

10-701: Introduction to Machine Learning

# Lecture 17 – Reinforcement Learning: Q-learning & Deep RL

Hoda Heidari

\* Slides adopted from F24 offering of 10701 by Henry Chai.

# Recall: Markov Decision Process (MDP)

- Assume the following model for our data:

1. Start in some initial state  $s_0$
2. For time step  $t$ :
  1. Agent observes state  $s_t$
  2. Agent takes action  $a_t = \pi(s_t)$
  3. Agent receives reward  $r_t \sim p(r \mid s_t, a_t)$
  4. Agent transitions to state  $s_{t+1} \sim p(s' \mid s_t, a_t)$

3. Total reward is  $\sum_{t=0}^{\infty} \gamma^t r_t$

- MDPs make the *Markov assumption*: the reward and next state only depend on the current state and action.

## Recall: Value Function

- $V^\pi(s) = \mathbb{E}[\text{discounted total reward of starting in state } s \text{ and executing policy } \pi \text{ forever}]$   
 $= \mathbb{E}[R(s_0, \pi(s_0)) + \gamma R(s_1, \pi(s_1)) + \gamma^2 R(s_2, \pi(s_2)) + \dots \mid s_0 = s]$   
 $= R(s, \pi(s)) + \gamma \mathbb{E}[R(s_1, \pi(s_1)) + \gamma R(s_2, \pi(s_2)) + \dots \mid s_0 = s]$   
 $= R(s, \pi(s)) + \gamma \sum_{s_1 \in \mathcal{S}} p(s_1 \mid s, \pi(s)) (R(s_1, \pi(s_1)) + \gamma \mathbb{E}[R(s_2, \pi(s_2)) + \dots \mid s_1])$

$$V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s_1 \in \mathcal{S}} p(s_1 \mid s, \pi(s)) V^\pi(s_1)$$

Bellman equations

## Recall: Optimality

- Optimal value function:

$$V^*(s) = \max_{a \in \mathcal{A}} R(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s' | s, a) V^*(s')$$

- System of  $|\mathcal{S}|$  equations and  $|\mathcal{S}|$  variables

- Optimal policy:

$$\pi^*(s) = \operatorname{argmax}_{a \in \mathcal{A}} \underbrace{R(s, a)}_{\text{Immediate reward}} + \underbrace{\gamma \sum_{s' \in \mathcal{S}} p(s' | s, a) V^*(s')}_{\text{(Discounted) Future reward}}$$

# Fixed Point Iteration

- Iterative method for computing fixed points of a fn
- Given some equations and initial values

$$x_1 = f_1(x_1, \dots, x_n)$$

$$\vdots$$

$$x_n = f_n(x_1, \dots, x_n)$$

$$x_1^{(0)}, \dots, x_n^{(0)}$$

- While not converged, do

$$x_1^{(t+1)} \leftarrow f_1(x_1^{(t)}, \dots, x_n^{(t)})$$

$$\vdots$$

$$x_n^{(t+1)} \leftarrow f_n(x_1^{(t)}, \dots, x_n^{(t)})$$

# Fixed Point Iteration

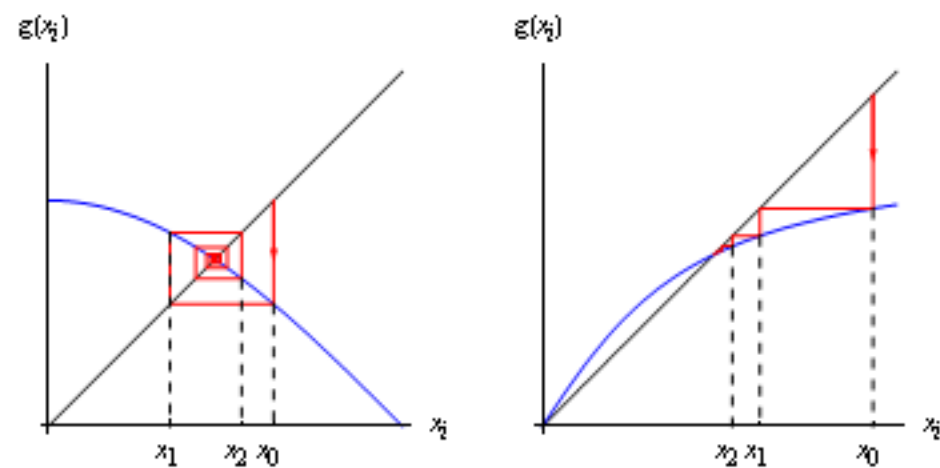


Diagram from: <https://alexgs.co.uk/maths/unit-3-pure/3.8-numerical-methods/fixed-point-iteration>

# Optimal Values as Fixed Points

- Optimal value function:

$$V^*(s) = \max_{a \in \mathcal{A}} R(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s' | s, a) V^*(s')$$

# Value Iteration

- Inputs:  $R(s, a), p(s' | s, a)$
- Initialize  $V^{(0)}(s) = 0 \forall s \in \mathcal{S}$  (or randomly) and set  $t = 0$
- While not converged, do:

- For  $s \in \mathcal{S}$

$$V^{(t+1)}(s) \leftarrow \max_{a \in \mathcal{A}} R(s, a) + \underbrace{\gamma \sum_{s' \in \mathcal{S}} p(s' | s, a) V^{(t)}(s')}_{Q(s, a)}$$

- $t = t + 1$

- For  $s \in \mathcal{S}$

$$\pi^*(s) \leftarrow \operatorname{argmax}_{a \in \mathcal{A}} R(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s' | s, a) V^{(t)}(s')$$

