

# Model Free Control

---

DAVIDE BACCIU – [BACCIU@DI.UNIPI.IT](mailto:BACCIU@DI.UNIPI.IT)



UNIVERSITÀ DI PISA

# Outline

---

- ✓ Introduction
- ✓ On-policy Vs Off-Policy
- ✓ On-policy Monte-Carlo
- ✓ On-policy TD learning (SARSA)
- ✓ Off-policy TD (Q-learning)

# Introduction

---

# Today's focus

---

- ✓ Last lecture
  - ✓ Model-free prediction
  - ✓ Estimate the value function of an unknown MDP
- ✓ Today's lecture
  - ✓ Model-free control
  - ✓ Optimise the value function of an unknown MDP

# Model Free Control – Where to find it

---

- ✓ Elevator
- ✓ Robot walking
- ✓ Vehicle Steering
- ✓ Bioreactor
- ✓ Molecule engineering
- ✓ Robocup Soccer
- ✓ Quake
- ✓ Portfolio management
- ✓ Protein Folding
- ✓ 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

# On-policy & Off-policy Learning

---

- ✓ **On-policy** learning

- ✓ *Learn on the job*

- ✓ Learn about policy  $\pi$  from experience sampled from  $\pi$

- ✓ **Off-policy** learning

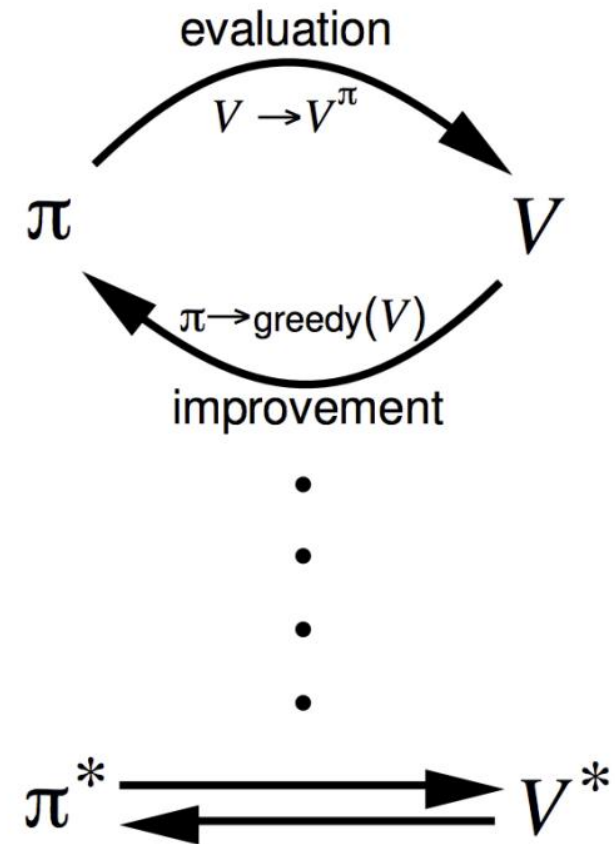
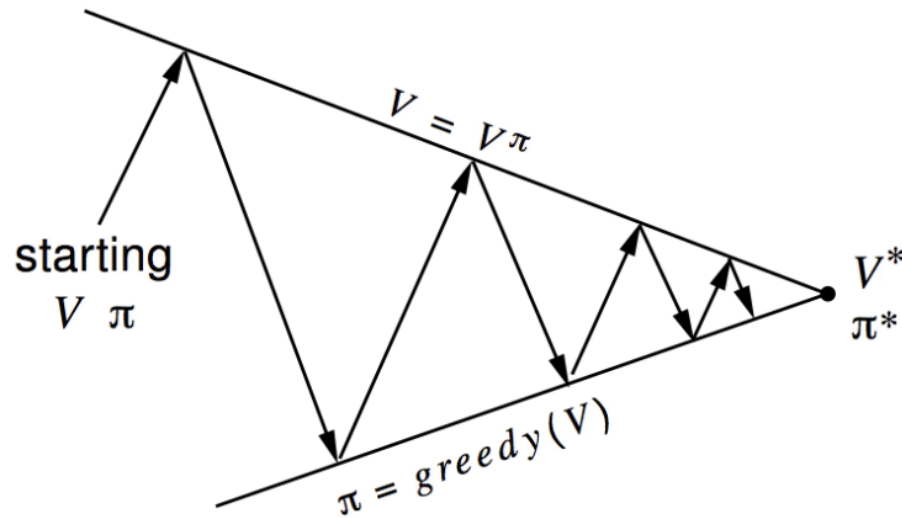
- ✓ *Look over someone's shoulder*

