Reinforcement Learning - Lecture 5

Profa. Dra. Esther Luna Colombini
esther@ic.unicamp.br

Prof. Dr. Alexandre Simoes
alexandre.simoes@unesp.br

LaRoCS – Laboratory of Robotics and Cognitive Systems

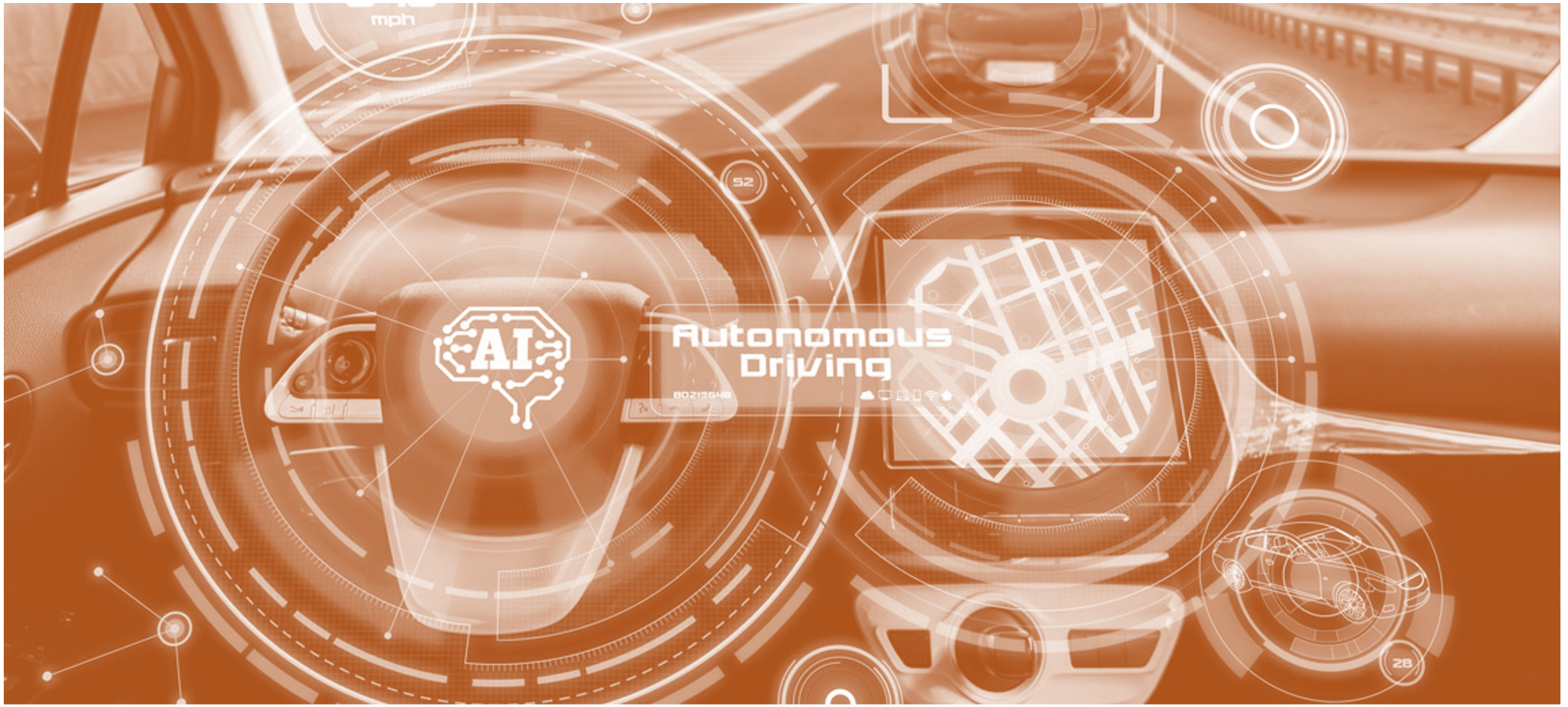
Summary

- Model-free Control
  - On-policy
    - MC Control
    - SARSA
    - SARSA(λ)
  - Off-policy
    - Q-learning
    - Double Q-learning

Disclaimer

- These slides were built upon David Silver's Lecture notes on Reinforcement Learning

- Planning by dynamic programming
  - Solve a known MDP
- **Model-free prediction**
  - Estimate the value function of an unknown MDP
- **Model-free control**
  - Optimize the value function of an unknown MDP



Model-free control

Model-free control

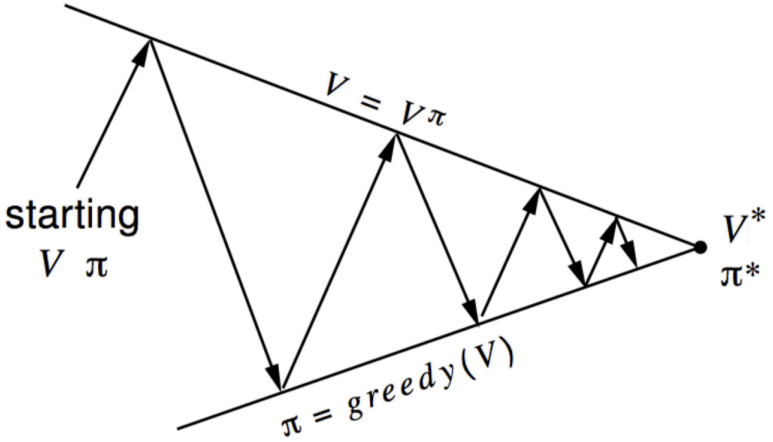
- Some example problems that can be modelled as MDPs
  - Elevator
  - Parallel Parking
  - Ship Steering
  - Bioreactor
  - Helicopter
  - Aeroplane
  - Logistics
  - RoboCup Soccer
  - Portfolio management
  - Protein Folding (Deepmind just released incredible results)
  - Robot walking
  - Game of 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

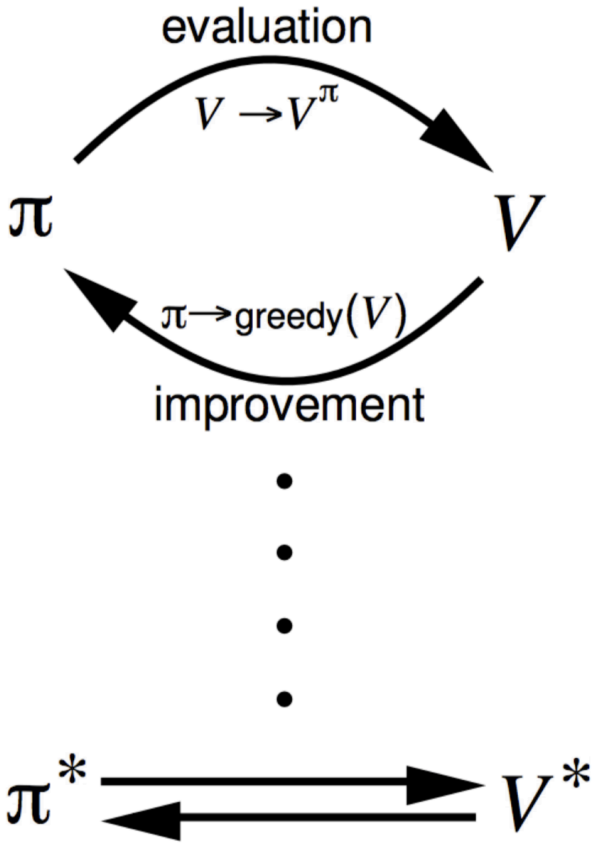
Model-free control

- 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

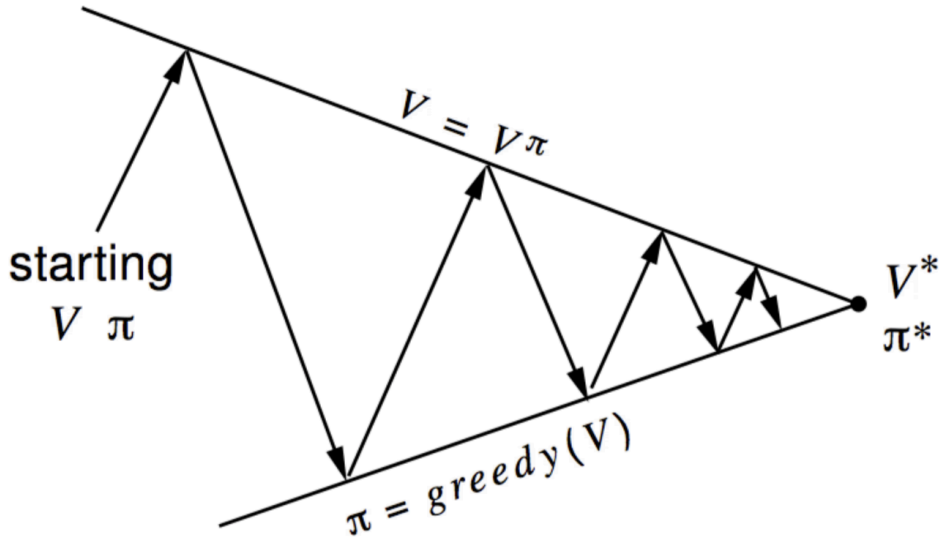
Generalized Policy Iteration



- **Generalized Policy Iteration**
  - Policy evaluation
    - Estimate v_π
      - **Any** policy evaluation algorithm
  - Policy improvement
    - Generate $\pi' \geq \pi$
      - **Any** policy improvement



Generalized Policy Iteration With Monte-Carlo Evaluation



- Policy evaluation
  - Monte-Carlo policy evaluation, $V = v_\pi$?
    - How can we use V if you are model-free?
- Policy improvement
  - Greedy policy improvement?
    - Problem regarding explroation