- Return  $\pi^*$



# Synchronous Value Iteration

- Inputs:  $R(s, a), p(s' | s, a)$
- Initialize  $V^{(0)}(s) = 0 \forall s \in \mathcal{S}$  (or randomly) and set  $t = 0$
- While not converged, do:
  - For  $s \in \mathcal{S}$ 
    - For  $a \in \mathcal{A}$ 
$$Q(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s' | s, a) V^{(t)}(s')$$
    - $V^{(t+1)}(s) \leftarrow \max_{a \in \mathcal{A}} Q(s, a)$
  - $t = t + 1$
- For  $s \in \mathcal{S}$ 
$$\pi^*(s) \leftarrow \operatorname{argmax}_{a \in \mathcal{A}} R(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s' | s, a) V^{(t)}(s')$$
- Return  $\pi^*$

# Asynchronous Value Iteration

- Inputs:  $R(s, a), p(s' | s, a)$
- Initialize  $V^{(0)}(s) = 0 \forall s \in \mathcal{S}$  (or randomly)
- While not converged, do:

- For  $s \in \mathcal{S}$

- For  $a \in \mathcal{A}$

$$Q(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s' | s, a) V(s')$$

- $V(s) \leftarrow \max_{a \in \mathcal{A}} Q(s, a)$

- For  $s \in \mathcal{S}$

$$\pi^*(s) \leftarrow \operatorname{argmax}_{a \in \mathcal{A}} R(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s' | s, a) V(s')$$

- Return  $\pi^*$

# Value Iteration Theory

- **Theorem 1:** Value function convergence

$V$  will converge to  $V^*$  if each state is “visited”  
infinitely often (Bertsekas, 1989)

- **Theorem 2:** Convergence criterion

$$\text{if } \max_{s \in \mathcal{S}} |V^{(t+1)}(s) - V^{(t)}(s)| < \epsilon,$$

$$\text{then } \max_{s \in \mathcal{S}} |V^{(t+1)}(s) - V^*(s)| < \frac{2\epsilon\gamma}{1-\gamma} \text{ (Williams \& Baird, 1993)}$$

- **Theorem 3:** Policy convergence

The “greedy” policy,  $\pi(s) = \operatorname{argmax}_{a \in \mathcal{A}} Q(s, a)$ , converges to the optimal  $\pi^*$  in a finite number of iterations, often before the value function has converged! (Bertsekas, 1987)

# Bellman Optimality Characterization

- A policy  $\pi$  is optimal if and only if it is greedy (optimal) w.r.t. its own value function  $V^\pi$ .
- Proof:
  - ( $\Rightarrow$ ) If  $\pi$  is optimal, then it must be greedy w.r.t  $V^\pi$ . If  $\pi$  were not greedy at some state, there would exist an action with strictly higher expected return  $\Rightarrow$  we could improve the policy  $\Rightarrow \pi$  was not optimal. Contradiction.
  - ( $\Leftarrow$ ) If  $\pi$  is greedy w.r.t  $V^\pi$ , then  $\pi$  is optimal.  
Greedy w.r.t its own value solves the Bellman *optimality* fixed point, which is known to have a unique solution. So  $V^\pi = V^*$  and  $\pi$  is optimal.

# Policy Iteration

- Inputs:  $R(s, a)$ ,  $p(s' | s, a)$
- Initialize  $\pi$  randomly
- While not converged, do:

- Solve the Bellman equations defined by policy  $\pi$

$$V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} p(s' | s, \pi(s)) V^\pi(s')$$

- Update  $\pi$

$$\pi(s) \leftarrow \operatorname{argmax}_{a \in \mathcal{A}} R(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s' | s, a) V^\pi(s')$$

- Return  $\pi$

# Policy Iteration Theory

- In policy iteration, the policy improves in each iteration.
- Given finite state and action spaces, there are finitely many possible policies
  - Thus, the number of iterations needed to converge is bounded!
- Value iteration takes  $O(|\mathcal{S}|^2|\mathcal{A}|)$  time / iteration
- Policy iteration takes  $O(|\mathcal{S}|^2|\mathcal{A}| + |\mathcal{S}|^3)$  time / iteration
  - However, empirically policy iteration requires fewer iterations to converge

# Key Takeaways

- In reinforcement learning, we assume our data comes from a Markov decision process
- The goal is to compute an optimal policy or function that maps states to actions
- Value function can be defined in terms of values of all other states; this is called the Bellman equations
- If the reward and transition functions are known, we can solve for the optimal policy (and value function) using value or policy iteration
  - Both algorithms are instances of fixed point iteration and are guaranteed to converge (under some assumptions)

# Two big Q's

1. What can we do if the reward and/or transition functions/distributions are unknown?
2. How can we handle infinite (or just very large) state/action spaces?



# Value Iteration

- Inputs:  $R(s, a)$ ,  $p(s' | s, a)$ ,  $\gamma$
- Initialize  $V^{(0)}(s) = 0 \forall s \in \mathcal{S}$  (or randomly) and set  $t = 0$
- While not converged, do:
  - For  $s \in \mathcal{S}$ 
    - For  $a \in \mathcal{A}$ 
$$Q(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s' | s, a) V(s')$$
    - $V(s) \leftarrow \max_{a \in \mathcal{A}} Q(s, a)$
  - For  $s \in \mathcal{S}$ 
$$\pi^*(s) \leftarrow \operatorname{argmax}_{a \in \mathcal{A}} R(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s' | s, a) V(s')$$
- Return  $\pi^*$

$Q^*(s, a)$  w/  
deterministic  
rewards

- $Q^*(s, a) = \mathbb{E}[\text{total discounted reward of taking action } a \text{ in state } s, \text{ assuming all future actions are optimal}]$

$$= R(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s' | s, a) V^*(s')$$

$$V^*(s') = \max_{a' \in \mathcal{A}} Q^*(s', a')$$

$$Q^*(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s' | s, a) \left[ \max_{a' \in \mathcal{A}} Q^*(s', a') \right]$$

$$\pi^*(s) = \operatorname{argmax}_{a \in \mathcal{A}} Q^*(s, a)$$

- Insight: if we know  $Q^*$ , we can compute an optimal policy  $\pi^*$ !