- ✓ Learn about policy  $\pi$  from experience sampled from  $\mu$

# On-policy MC

---

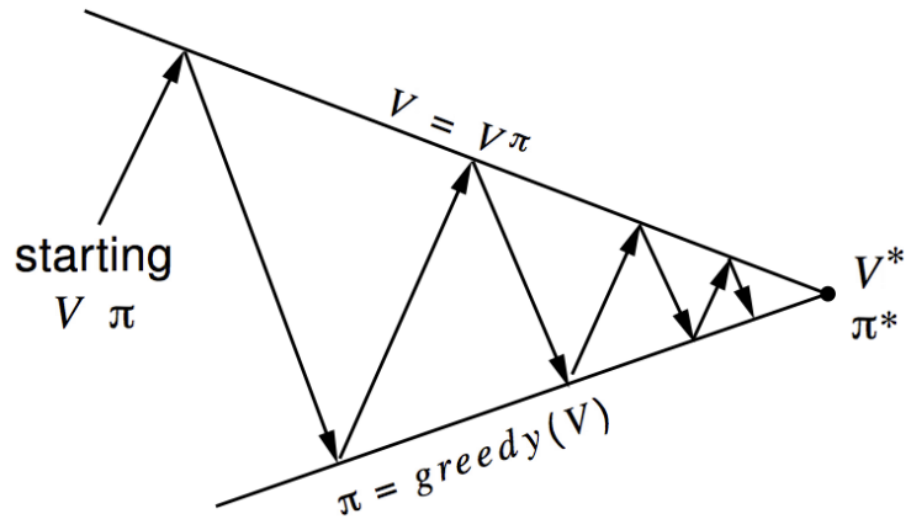
# Generalized Policy Iteration (Lecture 4)



- ✓ Policy evaluation - Estimate  $v_\pi$ 
  - ✓ Any policy evaluation
- ✓ Policy improvement - Generate  $\pi' \geq \pi$ 
  - ✓ Any policy improvement algorithm



# Generalized Policy Iteration with On-policy MC



- ✓ Policy evaluation - Monte-Carlo policy evaluation,  $V = v_\pi$ ?
- ✓ Policy improvement - Generate greedy policy improvement?

# Model-Free Policy Iteration Using Action-Value Function

---

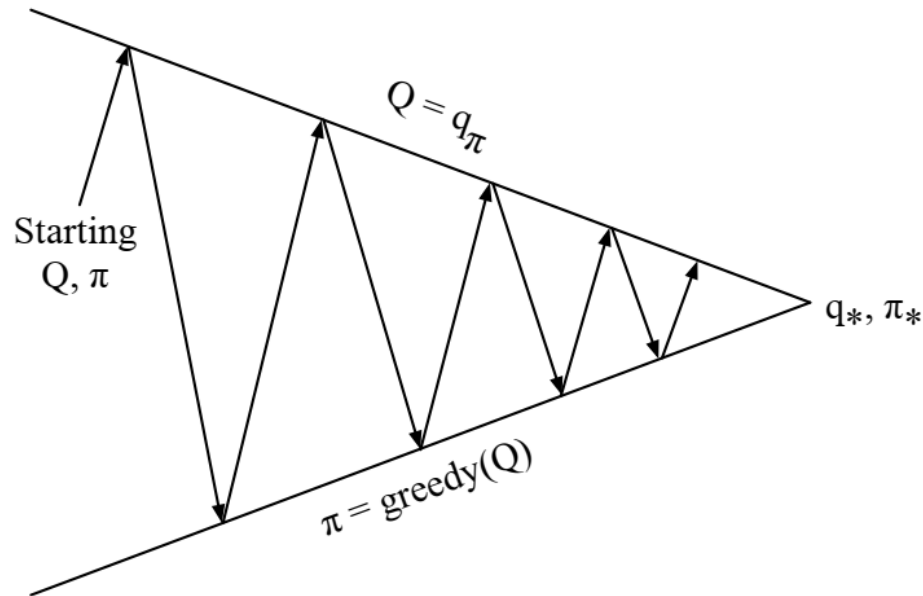
- ✓ Greedy policy improvement over  $V(s)$  requires model of MDP

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

- ✓ Greedy policy improvement over  $Q(s, a)$  is model-free

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

# Generalized Policy Iteration with Action-Value Function



- ✓ Policy evaluation - Monte-Carlo policy evaluation,  $Q = q_\pi$
- ✓ Policy improvement - Generate Greedy policy improvement?