Model-Free Policy Iteration Using Action-Value Function

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

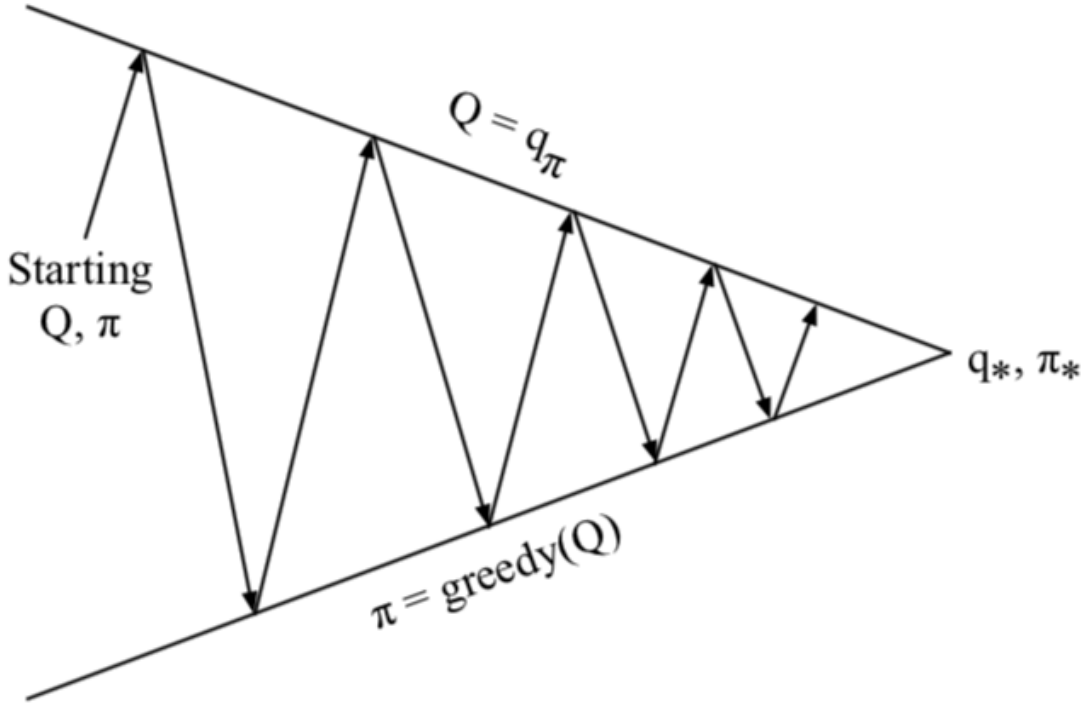
$$\pi'(s) = \underset{a \in A}{\operatorname{argmax}}\, R_s^a + P_{ss'}^a V(s')$$

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

$$\pi'(s) = \underset{a \in A}{\operatorname{argmax}}\, Q(s, a)$$

- Action-value functions enable us to run model-free control

Generalized Policy Iteration With Action-Value Function



- Policy evaluation
  - Monte-Carlo policy evaluation, $Q = q_\pi$
- Policy improvement
  - Greedy policy improvement? Does it work?

Example of Greedy Action Selection

- Bandit problem
  - Two choices
  - There are two doors in front of you.
- You open the left door and get reward 0
  - *V(left) = 0*
- You open the right door and get reward +1
  - *V(right) = +1*
- You open the right door and get reward +3
  - *V(right) = +2 (mean using MC)*
- You open the right door and get reward +2
  - *V(right) = +2 (mean using MC)*
- Are you sure you've chosen the best door?

ϵ-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 ϵ choose an action **at random**

$$\pi(a|s) = \begin{cases} \frac{\epsilon}{m} + 1 - \epsilon & \text{if } a^* = \underset{a \in A}{\operatorname{argmax}}\, Q(s,a) \\ \frac{\epsilon}{m} & \text{otherwise} \end{cases}$$

ϵ-Greedy Policy Improvement

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

$$q_\pi\left(s, \pi'(s)\right) = \sum_{a \in A} \pi'(a|s)\, q_\pi(s,a)$$

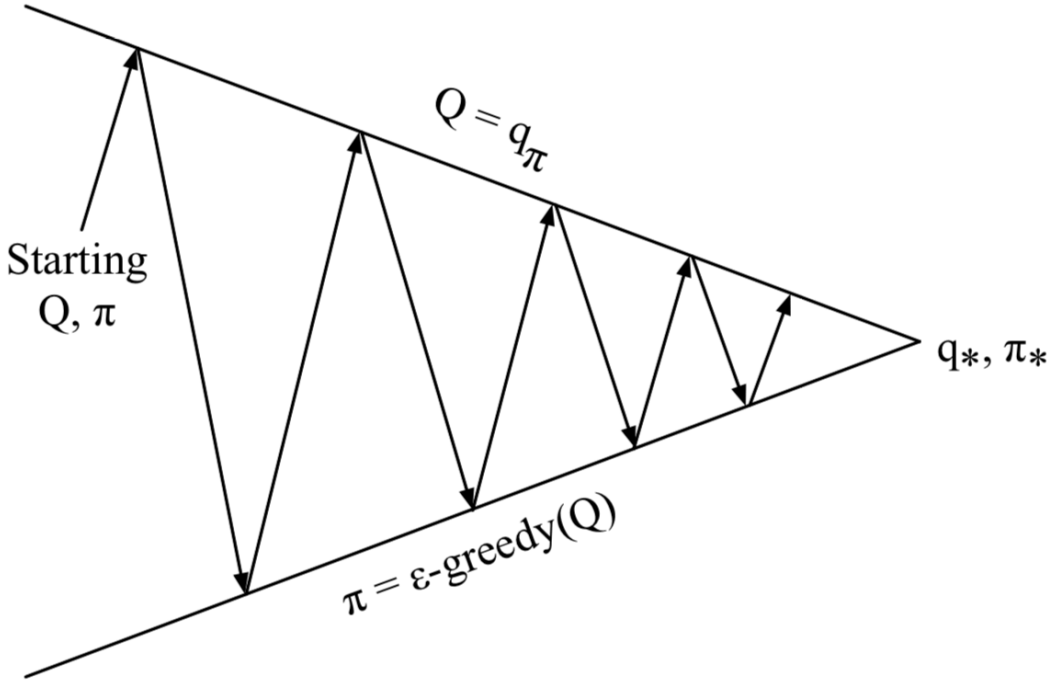
$$= \epsilon/m \sum_{a \in A} q_\pi(s,a) + (1-\epsilon) \max_{a \in A} q_\pi(s,a)$$

$$\geq \epsilon/m \sum_{a \in A} q_\pi(s,a) + (1-\epsilon) \sum_{a \in A} \frac{\pi(s,a) + \left(\frac{\epsilon}{m}\right)}{1-\epsilon} q_\pi(s,a)$$

$$= \sum_{a \in A} \pi(a|s)\, q_\pi(s,a) = v_\pi(s)$$

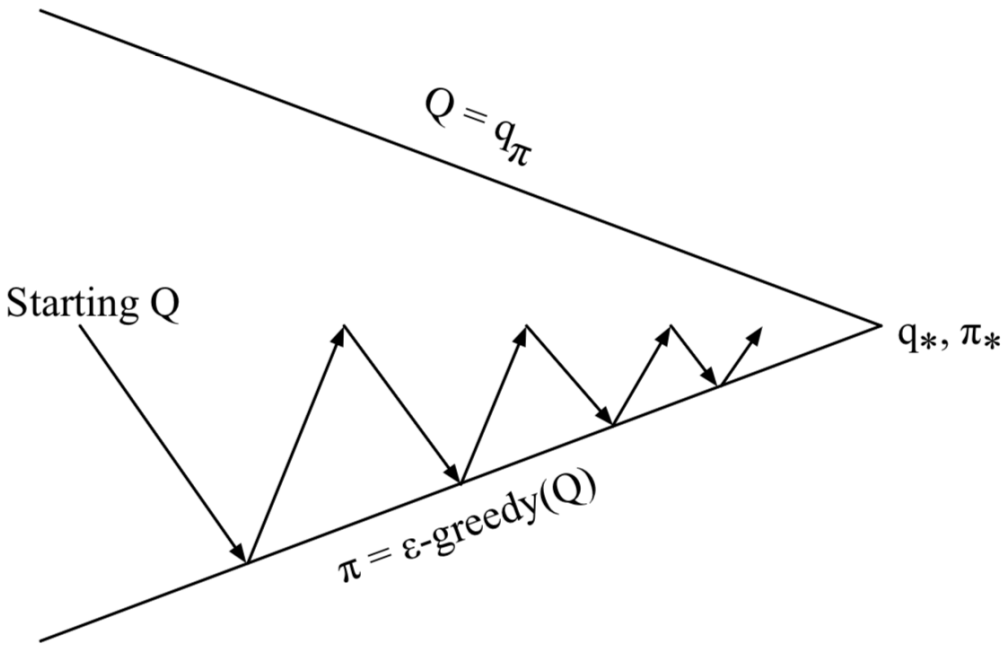
  - Therefore from policy improvement theorem, $v_{\pi'}(s) \geq v_\pi(s)$

Monte-Carlo Policy Iteration



- Policy evaluation
  - Monte-Carlo policy evaluation, $\boldsymbol{Q = q_\pi}$
- Policy improvement
  - ϵ -greedy policy improvement

Monte-Carlo Control



- Every episode:
  - Policy evaluation
    - Monte-Carlo policy evaluation, $\boldsymbol{Q \approx q_\pi}$
  - Policy improvement
    - ϵ -greedy policy improvement

GLIE - The Exploration/Exploitation Trade-off

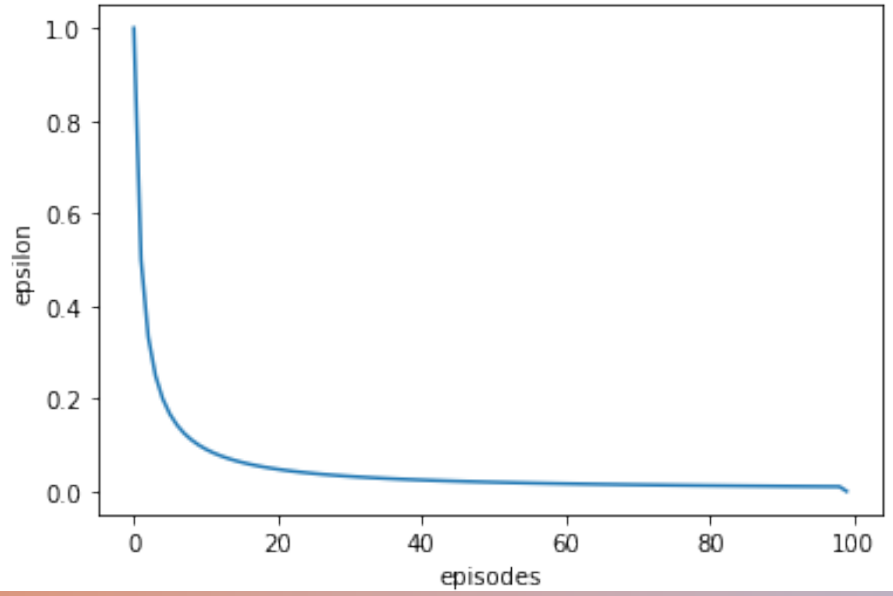
- Greedy in the Limit with Infinite Exploration (GLIE)
  - All state-action pairs are explored infinitely many times,

