Advanced Machine Learning and Autonomous Agents

Reinforcement Learning II

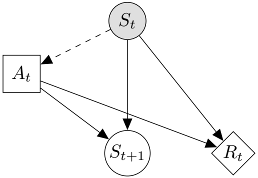
Jesse Read

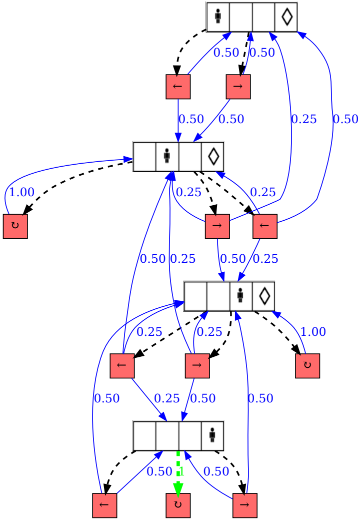
Version: February 5, 2025

Outline

(Re-)Introduction to Reinforcement Learning

The Environment / Markov Decision Process (MDP)





e.g.,

- $\mathcal{S} = \{1, 2, 3, 4\}$ state space
- $\mathcal{A} = \{\rightarrow, \circlearrowright, \leftarrow\}$ action space
- $r(s, a) = \begin{cases} 1 & \text{when } (s, a) = (4, \circlearrowright) \\ 0 & \text{otherwise} \end{cases}$
- $p(s' \mid s, a)$
- γ discount factor

We may not have an expression for p and r!

The Gain (Return): Value of the Current State

The return over a finite horizon

$$G_t = R_{t+1} + R_{t+2} + \cdots + R_T$$
$$= \sum_{k=0}^{T} R_{t+k+1}$$

The return over an infinite horizon

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$$
$$= \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \quad \triangleright \text{ discount factor } \gamma \in (0,1)$$

The Policy

A deterministic policy

$$\pi : \mathcal{S} \to \mathcal{A}$$
$$\pi(s) \in \mathcal{A}(s)$$

e.g., the greedy policy

$$\pi(s) = \operatorname*{argmax}_{a \in \mathcal{A}} Q(s, a)$$

A stochastic policy

$$\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$$
$$\pi(a|s) = p(A_t = a | S_t = s)$$

e.g., the ϵ-greedy policy

$$\pi(s) = \begin{cases} \operatorname{argmax}_{a \in \mathcal{A}(s)} Q(s, a) & \text{with probability } 1 - \epsilon \\ a \sim \mathcal{A}(s) & \text{with probability } \epsilon \end{cases}$$

Value Functions

The state-value function

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$$

The action-value function

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$$

Tasks

Evaluation/Prediction Problem

$$V \approx v_\pi \quad \text{and/or} \quad Q \approx q_\pi$$

Evaluate/predict how policy π performs.

Control/Improvement Problem

$$\pi \rightsquigarrow \pi^*$$

Search for optimal policy π^*.

General schema of generalised policy improvement (GPI): policy-evaluation and policy-improvement processes interact; Bellman optimality equations: The interactive processes stabilizes/converges $\Leftrightarrow$ we have found optimal π^*, v^*.

Dynamic Programming

- Policy Evaluation ($V \approx v_\pi$)
- Value Iteration ($\pi \approx \pi_*$)
- Policy Iteration ($\pi \approx \pi_*$)

The Markov property allows for an efficient recursion; solve the MDP.

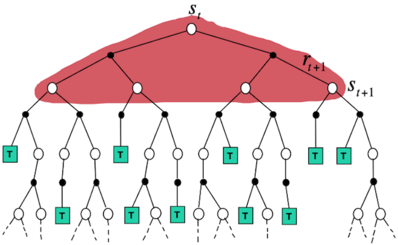


Image credits: David Silver's slides

However:

- Involves $|\mathcal{S}|$ equations (one for each $s \in \mathcal{S}$) $\Rightarrow$ expensive
- Requires knowledge of system dynamics $p(s', r \mid s, a)$

Monte Carlo Methods

Monte Carlo for Evaluation/Prediction ($V \approx v_\pi$)

Recall the value function for policy π:

$$v^\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s]$$

Monte Carlo: Replace the expectation with average over samples.