# Example of Greedy Action Selection



"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  $0 - V(\text{left}) = 0$
  - ✓ You open the right door and get reward  $+1 - V(\text{right}) = +1$
  - ✓ You open the right door and get reward  $+3 - V(\text{right}) = +2$
  - ✓ You open the right door and get reward  $+2 - V(\text{right}) = +2$
- ...
- ✓ Are you sure you've chosen the best door?

# $\epsilon$ -greedy Exploration

---

- ✓ Simplest idea for ensuring continual exploration
- ✓ All  $m$  actions are **tried with non-zero probability**
  - ✓ With probability  $1 - \epsilon$  choose the greedy action
  - ✓ With probability  $\epsilon$  choose an action at random

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

# $\epsilon$ -greedy Policy Improvement

## Theorem

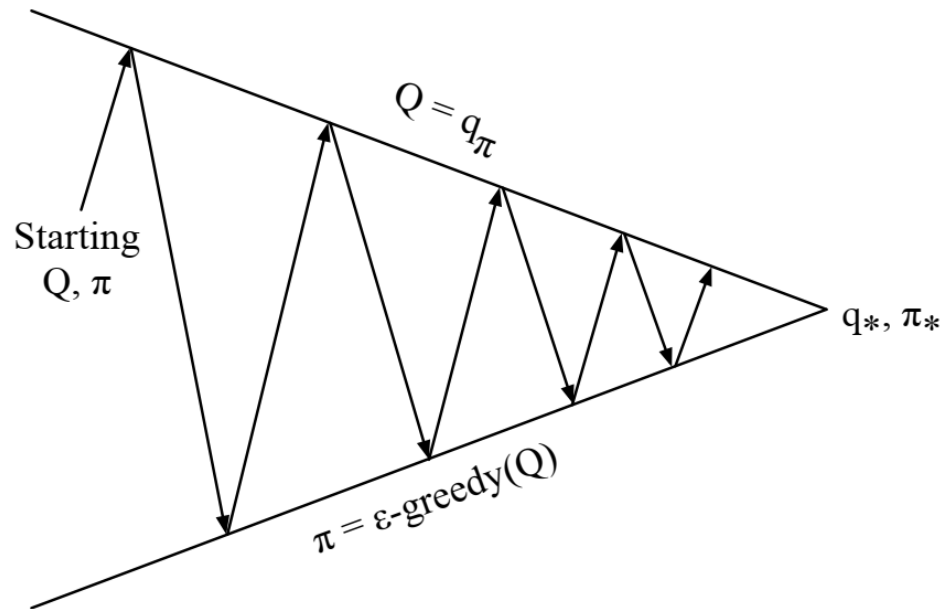
For any  $\epsilon$ -greedy policy  $\pi$ , the  $\epsilon$ -greedy policy  $\pi'$  with respect to  $q_\pi$  is an improvement,  $v_{\pi'}(s) \geq v_\pi(s)$