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

  - The policy converges on a greedy policy,

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

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



GLIE Monte-Carlo Control

- **Evaluate policy** sampling the *kth* episode using

$$\pi: \{S_1, A_1, R_2, \ldots, S_T\} \sim \pi$$

- 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

$$\epsilon_k \leftarrow 1/k$$

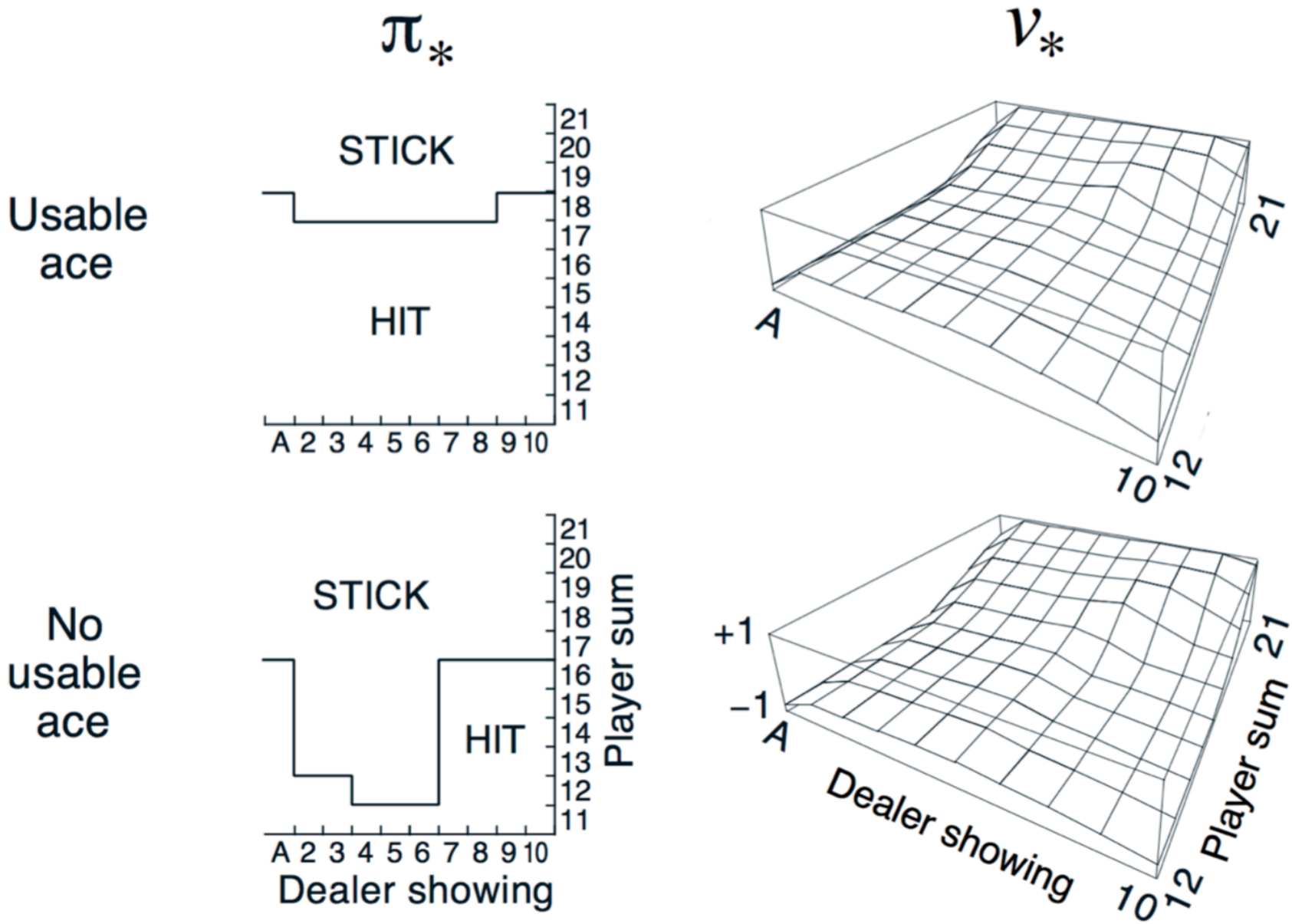
$$\pi \leftarrow \epsilon \text{-greedy(Q)}$$

- **Theorem**

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

Blackjack

Monte-Carlo Control in Blackjack



Monte-Carlo On-Policy Improvement

Algorithm: Monte-Carlo On-Policy Improvement

1: *Initialize* $Q(s,a) = 0, N(s,a) = 0\ \forall(s,a)$, *Set* $\epsilon = 1, k = 1$

2: $\pi_k = \epsilon$-*greedy*(Q) // *Create initial* ϵ-*greedy policy*

3: ***repeat (for each episode)***

4: Sample k-th episode $(s_{k,1}, a_{k,1}, r_{k,1}, s_{k,2}, \ldots, s_{k,T})$ given π_k

5: $G_{k,t} = R_{k,t} + \gamma R_{k,t+1} + \cdots + \gamma^{T-1} R_{k,T}$

6: **for** $(t=1,\ldots, T)$ **do**

7: **if** first visit to (s,a) in episode k then

8: $N(s, a) = N(s, a) + 1$

9: $Q(s_t,a_t)=Q(s_t,a_t)+ 1/N(s, a)\ (G_{k,t} - Q(s_t,a_t))$

10: $k=k+1, \epsilon = 1/k$

11: $\pi_k = \epsilon$-*greedy*(Q) // Policy improvement

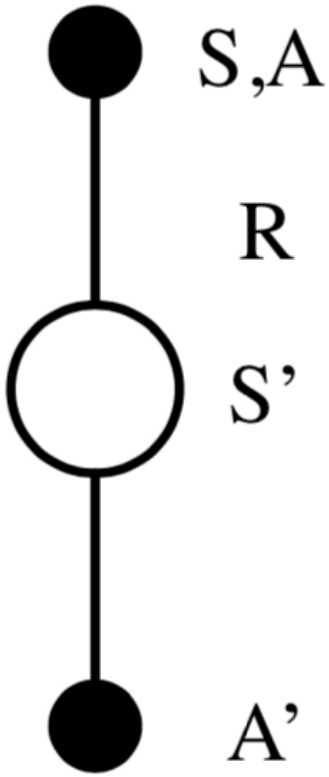
12: **until last episode**

MC vs. TD Control

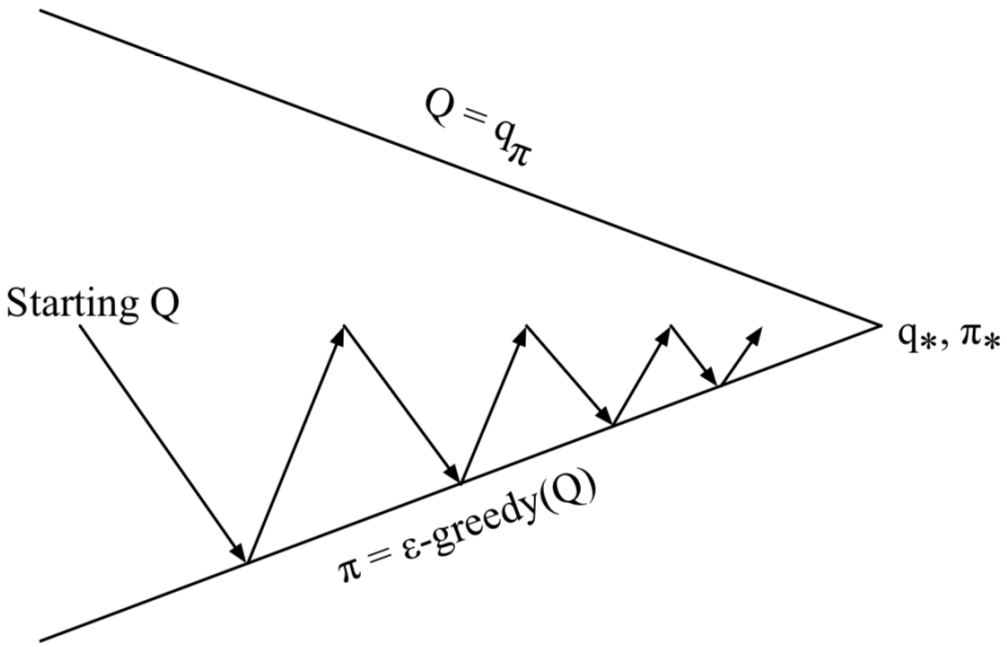
- Temporal-difference (TD) learning has several advantages over Monte-Carlo (MC)
  - Lower variance
  - Online
  - Incomplete sequences
- Natural idea: use TD instead of MC in our control loop
  - Apply TD to $Q(s,a)$
  - Use ϵ-*greedy* policy improvement
  - Update every time-step

Updating Action-Value Functions with Sarsa

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



On-Policy Control With Sarsa



- Every time-step:
  - Policy evaluation
    - Sarsa, $\boldsymbol{Q \approx q_\pi}$
  - Policy improvement
    - ϵ -greedy policy improvement

SARSA Algorithm

Algorithm: SARSA (on-policy TD control)

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$

Initialize $Q(s,a)$, for all $s \in S^+$, $a \in A(s)$, arbitrarily except that $Q(terminal, \cdot)=0$

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

On-policy term

Convergence of Sarsa

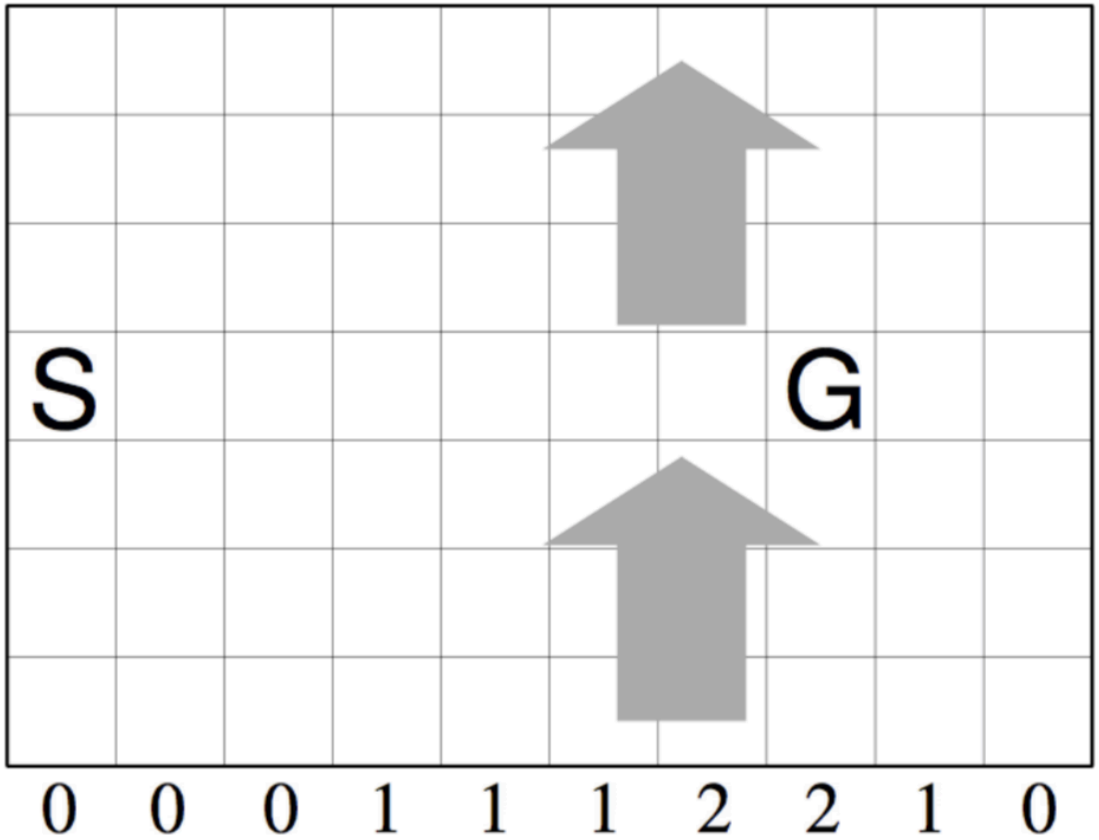
- **Theorem**
  - Sarsa converges to the optimal action-value function, $Q \quad (s,a) \rightarrow q_* \ (s,a)$, under the following conditions:
    - GLIE sequence of policies $\boldsymbol{\pi_t}(a|t)$
    - Robbins-Monro sequence of step-sizes $\boldsymbol{\alpha_t}$