Samples come from a trajectory (rollout, episode); π interacting with environment p:

$$\underbrace{S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T}_{\tau} \sim p_\pi$$

Basic recipe for evaluation (from π to v_π):

- Play many episodes with π
- Record actual returns from visit to each s.
- Return average as approximation of $v_\pi(s)$

First-Visit Monte Carlo for Prediction

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

  $V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

  $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):

  Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$

  $G \leftarrow 0$

  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

    $G \leftarrow \gamma G + R_{t+1}$

    Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:

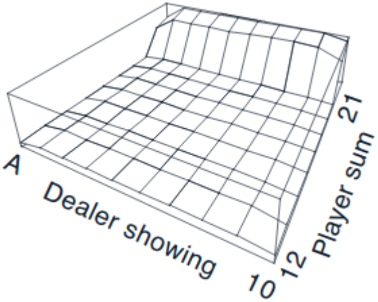
      Append G to $Returns(S_t)$

      $V(S_t) \leftarrow \text{average}(Returns(S_t))$

From Sutton & Barto, 2020

Example: Blackjack

V for policy π that sticks on 20 or 21:



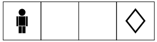
From Sutton & Barto, 2020: Blackjack, $V(s)$ with no usable ace; after 500,000 episodes

The agent has learned the game dynamics *indirectly* via V

V is only optimal if $V \approx v_{\pi*}$.

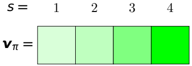
How to play now (what is π)?

V-Learning or Q-Learning?

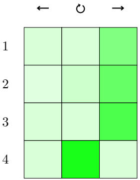


The value of state s, following policy π is

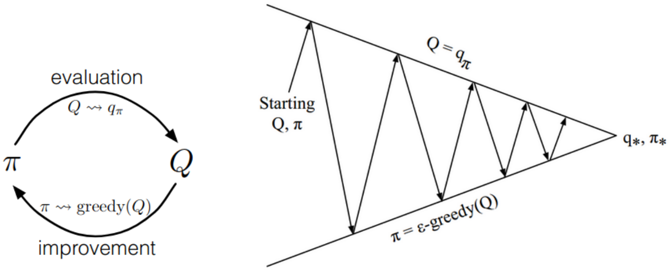
$$V^\pi(s) = Q^\pi(s, \pi(s)) = \max_{a \in \mathcal{A}(s)} Q^\pi(s, a)$$



V-learning only makes sense when we have access to the model of the environment!
The Q-function tells us the action to take!



Monte Carlo Control ($\pi \rightsquigarrow \pi^*$)



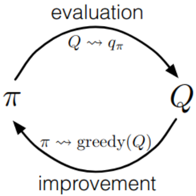
From Sutton & Barto, 2020

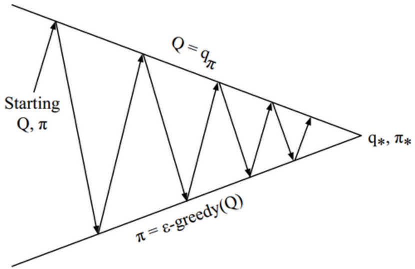
Recall the greedy policy for a given q):

$$\pi(s) = \underset{a \in \mathcal{A}(s)}{\text{argmax}}\, q(s, a)$$

$$\pi_0 \to q_{\pi_0} \to \pi_1 \to q_{\pi_1} \to \cdots \to \pi_k \to q_{\pi_k} \to \cdots \to \pi_*$$

Monte Carlo Control ($\pi \rightsquigarrow \pi^*$)





From Sutton & Barto, 2020

Recall the greedy policy for a given q):

$$\pi(s) = \operatorname*{argmax}_{a \in \mathcal{A}(s)} q(s, a)$$

$$\pi_0 \to q_{\pi_0} \to \pi_1 \to q_{\pi_1} \to \cdots \to \pi_k \to q_{\pi_k} \to \cdots \to \pi_*$$

Problem: With the greedy policy we typically lack observations for many (s, a)-pairs.

