


Machine Learning Department
School of Computer Science
Carnegie Mellon University

# Q-Learning

# +

# Deep RL

Matt Gormley
Lecture 23
Nov. 15, 2023

# Reminders

- Homework 7: Transformer in PyTorch
  - Out: Sat, Nov. 11
  - Due: Mon, Nov. 20 at 11:59pm
- Homework 8: Reinforcement Learning
  - Out: Mon, Nov. 20
  - Due: Fri, Dec. 1 at 11:59pm

# VALUE ITERATION

# Value Iteration

**Algorithm 1** Value Iteration

**procedure** VALUEITERATION($R(s,a)$ reward function, $p(\cdot|s,a)$ transition probabilities)
    Initialize value function $V(s) = 0$ or randomly
    **while** not converged **do**
        **for** $s \in \mathcal{S}$ **do**
            $V(s) = \max_a R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V(s')$
    Let $\pi(s) = \operatorname{argmax}_a R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V(s'), \ \forall s$
    **return** $\pi$

Variant 1: without Q(s,a) table

# Value Iteration

**Algorithm 1** Value Iteration

**procedure** VALUEITERATION($R(s,a)$ reward function, $p(\cdot|s,a)$ transition probabilities)
    Initialize value function $V(s) = 0$ or randomly
    **while** not converged **do**
        **for** $s \in \mathcal{S}$ **do**
            **for** $a \in \mathcal{A}$ **do**
                $Q(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V(s')$
            $V(s) = \max_a Q(s,a)$
    Let $\pi(s) = \operatorname{argmax}_a Q(s,a), \ \forall s$
    **return** $\pi$

Variant 2: with Q(s,a) table

# POLICY ITERATION

# Policy Iteration

**Algorithm 1** Policy Iteration

1: **procedure** POLICYITERATION($R(s,a)$ reward function, $p(\cdot|s,a)$ transition probabilities)

2: Initialize policy $\pi$ randomly

3: **while** not converged **do**

4: Solve Bellman equations for fixed policy $\pi$

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

5: Improve policy $\pi$ using new value function

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

6: **return** $\pi$

# Policy Iteration

**Algorithm 1** Policy Iteration

1: **procedure** POLICYITERATION($R(s,a)$ ..., $p(\cdot|s,a)$ transition probabilities)

2: Initialize policy $\pi$ randomly

3: **while** not converged **do**

4: Solve Bellman equations for fixed policy $\pi$

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

5: Improve policy $\pi$ using new value function

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

6: **return** $\pi$

Compute value function for fixed policy is easy

System of |S| equations and |S| variables

Greedy policy w.r.t. current value function

Greedy policy might **remain the same** for a particular state if there is no better action

# STOCHASTIC REWARDS AND VALUE ITERATION

# Q&A

**Q:** What if the rewards are also stochastic?

**A:** No problem. Everything we've been doing here still works just fine.

The Q-Learning algorithm doesn't need to change at all.

Let's consider how value iteration would look slightly different though…

# RL: Components

**From the Environment (i.e. the MDP)**

- State space, $\mathcal{S}$
- Action space, $\mathcal{A}$
- Reward function, $R(s, a, s'),\ R: \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$
- Transition probabilities, $p(s' \mid s, a)$
  - Deterministic transitions:

$$p(s' \mid s, a) = \begin{cases} 1 \text{ if } \delta(s, a) = s' \\ 0 \text{ otherwise} \end{cases}$$

where $\delta(s, a)$ is a transition function

Markov Assumption

$$p(s_{t+1} \mid s_t, a_t, \ldots, s_1, a_1) = p(s_{t+1} \mid s_t, a_t)$$

**From the Model**

- Policy, $\pi: \mathcal{S} \to \mathcal{A}$
- Value function, $V^{\pi}: \mathcal{S} \to \mathbb{R}$
  - Measures the expected total payoff of starting in some state $s$ and *executing* policy $\pi$

# Markov Decision Processes (MDP)

In RL, the source of our data is an MDP:

1. Start in some initial state $s_0 \in \mathcal{S}$
2. For time step $t$:
    1. Agent observes state $s_t \in \mathcal{S}$
    2. Agent takes action $a_t \in \mathcal{A}$ where $a_t = \pi(s_t)$
    3. Agent receives reward $r_t \in \mathbb{R}$ where $r_t = R(s_t, a_t, s_{t+1})$
    4. Agent transitions to state $s_{t+1} \in \mathcal{S}$ where $s_{t+1} \sim p(s' \mid s_t, a_t)$
3. Total reward is $\sum_{t=0}^{\infty} \gamma^t r_t$
    - The value $\gamma$ is the "discount factor", a hyperparameter $0 < \gamma < 1$

- Makes the same Markov assumption we used for HMMs! The next state only depends on the current state and action.
- *Def.*: we **execute** a policy $\pi$ by taking action $a = \pi(s)$when in state $s$