# Q-learning: Overview

- Q-learning
  - Learn the Q-table without requiring a model of the world, that is rewards and transition probabilities (**model-free**).
  - Utilizes *Monte Carlo sampling*: it approximates the expectation simply by taking more and more samples over time.
  - Is **off-policy** because it estimates the value of the greedy optimal policy while following a potentially different policy.

$Q^*(s, a)$  w/  
deterministic  
rewards

- $Q^*(s, a) = \mathbb{E}[\text{total discounted reward of taking action } a \text{ in state } s, \text{ assuming all future actions are optimal}]$

$$= R(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s' | s, a) V^*(s')$$

$$V^*(s') = \max_{a' \in \mathcal{A}} Q^*(s', a')$$

$$Q^*(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s' | s, a) \left[ \max_{a' \in \mathcal{A}} Q^*(s', a') \right]$$

$$\pi^*(s) = \operatorname{argmax}_{a \in \mathcal{A}} Q^*(s, a)$$

- Insight: if we know  $Q^*$ , we can compute an optimal policy  $\pi^*$ !

$Q^*(s, a)$  w/  
deterministic  
rewards and  
transitions

- $Q^*(s, a) = \mathbb{E}[\text{total discounted reward of taking action } a \text{ in state } s, \text{ assuming all future actions are optimal}]$

$$= R(s, a) + \gamma V^*(\delta(s, a))$$

- $V^*(\delta(s, a)) = \max_{a' \in \mathcal{A}} Q^*(\delta(s, a), a')$

$$Q^*(s, a) = R(s, a) + \gamma \max_{a' \in \mathcal{A}} Q^*(\delta(s, a), a')$$

$$\pi^*(s) = \operatorname{argmax}_{a \in \mathcal{A}} Q^*(s, a)$$

- Insight: if we know  $Q^*$ , we can compute an optimal policy  $\pi^*$ !

# Learning $Q^*(s, a)$ w/ deterministic rewards and transitions

## Algorithm 1: Online learning (table form)

- Inputs: discount factor  $\gamma$ , an initial state  $s$
- Initialize  $Q(s, a) = 0 \forall s \in \mathcal{S}, a \in \mathcal{A}$  ( $Q$  is a  $|\mathcal{S}| \times |\mathcal{A}|$  array)
- While TRUE, do
  - Take a random action  $a$
  - Receive reward  $r = R(s, a)$
  - Update the state:  $s \leftarrow s'$  where  $s' = \delta(s, a)$
  - Update  $Q(s, a)$ :

$$Q(s, a) \leftarrow r + \gamma \max_{a'} Q(s', a')$$

# Learning $Q^*(s, a)$ w/ deterministic rewards and transitions

## Algorithm 2: $\epsilon$ -greedy online learning (table form)

- Inputs: discount factor  $\gamma$ , an initial state  $s$ , greediness parameter  $\epsilon \in [0, 1]$
- Initialize  $Q(s, a) = 0 \forall s \in \mathcal{S}, a \in \mathcal{A}$  ( $Q$  is a  $|\mathcal{S}| \times |\mathcal{A}|$  array)
- While TRUE, do
  - With probability  $1 - \epsilon$ , take the greedy action
$$a = \operatorname{argmax}_{a' \in \mathcal{A}} Q(s, a')$$
Otherwise, with probability  $\epsilon$ , take a random action  $a$
  - Receive reward  $r = R(s, a)$
  - Update the state:  $s \leftarrow s'$  where  $s' = \delta(s, a)$
  - Update  $Q(s, a)$ :
$$Q(s, a) \leftarrow r + \gamma \max_{a'} Q(s', a')$$

# Learning $Q^*(s, a)$ w/ deterministic rewards

## Algorithm 3: $\epsilon$ -greedy online learning (table form)

- Inputs: discount factor  $\gamma$ , an initial state  $s$ , greediness parameter  $\epsilon \in [0, 1]$ , learning rate  $\alpha \in [0, 1]$  (“trust parameter”)
- Initialize  $Q(s, a) = 0 \forall s \in \mathcal{S}, a \in \mathcal{A}$  ( $Q$  is a  $|\mathcal{S}| \times |\mathcal{A}|$  array)
- While TRUE, do
  - With probability  $1 - \epsilon$ , take the greedy action
$$a = \operatorname{argmax}_{a' \in \mathcal{A}} Q(s, a')$$
Otherwise, with probability  $\epsilon$ , take a random action  $a$
  - Receive reward  $r = R(s, a)$
  - Update the state:  $s \leftarrow s'$  where  $s' \sim p(s' | s, a)$
  - Update  $Q(s, a)$ :

$$Q(s, a) \leftarrow \underbrace{(1 - \alpha)Q(s, a)}_{\text{Current value}} + \alpha \underbrace{\left( r + \gamma \max_{a'} Q(s', a') \right)}_{\text{Update w/ deterministic transitions}}$$



