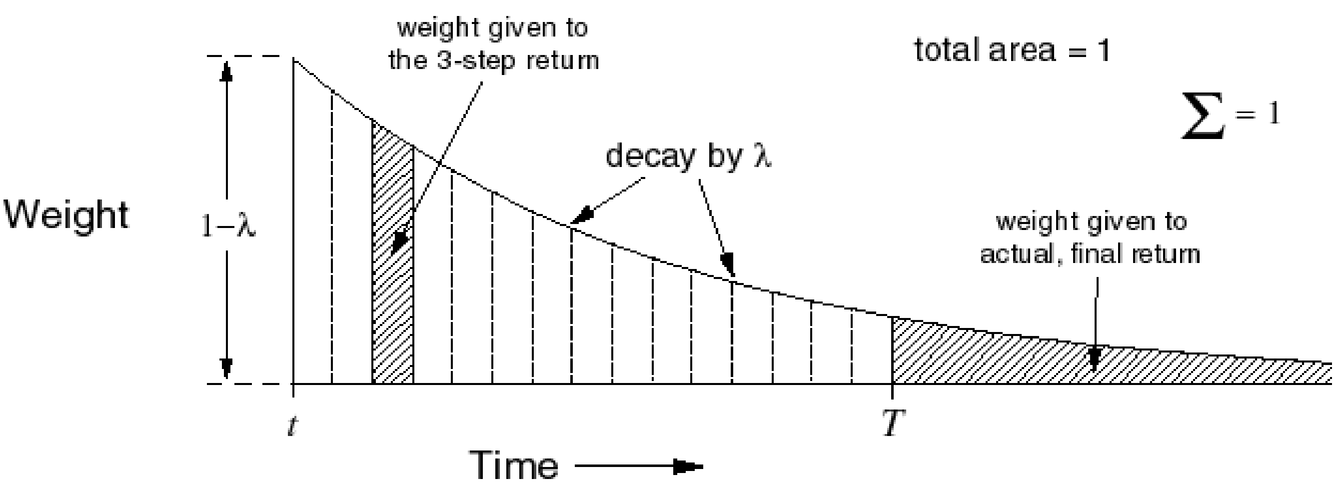
- Using weight $(1-\lambda)\lambda^{n-1}$

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$$

- Forward-view TD(λ)

$$V(S_t) \leftarrow V(S_t) + \alpha \left(G_t^\lambda - V(S_t) \right)$$





Model-free Control

- Model-Free Reinforcement Learning
  - Model-free prediction
    - Estimate the value function of an unknown MDP
  - Model-free control
    - Monte-Carlo control
    - Temporal Difference (TD) control
    - Off-Policy Learning
  - Optimise the value function of an unknown MDP

Uses of Model-Free Control

Some example problems that can be modelled as MDPs

- Elevator
- Parallel Parking
- Ship Steering
- Bioreactor
- Helicopter
- Aeroplane Logistics
- Robocup Soccer
- Quake
- Portfolio management
- Protein Folding
- Robot walking
- Game of Go

For most of these problems, either:

