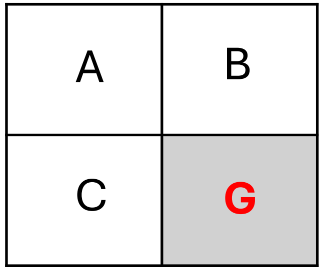
Deep Q-Network

Prof. Subrahmanya Swamy

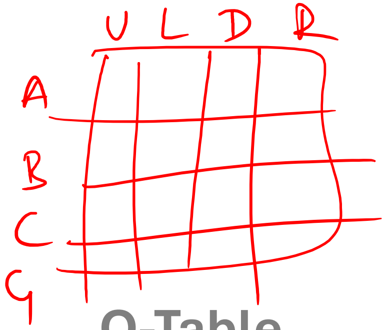
Infeasibility of Tabular Approaches

- Small state space: Q-Table Feasible



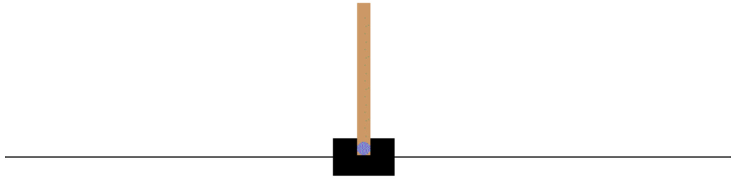
Grid Environment

→ $Q(s, a)$



Q-Table

- Continuous or Large state space: Not feasible!



Cart-Pole

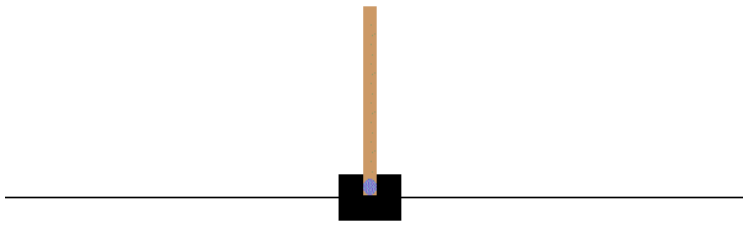
Go Game

Features: Cart-Pole Example

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

State Features $s = (s_1, s_2, s_3, s_4)$

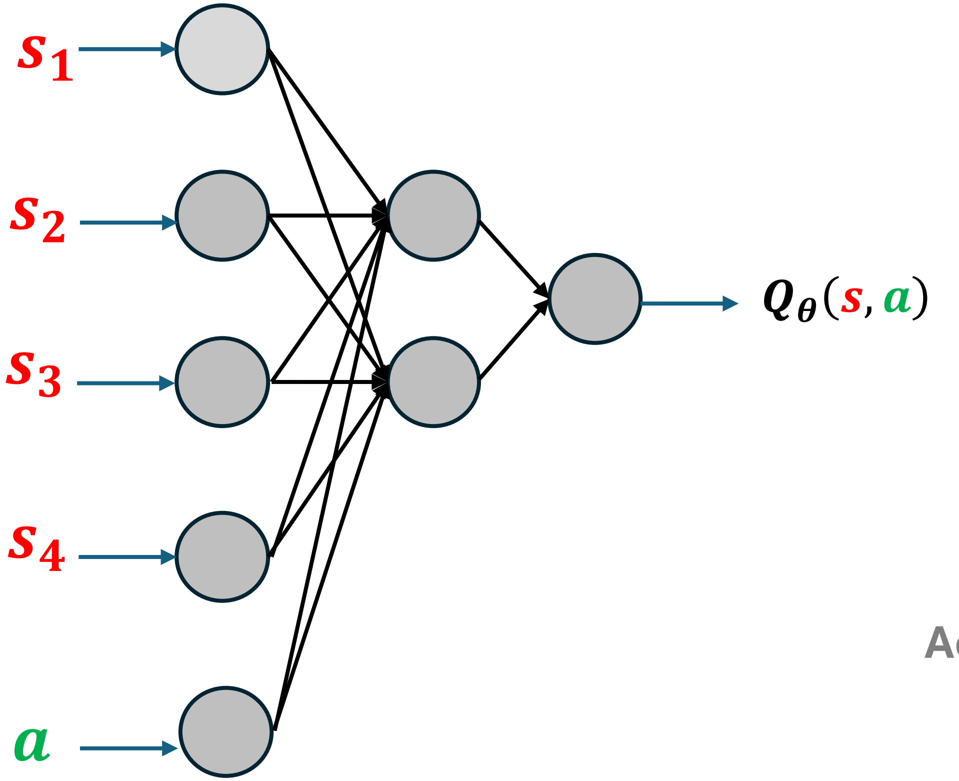
State $\bar{s}$	**Min**	**Max**
Cart Position s_1	-4.8	4.8
Cart Velocity s_2	-Inf	Inf
Pole Angle s_3	~ 24°	~ 24°
Pole Angular Velocity s_4	-Inf	Inf