# Optimal Value Function

For the optimal policy function $\pi^*$ we can compute its **value function** as:

$$\begin{aligned} V^{\pi^*}(s) &= V^*(s) \\ &= \mathbb{E}[R(s_0, \pi^*(s_0), s_1) + \gamma R(s_1, \pi^*(s_1), s_2) \\ &\quad + \gamma^2 R(s_2, \pi^*(s_2), s_3) \cdots \mid s_0 = s, \pi^*]. \end{aligned}$$

This **optimal value function** can be represented recursively as:

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

If $R(s,a,s') = R(s,a)$ (deterministic transition), then we have the form:

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

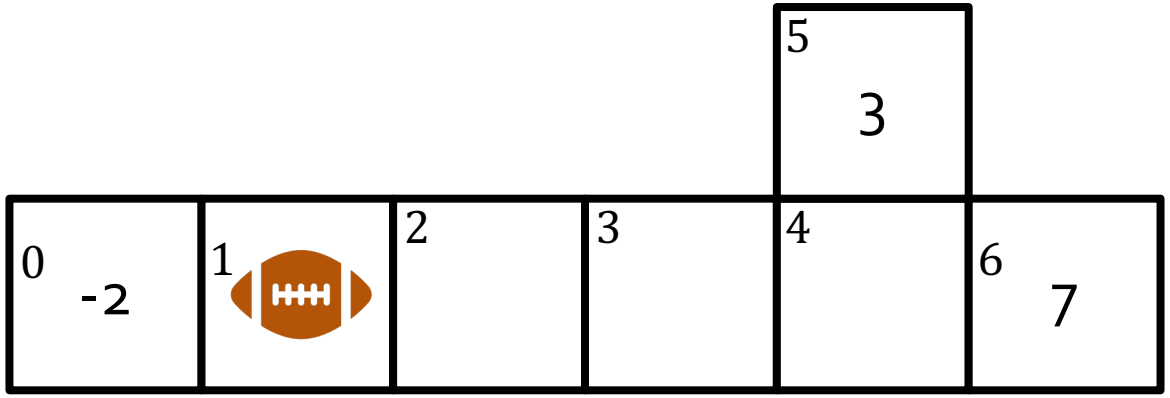
# Value Iteration

**Algorithm 1** Value Iteration with Stochastic Rewards

**procedure** VALUEITERATION($R(s, a, s')$ reward function, $p(\cdot|s, a)$ transition probabilities)
    Initialize value function $V(s) = 0$ or randomly
    **while** not converged **do**
        **for** $s \in \mathcal{S}$ **do**
            $V(s) = \max_a \sum_{s' \in \mathcal{S}} p(s'|s, a)(R(s, a, s') + \gamma V(s'))$
    Let $\pi(s) = \operatorname{argmax}_a \sum_{s' \in \mathcal{S}} p(s'|s, a)(R(s, a, s') + \gamma V(s')), \ \forall s$
    **return** $\pi$

This is just fixed point iteration applied to the recursive definition of the optimal value function.

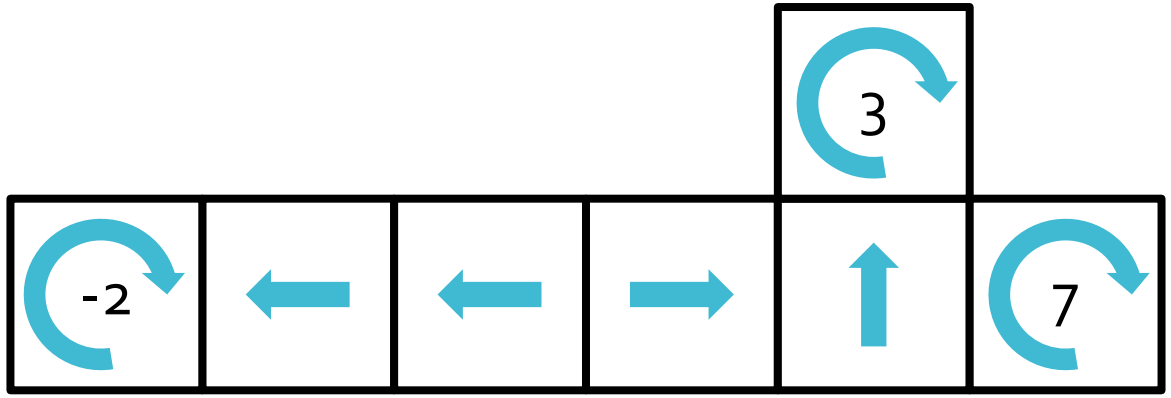
# RL: Value Function Example




$$R(s,a) = \begin{cases} -2 \text{ if entering state 0 (safety)} \\ 3 \text{ if entering state 5 (field goal)} \\ 7 \text{ if entering state 6 (touch down)} \\ 0 \text{ otherwise} \end{cases}$$