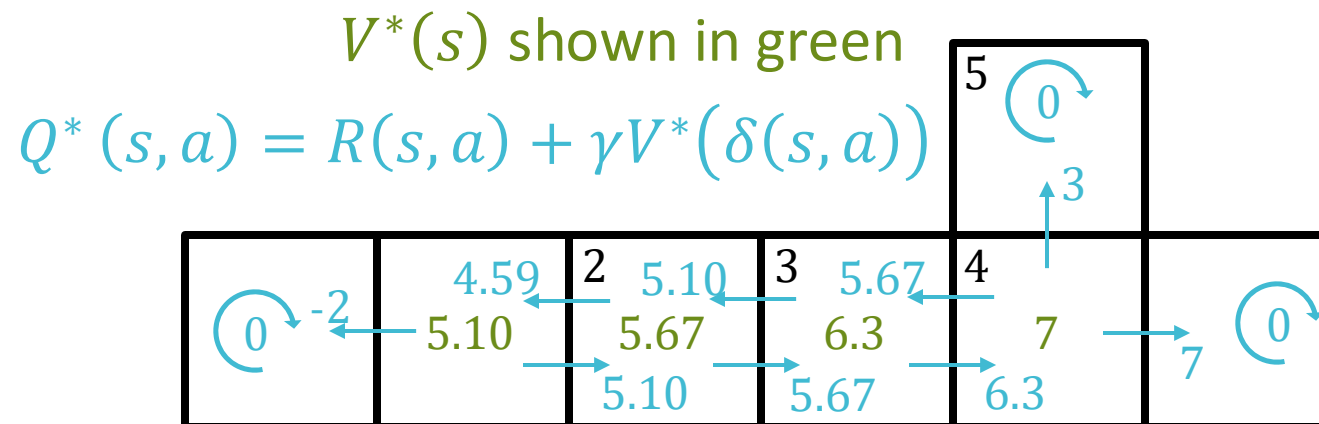
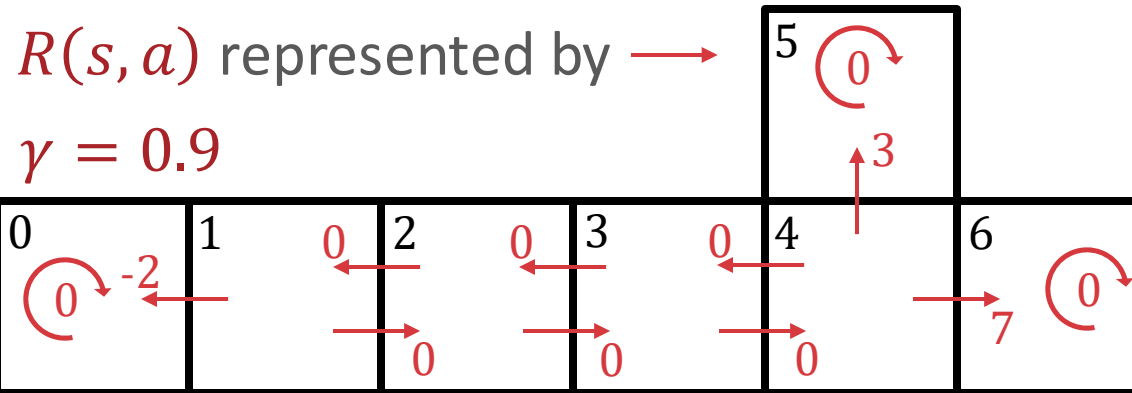
# Learning $Q^*(s, a)$ w/ deterministic rewards

## Algorithm 3: $\epsilon$ -greedy online learning (table form)

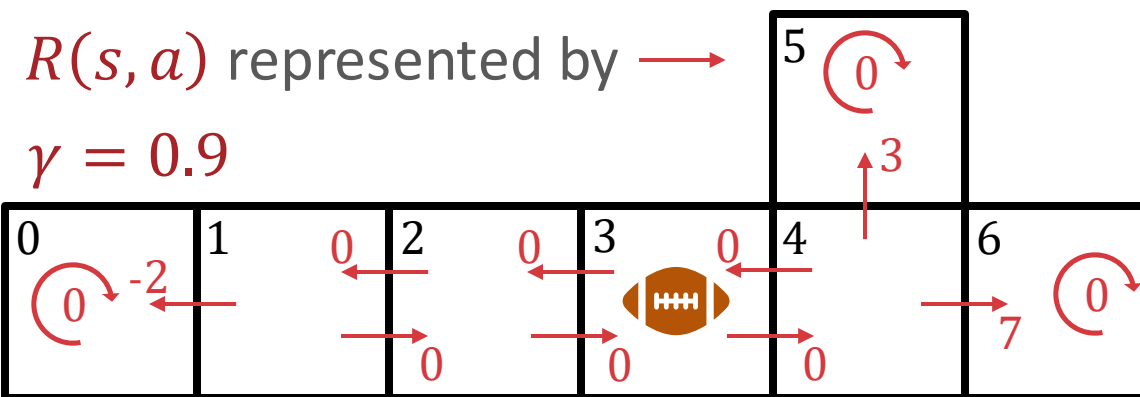
- Inputs: discount factor  $\gamma$ , an initial state  $s$ , greediness parameter  $\epsilon \in [0, 1]$ , learning rate  $\alpha \in [0, 1]$  (“trust parameter”)
- Initialize  $Q(s, a) = 0 \forall s \in \mathcal{S}, a \in \mathcal{A}$  ( $Q$  is a  $|\mathcal{S}| \times |\mathcal{A}|$  array)
- While TRUE, do
  - With probability  $\epsilon$ , take the greedy action
$$a = \operatorname{argmax}_{a' \in \mathcal{A}} Q(s, a')$$
  - Otherwise, with probability  $1 - \epsilon$ , take a random action  $a$
  - Receive reward  $r = R(s, a)$
  - Update the state:  $s \leftarrow s'$  where  $s' \sim p(s' | s, a)$
  - Update  $Q(s, a)$ :

$$Q(s, a) \leftarrow \underbrace{Q(s, a)}_{\text{Current value}} + \alpha \left( \underbrace{r + \gamma \max_{a'} Q(s', a')}_{\text{Temporal difference target}} - \underbrace{Q(s, a)}_{\text{Temporal difference}} \right)$$