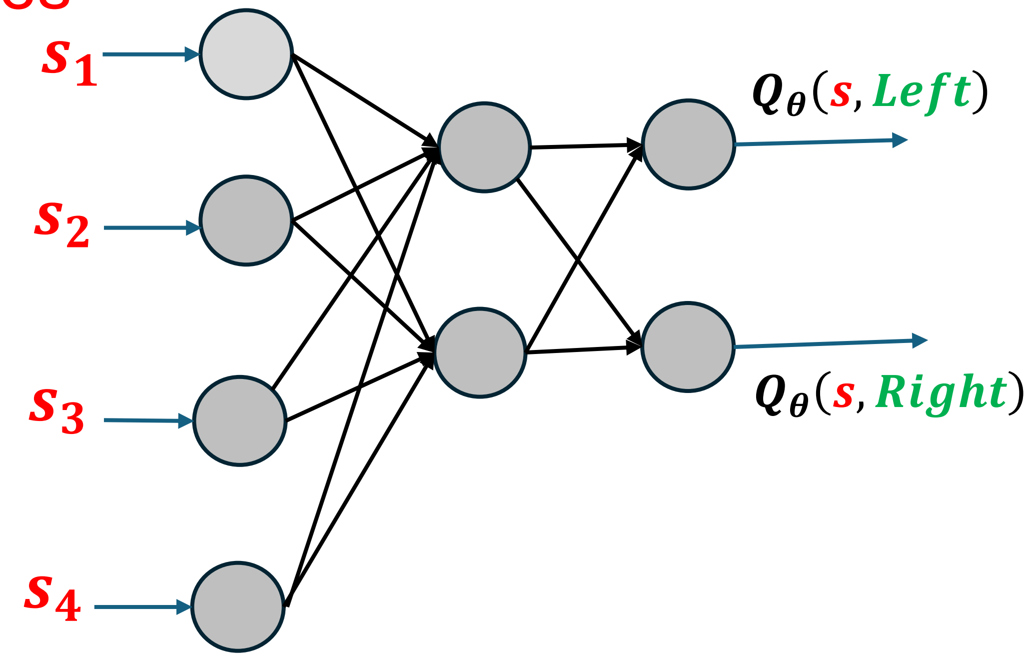
a Action Features

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

Q-Function Approximation: Architecture Choices



(OR)



Action- Value Network

Neural Network-based Function Approximation

State Features $s = (s_1, s_2, s_3, s_4)$

Actions Features a

Neural Network Weights θ

Function approx. for Q^*

$$\boxed{\theta^* = \operatorname{argmin}_\theta \mathbb{E}\left[\left(Q^*(s,a) - Q_\theta(s,a)\right)^2\right]}$$

SGD: $\theta_{new} = \theta_{old} + 2\,\alpha\left(Q^*(s,a) - Q_\theta(s,a)\right) \nabla Q_\theta(s,a)$

Challenge: Q^* unknown

Bellman Equation: $Q^*(S_t, A_t) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} Q^*(S_{t+1}, a')\right] \approx R_{t+1} + \gamma \max_{\mathrm{a}'} Q_\theta(S_{t+1}, a')$

Solution: $\theta_{new} = \theta_{old} + 2\,\alpha\left(R_{t+1} + \gamma \max_{\mathrm{a}'} Q_\theta(S_{t+1}, a') - Q_\theta(S_t, A_t)\right) \nabla Q_\theta(S_t, A_t)$

Q-Learning with Fn Approx: A Naïve Approach

- Initialize θ parameters randomly
- **Repeat** for each episode:
  - Initialize S_0 randomly
  - **Repeat** for each time-step $\boldsymbol{t}$ in the episode:
    - Obtain $Q_\theta(S_t, a)$ for all actions through a neural network forward pass
    - Sample action $A_t \sim \epsilon$-greedy w.r.t. $Q_\theta(S_t, a)$
    - Take action A_t and observe R_{t+1} and S_{t+1}
    - $target = R_{t+1} + \gamma \max_{a'} Q_\theta(S_{t+1}, a')$
    - Update $\theta = \theta + \alpha \left(target - Q_\theta(S_t, A_t)\right) \nabla Q_\theta(S_t, A_t)$ using backprop
- Output: $\pi^*(s) \approx greedy\ (Q_\theta(s, a))$

Issues with Naïve Q function approx.