Encouraging Exploration via ϵ-soft Policies and ϵ-greedy

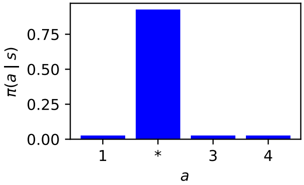
An ϵ-soft policy is *any* policy where

$$\pi(a \mid s) > 0 \quad \text{for all } s \in \mathcal{S} \text{ and } a \in \mathcal{A}(s)$$

For example: The ϵ-greedy policy; select optimal action a^* with probability $1 - \epsilon$, and some other action $a^{\neg *}$ with probability ϵ.

$$\pi(a_j^{\neg *}|s) = \frac{\epsilon}{|\mathcal{A}(s)|} \quad \text{and} \quad \pi(a^*|s) = (1 - \epsilon) + \frac{\epsilon}{|\mathcal{A}(s)|}$$

For example (where $\epsilon = 0.1$):



Other options to encourage exploration: softmax policy, exploring starts, initialize the value functions with high values.

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

Algorithm parameter: small $\varepsilon > 0$

Initialize:

$\pi \leftarrow$ an arbitrary ε-soft policy

$Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

$Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:

Append G to $Returns(S_t, A_t)$

$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

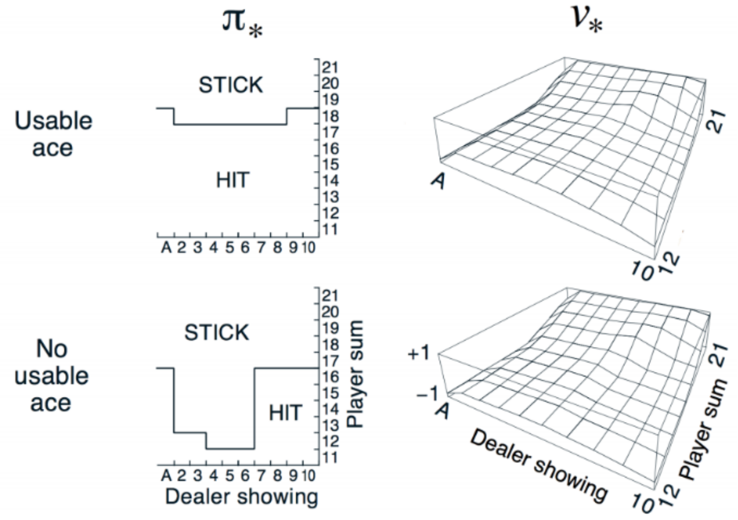
$A^* \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken arbitrarily)

For all $a \in \mathcal{A}(S_t)$:

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

From Sutton & Barto, 2020

Example: Playing Blackjack



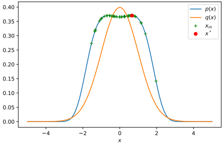
From *Sutton & Barto, 2020*. The state-value function v_* (right) has been calculated from action-value function q_*, which determines π_* (left).

Off-policy Monte-Carlo with Importance Sampling

Practical issue: we seek optimal behaviour, but are forced to behave non-optimally in order to do so (i.e., to explore)!

Consider off-policy (two-policy) methods: one policy for exploration (behavioural policy), another for exploitation (target policy).

Requires additional concepts; importance sampling.



Example: Off-Policy Monte Carlo Prediction

Again, we want expected gain,

$$v^{\pi}(s) = \mathbb{E}_{\mu}[G_t \mid S_t = s]$$

but this time, via behavioural policy μ.

Record trajectory $\tau_m = \{s_t = s, a_t, r_{t+1}, s_{t+1}, \ldots, r_T, s_T\} \sim \mu$ from state s at time t. Approximate the expectation:

$$V(s) = \frac{1}{M} \sum_{m=1}^{M} \omega_m G_t^{(m)} \approx \mathbb{E}[G_t | S_t = s]$$

with importance weight (to compensate the fact that $\pi \neq \mu$):

$$\begin{aligned} \omega_m &= \frac{p_{\pi}(\tau_m)}{p_{\mu}(\tau_m)} \\ &= \frac{\prod_{k=t}^{T-1} \pi(a_k|s_k) p(s_{k+1}|s_k, a_k)}{\prod_{k=t}^{T-1} \mu(a_k|s_k) p(s_{k+1}|s_k, a_k)} = \prod_{k=t}^{T-1} \frac{\pi(a_k|s_k)}{\mu(a_k|s_k)} \end{aligned}$$