# Learning $Q^*(s, a)$ : Example

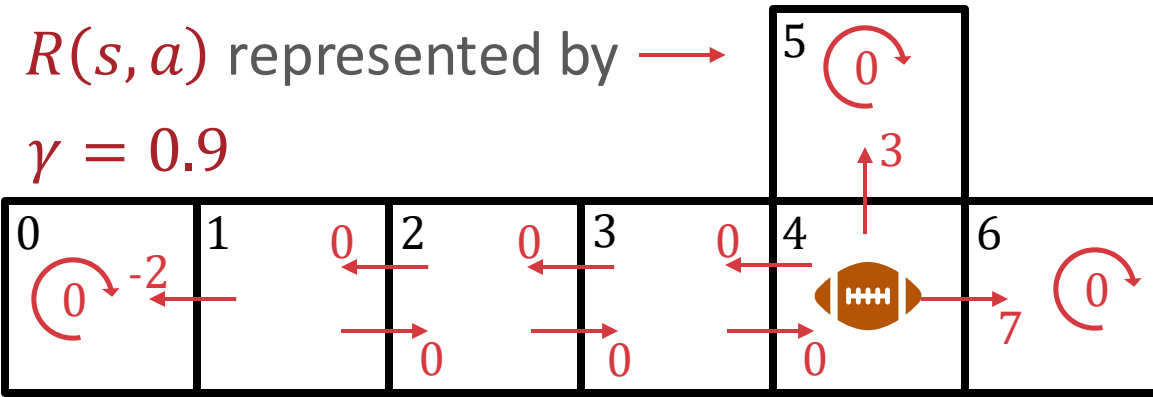


# Learning $Q^*(s, a)$ : Example



$Q(s, a)$	$\rightarrow$	$\leftarrow$	$\uparrow$	$\curvearrowright$
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0

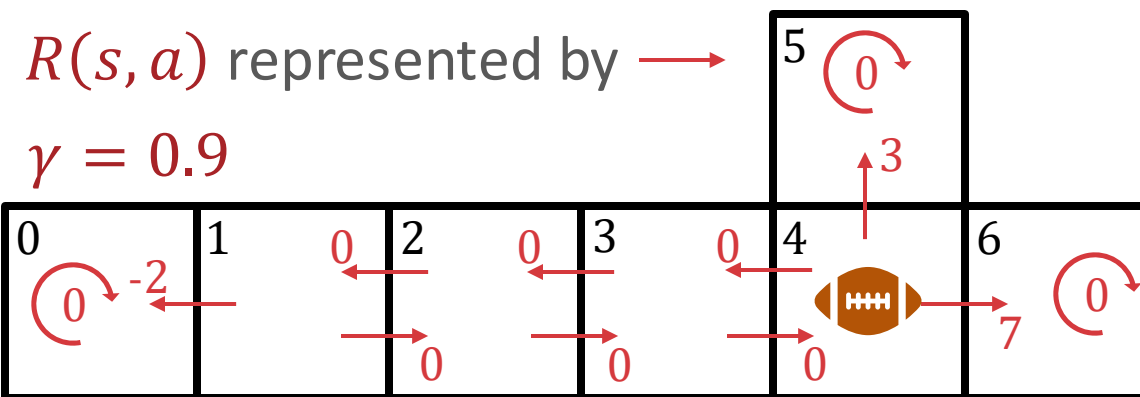
# Learning $Q^*(s, a)$ : Example



$$Q(3, \rightarrow) \leftarrow 0 + (0.9) \max_{a' \in \{\rightarrow, \leftarrow, \uparrow, \cup\}} Q(4, a') = 0$$

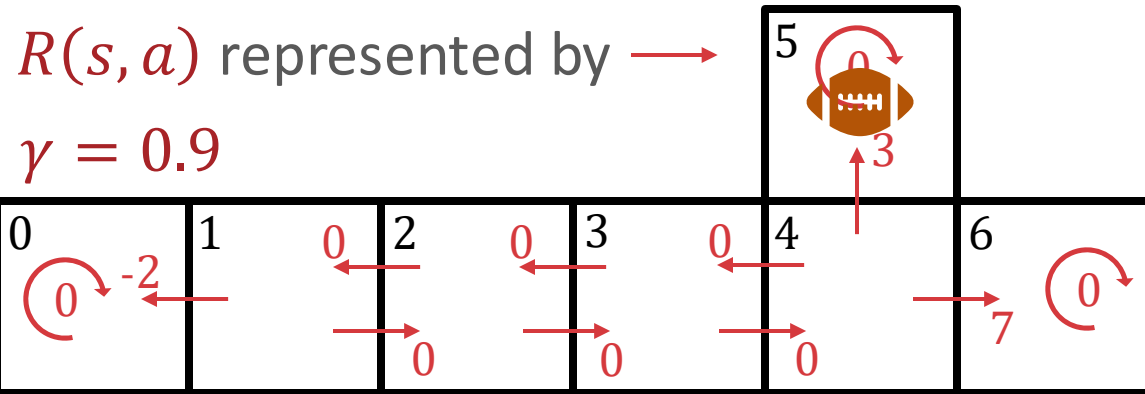
$Q(s, a)$	$\rightarrow$	$\leftarrow$	$\uparrow$	$\cup$
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0

# Learning $Q^*(s, a)$ : Example



$Q(s, a)$	$\rightarrow$	$\leftarrow$	$\uparrow$	$\downarrow$
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0