1. Non-Stationary Target

- Minimize $\mathbb{E}\left[\left(Q^*(s,a) - Q_\theta(s,a)\right)^2\right]$
- Target $Q^*(s,a) \approx r \quad + \gamma \max_{a'} Q_\theta(s',a')$
- Minimize $\mathbb{E}\left[\left(r + \gamma \max_{a'} Q_\theta(s',a') - Q_\theta(s,a)\right)^2\right]$

Solution: Fixed-Target Q-Network

Issues with Naïve Q function approx.

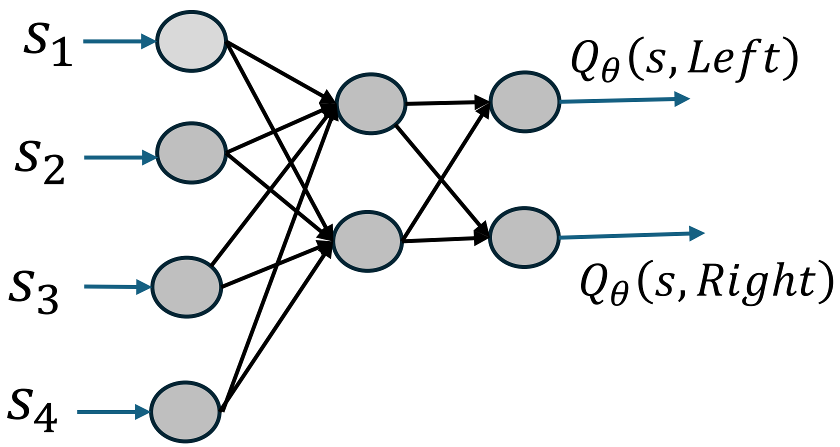
1. Non-Stationary Target

- $target = R_{t+1} + \gamma \max_{a'} Q_\theta(S_{t+1}, a')$
- Update $\theta = \theta + \alpha \left(target - Q_\theta(S_t, A_t)\right) \nabla Q_\theta(S_t, A_t)$
- During the training process θ keeps changing
- The target depends on θ
- Since target keeps changing making it difficult to converge

Solution: Fixed-Target Q-Network

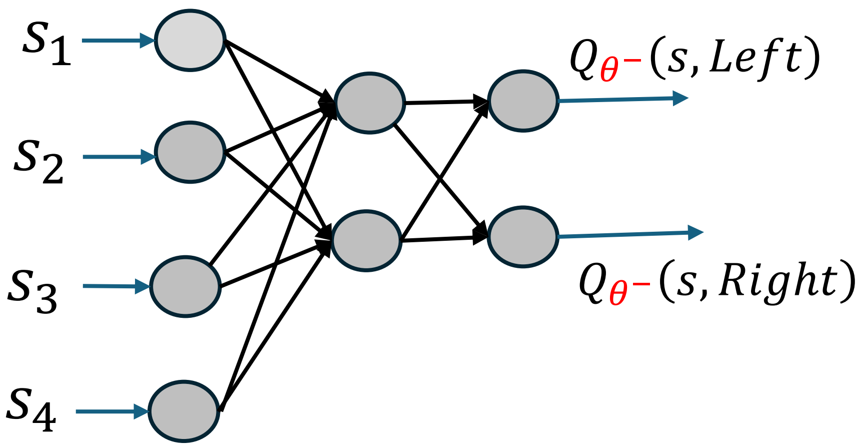
Fixed Target Q-Network

Maintain an additional neural network for calculating target



Train Q-Network

Weights: θ



Target Q-Network

Weights: θ^-

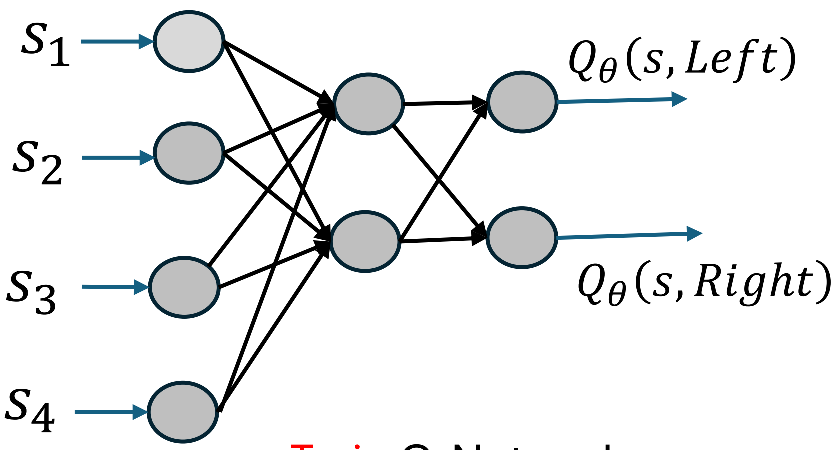
Calculate $target = R_{t+1} + \gamma \max_{a'} Q_{\theta^-}(S_{t+1}, a')$

