08 Model-Free Control (SARSA & Q-Learning)

Table of contents

Model-Free Reinforcement Learning

Last Module (6):

- Model-free prediction
- Prediction: *Optimise* the value function of an unknown MDP

This Module (7):

- Model-free control
- Control: Learn model directly from *experience*

Model-Free Control (SARSA & Q-Learning)

Uses of Model-Free Control

Example problems that can be naturally modelled as MDPs

- Elevator
- Parallel Parking
- Ship Steering
- Bioreactors
- Power stations
- Computer Programming
- Fine tuning LLMs
- Portfolio management
- Protein Folding
- Robot walking

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

On and Off-Policy Learning

On-policy learning

- "Learn on the job"
- Learn about policy π from experience sampled from π

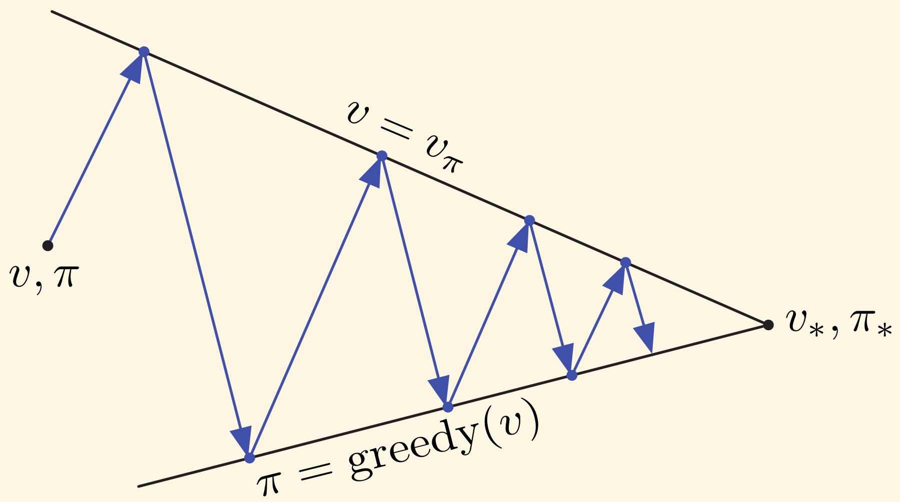
Off-policy learning

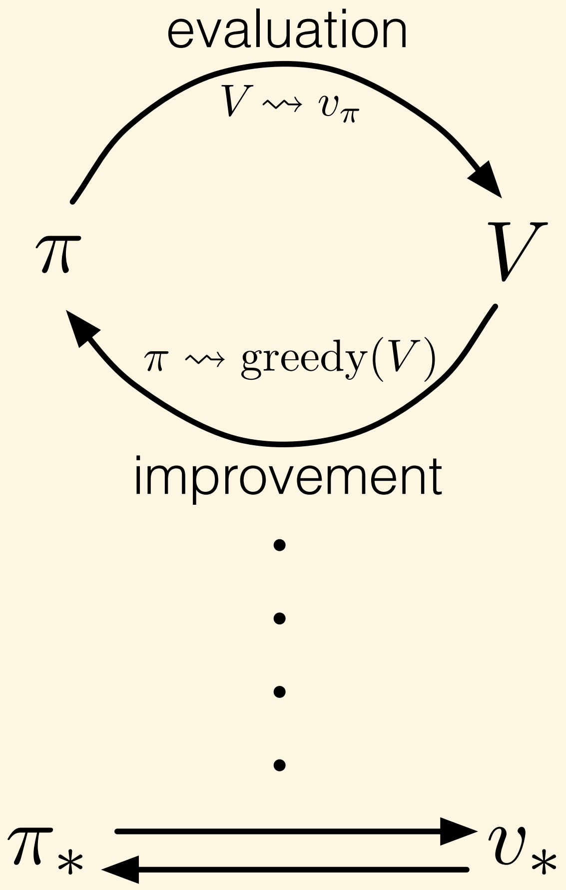
- "Look over someone's shoulder"
- Learn about policy π from experience sampled from μ

Off-policy learning uses trajectories sampled from policy μ, e.g. from another robot, AI agent, human, or simulator.

On-Policy Monte-Carlo Control

Generalised Policy Iteration





Alternation *converges* on optimal policy π_*

- Policy evaluation Estimate v_π
  e.g. Iterative policy evaluation, *going up*
- Policy improvement Generate $\pi' \geq \pi$
  e.g. Greedy policy improvement, act greedily with respect to value function, *going down*

Principle of Optimality

Any optimal policy can be subdivided into two components:

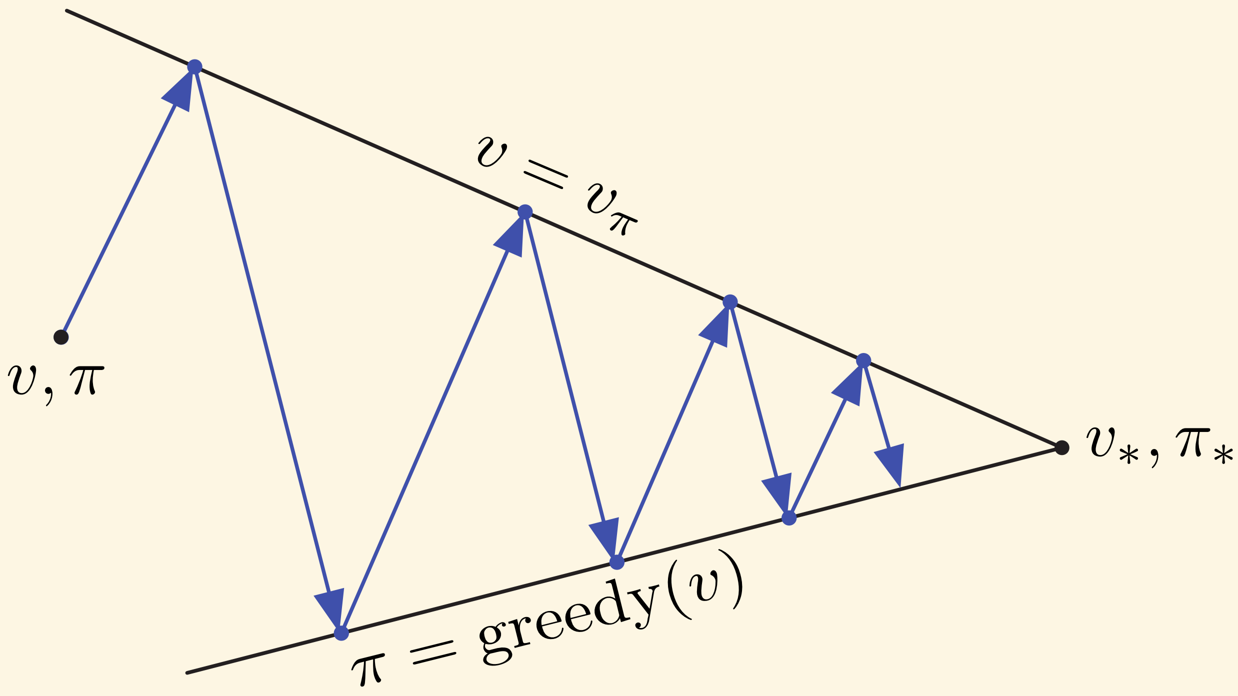
- An optimal first action A_*
- Followed by an optimal policy from successor state S'

Theorem (Principle of Optimality)

A policy $\pi(a|s)$ achieves the optimal value from state s, $v_\pi(s) = v_*(s)$, if and only if

- For any state s' reachable from s
- π achieves the optimal value from state s', $v_\pi(s') = v_*(s')$

Generalised Policy Iteration with Monte-Carlo Evaluation



Policy evaluation 1. Can we use Monte-Carlo policy evaluation to estimate $V = v_\pi$(running multiple episodes/rollouts)?