Off-policy MC control, for estimating $\pi \approx \pi_*$

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

- $Q(s,a) \in \mathbb{R}$ (arbitrarily)
- $C(s,a) \leftarrow 0$
- $\pi(s) \leftarrow \arg\max_a Q(s,a)$ (with ties broken consistently)

Loop forever (for each episode):

- $b \leftarrow$ any soft policy
- Generate an episode using b: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
- $G \leftarrow 0$
- $W \leftarrow 1$
- Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
  - $G \leftarrow \gamma G + R_{t+1}$
  - $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
  - $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$
  - $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken consistently)
  - If $A_t \neq \pi(S_t)$ then exit inner Loop (proceed to next episode)
  - $W \leftarrow W \frac{1}{b(A_t|S_t)}$

From Sutton & Barto, 2020

Monte Carlo Reinforcement Learning: Summary

- No prior knowledge about the environment dynamics is necessary
- Use samples (empirical episodes) from the environment
- We obtain unbiased estimates $V(s)$ (or $Q(s, a)$)
- After infinite samples, we converge!

but,

- Observing empirical return implies that the agent has to wait until the end of an episode to improve the policy
- High variance
- Infinite samples is a lot

i.e., can take a long time to learn.

Temporal Difference Learning

Temporal Difference (TD)

Recall: V is an estimate of v_π or v_*.

$$\begin{aligned} V(S_t) &= R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \cdots \\ &= R_t + \gamma V(S_{t+1}) \end{aligned}$$

Temporal Difference (TD)

Recall: V is an estimate of v_π or v_*.

$$
\begin{aligned}
V(S_t) &= R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \cdots \\
&= R_t + \gamma V(S_{t+1}) \\
0 &= R_t + \gamma V(S_{t+1}) - V(S_t) \qquad ???
\end{aligned}
$$

In practice, is not the case!

$$
\underbrace{\underbrace{[R_t + \gamma V(S_{t+1})]}_{\text{target (future) } V} - \underbrace{V(S_t)}_{\text{current } V}}_{\text{TD error } \delta} \neq 0
$$

TD Learning: Learn from the future; reduce the TD error toward 0

Let y be the target, $\hat{q}$ be the estimate:

$$\begin{aligned} E &= \frac{1}{2}(y - \hat{q})^2 \\ \nabla_{\hat{q}} E &= \nabla_{\hat{q}} \frac{1}{2}(y - \hat{q})^2 \\ &= (y - \hat{q}) \nabla_{\hat{q}} (y - \hat{q}) \quad \triangleright \text{ Chain rule} \\ &= \hat{q} - y \end{aligned}$$

We can do updates (gradient descent):

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

with learning rate α.

- Note: We use an estimate to compute next estimate;
  - a form of bootstrapping,
  - introduces bias,
  - related to dynamic programming (and Monte Carlo)
- Related to the way dopamine cells operate in animal learning:

Wikipedia.

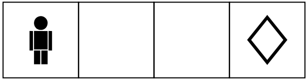
TD for Prediction (Estimating $V \approx v_\pi$)

Tabular TD(0) for estimating v_π

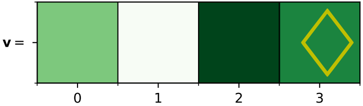
Input: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\textit{terminal}) = 0$

Loop for each episode:
    Initialize S
    Loop for each step of episode:
        $A \leftarrow$ action given by π for S
        Take action A, observe R, S'
        $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
        $S \leftarrow S'$
    until S is terminal

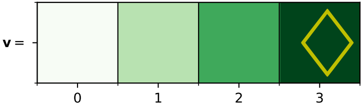
From Sutton & Barto, 2020. TD(0) $\Rightarrow$ one-step ahead