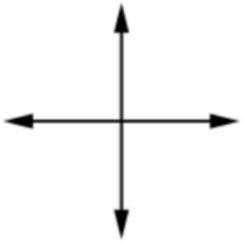
$$\sum_{t=1}^{\infty} \alpha_t = \infty$$

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

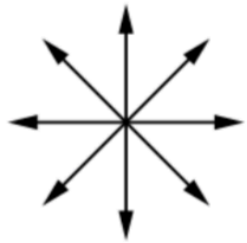
      - Step-size is sufficient large to get you to move your Q-value whatever you want
      - The changes on your Q-values will get smaller with time

Windy Gridworld Example





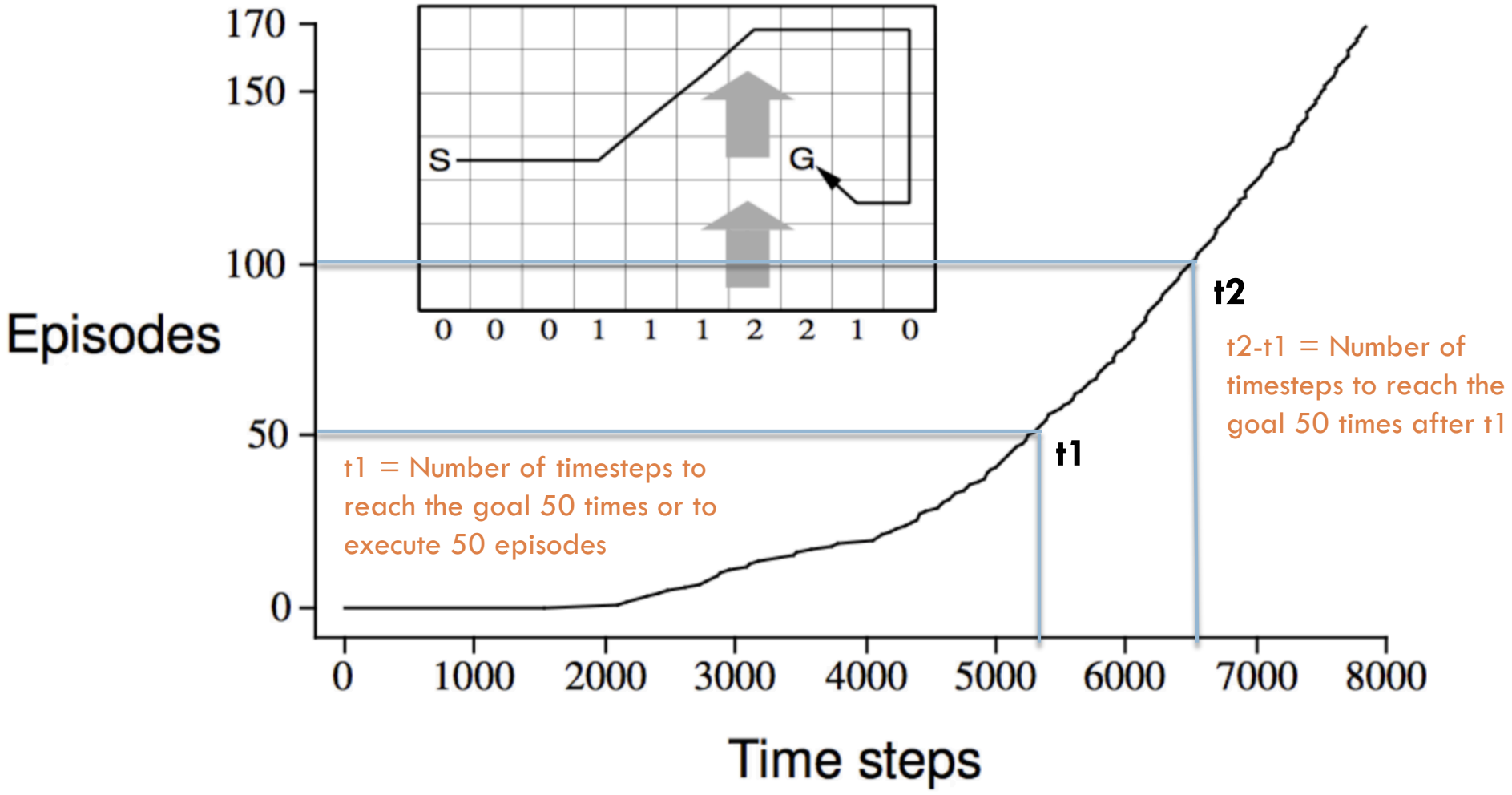
standard moves



king's moves

- Reward = -1 per time-step until reaching goal
- Undiscounted
- What do you think is the optimal policy?

Sarsa on the Windy Gridworld



n-Step Sarsa

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

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

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

$\vdots$ $\vdots$

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

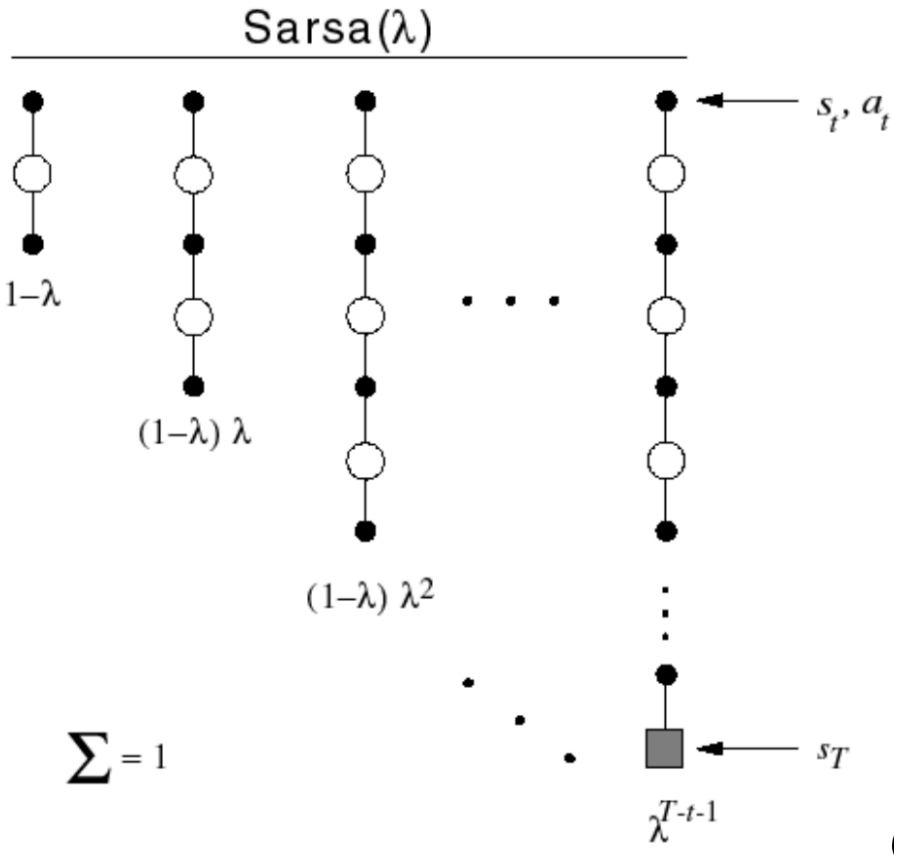
- Define the n-step return

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

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

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

Forward-view Sarsa(λ)



- The $q^{(\lambda)}$ combines all n-step returns 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)

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

Backward-view Sarsa(λ)

- Just like TD(λ), we use **eligibility traces** gives an online algorithm
- But Sarsa(λ) has one eligibility trace for each state-action pair

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

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

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

$$\delta_t = \mathrm{R}_{t+1} + \gamma Q(S_{t+1}, \mathrm{A}_{t+1}) - Q(S_t, \mathrm{A}_t)$$

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

SARSA(λ) Algorithm

Algorithm: SARSA(λ)

1: *Initialize* $Q(s,a)$, for all $s \in S^+$, $a \in A(s)$, arbitrarily except that $Q(terminal, \cdot)=0$

2: ***repeat (for each episode)***

3: $E(S, A)=0$, *for all* $s \in S$, $a \in A(s)$

4: Initialize S, A

5: **repeat** (for each step of episode)

6: Take action A, observe R, S'

7: Choose A' from S' using policy derived from Q (e.g. ϵ***-greedy)***

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

9: $E(S, A) \leftarrow E(S, A) + 1$

10: **for** all $s \in S$, $a \in A(s)$

11: $Q(S, A) \leftarrow Q(S, A) + \alpha \delta E(s, a)$

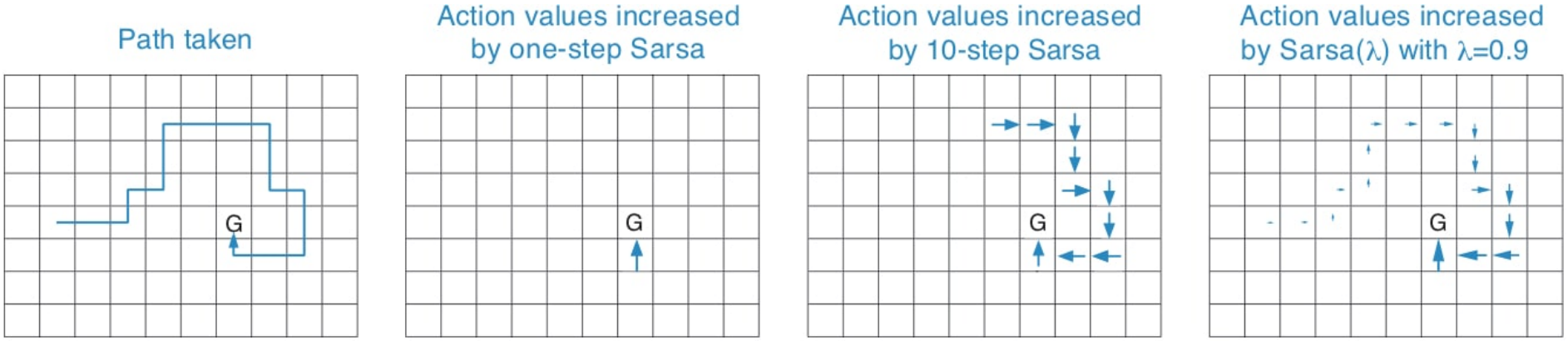
12: $E(S, A) \leftarrow \gamma \lambda E(s\ a)$

13: $S \leftarrow S'$; $A \leftarrow A'$

14: **until** S is terminal

SARSA(λ) Gridworld Example

- The fading-trace bootstrapping strategy of Sarsa(λ) increases the learning efficiency, which is shown in the gridworld example below (consider all Q initialized as 0)
- All rewards were zero except a positive at Goal State
- One-step would increment only the last action value, whereas n-step method would equally increment the last n actions' values, and an eligibility trace method would update all the action values up to the beginning of the episode, to different degrees, fading with recency.