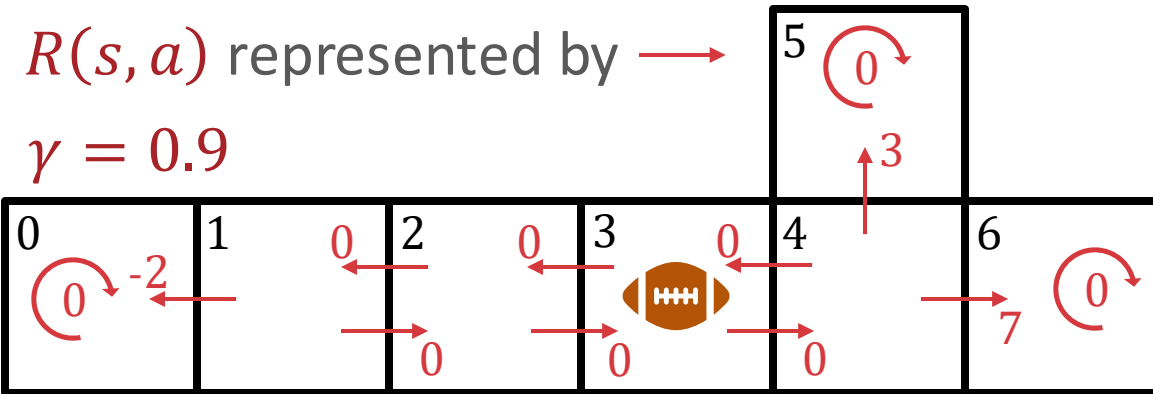
# Learning $Q^*(s, a)$ : Example



$$Q(4, \uparrow) \leftarrow 3 + (0.9) \max_{a' \in \{\rightarrow, \leftarrow, \uparrow, \cup\}} Q(5, a') = 3$$

$Q(s, a)$	$\rightarrow$	$\leftarrow$	$\uparrow$	$\cup$
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0

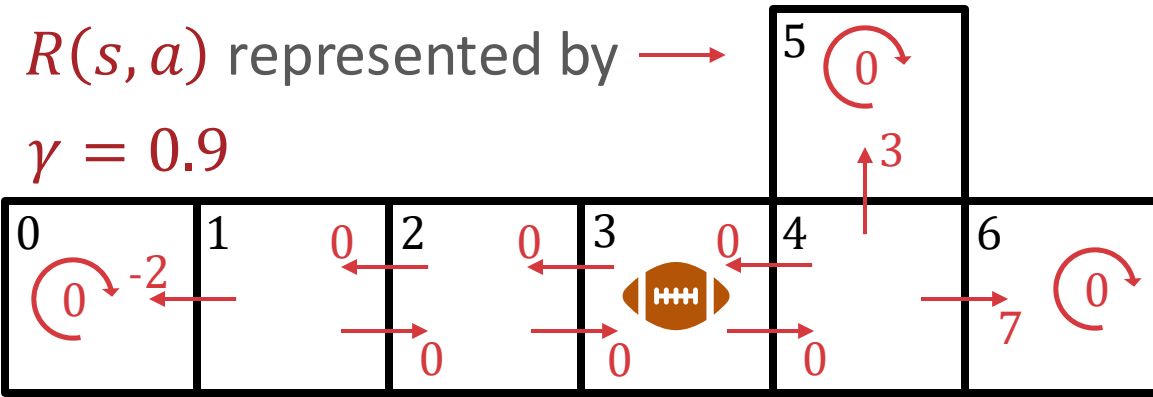
# Learning $Q^*(s, a)$ : Example



$$Q(3, \rightarrow) \leftarrow 0 + (0.9) \max_{a' \in \{\rightarrow, \leftarrow, \uparrow, \cup\}} Q(4, a') = 2.7$$

$Q(s, a)$	$\rightarrow$	$\leftarrow$	$\uparrow$	$\cup$
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	3	0
5	0	0	0	0
6	0	0	0	0

# Learning $Q^*(s, a)$ : Example



$$Q(3, \rightarrow) \leftarrow 0 + (0.9) \max_{a' \in \{\rightarrow, \leftarrow, \uparrow, \cup\}} Q(4, a') = 2.7$$

$Q(s, a)$	$\rightarrow$	$\leftarrow$	$\uparrow$	$\cup$
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	2.7	0	0	0
4	0	0	3	0
5	0	0	0	0
6	0	0	0	0



# Learning $Q^*(s, a)$ : Convergence

- For Algorithms 1 & 2 (deterministic transitions),  
 $Q$  converges to  $Q^*$  if
  1. Every valid state-action pair is visited infinitely often
    - Q-learning is exploration-insensitive: any visitation strategy that satisfies this property will work!
  2.  $0 \leq \gamma < 1$
  3.  $\exists \beta$  s.t.  $|R(s, a)| < \beta \forall s \in \mathcal{S}, a \in \mathcal{A}$
  4. Initial  $Q$  values are finite