Policy improvement 2. Can we do greedy policy improvement with MC evaluation?

Model-Free Policy Iteration Using Action-Value Function

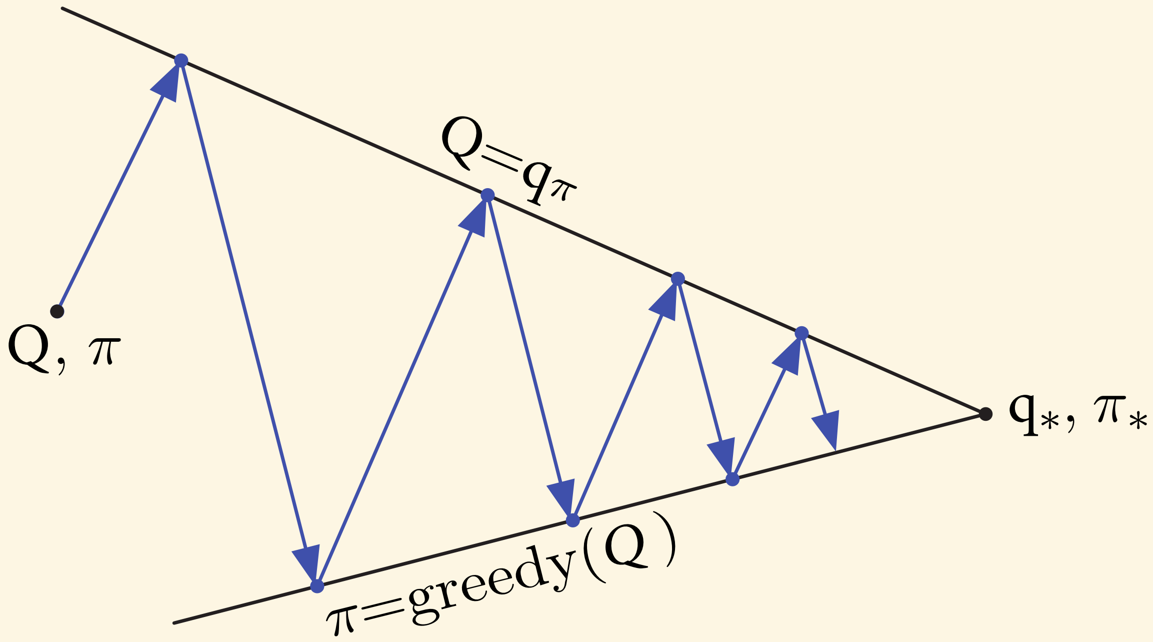
- Problem 1: Greedy policy improvement over $V(s)$ requires a model of MDP

$$\pi'(s) = \arg\max_{a \in \mathcal{A}} \left[\mathcal{R}_s^a + \sum_{s'} P_{ss'}^a V(s') \right]$$

- Alternative: use action-value functions in place of model
  Greedy policy improvement over $Q(s, a)$ is *model-free*

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

Generalised Policy Iteration with Action-Value Function



Policy evaluation We run Monte-Carlo policy evaluation using $Q = q_\pi$

- For each state-action pair $Q(A, S)$ we take *mean* return
- We do this for *all* states and actions, i.e. we don't need model

Policy improvement Greedy policy improvement?

Problem 2: We are acting *greedily* which means you can get stuck in *local minima*

- Note that at each step we are running *episodes* for the policy by trial and error, so we might not see some states
- i.e. you won't necessarily see the states you need in order to get get correct estimate of value function
- Unlike in dynamic programming where you see all states

Example of Greedy Action Selection (Bandit problem)

"Behind one door is tenure - behind the other is flipping burgers at McDonald's."

There are two doors in front of you.

- You open the left door and get reward 0
  $V(left) = 0\ (Monte\ Carlo\ Estimate)$
- You open the right door and get reward $+1$
  $V(right) = +1$
- You open the right door and get reward $+3$
  $V(right) = +2$
- You open the right door and get reward $+2$
  $V(right) = +2$

$\vdots$

You may never explore left door again!

- i.e. are you sure you've chosen the best door?

ε-Greedy Exploration

The simplest idea for ensuring continual exploration:

All m actions are tried with non-zero probability

- with probability $1 - \varepsilon$ choose the best action, *greedily*
- with probability ε choose a *random*

$$\pi(a \mid s) = \begin{cases} \dfrac{\varepsilon}{m} + 1 - \varepsilon & \text{if } a^* = \arg\max\limits_{a \in \mathcal{A}} Q(s,a) \\ \dfrac{\varepsilon}{m} & \text{otherwise} \end{cases}$$