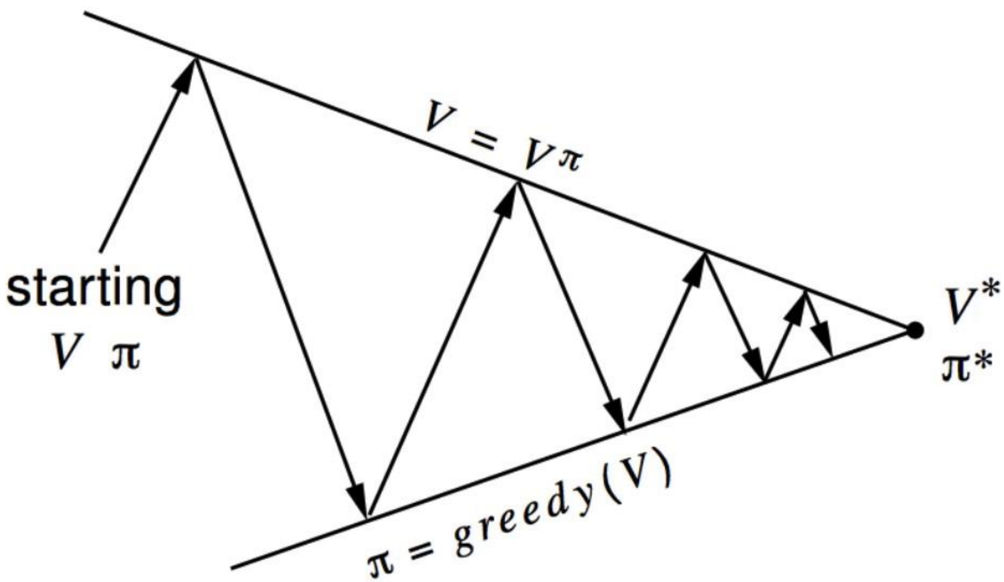
- MDP model is unknown, but experience can be sampled
- MDP model is known, but is too big to use, except by samples

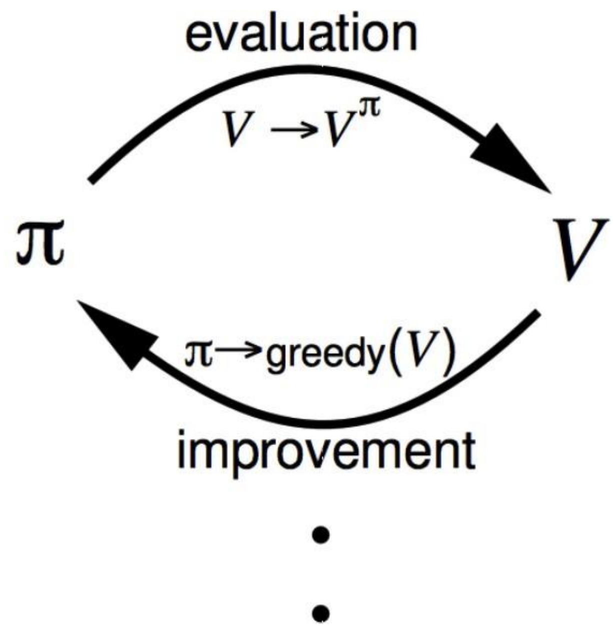
Model-free control can solve these problems

Policy Iteration

- Iteration through the two steps
  - Evaluate the policy π (computing v given current π)
  - Improve the policy by acting greedily with respect to v_π

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





Policy Iteration for a Known MDP

- Compute the state-action value of a policy π:

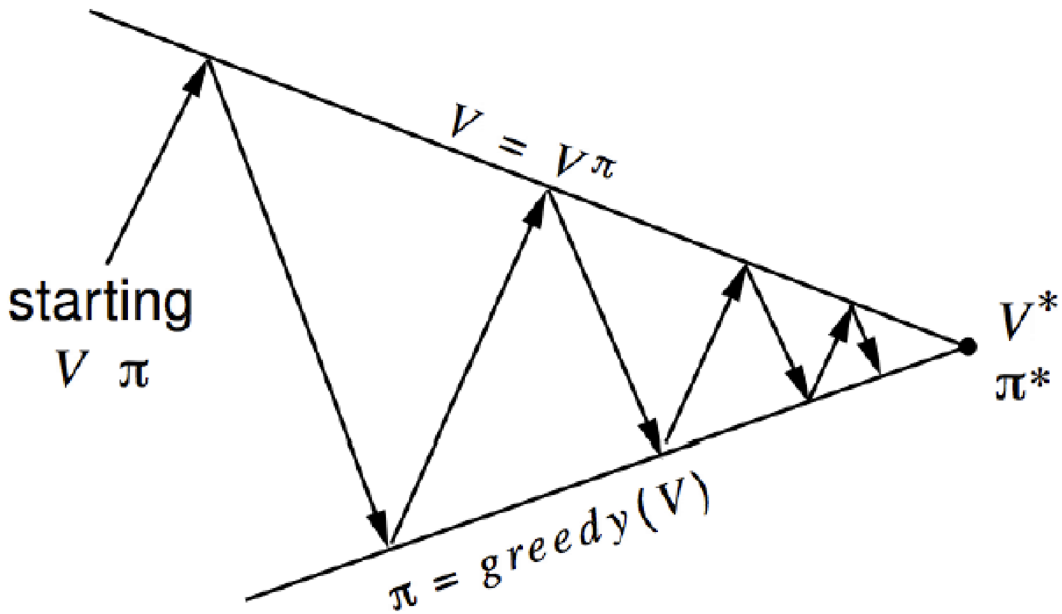
$$q_{\pi_i}(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) v_{\pi_i}(s')$$

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

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

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

General PI With Monte-Carlo Evaluation



Policy evaluation Monte-Carlo policy evaluation, $V = v_\pi$?

Policy improvement Greedy policy improvement?

Monte Carlo with $\epsilon - Greedy$ Exploration

- $\epsilon - greedy$ Exploration: Ensuring continual exploration
  - All actions are tried with non-zero probability
  - With probability $1 - \epsilon$ choose the greedy action
  - With probability ϵ choose an action at random

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

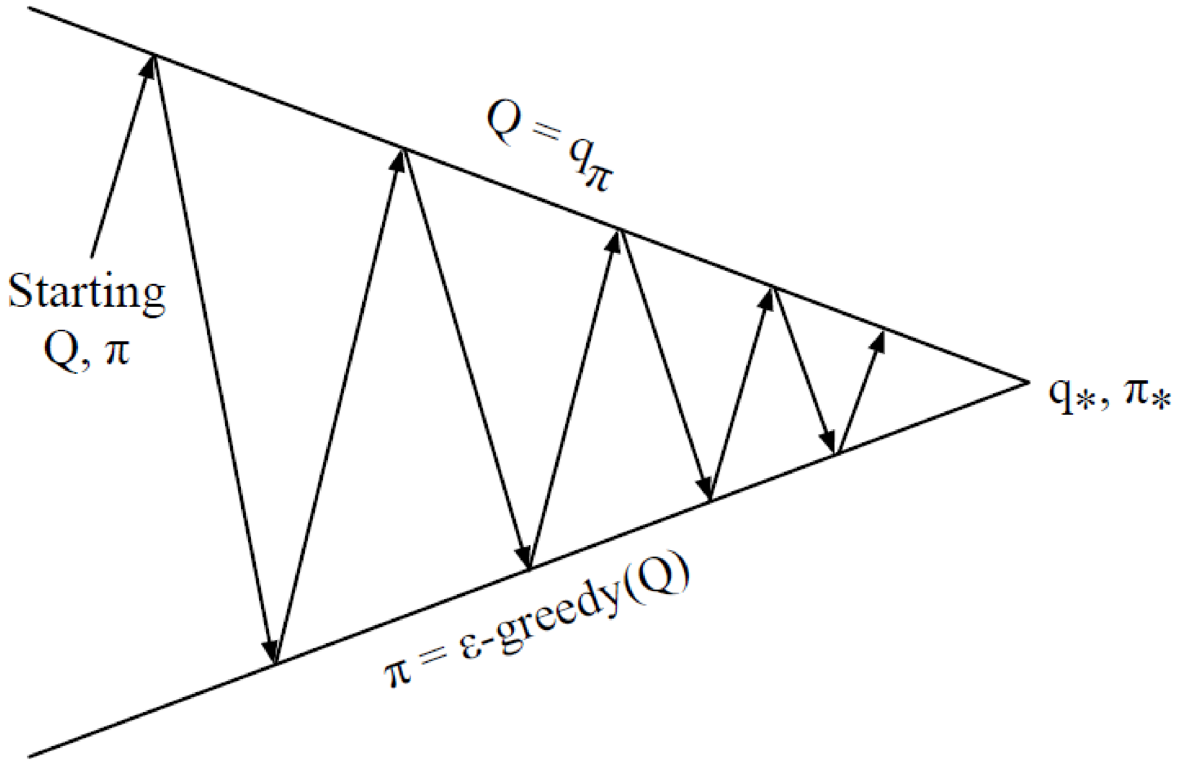
Monte Carlo with $\epsilon - Greedy$ Exploration

- **Policy improvement theorem**: For any policy π, the $\epsilon - greedy$ 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 Policy Iteration



Policy evaluation Monte-Carlo policy evaluation, $Q = q_\pi$

Policy improvement ϵ-greedy policy improvement

GLIE

Definition

Greedy in the Limit with Infinite Exploration (GLIE)

- All state-action pairs are explored infinitely many times,

$$\lim_{k \to \infty} N_k(s, a) = \infty$$

- The policy converges on a greedy policy,

$$\lim_{k \to \infty} \pi_k(a|s) = \mathbf{1}(a = \underset{a' \in \mathcal{A}}{\operatorname{argmax}}\ Q_k(s, a'))$$

- For example, ϵ-greedy is GLIE if ϵ reduces to zero at $\epsilon_k = \frac{1}{k}$

GLIE Monte-Carlo Control

- Sample kth episode using π: $\{S_1, A_1, R_2, ..., S_T\} \sim \pi$
- For each state S_t and action A_t in the episode,

$$N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$$
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)} (G_t - Q(S_t, A_t))$$

- Improve policy based on new action-value function

$$\epsilon \leftarrow 1/k$$
$$\pi \leftarrow \epsilon\text{-greedy}(Q)$$

Theorem

GLIE Monte-Carlo control converges to the optimal action-value function, $Q(s, a) \rightarrow q_*(s, a)$

MC VS. TD for Prediction and Control

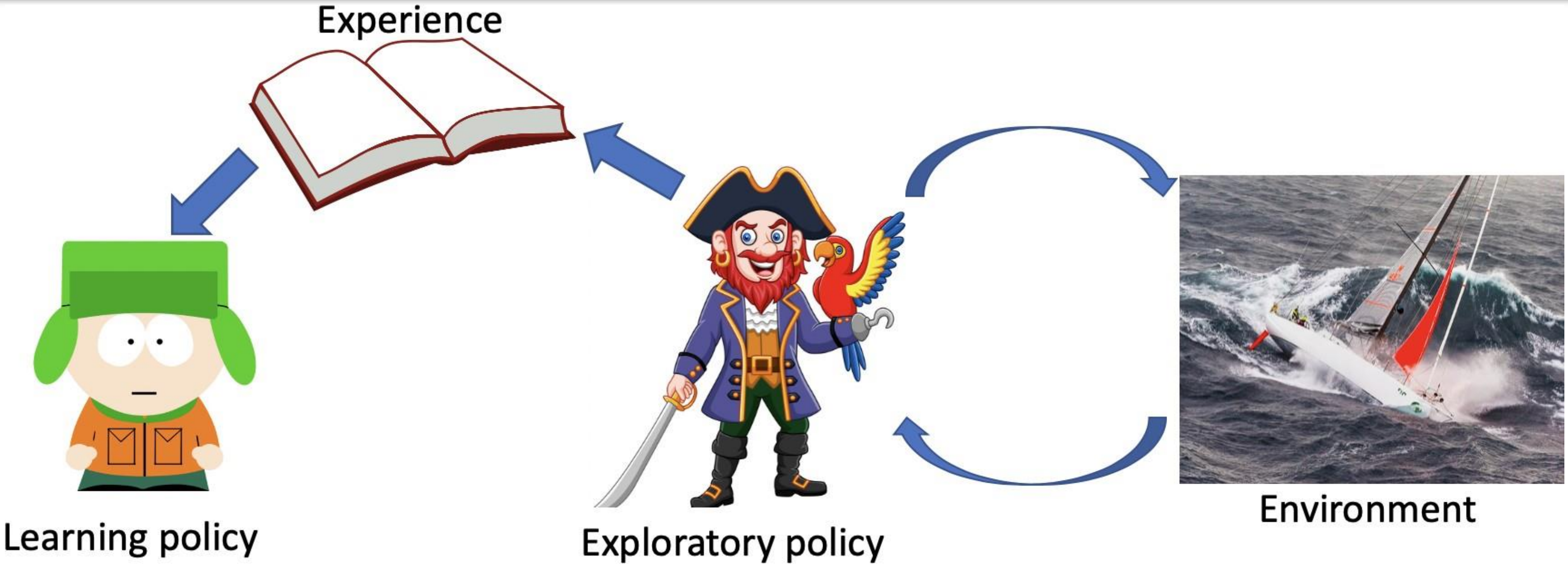
- Temporal-difference(TD) learning has several advantages over Monte-Carlo(MC)
  - Lower variance
  - Online
  - Incomplete sequences

- So we can use TD instead of MC in our control loop
  - Apply TD to Q(S, A)
  - Use $\epsilon - greedy$ policy improvement
  - Update every time-step rather than at the end of one episode

On-policy learning and Off-policy learning

- **On-policy learning**
  - Learn on the job
  - Learn about policy π from experience sampled from π
- **Off-policy learning**
  - Look over someone's shoulder
  - Learn about policy π from experience sampled from μ

Off-policy learning



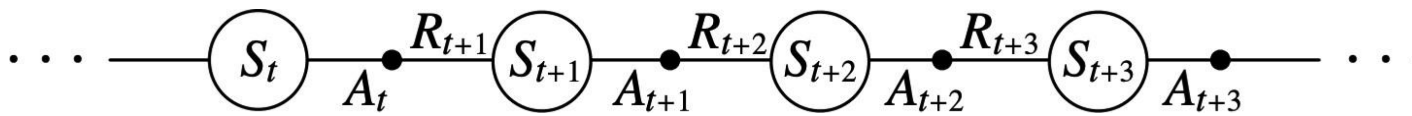
- Following behavior policy $\mu(a|s)$ to collect data

$$S_1, A_1, R_2, ..., S_T \sim \mu$$
$$\text{Update } \pi \text{ using } S_1, A_1, R_2, ..., S_T$$

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

Sarsa: On-policy TD Control

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



- $\epsilon - Greedy$ policy for one step, then bootstrap the action value function:

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

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

N-step Sarsa

- Consider the following *n-step* Q-returns for n=1,2,∞

$$
\begin{aligned}
n = 1 (Sarsa) \quad q_t^{(1)} &= R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) \\
n = 2 \quad q_t^{(2)} &= R_{t+1} + \gamma R_{t+2} + \gamma^2 Q(S_{t+2}, A_{t+2}) \\
&\vdots \\
n = \infty (MC) \quad q_t^{\infty} &= R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-t-1} R_T
\end{aligned}
$$

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

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

- N-step Sarsa updates Q(s, a) towards the n-step Q-return:

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

Off-policy control with Q learning

- We allow both behavior and target policies to improve

- The target police π is greedy on Q(s, a)

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

- The behavior policy μ could be totally random, but we let it improve following $\epsilon - greedy$ on Q(s, a)

- Thus Q-learning target

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

- Thus the Q-learning update

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

Comparison of Sarsa and Q-learning

□ Sarsa: On-Policy TD control

Choose action A_t from S_t using policy derived from Q with $\epsilon - greedy$
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 $\epsilon - greedy$

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

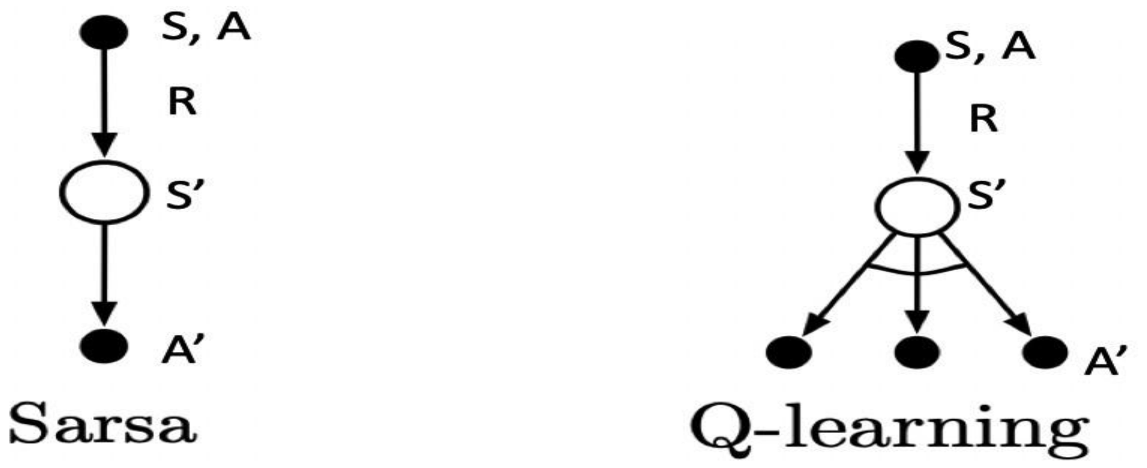
□ Q-learning: Off-Policy TD control

Choose action A_t from S_t using policy derived from Q with $\epsilon - greedy$
Take action A_t, observe R_{t+1} and S_{t+1}
Then 'imagine' A_{t+1} as argmax Q(S_{t+1}, a') in the update target