Update train network $\theta = \theta + \alpha \left(target - Q_\theta(S_t, A_t)\right) \nabla Q_\theta(S_t, A_t)$

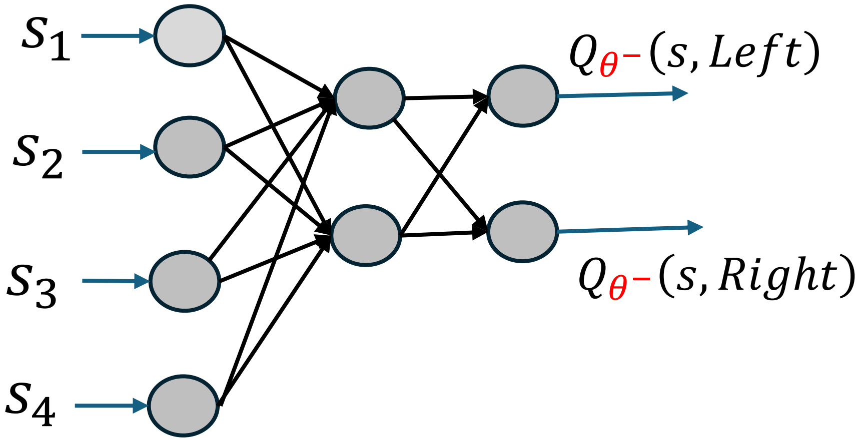
How to choose the target network weights θ^-?

How to choose target weights θ^-?

- Initialize $\theta^- = \theta$
- Repeat:
  - keep θ^- fixed for N time steps and update train weights θ
  - Update $\theta^- = \theta$ target weights to the latest train weights



Train Q-Network



Target Q-Network

2. Train for N time steps

$$target = R_{t+1} + \gamma \max_{a'} Q_{\theta^-}(S_{t+1}, a')$$
$$\theta = \theta + \alpha\left(target - Q_\theta(S_t, A_t)\right) \nabla Q_\theta(S_t, A_t)$$

1. Freeze

3. Update $\theta^- = \theta$

Issues with Naïve Q function approx.

2. Non i.i.d training samples: Leads to Instability

Sequence of samples during training in

- **Supervised learning:** i.i.d
- **Reinforcement learning:** Correlated

Solution:

- Store Last D time-steps data in a replay buffer
- Pick a random data sample from the replay buffer to train

Memory Replay Buffer

$(S_t, A_t, R_{t+1}, S_{t+1})$

$(S_{t-1}, A_{t-1}, R_t, S_t)$

$\vdots$

$\vdots$

$(S_{t-D-1}, A_{t-D-1}, R_{t-D}, S_{t-1})$

D

DQN Pseudo Code (with Target Network and Replay Buffer)

- Initialize train and target Q-network weights θ and θ^-
- **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. $Q_\theta(S_t, a)$
    - Take action A_t and observe R_{t+1} and S_{t+1}
    - Store the data $(S_t\ , A_t, R_{t+1}, S_{t+1})$ in the replay buffer
    - Select a random data sample (s, a, r, s') from the replay buffer
    - $target = r + \gamma\ \max_{a'}\ Q_{\theta^-}(s', a')$
    - Update train weights $\theta = \theta +\ \alpha\left(target - Q_\theta(s,a)\right)\nabla\, Q_\theta(s,a)$
    - If $t\ (mod)N == 0$:
      - Update target weights $\theta^- = \theta$
- Output: $\pi^*(s) \approx greedy\ (Q_\theta(s,a))$

Enhancements to DQN

- Double DQN
  - DQN overestimates Q-values due to maximization bias

$$\max_a Q(s,a)$$

  - Uses two Q-networks to resolve bias

$$Q_1(s, \arg\max_a Q_2(s,a))$$

- Duelling DQN
  - Splits Q-value into state-value $V(s)$ and advantage $A(s,a)$ functions:

$$Q(s,a) = V(s) + A(s,a)$$

  - This separation improves learning stability and efficiency