ε-Greedy Policy Improvement

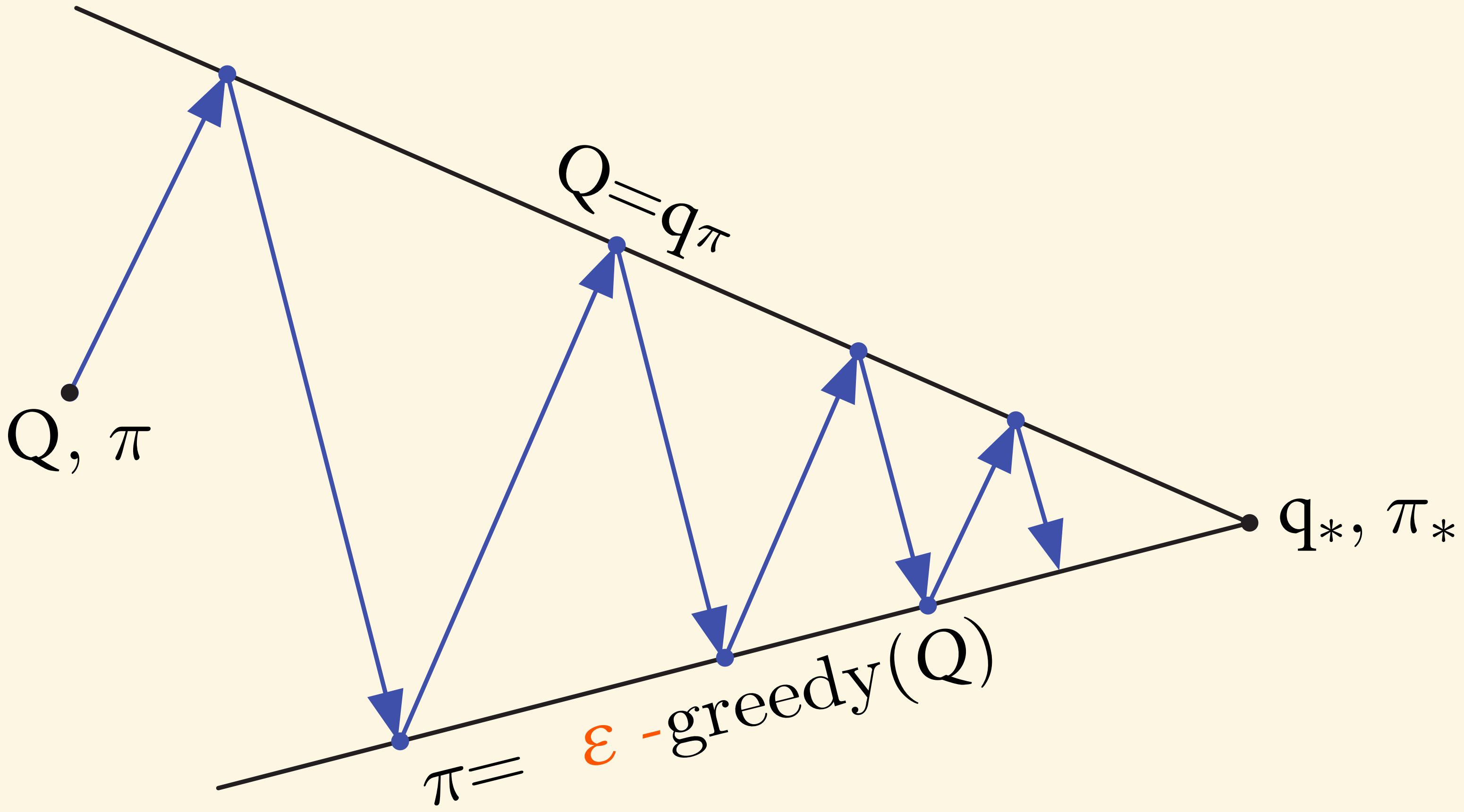
Theorem

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

$$
\begin{aligned}
q_\pi(s, \pi'(s)) &= \sum_{a \in \mathcal{A}} \pi'(a \mid s)\, q_\pi(s, a) \\
&= \frac{\varepsilon}{m} \sum_{a \in \mathcal{A}} q_\pi(s, a) + (1 - \varepsilon) \max_{a \in \mathcal{A}} q_\pi(s, a) \\
&\geq \frac{\varepsilon}{m} \sum_{a \in \mathcal{A}} q_\pi(s, a) + (1 - \varepsilon) \sum_{a \in \mathcal{A}} \frac{\pi(a \mid s) - \frac{\varepsilon}{m}}{1 - \varepsilon} q_\pi(s, a) \\
&= \sum_{a \in \mathcal{A}} \pi(a \mid s)\, q_\pi(s, a) = v_\pi(s)
\end{aligned}
$$

Proof idea: $\max_{a \in \mathcal{A}} q_\pi(a, a)$ is at least as good as any weighted sum of all of your actions; therefore from the policy improvement theorem, $v_{\pi'}(s) \ \geq \ v_\pi(s)$

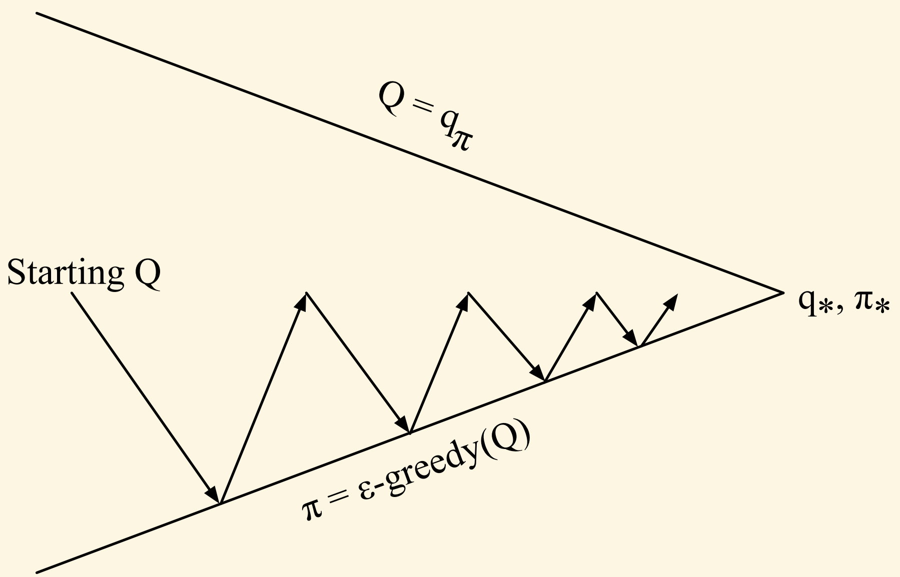
Monte-Carlo Policy Iteration



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

Policy improvement $\varepsilon-$Greedy policy improvement

Monte-Carlo Control



Every episode:

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

- Not necessary to fully evaluate policy every time, *going all the way to the top*, instead, immediately improve policy for every *episode*

Policy improvement $\varepsilon -$Greedy policy improvement

Greedy in the Limit with Infinite Exploration (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,

$$\lim_{k \to \infty} \pi_k(a \mid s) = \mathbf{1}\left(a = \arg\max_{a' \in \mathcal{A}} Q_k(s, a')\right)$$

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

- i.e. decay ε over time according to a *hyperbolic schedule*

Note that the term $\mathbf{1}(S_t = s)$ is an *indicator function* that equals 1 if the condition inside is true, and 0 otherwise.

$$\mathbf{1}(S_t = s) = \begin{cases} 1, & \text{if } S_t = s \\ 0, & \text{otherwise} \end{cases}$$

It acts as a *selector* that ensures the update is applied only to the state currently being visited.

- The boldface notation $\mathbf{1}(S_t = s)$ simply emphasises that this is a function, not a constant.