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

Therefore from [policy improvement theorem](#)  
 $v_{\pi'}(s) \geq v_\pi(s)$

# Monte-Carlo Policy Iteration

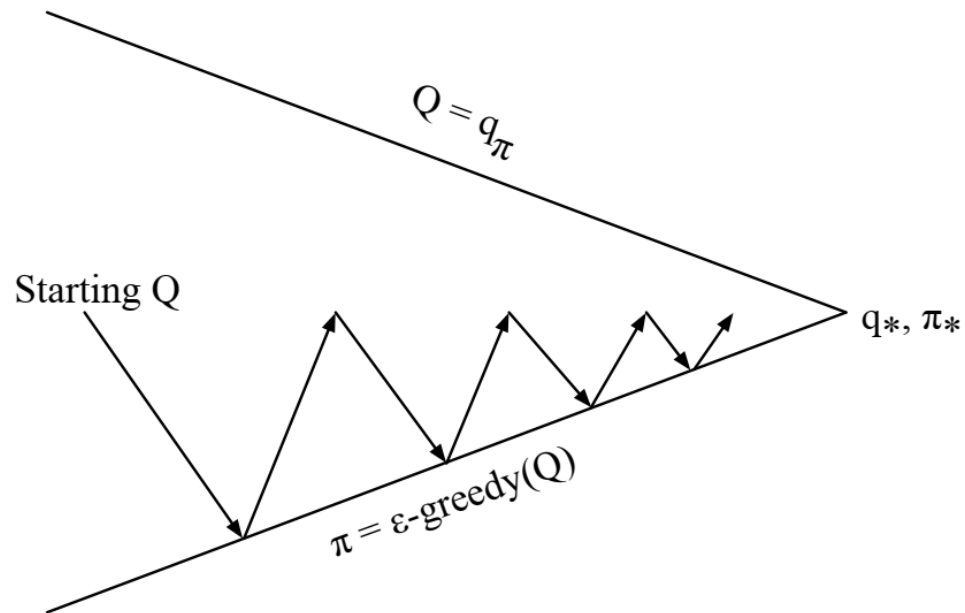
---



- ✓ Policy evaluation - Monte-Carlo policy evaluation,  $Q = q_\pi$
- ✓ Policy improvement -  $\epsilon$ -greedy policy improvement

# Monte-Carlo Control

---



## Every Episode

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

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

The policy converges on a greedy policy

$$\lim_{k \rightarrow \infty} \pi_k(a|s) = \mathbf{1}(a; \arg \max_{a' \in \mathcal{A}} Q_k(s, a'))$$

$\epsilon$ -greedy is GLIE if  $\epsilon$  reduces to zero at  $\epsilon_k = \frac{1}{k}$

# GLIE Monte Carlo Control

---

✓ Sample  $k$ -th episode using  $\pi$ :  $\{S_1, A_1, R_2, \dots, S_T\} \sim \pi$

✓ 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

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

## Theorem

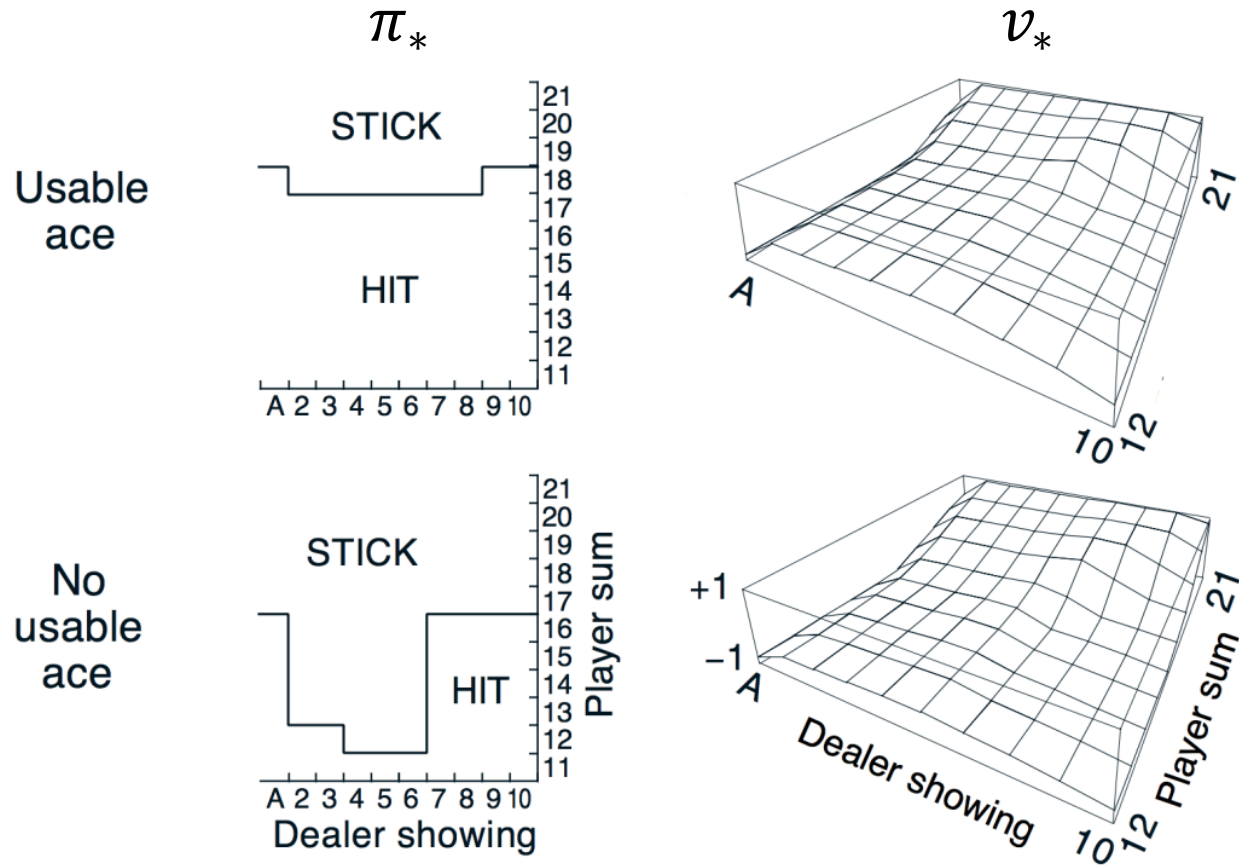
GLIE Monte-Carlo control converges to the optimal action-value function  $Q(s, a) \rightarrow q_*(s, a)$