Off-Policy Learning

- Evaluate target policy $\pi(a|s)$ to compute $v_\pi(s)$ or $q_\pi(s,a)$
  - While following **behaviour policy** $\boldsymbol{\mu(a|s)}$

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

- Why is this important?
  - Learn from observing humans or other agents
  - Re-use experience generated from old policies $\pi_1, \pi_2, \pi_3, \ldots, \pi_{t-1}$
    - Batch data (replay memory)
  - Learn about **optimal policy (usually deterministic)** while following **exploratory policy (stochastic)**
  - Learn about *multiple policies* while following one policy

- Estimate the expectation of a different distribution

$$\mathbb{E}_{X\sim P}[f(X)] = \sum P(X)f(X)$$

$$= \sum Q(X)\frac{P(X)}{Q(X)}f(X)$$

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

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 whole episode

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

- Update value towards corrected return

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

- Cannot use if μ is zero when π is non-zero
- Importance sampling can **dramatically increase variance**
  - **Not feasible in practice**

Importance Sampling for Off-Policy TD

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

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

- Importance sampling can dramatically increase variance
- Much lower variance than Monte-Carlo importance sampling
- Policies only need to be similar over a single step

Q-Learning

- We now consider off-policy learning of action-values $Q(s, a)$
- **No** importance sampling is required
- Next action is chosen using behavior policy $A_{t+1} \sim \mu(\cdot \mid S_t)$
- But we consider alternative successor action $A' \sim \pi(\cdot \mid S_t)$
- And update $Q(S_t, A_t)$ towards value of alternative action

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

The policy I care about
We bootstarp from it

Off-Policy Control with Q-Learning

- We now allow both behavior and target policies to **improve**
- The target policy π is **greedy** w.r.t. $Q(s,a)$

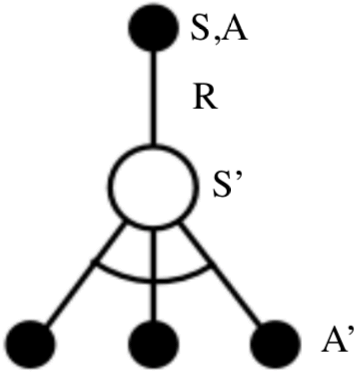
$$\pi(S_{t+1}) = \underset{a'}{\operatorname{argmax}}\, Q(S_{t+1}, a')$$

- The behavior policy μ is e.g. ***ϵ-greedy*** w.r.t. $Q(s,a)$
- The Q-learning target then simplifies:

$$R_{t+1} + \gamma Q(S_{t+1}, A')$$
$$= R_{t+1} + \gamma Q\left(S_{t+1}, \underset{a'}{\operatorname{argmax}}\, Q(S_{t+1}, a')\right)$$
$$= R_{t+1} + \max_{a'} \gamma Q(S_{t+1}, a')$$

Q-Learning Control Algorithm (SARSAMAX)

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



- Just like the Bellman optimality equation
- **Theorem**

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

Q-learning Algorithm

Algorithm: Q-learning (off-policy TD control)

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$

Initialize $Q(s,a)$, for all $s \in S^+$, $a \in A(s)$, arbitrarily except that $Q(terminal, \cdot)=0$

repeat (for each episode)

    Initialize S

    repeat (for each step of episode)

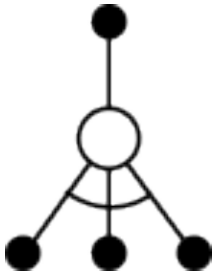
        Choose A from S using policy derived from Q (e.g. ***ϵ-greedy***)

        Take action A, observe R, S'

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

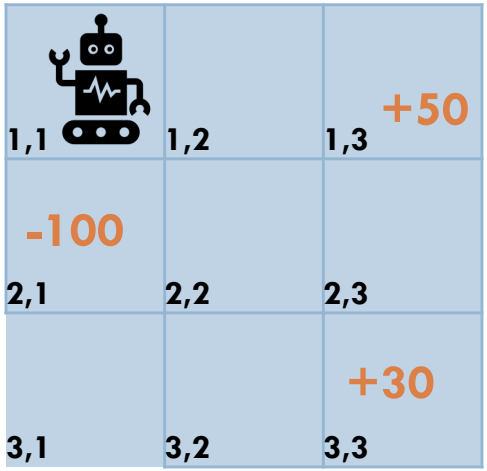
        $S \leftarrow S'$

    until S is terminal



Q-Learning Algorithm

- Consider the grid-world below and an agent who is trying to learn the optimal policy.



- The possible actions are:
  - R (right), L (left), N (north) and S (south).

Q-Learning Algorithm

- The Q table has been initialized with the following values:

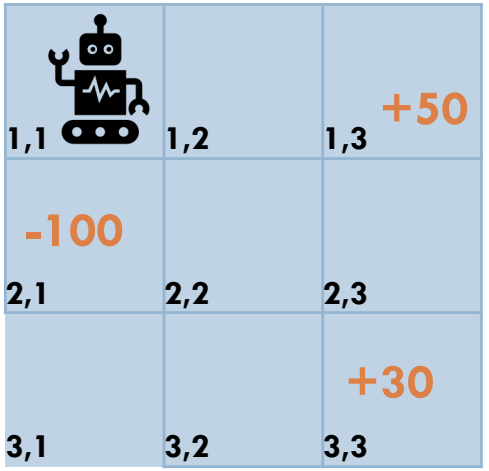
	R	L	N	S
1,1	0.4	0.2	0.2	0.3
1,2	0.4	0.2	0.1	0.2
1,3	0.2	0.1	0.2	0.3
2,1	0.2	0.1	0.5	0.1
2,2	0.3	0.4	0.1	0.2
2,3	0.2	0.5	0.1	0.2
3,1	0.4	0.2	0.4	0.2
3,2	0.1	0.2	0.3	0.2
3,3	0.2	0.2	0.3	0.1

- Reinforcements (positive and negative) will be given only in the indicated regions.
- Assume γ=1 and α=0.5 for all calculations. Consider the agent in the initial position indicated.
- Perform 5 greedy actions in sequence, performing the required updates on Table Q.

Q-Learning Algorithm

- **Action 1:**
  - Which action to perform in state 1,1?
  - According to greedy policy, the one with the highest value of Q.
    - Hence, action R.
  - Updating the value
    - $Q(s_{1,1}, a_R) = (1 - \alpha)\, Q(s_{1,1}, a_R) + \alpha(r_t + \gamma.Q(s_{1,2}, a_R))$
    - $Q(s_{1,1}, a_R) = (1 - 0.5)*(0.4) + 0.5(0 + 1*0.4) = 0.4$

	R	L	N	S
1,1	0.4	0.2	0.2	0.3
1,2	0.4	0.2	0.1	0.2
1,3	0.2	0.1	0.2	0.3
2,1	0.2	0.1	0.5	0.1
2,2	0.3	0.4	0.1	0.2
2,3	0.2	0.5	0.1	0.2
3,1	0.4	0.2	0.4	0.2
3,2	0.1	0.2	0.3	0.2
3,3	0.2	0.2	0.3	0.1



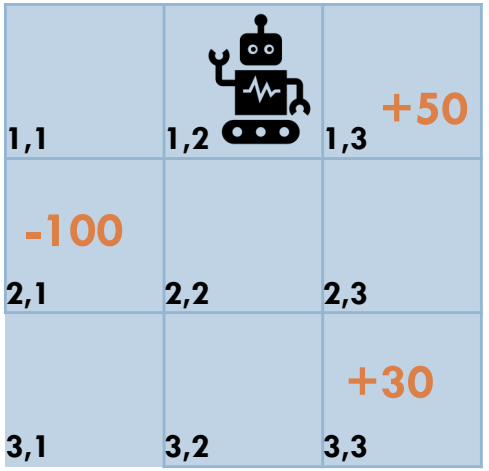
Greedy action

maxQ in s_{t+1}

Q-Learning Algorithm

- **Action 2:**
  - Which action to perform in state 1,2?
  - According to greedy policy, the one with the highest value of Q.
    - Hence, action R.
  - Updating the value
    - $Q(s_{1,2}, a_R) = (1 - \alpha)\, Q(s_{1,2}, a_R) + \alpha(r_t + \gamma Q(s_{1,3}, a_S))$
    - $Q(s_{1,2}, a_R) = (1 - 0.5)*(0.4) + 0.5(50 + 1*0.3) = 25.35$

	R	L	N	S
1,1	0.4	0.2	0.2	0.3
1,2	0.4	0.2	0.1	0.2
1,3	0.2	0.1	0.2	0.3
2,1	0.2	0.1	0.5	0.1
2,2	0.3	0.4	0.1	0.2
2,3	0.2	0.5	0.1	0.2
3,1	0.4	0.2	0.4	0.2
3,2	0.1	0.2	0.3	0.2
3,3	0.2	0.2	0.3	0.1