GLIE Monte-Carlo Control

Sample kth episode using π: $\{S_1, A_1, R_2, \ldots, S_T\} \sim \pi$

For each state S_t and action A_t in the episode update an incremental mean,

$$\begin{aligned} N(S_t, A_t) &\leftarrow N(S_t, A_t) + 1 \\ Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)}\Big(G_t - Q(S_t, A_t)\Big) \end{aligned}$$

Improve policy based on new action–value function, replacing Q values at each step

$$\begin{aligned} \varepsilon &\leftarrow \frac{1}{k} \\ \pi &\leftarrow \varepsilon\text{-greedy}(Q) \end{aligned}$$

- In practice don't need to store π, just store Q (π becomes implicit)

Theorem

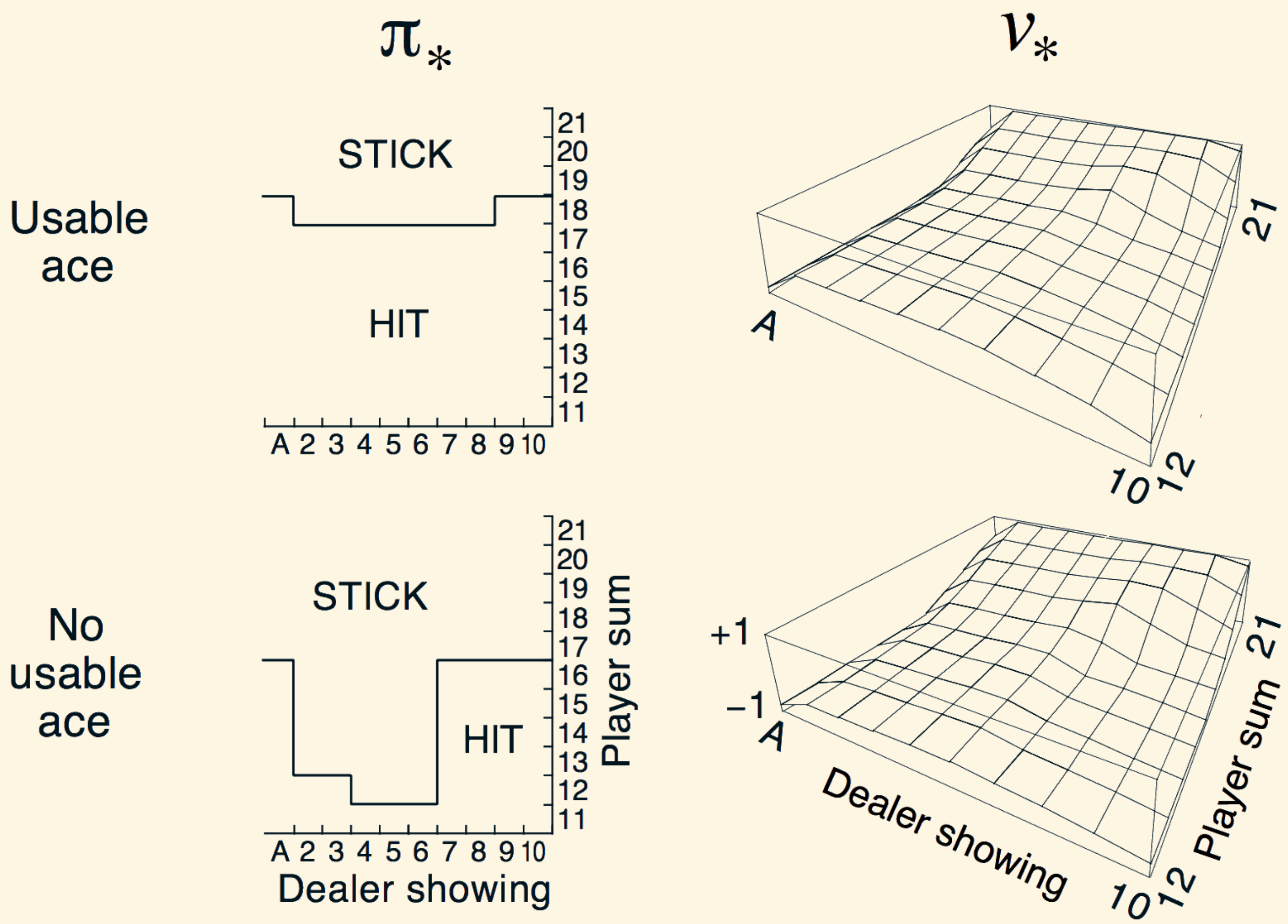
GLIE Monte Carlo control converges to the optimal action–value function,

$$Q(s, a) \quad \rightarrow \quad q_*(s, a)$$

- Converges to the optimal policy, π_*
- Every episode Monte-Carlo is substantially more efficient than running multiple episodes at each step

Back to the Blackjack Example

Monte-Carlo Control in Blackjack



Monte-Carlo Control algorithm finds the optimal policy!

(Note: *Stick* is equivalent to *hold* in this Figure)

On-Policy Temporal-Difference Learning

MC versus TD Control (Gain efficiency by Bootstrapping)

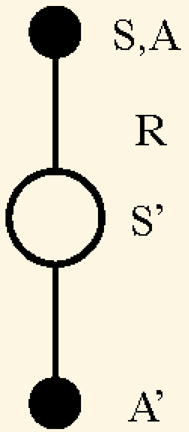
Temporal-difference (TD) learning has several advantages over Monte-Carlo (MC)

- Lower variance
- Online (including non-terminating)
- Incomplete sequences

Natural idea: use TD instead of MC in our control loop

- Apply TD to $Q(S, A)$
- Use ε-greedy policy improvement
- Update *every* time-step

Updating Action-Value Functions with SARSA

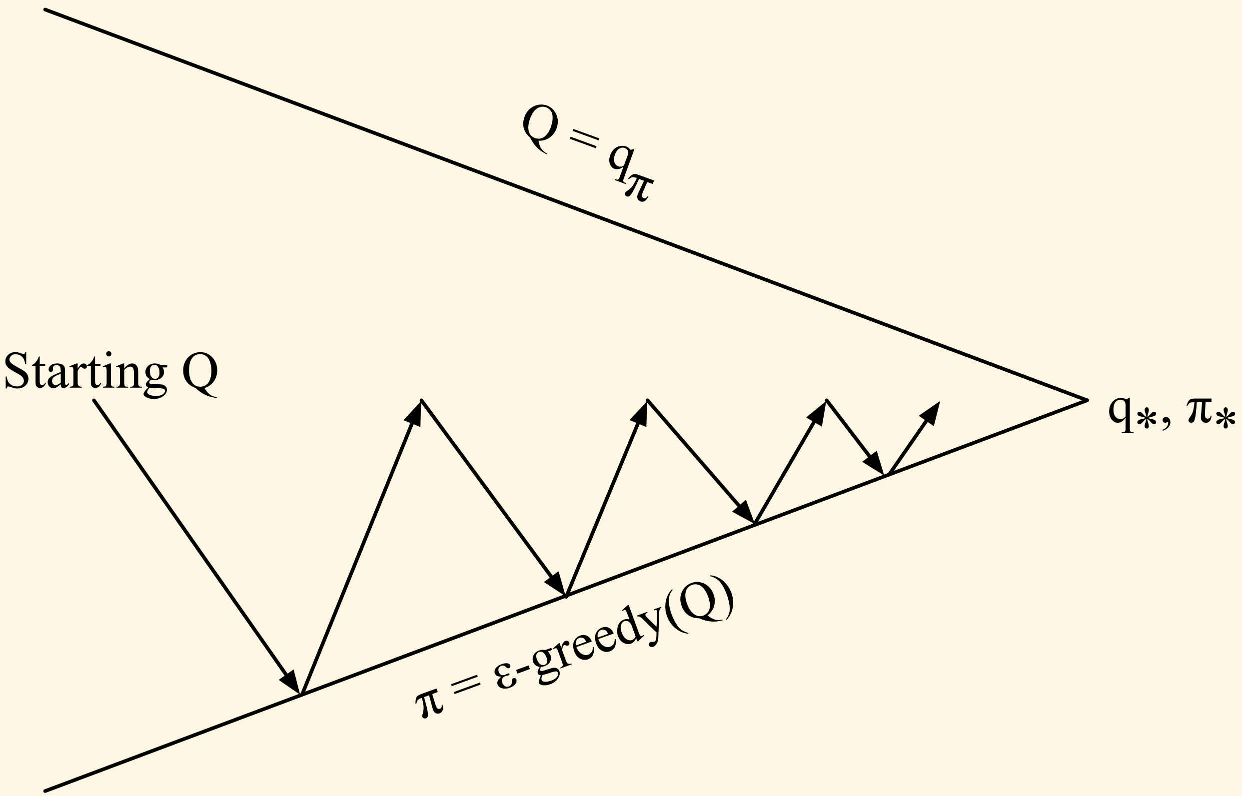


$$Q(S,A) \leftarrow Q(S,A) + \alpha\Big(R + \gamma Q(S',A') - Q(S,A)\Big)$$

Starting in state-action pair S, A, sample reward R from environment, then sample our own policy in S' for A' (note S' is chosen by the environment)

- Moves $Q(S,A)$ value in direction of *TD Target* - $Q(S,A)$ (as in Bellman equation for Q).

On-Policy Control with SARSA



Every time-step:

Policy evaluation SARSA, $Q \approx q_\pi$

Policy improvement $\varepsilon-$Greedy policy improvement

SARSA Algorithm for On-Policy Control

SARSA (On-Policy)

Initialise $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\text{terminal-state}, \cdot) = 0$
Loop for each episode:
  Initialise 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' (environment takes us to state S')
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma Q(S', A') - Q(S, A)\right]$
    $S \leftarrow S'; A \leftarrow A'$
until S is terminal

RHS of $Q(S, A)$ update is on-policy version of Bellman equation—expectation of what happens in environment to state S' and what happens under our own policy from that state S' onwards.

Convergence of SARSA (Stochastic optimisation theory)

Theorem

SARSA converges to the optimal action-value function, $Q(s, a) \rightarrow q_*(s, a)$*, under the following conditions:*

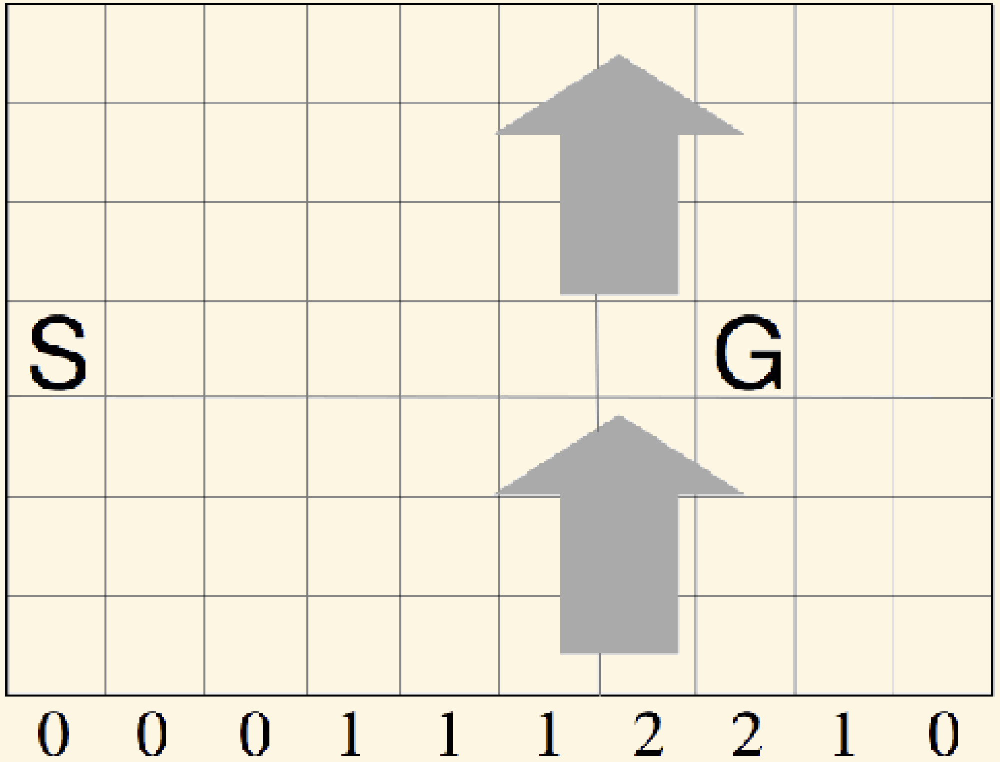
- GLIE sequence of policies $\pi_t(a \mid s)$
- Robbins-Monro sequence of step-sizes α_t

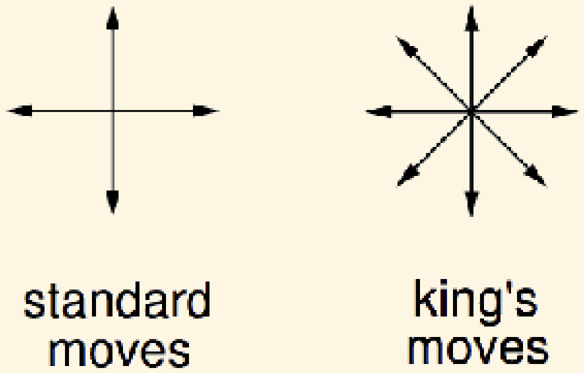
$$\sum_{t=1}^{\infty} \alpha_t = \infty$$

$$\sum_{t=1}^{\infty} \alpha_t^2 < \infty$$

Tells us that step sizes must be sufficiently large to move us as far as you want; and changes to step sizes must result in step eventually vanishing

Windy Gridworld Example



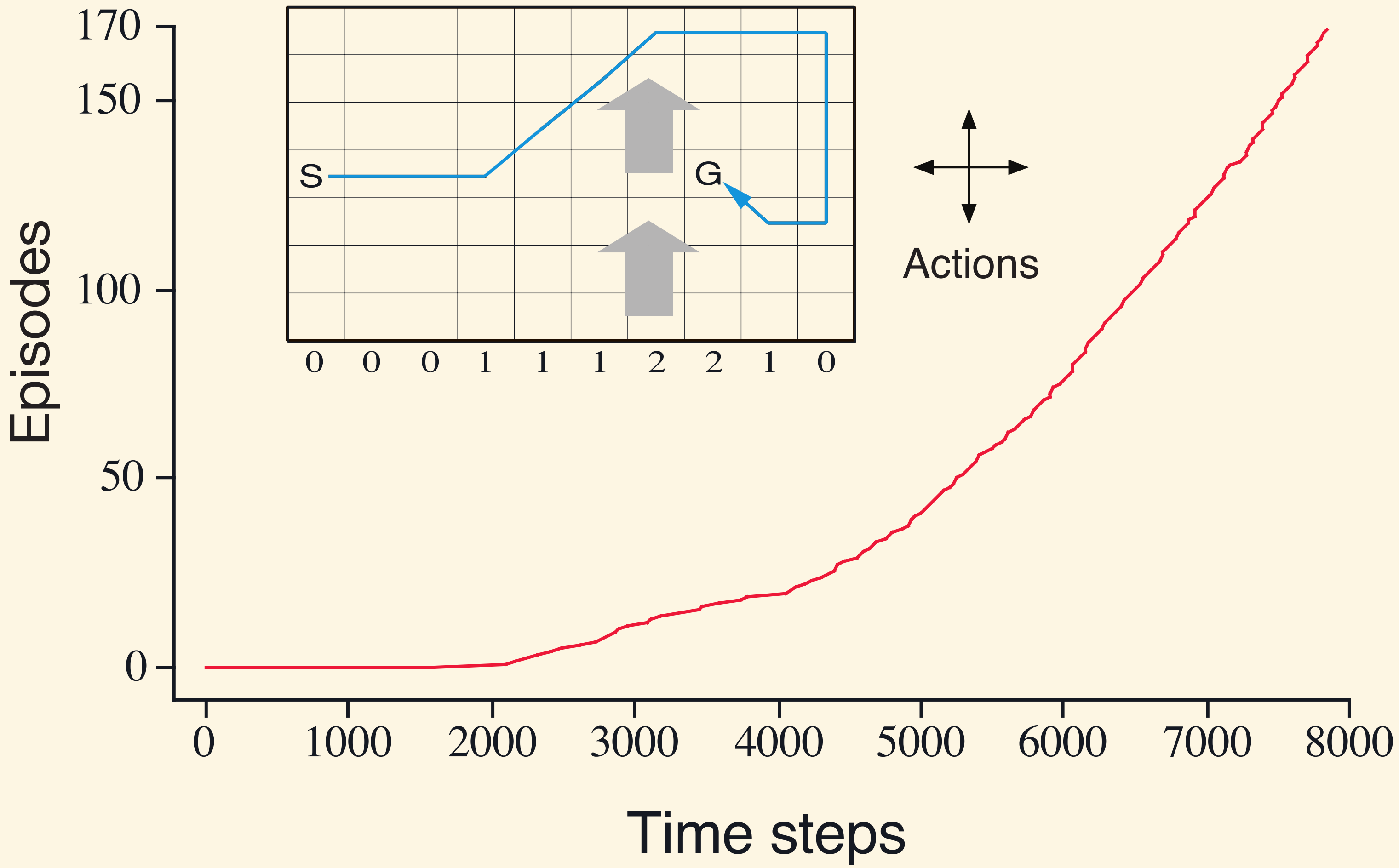


Numbers under each column is how far you get blown up per time step

- Reward $= -1$ per time-step until reaching goal

(Undiscounted and uses fixed step size α in this example)

SARSA on the Windy Gridworld



Episodes completed (vertical axis) versus time steps (horizontal axis)

n-Step SARSA (Bias-variance trade-off)

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

$$\begin{array}{lll} n = 1 & \text{(SARSA)} & q_t^{(1)} = R_{t+1} + \gamma Q(S_{t+1}) \\ n = 2 & & q_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 Q(S_{t+2}) \\ \vdots & & \vdots \\ n = \infty & \text{(MC)} & q_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-1} R_T \end{array}$$

Define the n-step Q-return:

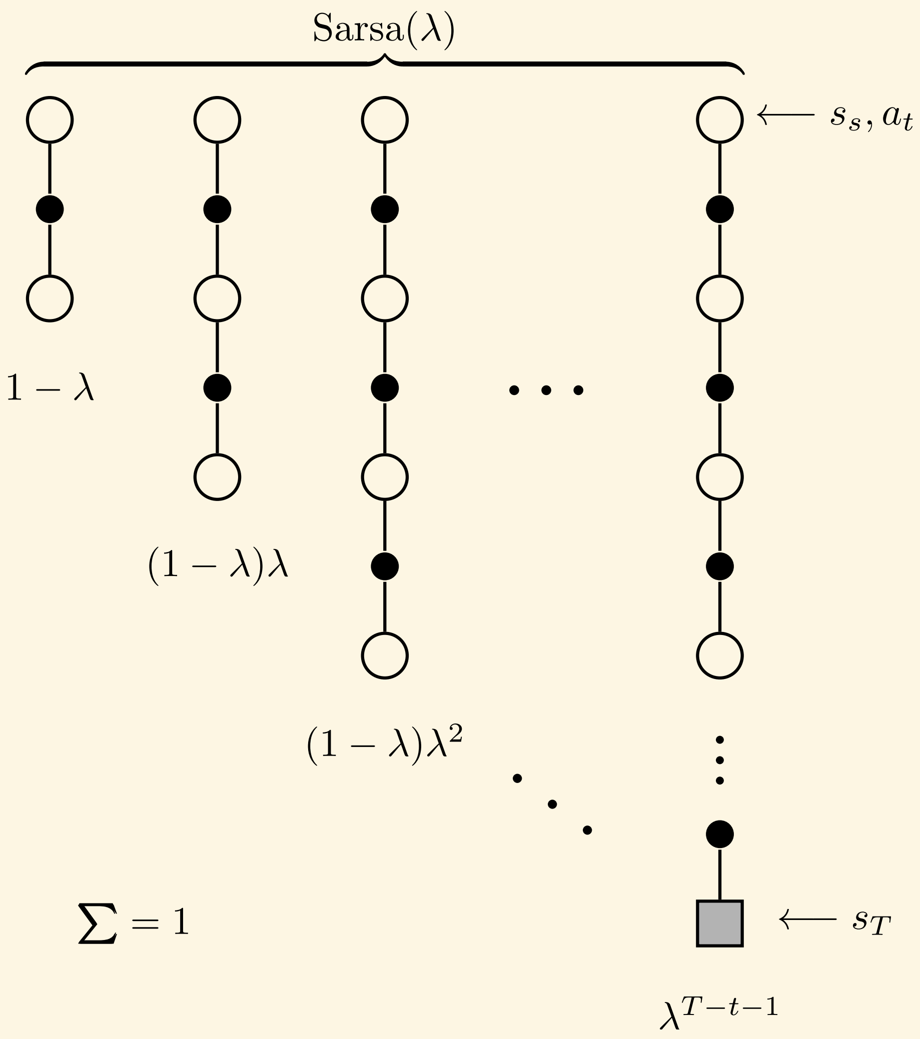
$$q_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n Q(S_{t+n})$$

n-step SARSA updates $Q(s, a)$ towards the n-step Q-return:

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

- $n = 1$:   high bias, low variance
- $n = \infty$: no bias, high variance

Forward View SARSA(λ)



We can do the same thing for control as we did in model-free prediction:

The q^λ *return* combines all n-step Q-returns $q_t^{(n)}$

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

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

Forward-view SARSA(λ):

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

Backward View SARSA(λ)

Just like TD(λ), we use eligibility traces in an online algorithm

- However SARSA(λ) has one eligibility trace for each *state–action pair*

$$
\begin{aligned}
E_0(s, a) &= 0 \\
E_t(s, a) &= \gamma\lambda E_{t-1}(s, a) + \mathbf{1}(S_t = s, A_t = a)
\end{aligned}
$$

$Q(s, a)$ is updated for every state s and action a

- In proportion to TD-error δ_t and eligibility trace $E_t(s, a)$

$$\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)$$

$$Q(s, a) \leftarrow Q(s, a) + \alpha\, \delta_t\, E_t(s, a)$$

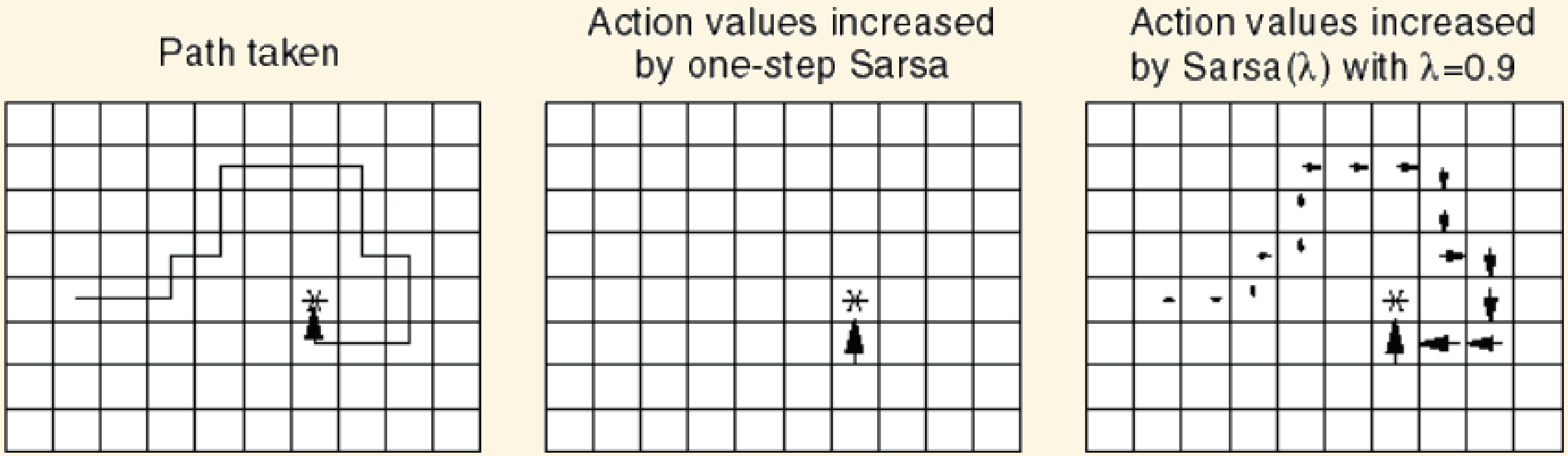
SARSA(λ) Algorithm

SARSA(λ)

Initialise $Q(s,a)$ arbitrarily, $\forall s \in \mathcal{S}, a \in \mathcal{A}(s)$
Loop for each episode:
  $E(s,a) = 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
  Initialise S, A
  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)
    $\delta \;\leftarrow\; R + \gamma Q(S', A') - Q(S, A)$
    $E(S, A) \;\leftarrow\; E(S, A) + 1$
    For all $s \in \mathcal{S}, a \in \mathcal{A}(s)$:
      $Q(s,a) \;\leftarrow\; Q(s,a) + \alpha\,\delta\,E(s,a)$
      $E(s,a) \;\leftarrow\; \gamma\lambda E(s,a)$
    $S \;\leftarrow\; S'$; $A \;\leftarrow\; A'$
until S is terminal

Algorithm updates *all* action-state pairs $Q(s,a)$ at *each* step of an episode

SARSA(λ) Gridworld Example



SARSA(0) SARSA(λ)

- Assume initialise action-values to zero
- Size of arrow indicates magnitude of $Q(A, S)$ value for that state
- SARSA updates *all* action-state pairs $Q(s, a)$ at *each* step of episode

Off-Policy Learning

Off-Policy Learning

Evaluate target policy $\pi(a \mid s)$ to compute $v_\pi(s)$ or $q_\pi(s, a)$ while following a *behaviour* policy $\mu(a \mid s)$

$$\{S_1, A_1, R_2, \ldots, S_T\} \sim \mu$$

Why is this important?

- Learning from observing humans or other agents (either AI or simulated)
- Re-use experience generated from old policies $\pi_1, \pi_2, \ldots, \pi_{t-1}$
- Learn about *optimal* policy while following *exploratory* policy
- Learn about *multiple* policies while following *one* policy

Importance Sampling

Estimate the expectation of a different distribution

$$\begin{aligned}
\mathbb{E}_{X\sim P}[f(X)] &= \sum P(X)f(X) \\
&= \sum Q(X)\,\frac{P(X)}{Q(X)}\,f(X) \\
&= \mathbb{E}_{X\sim Q}\left[\frac{P(X)}{Q(X)}f(X)\right]
\end{aligned}$$

A technique for estimating expectations by sampling from different distributions

- Re-weights by dividing and multiplying samples to correct mismatch between expectations of the different distributions

Importance Sampling for Off-Policy Monte-Carlo

Use returns generated from μ to evaluate π

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

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

Updates values towards *corrected* return

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

- Cannot use if μ is zero when π is non-zero
- However, importance sampling *dramatically* increases variance, in case of Monte-Carlo learning

Importance Sampling for Off-Policy TD

Use TD targets generated from μ to evaluate π

- Weight TD target $R + \gamma V(S')$ by importance sampling

Only need a single importance sampling correction

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

- Much lower variance than Monte Carlo importance sampling (policies only need to be similar over a *single* step)
- In practice you have to use TD learning when working off-policy (it becomes imperative to bootstrap)

Q-Learning

We now consider off-policy learning of action-values, $Q(s, a)$

No importance sampling is required using action-values as you can bootstrap as follows

- Next action is chosen using *behaviour* policy $A_{t+1} \sim \mu(\cdot \mid S_t)$
- But we consider *alternative* successor action $A' \sim \pi(\cdot \mid S_t)$
- $\cdots$ and we update $Q(S_t, A_t)$ towards value of alternative action

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

Q-learning is the technique that works best with off-policy learning

Off-Policy Control with Q-Learning

We can now allow *both* behaviour and target policies to improve

- The target policy π is greedy w.r.t. $Q(s, a)$

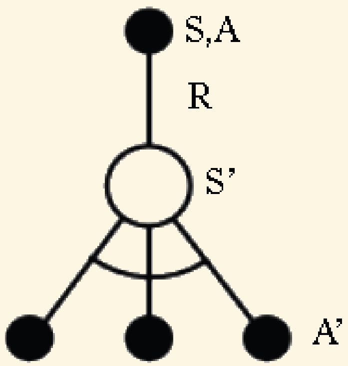
$$\pi(S_{t+1}) = \arg\max_{a'} Q(S_{t+1}, a')$$

- The behaviour policy μ is e.g. ε-greedy w.r.t. $Q(s, a)$

The Q-learning target then simplifies according to:

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

Q-Learning Control Algorithm



$$Q(S, A) \leftarrow Q(S, A) + \alpha \left(R + \gamma \max_{a'} Q(S', a') - Q(S, A) \right)$$

Theorem

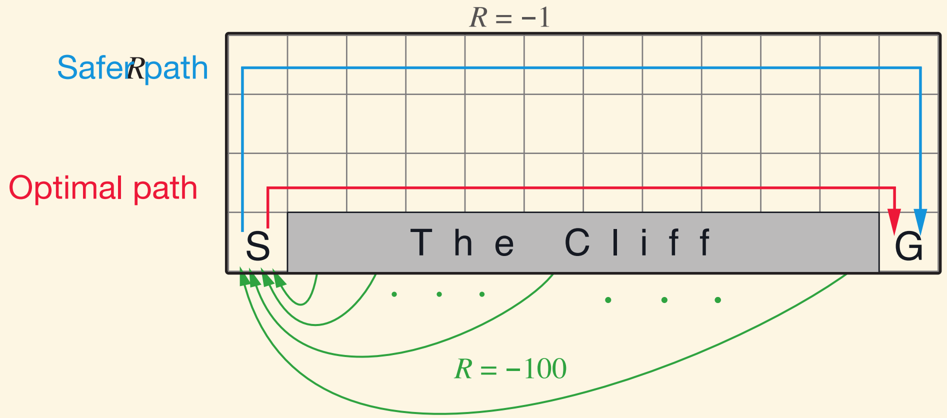
Q-learning control converges to the optimal action-value function, $Q(s, a) \rightarrow q_*(s, a)$

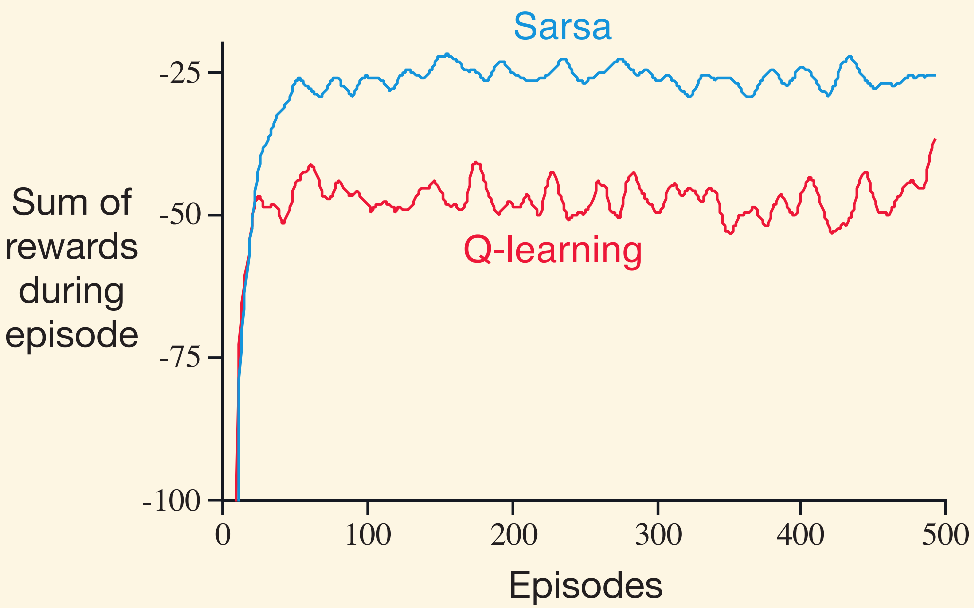
Q-Learning Algorithm for Off-Policy Control

$Q-$Learning (Off-Policy)

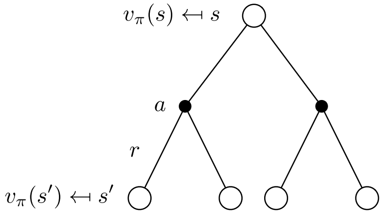
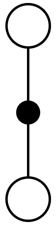
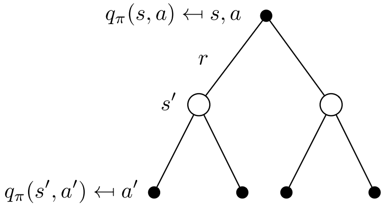
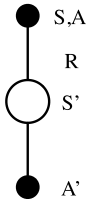
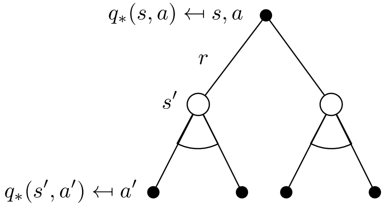
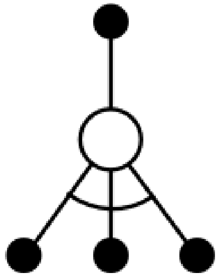
Initialise $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$
Loop for each episode:
  Initialise S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $$Q(S,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

Cliff Walking Example (SARSA versus Q-Learning)





Relationship between Tree Backup and TD

	Full Backup (DP)	*Sample Backup (TD)*
Bellman Expectation Equation for $v_\pi(s)$	 Iterative Policy Evaluation	 TD Learning
Bellman Expectation Equation for $q_\pi(s, a)$	 Q-Policy Iteration	 Sarsa
Bellman Optimality Equation for $q_*(s, a)$	 Q-Value Iteration	 Q-Learning

Relationship between DP and TD

Dynamic Programming (DP)	*Sample Backup (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 $x \xleftarrow{\alpha} y \equiv x \leftarrow x + \alpha(y - x)$

Reference Topics

See the Appendix for details of Convergence & Contraction Mapping Theorem, and the relationship between Forward and Backward TD.