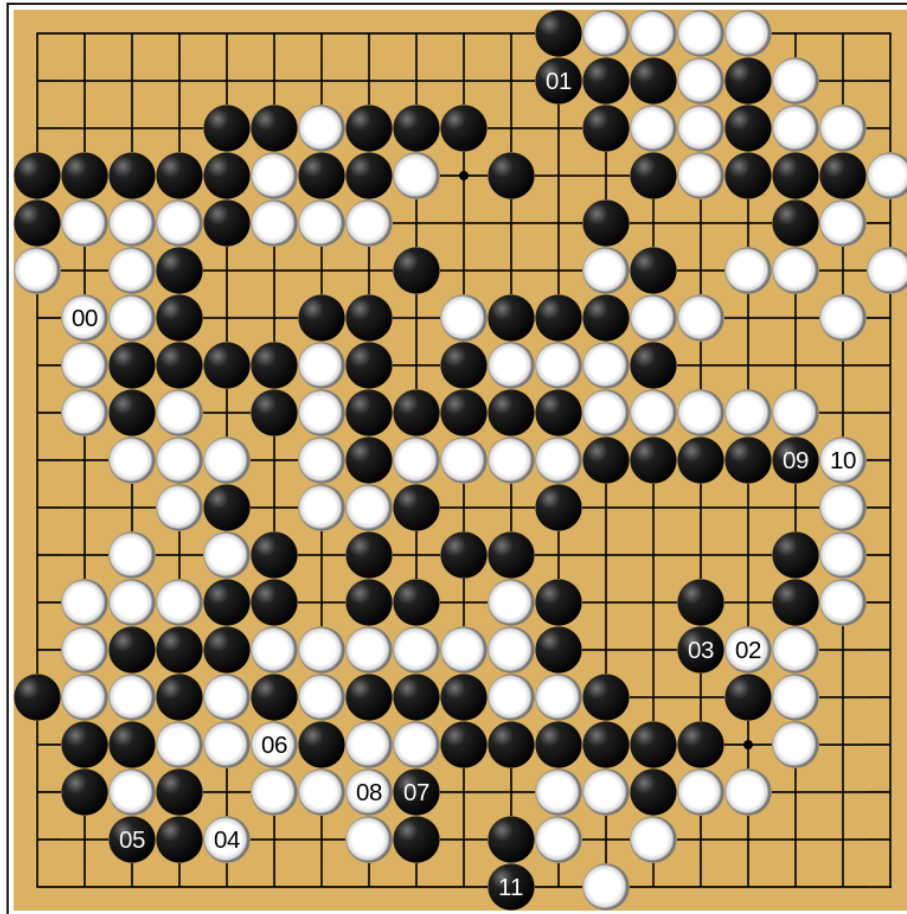
# Learning $Q^*(s, a)$ : Convergence

- For Algorithm 3 (temporal difference learning),  $Q$  converges to  $Q^*$  if
  1. Every valid state-action pair is visited infinitely often
    - Q-learning is exploration-insensitive: any visitation strategy that satisfies this property will work!
  2.  $0 \leq \gamma < 1$
  3.  $\exists \beta$  s.t.  $|R(s, a)| < \beta \forall s \in \mathcal{S}, a \in \mathcal{A}$
  4. Initial  $Q$  values are finite
  5. Learning rate  $\alpha_t$  follows some “schedule” s.t.  
 $\sum_{t=0}^{\infty} \alpha_t = \infty$  and  $\sum_{t=0}^{\infty} \alpha_t^2 < \infty$  e.g.,  $\alpha_t = 1/t+1$

# Two big Q's

1. What can we do if the reward and/or transition functions/distributions are unknown?
  - Use online learning to gather data and learn  $Q^*(s, a)$
2. How can we handle infinite (or just very large) state/action spaces?

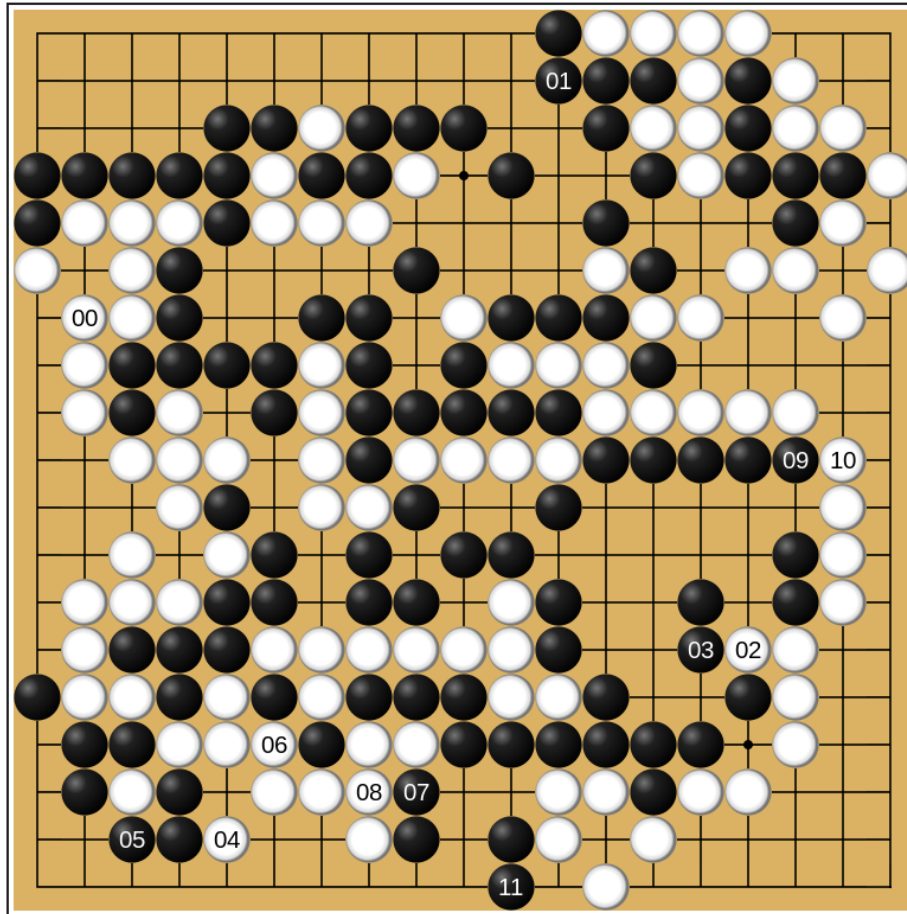
## AlphaGo (Black) vs. Lee Sedol (White) Game 2 final position (AlphaGo wins)



## Playing Go

- 19-by-19 board
- Players alternate placing black and white stones
- The goal is claim more territory than the opponent
- How many legal Go board states are there?

## AlphaGo (Black) vs. Lee Sedol (White) Game 2 final position (AlphaGo wins)



## Playing Go

- 19-by-19 board
- Players alternate placing black and white stones
- The goal is claim more territory than the opponent
- There are  $\sim 10^{170}$  legal Go board states!