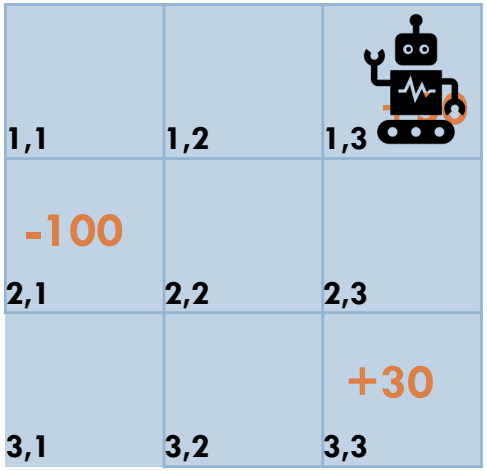


Greedy action

maxQ in s_{t+1}

- **Action 3:**
  - Which action to perform in state 1,3?
  - According to greedy policy, the one with the highest value of Q.
    - Hence, action S.
  - Updating the value
    - $Q(s_{1,3}, a_S) = (1 - \alpha)\, Q(s_{1,3}, a_S) + \alpha(r_t + \gamma Q(s_{2,3}, a_L))$
    - $Q(s_{1,3}, a_S) = (1 - 0.5)*(0.3) + 0.5(0 + 1*0.5) = 0.4$

	R	L	N	S
1,1	0.4	0.2	0.2	0.3
1,2	25.4	0.2	0.1	0.2
1,3	0.2	0.1	0.2	0.3
2,1	0.2	0.1	0.5	0.1
2,2	0.3	0.4	0.1	0.2
2,3	0.2	0.5	0.1	0.2
3,1	0.4	0.2	0.4	0.2
3,2	0.1	0.2	0.3	0.2
3,3	0.2	0.2	0.3	0.1



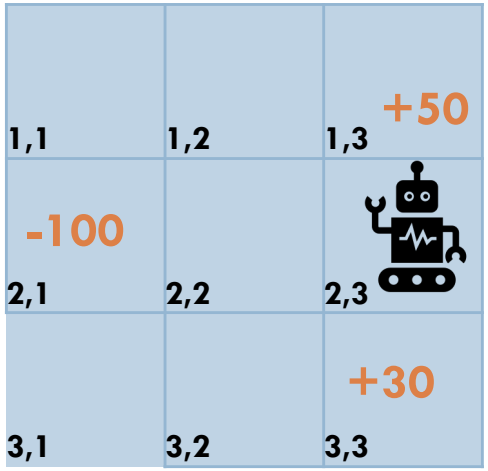
Greedy action

maxQ in s_{t+1}

Q-Learning Algorithm

- **Action 4:**
  - Which action to perform in state 2,3?
  - According to greedy policy, the one with the highest value of Q.
    - Hence, action L.
  - Updating the value
    - $Q(s_{2,3}, a_L) = (1 - \alpha)\, Q(s_{2,3}, a_L) + \alpha(r_t + \gamma Q(s_{2,2}, a_L))$
    - $Q(s_{2,3}, a_L) = (1 - 0.5)*(0.5) + 0.5(0 + 1*0.4) = 0.45$

	R	L	N	S
1,1	0.4	0.2	0.2	0.3
1,2	25.4	0.2	0.1	0.2
1,3	0.2	0.1	0.2	0.4
2,1	0.2	0.1	0.5	0.1
2,2	0.3	0.4	0.1	0.2
2,3	0.2	0.5	0.1	0.2
3,1	0.4	0.2	0.4	0.2
3,2	0.1	0.2	0.3	0.2
3,3	0.2	0.2	0.3	0.1



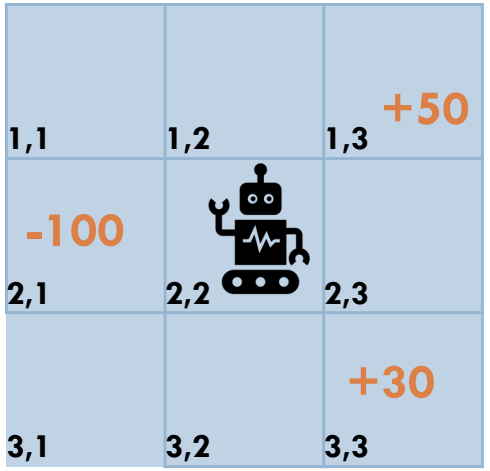
Greedy action

maxQ in s_{t+1}

Q-Learning Algorithm

- **Action 5:**
  - Which action to perform in state 2,2?
  - According to greedy policy, the one with the highest value of Q.
    - Hence, action L.
  - Updating the value
    - $Q(s_{2,2}, a_L) = (1 - \alpha)\, Q(s_{2,2}, a_L) + \alpha(r_t + \gamma Q(s_{2,1}, a_N))$
    - $Q(s_{2,2}, a_L) = (1 - 0.5)*(0.4) + 0.5(-100 + 1*0.4) = -49.55$

	R	L	N	S
1,1	0.4	0.2	0.2	0.3
1,2	25.4	0.2	0.1	0.2
1,3	0.2	0.1	0.2	0.4
2,1	0.2	0.1	0.5	0.1
2,2	0.3	0.4	0.1	0.2
2,3	0.2	0.45	0.1	0.2
3,1	0.4	0.2	0.4	0.2
3,2	0.1	0.2	0.3	0.2
3,3	0.2	0.2	0.3	0.1

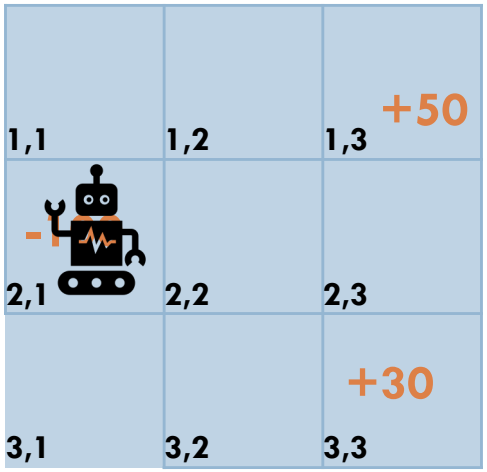


Greedy action

maxQ in s_{t+1}

Q-Learning Algorithm

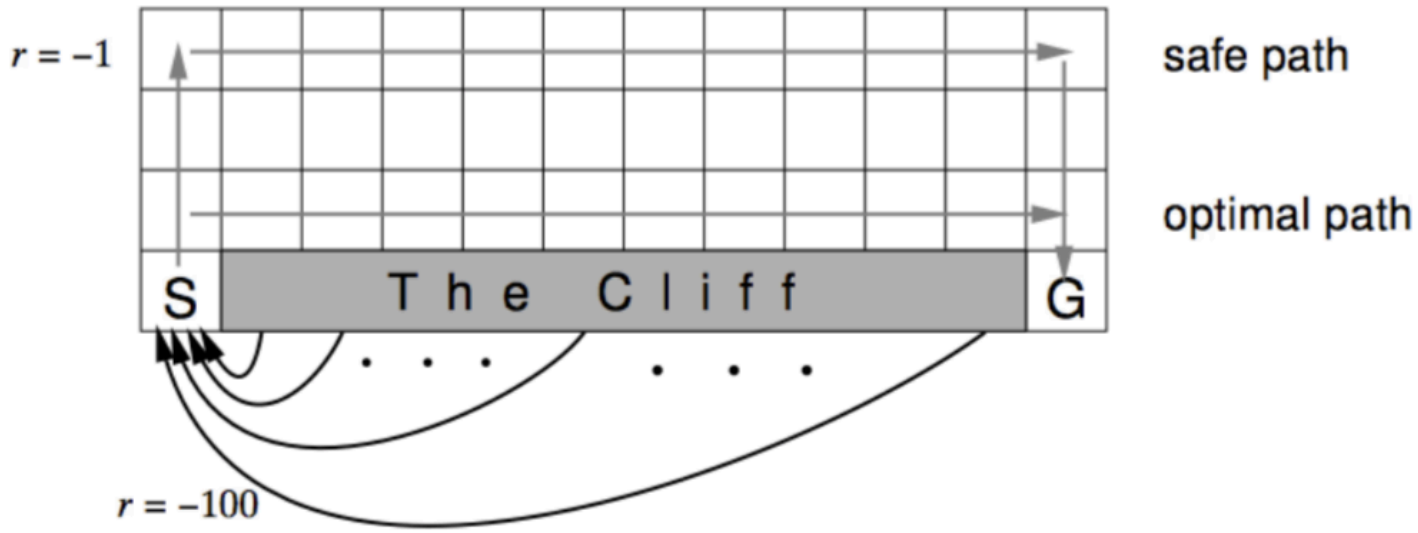
- After 5 actions

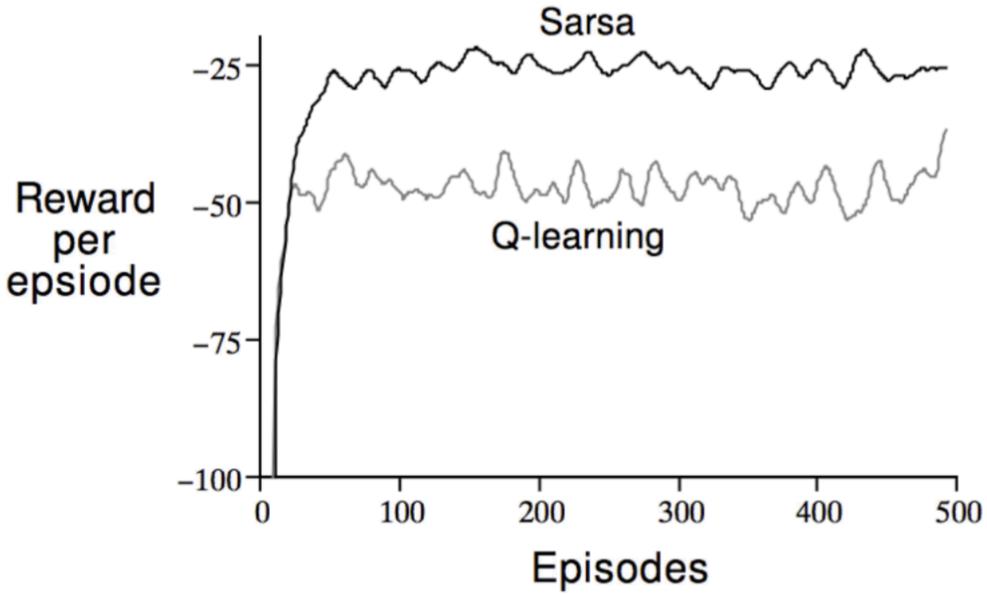


	R	L	N	S
1,1	0.4	0.2	0.2	0.3
1,2	25.4	0.2	0.1	0.2
1,3	0.2	0.1	0.2	0.4
2,1	0.2	0.1	0.5	0.1
2,2	0.3	-49.5	0.1	0.2
2,3	0.2	0.45	0.1	0.2
3,1	0.4	0.2	0.4	0.2
3,2	0.1	0.2	0.3	0.2
3,3	0.2	0.2	0.3	0.1

- Table Q has been updated for the 5 actions performed in the respective states
- What did the agent learn?
  - Going to the **right** when it is **1,2** is good because it will receive a high reinforcement
  - Going to the **left** when it is **2,2** is bad because it will receive a negative reinforcement
- This information is now embedded in the Q table
  - The value Q of action R in state 1,2 is high. Hence, it tends to be chosen
  - The Q value of the L action in the 2,2 state is low. It will be avoided by a greedy policy.