# Blackjack Example

---

# Blackjack – MC Control



# On-Policy TD Control

---

# MC Vs TD Control

---

- ✓ TD learning has several advantages over MC
  - ✓ Lower variance
  - ✓ Online
  - ✓ Incomplete sequences
- ✓ Straightforward intuition - Use TD instead of MC in our control loop
  - ✓ Apply TD to  $Q(s, a)$
  - ✓ Use  $\epsilon$ -greedy policy improvement
  - ✓ Update every time-step

# Updating Action-Value Functions with SARSA

●  $(S, A)$

$R$

○  $S'$

●  $A'$

Expected SARSA

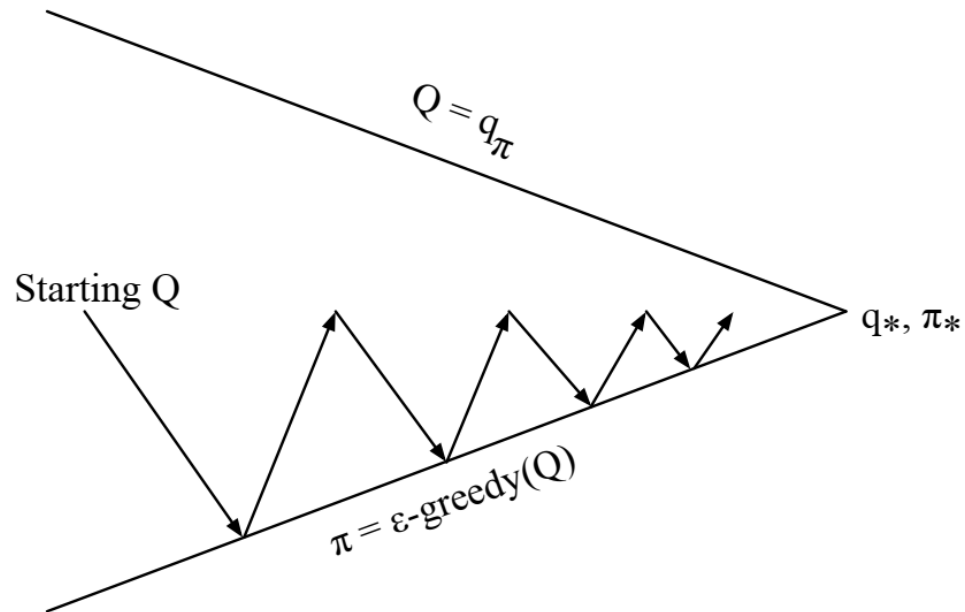
$$q_*(s, a) = \mathcal{R}_s^a + \gamma \sum_{a' \in \mathcal{A}} \pi(a'|s') q_*(s', a')$$

We sample also future action  $A'$   
(instead of leveraging policy to compute expectation)

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

# On-Policy Control with SARSA

---



Every **time-step**

✓ Policy evaluation - **SARSA**,  $Q \approx q_\pi$

✓ Policy improvement -  $\epsilon$ -greedy policy improvement



# SARSA Algorithm for On-Policy Control

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

Loop for each episode:

Initialize  $S$

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

Loop for each step of episode:

Take action  $A$ , observe  $R, S'$

Choose  $A'$  from  $S'$  using policy derived from  $Q$  (e.g.,  $\varepsilon$ -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

# Convergence of SARSA

## Theorem

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

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

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

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

# Time for TD Demo

---

[https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld\\_td.html](https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_td.html)

# SARSA( $\lambda$ )

---

# $n$ -step SARSA

---

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

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

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

...