# Two big Q's

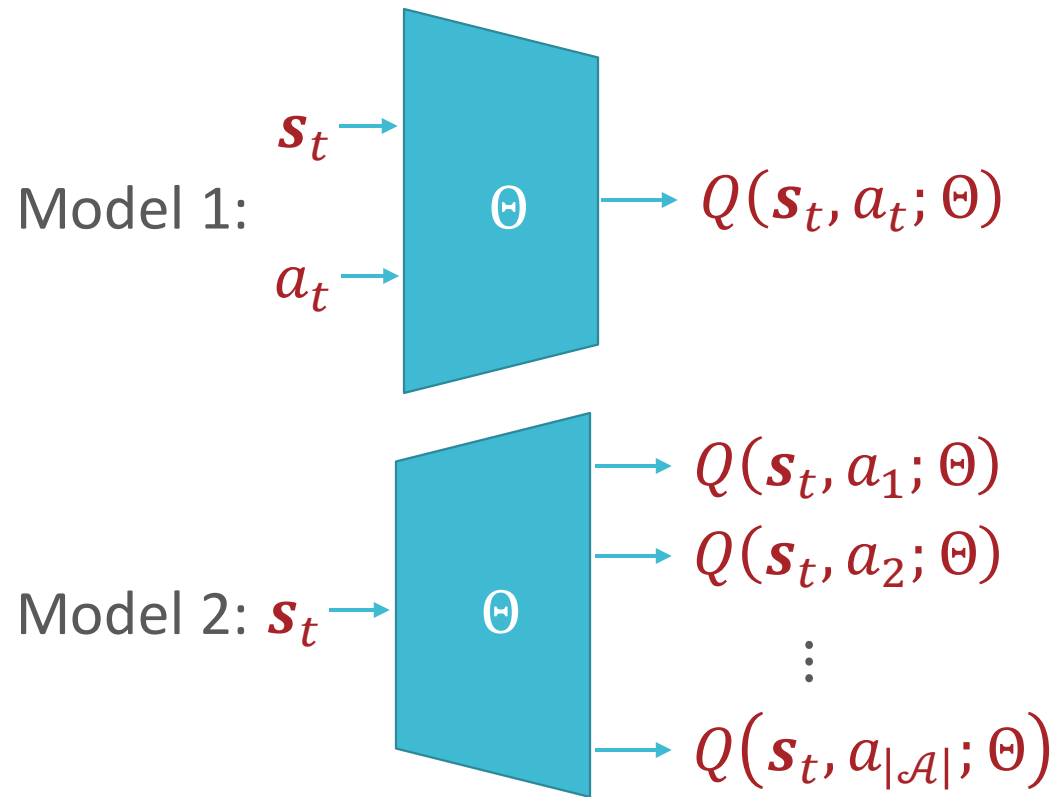
1. What can we do if the reward and/or transition functions/distributions are unknown?
  - Use online learning to gather data and learn  $Q^*(s, a)$
2. How can we handle infinite (or just very large) state/action spaces?
  - Throw a neural network at it!

# Deep Q-learning

- Use a parametric function,  $Q(s, a; \Theta)$ , to approximate  $Q^*(s, a)$ 
  - Learn the parameters using *stochastic* gradient descent (SGD)
  - Training data  $(\mathbf{s}_t, a_t, r_t, \mathbf{s}_{t+1})$  gathered online by the agent/learning algorithm

# Deep Q-learning: Model

- Represent states using some feature vector  $\mathbf{s}_t \in \mathbb{R}^M$   
e.g. for Go,  $\mathbf{s}_t = [1, 0, -1, \dots, 1]^T$
- Define a *differentiable* function that approximates  $Q$





# Deep Q-learning: Loss Function

- “True” loss

$$\ell(\Theta) = \sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} \underbrace{\left( \overbrace{Q^*(s, a)}^{2. \text{ Don't know } Q^*} - Q(s, a; \Theta) \right)^2}$$

1.  $\mathcal{S}$  too big to compute this sum

1. Use stochastic gradient descent: just consider one state-action pair in each iteration

2. Use temporal difference learning:

- Given current parameters  $\Theta^{(t)}$  the temporal difference target is

$$Q^*(s, a) \approx r + \gamma \max_{a'} Q(s', a'; \Theta^{(t)}) := y$$

- Set the parameters in the next iteration  $\Theta^{(t+1)}$  such that  $Q(s, a; \Theta^{(t+1)}) \approx y$

$$\ell(\Theta^{(t)}, \Theta^{(t+1)}) = \left( y - Q(s, a; \Theta^{(t+1)}) \right)^2$$

# Deep Q-learning

## Algorithm 4: Online learning (parametric form)

- Inputs: discount factor  $\gamma$ , an initial state  $s_0$ ,  
learning rate  $\alpha$
- Initialize parameters  $\Theta^{(0)}$
- For  $t = 0, 1, 2, \dots$ 
  - Gather training sample  $(s_t, a_t, r_t, s_{t+1})$
  - Update  $\Theta^{(t)}$  by taking a step opposite the gradient

$$\Theta^{(t+1)} \leftarrow \Theta^{(t)} - \alpha \nabla_{\Theta} \ell(\Theta^{(t)}, \Theta)$$

where

$$\nabla_{\Theta} \ell(\Theta^{(t)}, \Theta) = 2(y - Q(s, a; \Theta)) \nabla_{\Theta} Q(s, a; \Theta)$$

# Deep Q-learning: Experience Replay

- SGD assumes i.i.d. training samples but in RL, samples are *highly* correlated
- Idea: keep a “replay memory”  $\mathcal{D} = \{e_1, e_2, \dots, e_N\}$  of the  $N$  most recent experiences  $e_t = (s_t, a_t, r_t, s_{t+1})$  (Lin, 1992)
  - Also keeps the agent from “forgetting” about recent experiences
- Alternate between:
  1. Sampling some  $e_i$  uniformly at random from  $\mathcal{D}$  and applying a Q-learning update (repeat  $T$  times)
  2. Adding a new experience to  $\mathcal{D}$
- Can also sample experiences from  $\mathcal{D}$  according to some distribution that prioritizes experiences with high error (Schaul et al., 2016)

# Key Takeaways

- We can use (deep) Q-learning when the reward/transition functions are unknown and/or when the state/action spaces are too large to be modelled directly
  - Also guaranteed to converge under certain assumptions
  - Experience replay can help address non-i.i.d. samples