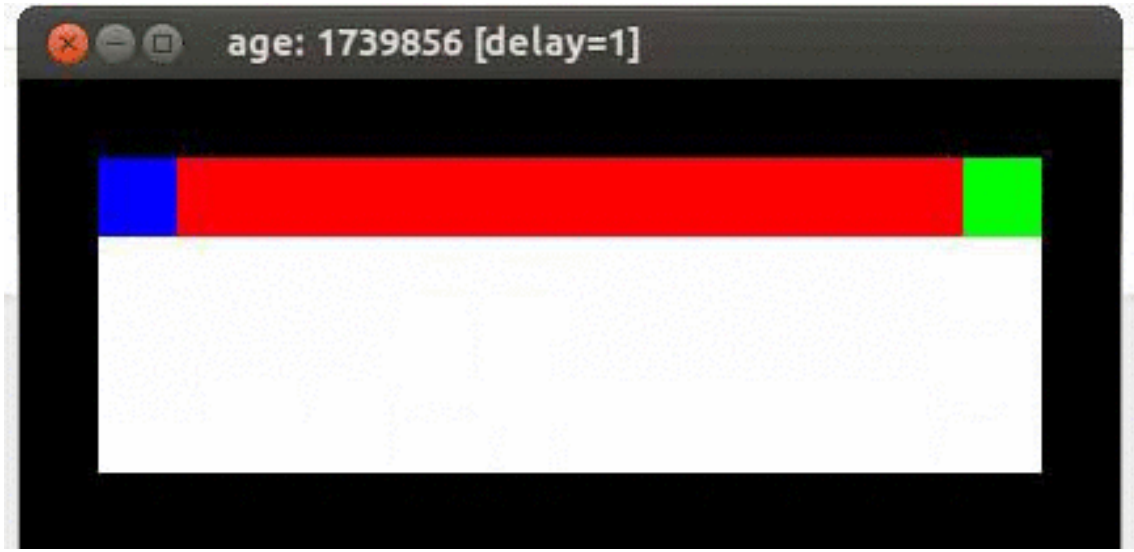
Cliff Walking Example





Cliff Walking Example

- Which of the following is Q-learning and which is Sarsa?

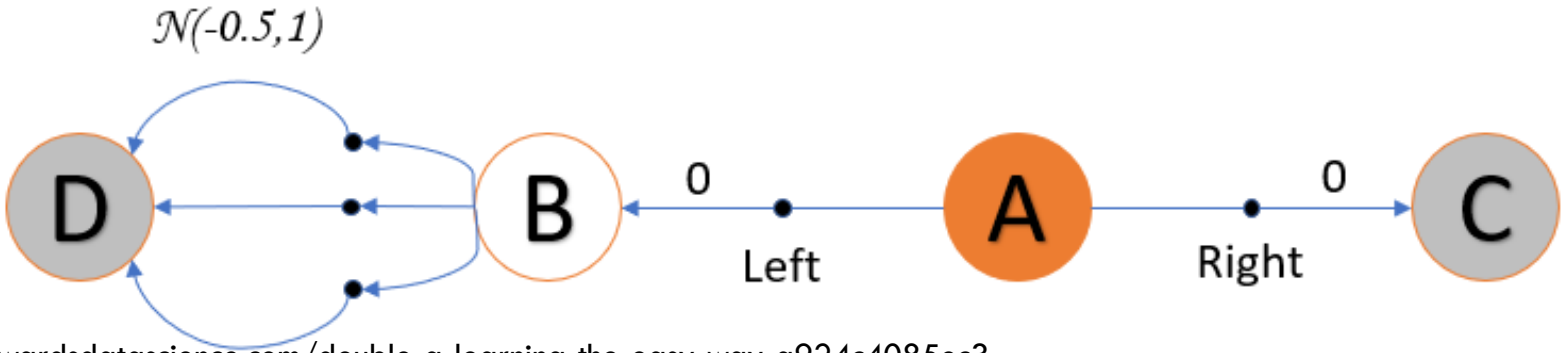


Double Q-learning

- **Proposed by** Hado V. Hasselt in NeurIPS 2010
  - https://paperswithcode.com/paper/double-q-learning
- **Double Q-learning** is an off-policy reinforcement learning algorithm that uses **double estimation** to counteract **overestimation** problems with traditional Q-learning
  - Q-Learning performs very poorly in some stochastic environments
  - The author pointed out that the poor performance is caused by large overestimation of action values due to the use of $\max Q(s', a)$ in Q-learning

Double Q-learning

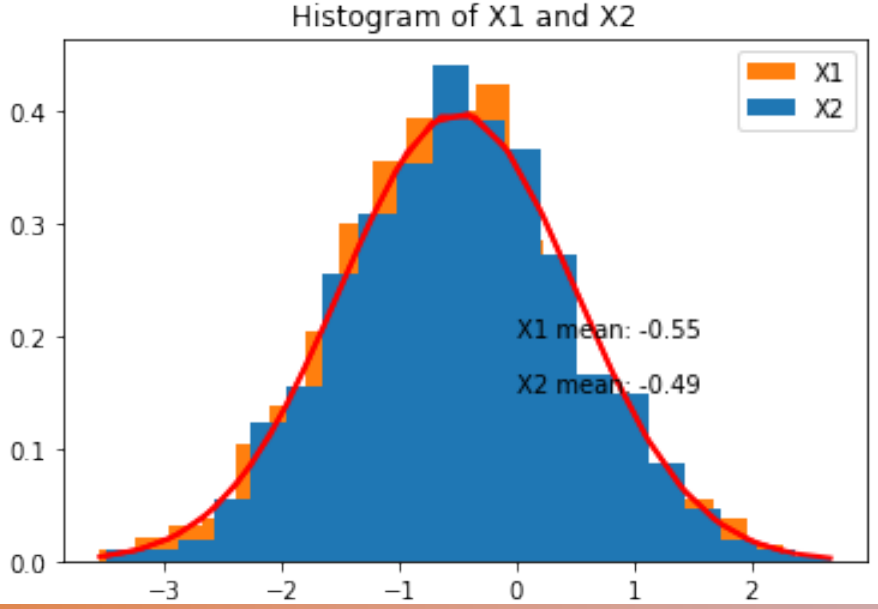
- **The problem:**
  - Consider an MDP having four states two of which are terminal states
  - State **A** is always considered as start state, and has two actions, either Right or Left. The Right action gives zero reward and lands in terminal state **C**
  - The Left action moves the agent to state **B** with zero reward
  - State **B** has a number of actions, they move the agent to the terminal state **D**
  - However (this is important) the reward R of each action from **B** to **D** has a **random value that follows a normal distribution with mean -0.5 and a variance 1.0.** The expected value of R is known to be negative (-0.5). This means that over a large number of experiments the average value of R is less than zero.

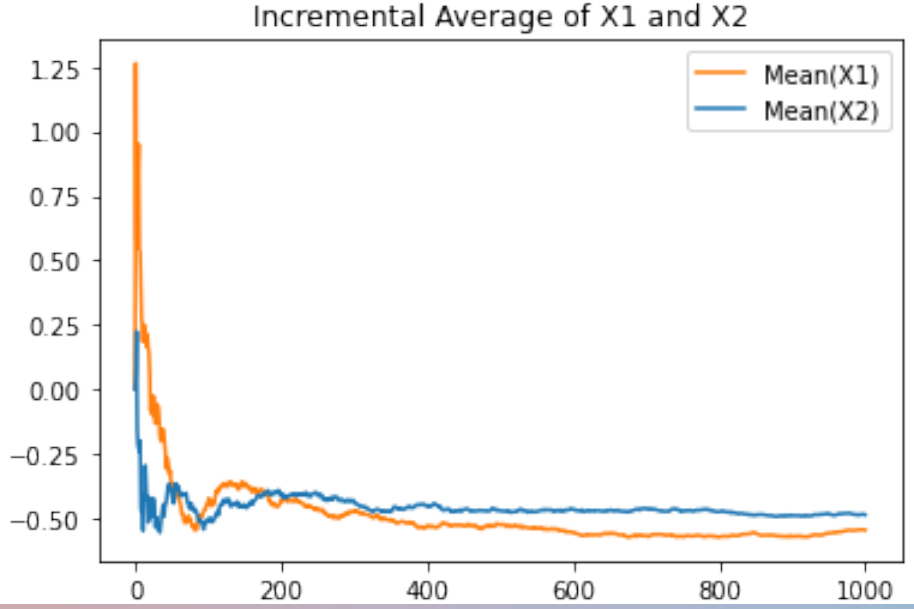


Source: https://towardsdatascience.com/double-q-learning-the-easy-way-a924c4085ec3

Double Q-learning

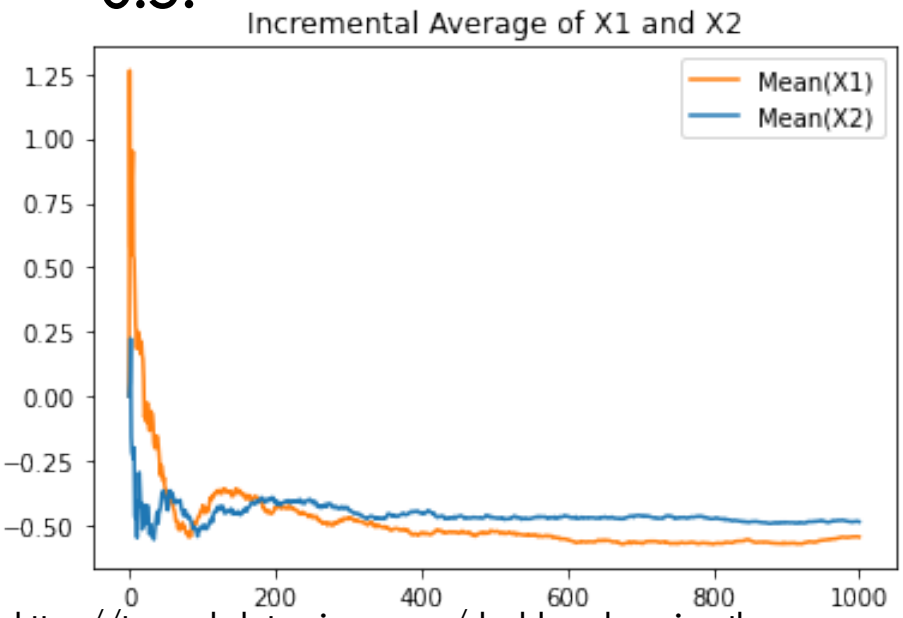
- Let **X1** and **X2** be two random variables that represent the reward of two actions at state **B**
- Since they are random variables, we will compute their expected values **E(X1)** and **E(X2)**
  - As we don't know them, we will do this by computing incremental average $\mu 1$ and $\mu 2$
    - Those estimates are unbiased because, as the number of samples increase, the average over the whole set of values approaches E(X1) and E(X2)