Environment



V via TD at $t = 0$

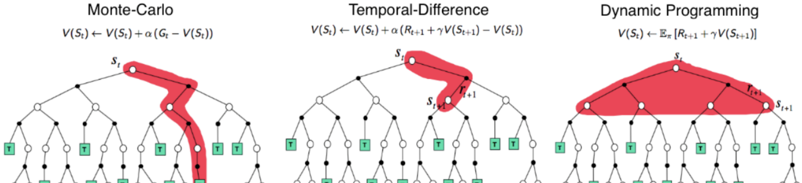


V via TD at $t = 100$

TD vs Monte Carlo (MC), Dynamic Programming (DP)

$$
\begin{aligned}
G_t &= R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \cdots && \triangleright \text{ empirical} \\
V^\pi(s) &= \mathbb{E}_\pi[G_t | S_t = s] && \triangleright \text{ MC} \\
&= \mathbb{E}_\pi[R_t + \gamma G_{t+1} | S_t = s] && \triangleright \text{ DP} \\
&= \mathbb{E}_\pi[R_t + \gamma V^\pi(S_{t+1}) | S_t = s] && \triangleright \text{ TD}
\end{aligned}
$$

- MC estimates $\mathbb{E}_\pi[G_t | S_t = s]$ with full-episode samples
- DP leverages the recursion inherent to G_t
- TD target samples one-step *and* uses current estimate V
    - No knowledge of model required (unlike DP methods)
    - Can be implemented naturally online (unlike MC methods)



TD for Control: Learning $Q(S_t, A_t) \approx q_\pi(s, a)$

TD estimation:

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

To know which action to take, we need either knowledge of the MDP or. . . to learn Q.

Off-policy TD control (**Q-Learning**):

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha \left[(R_t + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a)) - Q(S_t, A_t) \right]$$

On-Policy TD control (**SARSA**):

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

Note: SARSA is also Q-Learning. 'Q-Learning' could be called 'SARS' or 'SARSa'.

SARSA

SARSA (On-Policy TD Control)

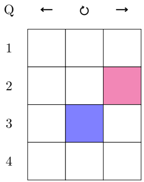
SARSA is an on-policy TD method for learning Q. Its name comes from the sequences:

$$\ldots, S_t\, A_t\, R_t\, S_{t+1}\, A_{t+1}, \ldots \sim p_\pi$$

(from the trajectories generated by policy π interacting with environment p).

The SARSA TD update:

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



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

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize S
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Loop 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

From Sutton & Barto, 2020

Q-Learning

Q-Learning (Off-policy TD Control)

Q-Learning is a TD method, or class of methods (including SARSA) for learning Q. We refer here to vanilla off-policy Q-Learning.

Like SARSA, we have

$$\ldots, S_t\, A_t\, R_t\, S_{t+1}\, A_{t+1}, \ldots \sim p_\pi$$

but the update is:

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

Like SARSA (on-policy), we *follow* an ϵ-greedy policy to select actions; but here we use a different action (the max action) as our estimate of value (Q) in our target.

Q	←	↻	→
1			
2			
3			
4			

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

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\mathit{terminal}, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop 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

From Sutton & Barto, 2020

Off-Policy vs On-Policy TD Control

Why is Q-learning an off-policy method? (Why is SARSA on-policy?)

Both policies *follow* (generate data

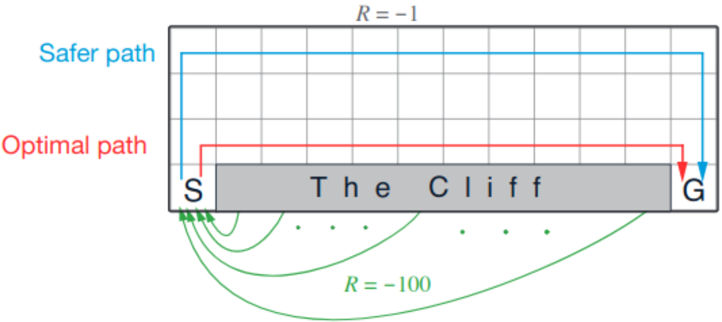
$$\ldots, S_t\, A_t\, R_t\, S_{t+1}\, A_{t+1}, \ldots \sim \pi_{\epsilon\text{-greedy}}$$

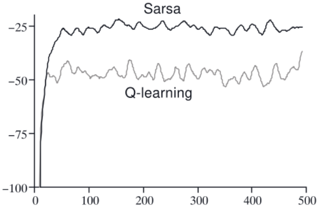
from) policy π_t (ϵ-soft). But Q-learning uses a different policy (greedy) to determine value:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\big[R_t + \gamma \underbrace{\max_a Q(S_{t+1}, a)}_{\text{via } \pi_{\text{greedy}}} - Q(S_t, A_t)\big]$$

So in Q-Learning we learn $\tilde{\pi}_t^* \rightsquigarrow \pi^*$ (note how it is used in the target), rather than $\pi_t \rightsquigarrow \pi^*$.