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

- ✓ Define the  $n$ -step Q-return

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

- ✓  $n$ -step SARSA updates  $Q(S, A)$  towards the  $n$ -step Q-return

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

1-step Sarsa  
aka Sarsa(0)



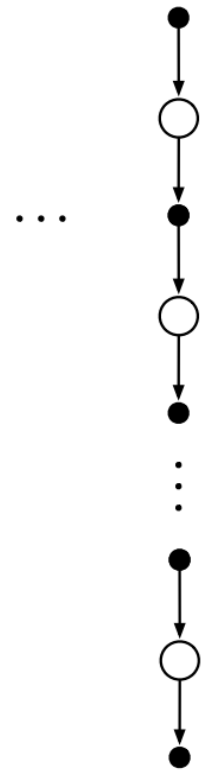
2-step Sarsa



3-step Sarsa



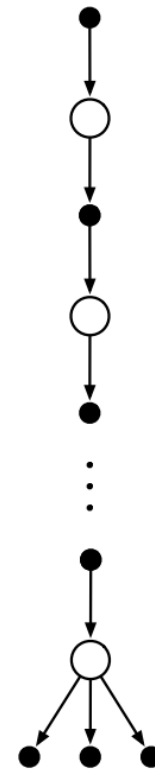
n-step Sarsa



$\infty$ -step Sarsa  
aka Monte Carlo



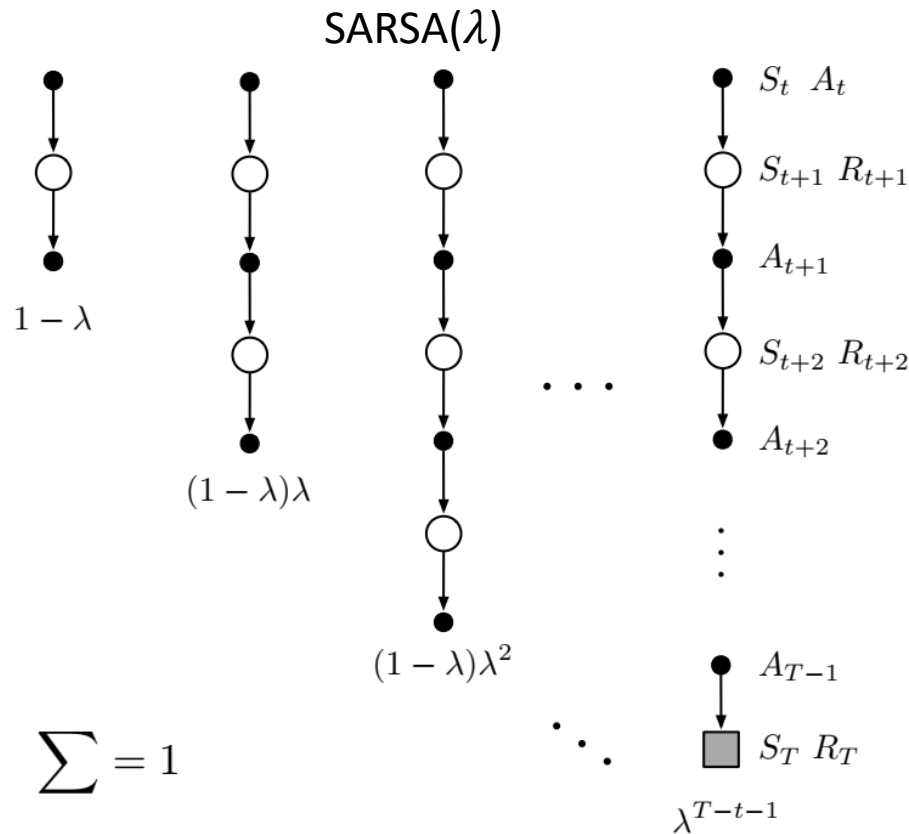
n-step  
Expected Sarsa



# SARSA backups

---

# SARSA( $\lambda$ ) - Forward View



✓ The  $q^\lambda$  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 SARSA update

$$Q(S, A) \leftarrow Q(S, A) + \alpha (q_t^\lambda - Q(S, A))$$

# SARSA( $\lambda$ ) - Backward View

---

- ✓ The return of eligibility traces
- ✓ SARSA( $\lambda$ ) needs **one eligibility trace for each state-action pair**

$$E_0(s, a) = 0$$

$$E_t(s, a) = \gamma\lambda E_{t-1}(s, a) + \mathbf{1}(S_t, A_t; s, a)$$

- ✓  $Q(s, a)$  is updated for every state  $s$  and action  $a$  in proportion to TD-error  $\delta_t$  and eligibility trace  $E_t(s, a)$

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

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



# SARSA( $\lambda$ ) Algorithm

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

Repeat (for each episode):

$E(s, a) = 0$ , for all  $s \in \mathcal{S}, a \in \mathcal{A}(s)$

Initialize  $S, A$

Repeat (for each step of episode):

Take action  $A$ , observe  $R, S'$

Choose  $A'$  from  $S'$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)

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

$E(S, A) \leftarrow E(S, A) + \delta$

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

$Q(s, a) \leftarrow Q(s, a) + \alpha \delta E(s, a)$

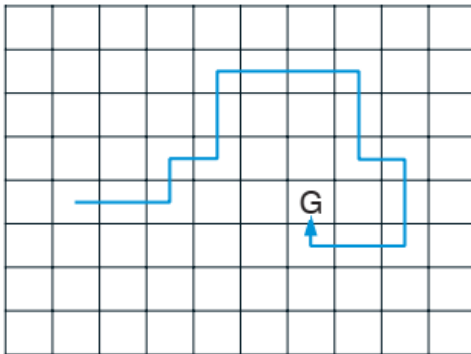
$E(s, a) \leftarrow \gamma \lambda E(s, a)$

$S \leftarrow S'; A \leftarrow A'$

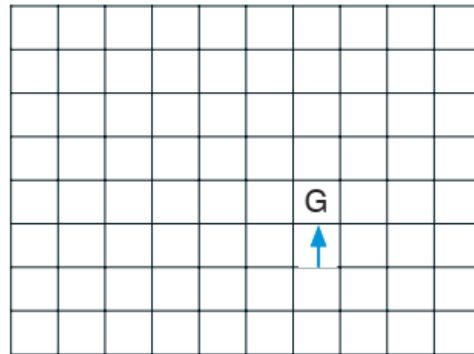
until  $S$  is terminal

# SARSA( $\lambda$ ) on Gridworld

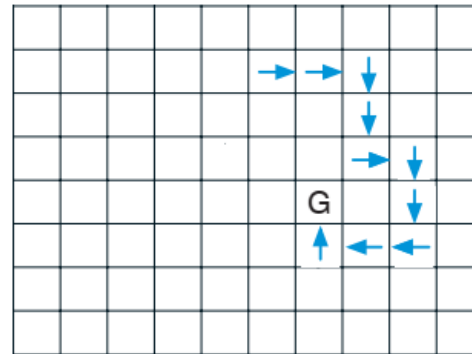
Path taken



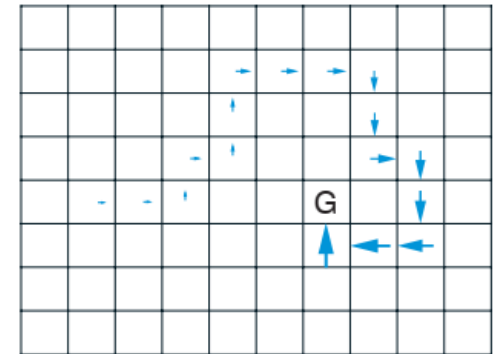
Action values increased by one-step Sarsa



Action values increased by 10-step Sarsa



Action values increased by Sarsa( $\lambda$ ) with  $\lambda=0.9$



# Off-policy TD Learning

---

# Off-Policy Learning

---

- ✓ Evaluate **target policy**  $\pi(a|s)$  to compute  $v_\pi(s)$  or  $q_\pi(s, a)$
- ✓ While **following behaviour policy**  $\mu(a|s)$   
 $\{S_1, A_1, R_2, \dots, S_T\} \sim \mu$
- ✓ Why is this important?
  - ✓ Learn from **imitation** (humans, other agents,...)
  - ✓ Re-use experience generated from **old policies**  $\pi_1, \pi_2, \dots, \pi_{t-1}$
  - ✓ Learn about **optimal policy** while following **exploratory policy**
  - ✓ Learn about **multiple policies** while following **one policy**

# (Repetita?) Importance Sampling

Estimate the expectation leveraging an external (importance) distribution

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

Draw samples from importance distribution  $Q(X)$  rather than from  $P(X)$

Assign weights such that the empirical expectation (on  $Q(X)$  samples) matches the expectation under  $P(X)$

# Importance Sampling for Off-Policy Monte Carlo

---

- ✓ Use returns generated from  $\mu$  to evaluate  $\pi$
- ✓ 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 corrected return

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

- ✓ Importance sampling can dramatically increase variance

# Importance Sampling for Off-Policy TD

---

- ✓ Use TD targets generated from  $\mu$  to evaluate  $\pi$
- ✓ Weight TD targets  $R + \gamma V(S')$  by importance sampling
- ✓ Only need a single importance sampling correction

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

- ✓ Much lower variance than MC
- ✓ Policies only need to be similar over a single step

# Q-Learning

---

Off-policy learning of action-values  $Q(s, a)$

- ✓ No importance sampling is required
- ✓ Next action is chosen using behaviour policy  $A_{t+1} \sim \mu(\cdot | S_t)$
- ✓ But we consider alternative successor action  $A' \sim \pi(\cdot | S_t)$
- ✓ And 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 by Q-Learning

---

✓ Allow both **behaviour and target policies to improve**

✓ The target policy  $\pi$  is **greedy** w.r.t.  $Q(S_t, A_t)$

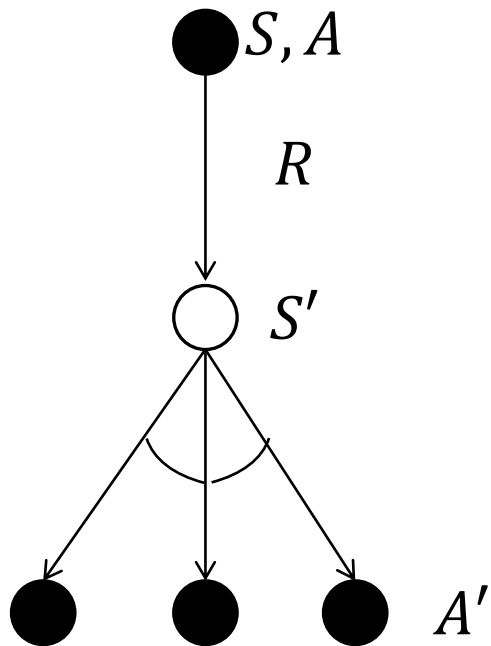
$$\pi(S_{t+1}) = \arg \max_{a'} Q(S_{t+1}, a')$$

✓ The behaviour policy  $\mu$  is  **$\epsilon$ -greedy** w.r.t.  $Q(s, a)$

✓ The Q-learning target then simplifies to

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

# Q-Learning Control Algorithm



## Theorem

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

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

# Q-Learning Algorithm for Off-policy Control

---

Initialize  $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ , 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.,  $\epsilon$ -greedy)

Take action  $A$ , observe  $R, S'$

$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$

$S \leftarrow S'$ ;

until  $S$  is terminal

# Q-learning & Exploration Demo

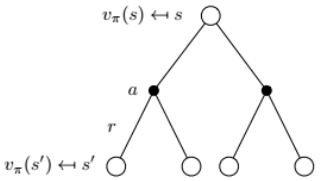

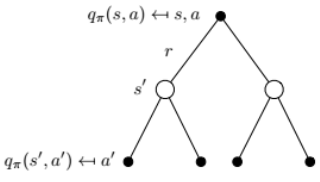
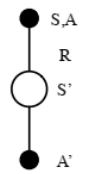
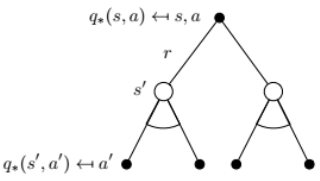
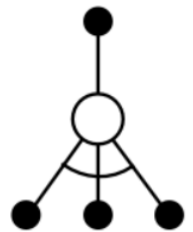
---

<https://www.aslanides.io/aixijs/demo.html>

# Wrap-up

---

# Dynamic Programming Vs Temporal Difference Learning

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

# Take (stay) home messages

---

- ✓ Model-Free control leverages **action-value function**
  - ✓ Greedy policy improvement does not need MDP
  - ✓ Generalized policy iteration
- ✓ Need to maintain sufficient **exploration** ( $\epsilon$ -greedy)
- ✓ Off-policy control
  - ✓ Learning value function of a target policy from data generated by a different behaviour policy
  - ✓ Importance sampling to match the expectations of two policies
- ✓ TD control
  - ✓ On-policy: SARSA( $\lambda$ )
  - ✓ Off-policy: Q-learning

# Next Lecture

---

## Value-function approximation

- ✓ Leave aside tabular environments
- ✓ Estimate value function with function approximation
- ✓ Linear models & neural networks
- ✓ MC & TD with Stochastic Gradient
- ✓ Experience replay buffers