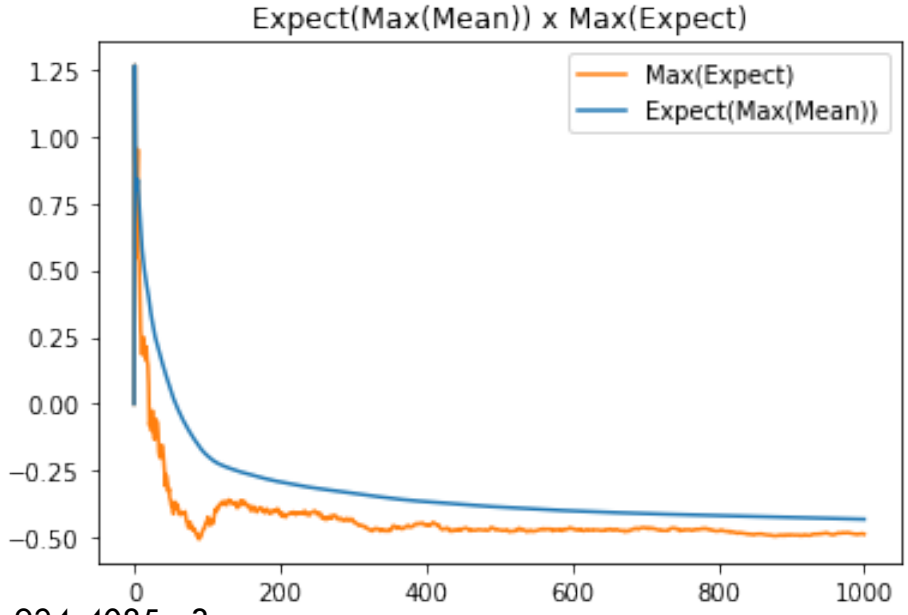




Double Q-learning

- **The problem:**
  - However, Q-Learning uses $\max \mathrm{Q}(s', a)$ (Here, $\mathbb{E}(\max(\boldsymbol{\mu}1, \boldsymbol{\mu}2))$
  - However, $\mathbb{E}(\max(\boldsymbol{\mu}1, \boldsymbol{\mu}2))$ is different from $\max(\mathbb{E}(X1), \mathbb{E}(X2))$
  - This tells that $\max(\boldsymbol{\mu}1, \boldsymbol{\mu}2)$ is not a good estimator for $\max(\mathbb{E}(X1), \mathbb{E}(X2))$
    - It is biased
  - In other words, when updating Q(s,a) with Max Q(s',a), Q(s,a) is not moving towards the expected value of the actions at state **B** which is -0.5.

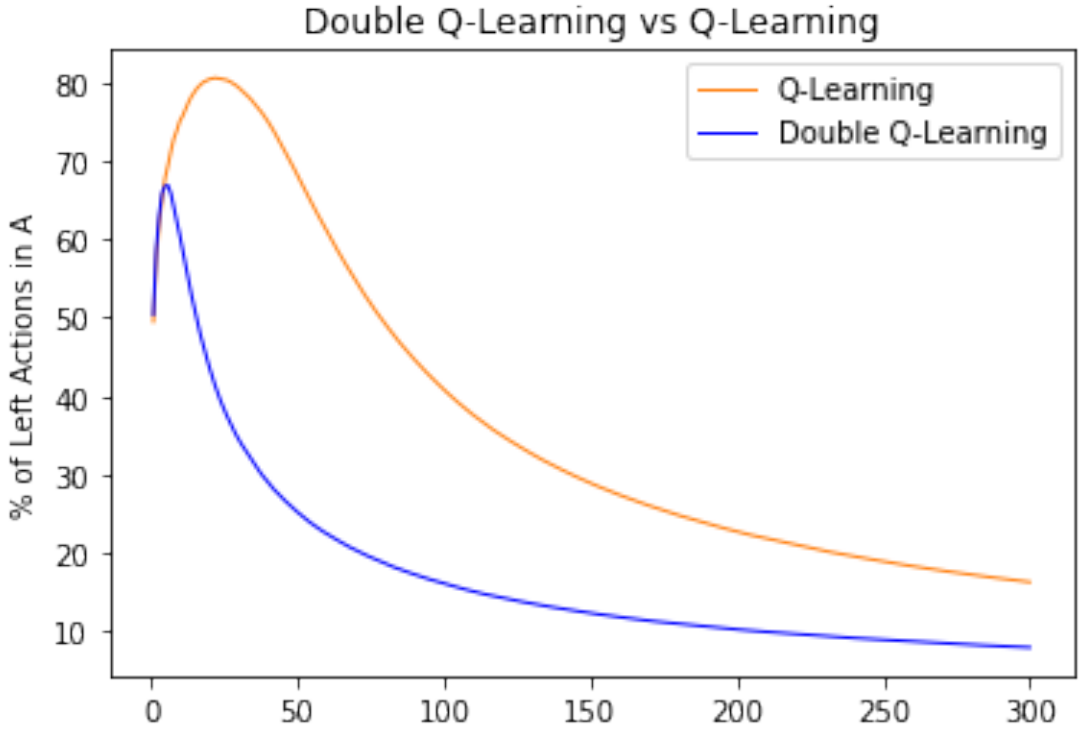




Source: https://towardsdatascience.com/double-q-learning-the-easy-way-a924c4085ec3

Double Q-learning x Q-learning

- Double-Q maintain two Q-value functions QA and QB
- Each one gets update from the other for the next state
- The update consists of finding the action a* that maximizes QA in the next state (Q(s', a*) = Max Q(s', a)), then use a* to get the value of QB(s', a*) in order to update QA(s, a).



Source: https://towardsdatascience.com/double-q-learning-the-easy-way-a924c4085ec3

Double Q-learnin: Algorithm

Algorithm: Double Q-learning

1: *Initialize* Q^A, Q^B, s

2: ***repeat***

3: Choose a, based on $Q^A(s, \cdot)$ and $Q^B(s, \cdot)$, observe r, s'

4: Choose (e.g. random) either UPDATE(A) or UPDATE(B)

5: **if** UPDATE(A) **then**

6: Define $a^* \leftarrow \arg\max_a Q^A(s', a)$

7: $Q^A(s, a) \leftarrow Q^A(s, a) + \alpha(s,a)\,(r + \gamma Q^B(s', a^*) - Q^A(s,a))$

8: **else if** UPDATE(B) **then**

9: Define $b^* \leftarrow \arg\max_a Q^B(s', a)$

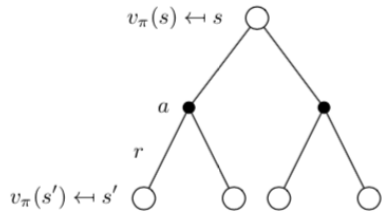
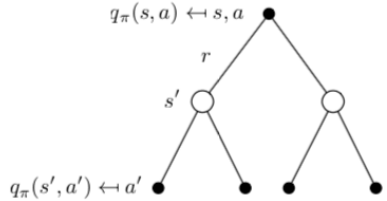
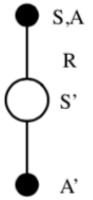
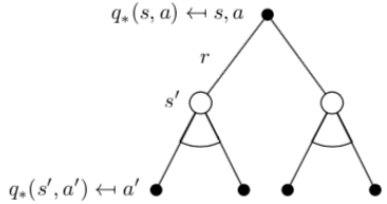
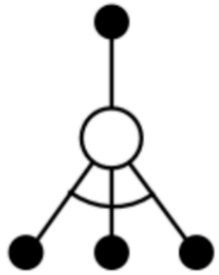
10: $Q^B(s, a) \leftarrow Q^B(s, a) + \alpha(s,a)\,(r + \gamma Q^A(s', b^*) - Q^B(s,a))$

11: **end if**

12: $s \leftarrow s'$

13: **until** end

Relationship Between DP 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

Full Backup (DP)	*Sample Backup (TD)*
Iterative Policy Evaluation	TD Learning
$V(s) \leftarrow \mathbb{E}\left[R + \gamma V(S') \mid s\right]$	$V(S) \overset{\alpha}{\leftarrow} R + \gamma V(S')$
Q-Policy Iteration	Sarsa
$Q(s,a) \leftarrow \mathbb{E}\left[R + \gamma Q(S',A') \mid s,a\right]$	$Q(S,A) \overset{\alpha}{\leftarrow} R + \gamma Q(S',A')$
Q-Value Iteration	Q-Learning
$Q(s,a) \leftarrow \mathbb{E}\left[R + \gamma \max_{a' \in \mathcal{A}} Q(S',a') \mid s,a\right]$	$Q(S,A) \overset{\alpha}{\leftarrow} R + \gamma \max_{a' \in \mathcal{A}} Q(S',a')$

where $x \overset{\alpha}{\leftarrow} y \equiv x \leftarrow x + \alpha(y - x)$

What you be able by now?

- To implement MC on policy control and SARSA and Q-learning
- To compare them according to properties of how quickly they update, bias and variance, computational cost
- To define conditions for these algorithms to converge to the optimal Q and optimal π and give at least one way to guarantee such conditions are met.

Lecture 5

- Reading:
  - RUSSELL, S. NORVIG, P. Artificial Intelligence. 3a edição. Chapter 21.
  - BARTO, A., SUTTON, R. Reinforcement Learning: An Introduction. Second Edition. Freely Available at: http://www.incompleteideas.net/book/RLbook2020.pdf

References

Lecture 5

- BARTO, A., SUTTON, R. Reinforcement Learning: An Introduction. Second Edition.
- MURPHY, R. R. Introduction to AI robotics. MIT Press, 2002.
- Lex Fridman, MIT Deep Learning Course, MIT, 2019.
- DUDEK, G.; JENKIN, M. Computational Principles of mobile robotics. Cambridge Press, 2000.
- ROMERO, R. A. F.; PRESTES, E.; OSÓRIO, F.; WOLF, D. (Orgs) Robótica móvel. LTC, 2014.
- BROOKS, R. Intelligence without representation. Artificial Intelligence, 47:139-159, 1991.
- RUSSEL, S. NORVIG, P. Artificial Intelligence: a modern approach. Prentice Hall, 2002.
- BRATKO, I. PROLOG: programming for artificial intelligence. Addison Wesley, 2nd edition, 1990.

This material is part of the Machine Learning Course

By Esther Colombini and Alexandre Simões