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

Comparison of Sarsa and Q-learning

- Backup diagram for Sarsa and Q-learning



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

Comparison of Sarsa and Q-learning

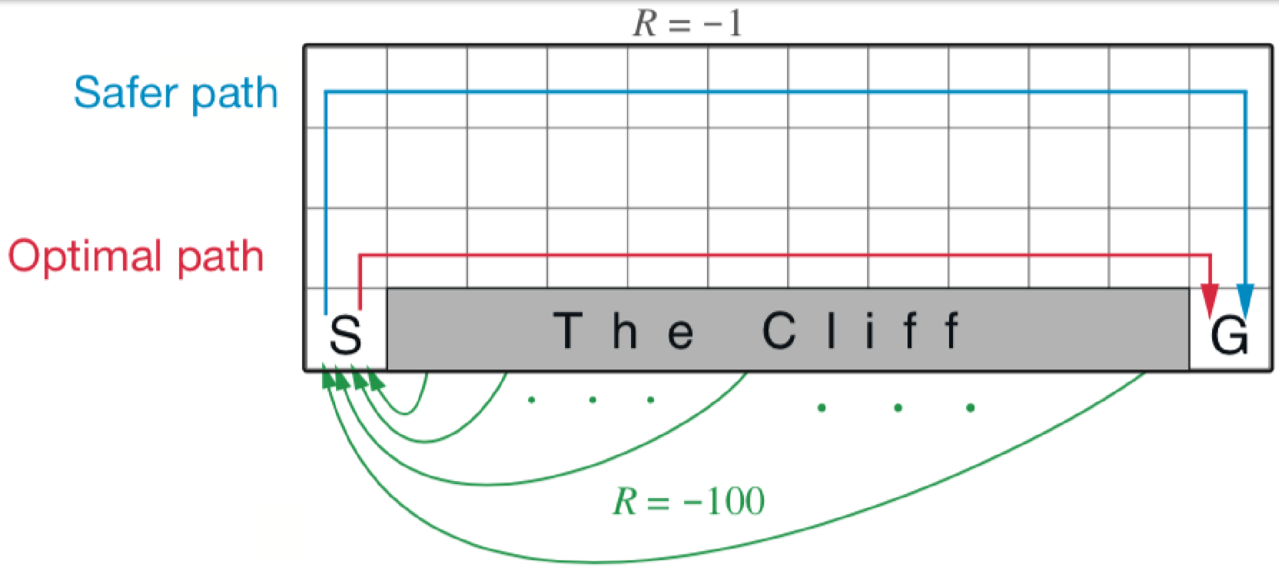
Sarsa

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, 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., 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)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Q learning

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, 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., ε-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
    $S \leftarrow S'$;
  until S is terminal

Example on Cliff Walk

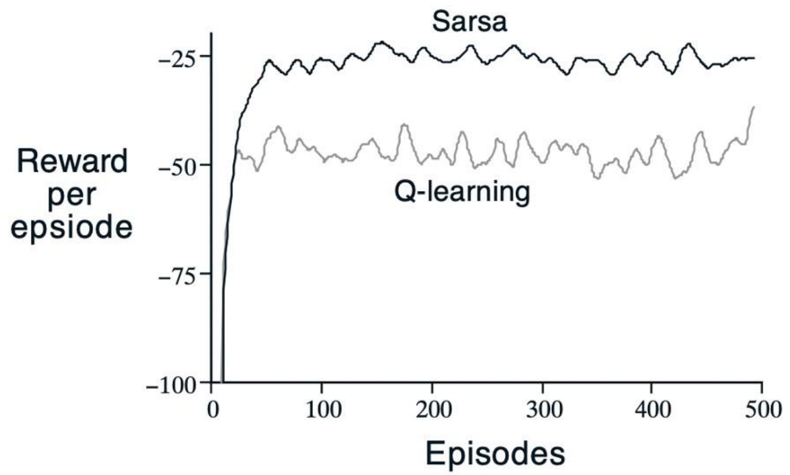


0	0	0	0	R	R	R	R	R	R	R	R
R	R	R	R	R	0	0	0	0	0	0	R
R	0	0	0	0	0	0	0	0	0	0	R
R	*	*	*	*	*	*	*	*	*	*	G

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
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 $V(s) \leftarrow \mathbb{E}[R + \gamma V(S') \mid s]$	TD Learning $V(S) \xleftarrow{\alpha} R + \gamma V(S')$
Q-Policy Iteration $Q(S, A) \leftarrow \mathbb{E}[R + \gamma Q(S', A') \mid s, a]$	Sarsa $Q(S, A) \xleftarrow{\alpha} R + \gamma Q(S', A')$
Q-Value Iteration $Q(S, A) \leftarrow \mathbb{E}[R + \gamma \max_{a' \in \mathcal{A}} Q(S', A') \mid s, a]$	Q-Learning $Q(S, A) \xleftarrow{\alpha} R + \gamma \max_{a' \in \mathcal{A}} Q(S', a')$

where $x \xleftarrow{\alpha} y$ is defined as $x \leftarrow x + \alpha(y - x)$

Importance Sampling

- Estimate the expectation of a function

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

- 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

- Estimate the expectation of a 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 MC

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

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

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

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

- Update value towards correct return

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

Importance Sampling for Off-Policy TD

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

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

- Policies only need to be similar over a single step

Importance Sampling on Q-learning

- Off-policy TD

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

- Why don't use importance sampling on Q-learning?
- Short answer: because Q-learning does not make expected value estimates over the policy distribution.
- Remember bellman optimality backup from value iteration

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

  - 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

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

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