SARSA vs Q-Learning: The Cliff Example



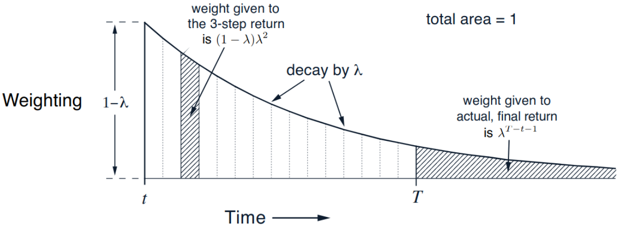


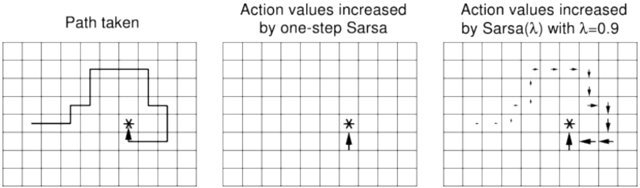
From Sutton & Barto, 2020

Eligibility Traces: TD(λ)

Problem of TD(0): information takes a long time to reach all the states (or might not even reach them) in a large state space.

Solution: weighting factor λ.





From Sutton & Barto, 2020

Summary

Summary: Value-based Learning

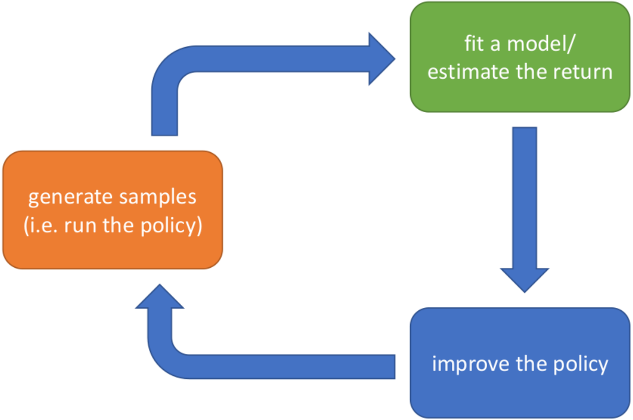


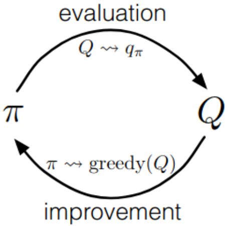
Image: Sergey Levine

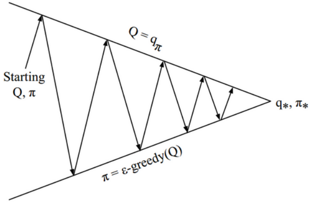
Fit $Q(s, a)$, then set $\pi(s) = \text{argmax}_a Q(s, a)$.

Generalised Policy Improvement: The policy is at least as good as the previous one!

Summary: MC and TD

Monte Carlo (MC) and Temporal Difference (TD) learning.





TD estimation: evaluate the value function; $V(s)$

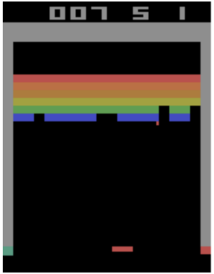
TD control: optimize the value function; $Q(s, a)$

- on-policy (learn π and follow π): SARSA
- off-policy (learn π^*, follow π_t): Q-Learning

MC vs TD? Both are sound. But which is better? Bias vs Variance tradeoff.

Limitations and Upcoming Material

Producing and updating a Q-table (and associated exploration) is tricky with a big state space/many possible state-action pairs.



What's next?

- Deep Q Learning and variants
- Policy Gradient methods
- Actor-Critic methods and variants

Advanced Machine Learning and Autonomous Agents

Reinforcement Learning II

Jesse Read

Version: