MC Control
SARSA
Q-Learning
Function Approx for V_π

Prof. Subrahmanya Swamy

Model-Free Setting so far..

- Prediction (To find V_π for a given π)
  - MC First-Visit
  - MC Every-Visit
  - TD
  - N-step TD
  - MC Off-Policy

- Control (To find π^* of an MDP)
  - GPI with MC
  - GPI with TD

Model Known: Policy Iteration (PI)

Policy Iteration

$$\pi_0 \xrightarrow{\text{Policy Evaluation}} V_{\pi_0} \xrightarrow{\text{Policy Improvement}} \pi_1 \rightarrow V_{\pi_1} \rightarrow \pi_2 \rightarrow V_{\pi_2} \rightarrow \dots$$

Repeat till $\pi_{k+1} = \pi_k$

$\Downarrow$

$\pi_k = \pi_*$

- *Policy Evaluation:* $V_{k+1}(s) = R_s^\pi + \sum_{s'} P_{ss'}^\pi V_k(s')$

Repeat till $V_{k+1} = V_k$

- *Policy Improvement:* $\pi_{i+1}(s) \coloneqq argmax_a \; R_s^a + \sum_{s'} P_{ss'}^a V_{\pi_i}(s')$

Policy Evaluation is stopped when $V_{k+1} \approx V_k$

Model Unknown: Generalized Policy Iteration (GPI)

$$\pi_0 \xrightarrow{\text{Approx. Policy Evaluation}} Q_{\pi_0} \xrightarrow{\epsilon-\text{greedy Policy Improvement}} \pi_1 \rightarrow Q_{\pi_1} \rightarrow \pi_2 \rightarrow Q_{\pi_2} \rightarrow \ldots$$

When to stop the Approx. Policy Evaluation?

MC: After one episode

TD: After one time-step

MC (every visit) GPI : Pseudo Code

$$\pi_0 \xrightarrow{\text{MC-PE for one episode}} Q_{\pi_0} \xrightarrow{\epsilon-\text{greedy PI}} \pi_1 \rightarrow Q_{\pi_1} \rightarrow \pi_2 \rightarrow Q_{\pi_2} \rightarrow \dots$$

- Initialize $Q(s,a) = 0, \;\; \forall\,(s,a)$
- **Repeat** for each *episode*:
  - $\pi(s) = \epsilon$-greedy w.r.t. $Q(s,a)$
  - Generate an episode following π : $S_0\,, A_0, R_1, S_1, A_1, R_2, S_2, \dots, S_T$
  - **Repeat** for each *time-step* ***t*** in the episode:
    - Compute $G_t = \sum_{i=t+1}^{T} \gamma^{i-t-1} R_i$
    - Update $Q(S_t, A_t) = Q(S_t, A_t) + \alpha\,(G_t - Q(S_t, A_t))$
- Output: $\pi^* = greedy\,(Q)$

SARSA for π^*: Pseudo Code

$$\pi_0 \rightarrow Q_{\pi_0} \rightarrow \pi_1 \rightarrow Q_{\pi_1} \rightarrow \pi_2 \rightarrow Q_{\pi_2} \rightarrow \ldots$$

TD-PE for one time-step $\qquad$ **ϵ −greedy PI**

- Initialize $Q(s,a) = 0, \ \forall \, (s,a)$
- **Repeat** for each *episode*:
  - Initialize S_0 randomly
  - Sample $A_0 \sim \epsilon$-greedy w.r.t. $Q(S_0, a)$
  - **Repeat** for each *time-step* ***t*** in the episode:
    - Take action A_t and observe R_{t+1} and S_{t+1}
    - Sample action $A_{t+1} \sim \epsilon$−greedy w.r.t. $Q(S_{t+1}, a)$
    - $Q(S_t, A_t) = Q(S_t, A_t) + \alpha \, (R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$
- Output: $\pi^* = greedy \, (Q)$

What ϵ and α to use?

- Sufficient conditions on α and ϵ to ensure MC/SARSA-based GPI converges to π^*
- k – episode number
- $\epsilon_k \rightarrow 0 \text{ as k} \rightarrow \infty$
- $\sum_k \alpha_k = \infty$ and $\sum_k \alpha_k^2 < \infty$

Example:

In the k^{th} episode, $\epsilon_k = \frac{1}{k}$ and $\alpha_k = \frac{1}{k}$ will satisfy the sufficient conditions.

Q-Learning

- SARSA: Based on Policy Iteration
- Q-Learning: Based on Value Iteration (Asynchronous)
- Value Iteration:

$$Q^*(s,a) = R_s^a + \gamma \sum_{s'} P_{ss'}^a V^*(s')$$
$$= R_s^a + \gamma \sum_{s'} P_{ss'}^a \max_{a'} Q^*(s',a')$$

- Q-Learning:

$$Q_{new}(S_t\,, A_t) = Q_{old}(S_t\,, A_t) + \alpha(R_{t+1} + \gamma \max_{a'} Q_{old}(S_{t+1}, a') - Q_{old}(S_t\,, A_t))$$

Which state and action pair should be updated in Q-learning?

- Random policy
- $\epsilon - greedy\; w.r.t\; Q$

Q-Learning: Pseudo Code

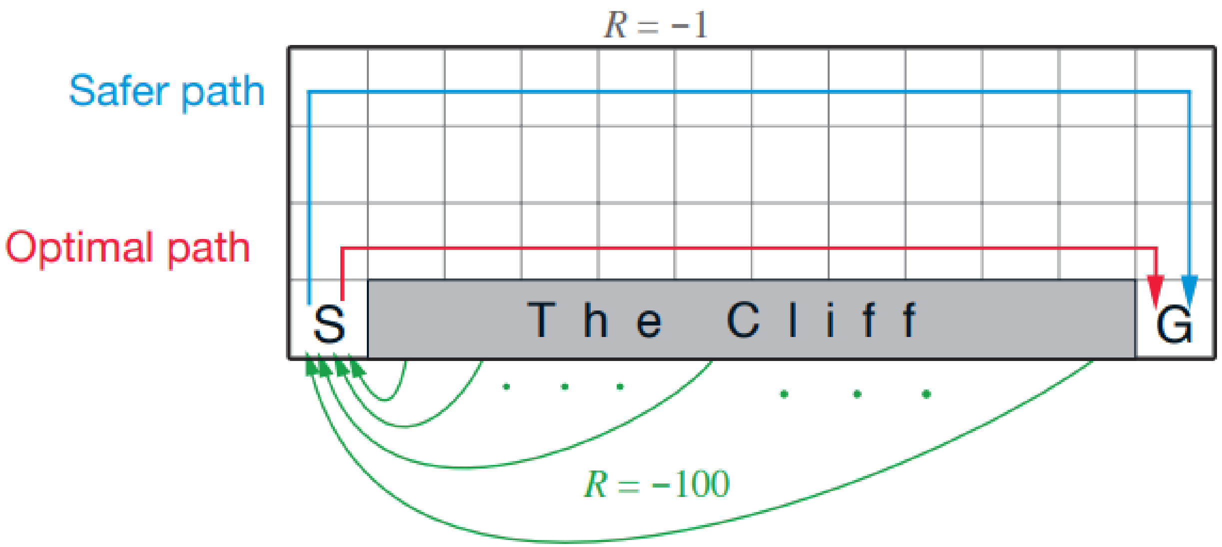
- Initialize $Q(s, a) = 0, \forall (s, a)$
- **Repeat** for each episode:
  - Initialize S_0 randomly
  - **Repeat** for each time-step $\boldsymbol{t}$ in the episode:
    - Sample action $A_t \sim \epsilon$-greedy w.r.t. $Q(S_t, a)$
    - Take action A_t and observe R_{t+1} and S_{t+1}
    - $Q(S_t, A_t) = Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right)$
- Output: $\pi^* = greedy\ (Q)$

SARSA Vs Q-Learning

- SARSA
    - On-Policy
    - Based on Policy Iteration
    - Converges to the best among ϵ-soft policies if fixed ϵ is chosen
    - Converges to π^* if ϵ is decreased to zero with time

- Q-Learning
    - Off-Policy
    - Based on Value Iteration
    - Converges to π^* even for fixed ϵ

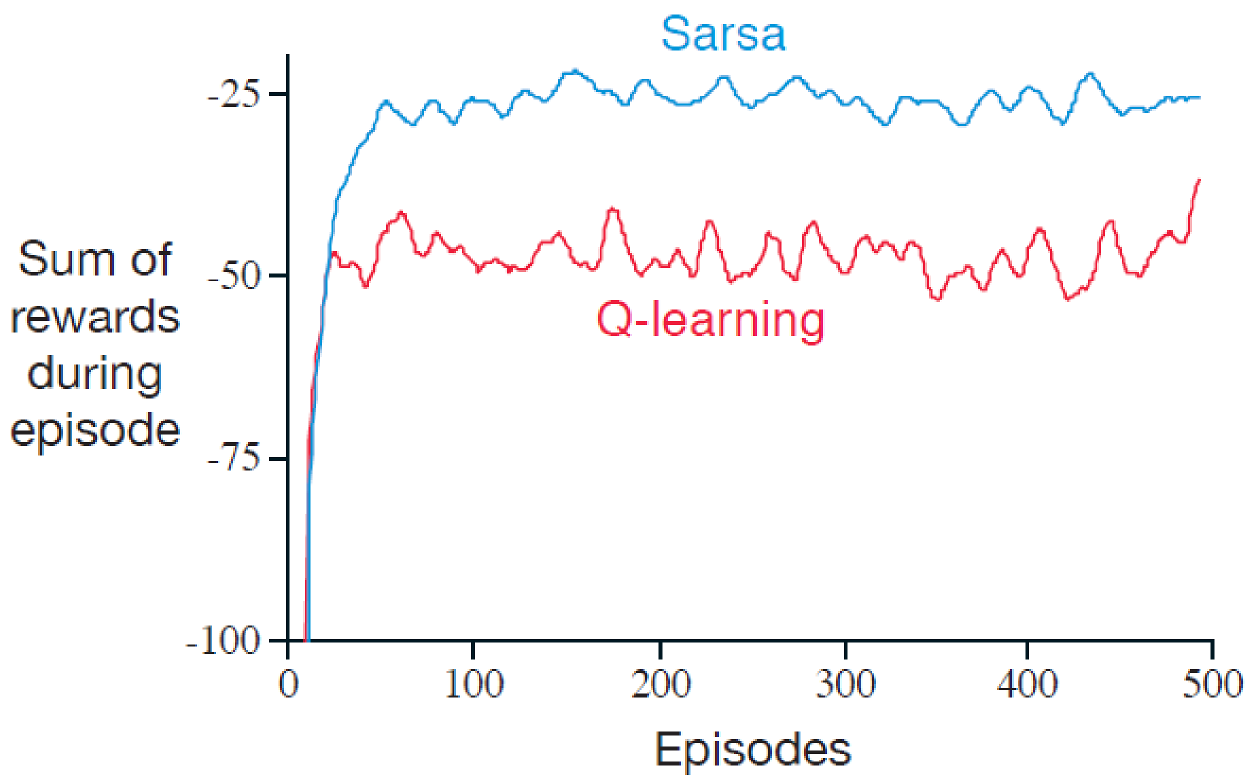
SARSA Vs Q-Learning: Example



- **SARSA**: Learns Safer path
- **Q-Learning**: Learns Optimal path

- **Aim:** To go in the shortest path from the **S**tart state to the **G**oal state
- Reward of -100 for transition into the Cliff region
- Reward of -1 for every other transition

SARSA Vs Q-Learning: Example



- SARSA: Learns Safer path
- Q-Learning: Learns Optimal path

- Why does Q-learning have a worse reward, although it learned the optimal policy?

Function Approximation

Prof. Subrahmanya Swamy

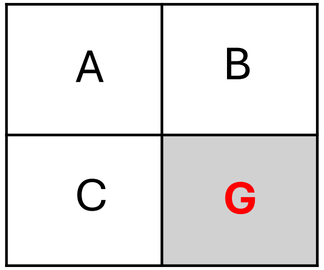
Large State Spaces in Real-time Applications

Go game

$\sim 10^{170}$ states > # Atoms in the universe

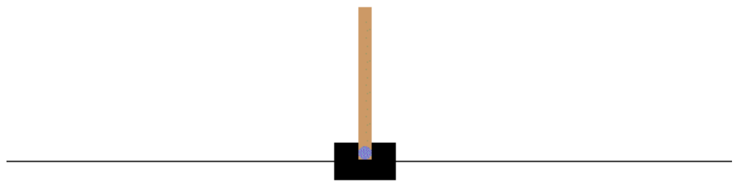
Tabular Methods

- Small state/action space: $Q(s, a)$ Table maintained for each (s, a) explicitly

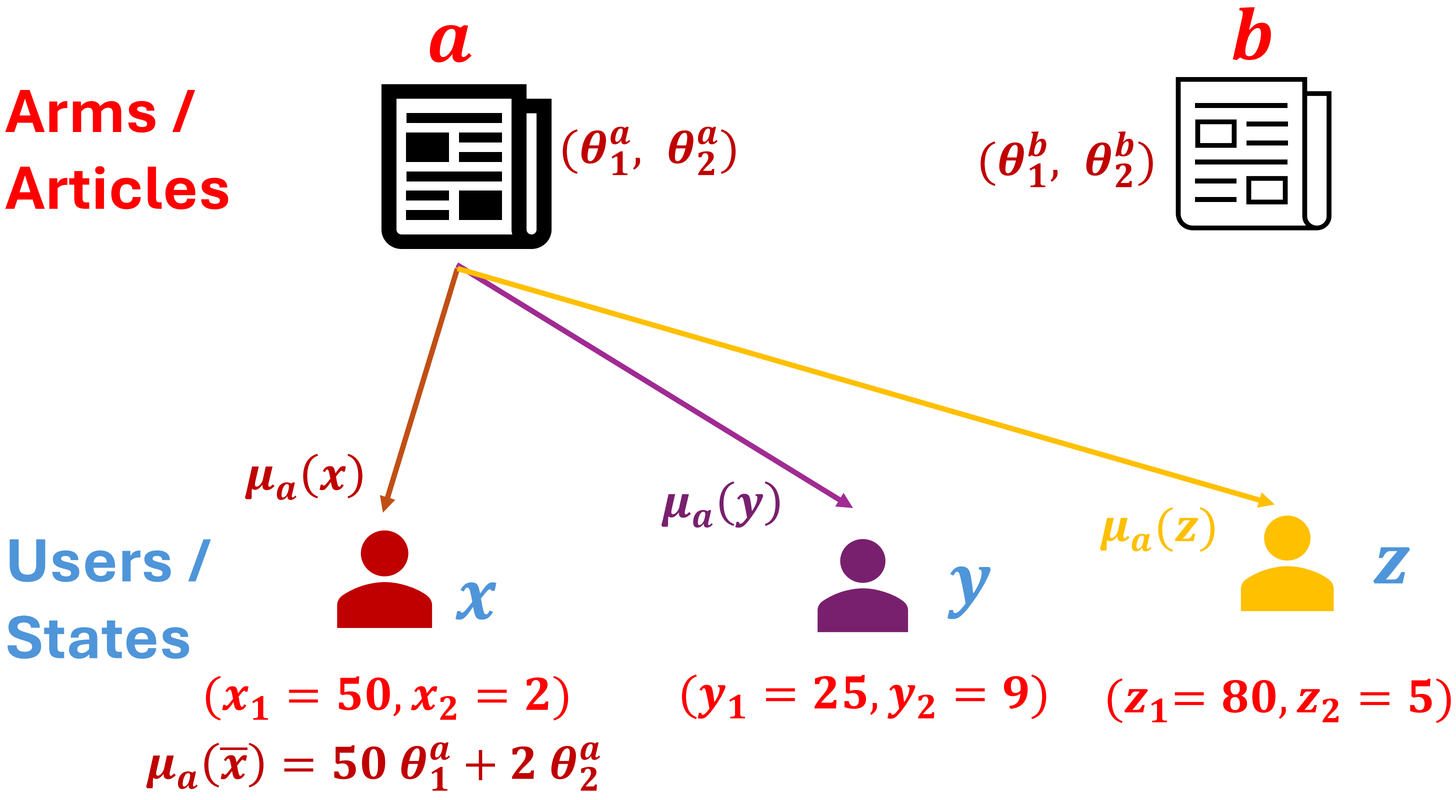


Q-Table has 4 states x 4 actions = 16 entries

- Large state/action spaces: Not feasible!
- Continuous state/action spaces: Not feasible!



Bandits + Supervised Learning



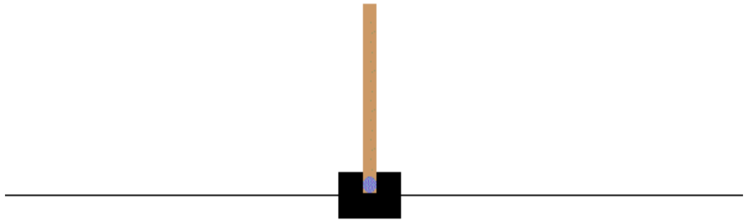
- User (state) represented by features such as age, income
  $\bar{x} = (x_1, x_2)$
- Model the expected reward for user $\bar{x}$ for pulling arm a as
  $\mu_a(\bar{x}) = \theta_1^a x_1 + \theta_2^a x_2$

Features and Function Approximation

Cartpole: The goal is to balance the pole by applying forces in the left and right direction

State Features $\boldsymbol{s} = (\boldsymbol{s_1}, \boldsymbol{s_2}, \boldsymbol{s_3}, \boldsymbol{s_4})$

State $\bar{\boldsymbol{s}}$	**Min**	**Max**
Cart Position $\boldsymbol{s_1}$	-4.8	4.8
Cart Velocity $\boldsymbol{s_2}$	-Inf	Inf
Pole Angle $\boldsymbol{s_3}$	~ -0.418 rad (-24°)	~ 0.418 rad (24°)
Pole Angular Velocity $\boldsymbol{s_4}$	-Inf	Inf



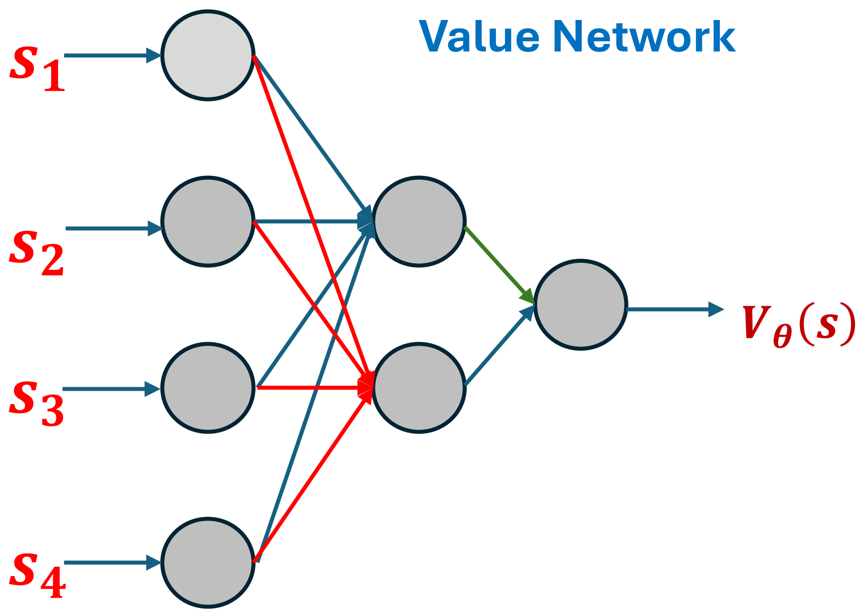
Value fn Approx $\boldsymbol{V_\theta(s) \approx s_1\theta_1 + s_2\theta_2 + s_3\theta_3 + s_4\theta_4}$

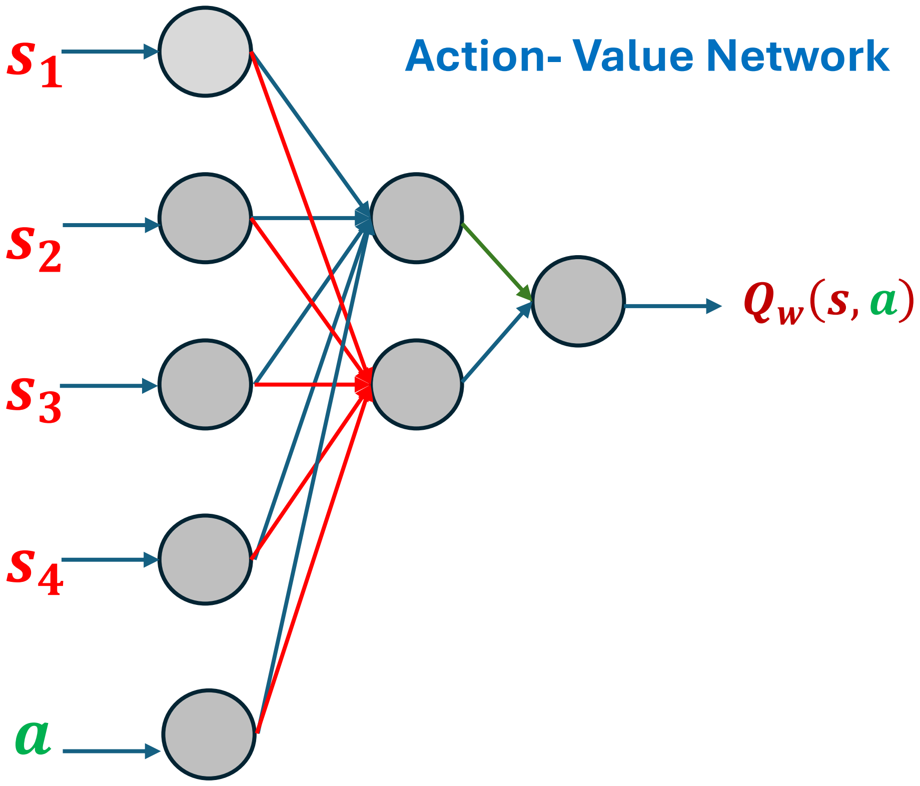
$\boldsymbol{a}$ Action Features

0:	Push the cart to the LEFT
1:	Push the cart to the RIGHT

Q fn Approx $\boldsymbol{Q_w(s,a) \approx s_1w_1 + s_2w_2 + s_3w_3 + s_4w_4 + a\,w_5}$

Non-Linear Function Approximation





Neural Network-based Function Approximation

How to find the weights of the neural network?

Function approx. for V_π

$$\boxed{\theta^* = \text{argmin}_\theta \; \mathbb{E}_\pi\left[\left(V_\pi(s) - V_\theta(s)\right)^2\right]}$$

Stochastic Gradient descent:

$$\theta_{new} = \theta_{old} + \; 2\,\alpha \left(V_\pi(s) - V_\theta(s)\right) \nabla\, V_\theta(s)$$

Challenge:

- $V_\pi(s)$ unknown
- No training data available

MC Function approx. for V_π

$$\boxed{\theta^* = \operatorname{argmin}_\theta \ \mathbb{E}_\pi\left[\left(V_\pi(s) - V_\theta(s)\right)^2\right]}$$

$$\theta_{new} = \theta_{old} + 2\,\alpha\left(V_\pi(s) - V_\theta(s)\right) \nabla V_\theta(s)$$

$V_\pi(s) \approx G_t$ starting from state s

$$\theta_{new} = \theta_{old} + 2\,\alpha\left(G_t - V_\theta(s)\right) \nabla V_\theta(s)$$

TD Function approx. for V_π

$$\boxed{\theta^* = \operatorname{argmin}_\theta \ \mathbb{E}_\pi\left[\left(V_\pi(s) - V_\theta(s)\right)^2\right]}$$

$$\theta_{new} = \theta_{old} + 2\,\alpha \left(V_\pi(s) - V_\theta(s)\right) \nabla V_\theta(s)$$

$$V_\pi(S_t) \approx R_{t+1} + \gamma\, V_\theta(S_{t+1})$$

$$\theta_{new} = \theta_{old} + 2\,\alpha \left(R_{t+1} + \gamma\, V_\theta(S_{t+1}) - V_\theta(s)\right) \nabla V_\theta(s)$$