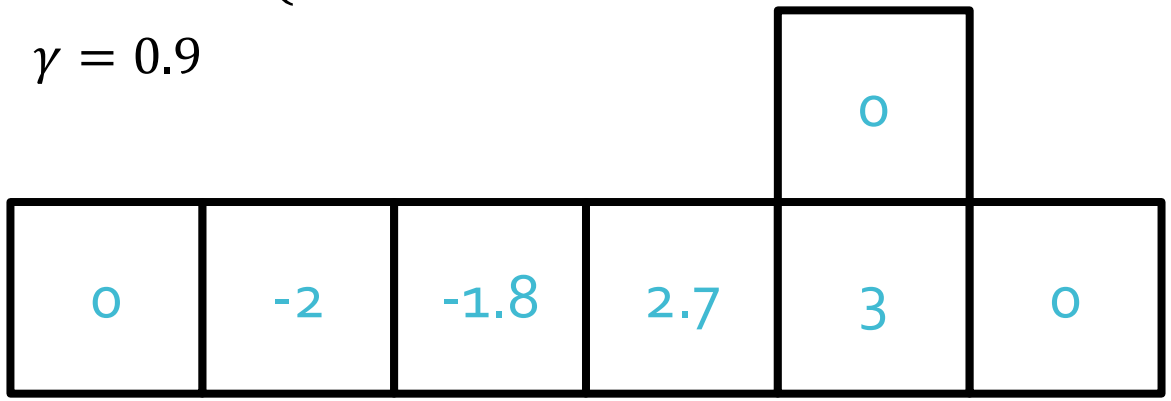
$\gamma = 0.9$

# RL: Value Function Example

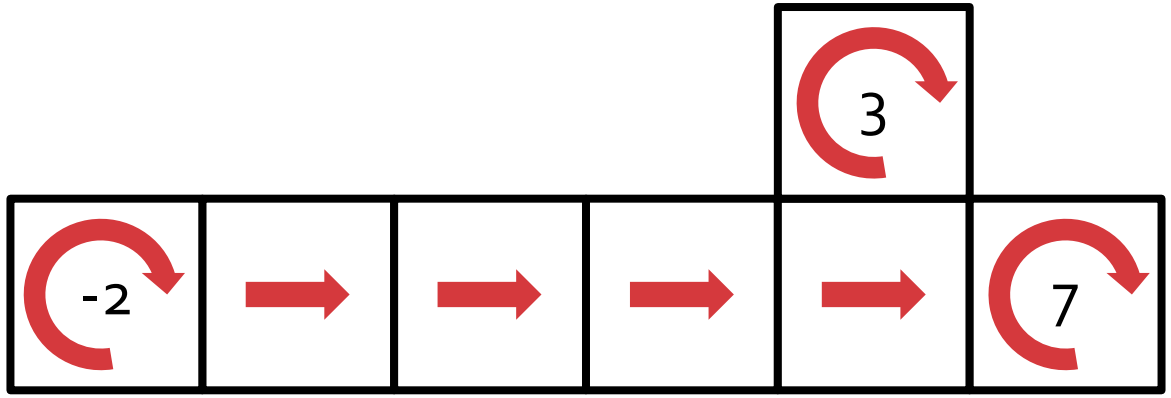



$$R(s,a) = \begin{cases} -2 \text{ if entering state 0 (safety)} \\ 3 \text{ if entering state 5 (field goal)} \\ 7 \text{ if entering state 6 (touch down)} \\ 0 \text{ otherwise} \end{cases}$$

$\gamma = 0.9$

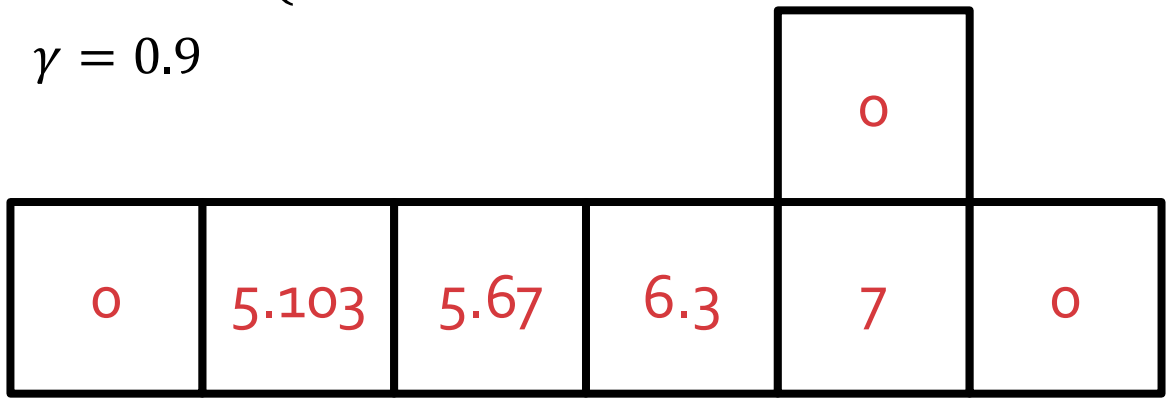


# RL: Value Function Example

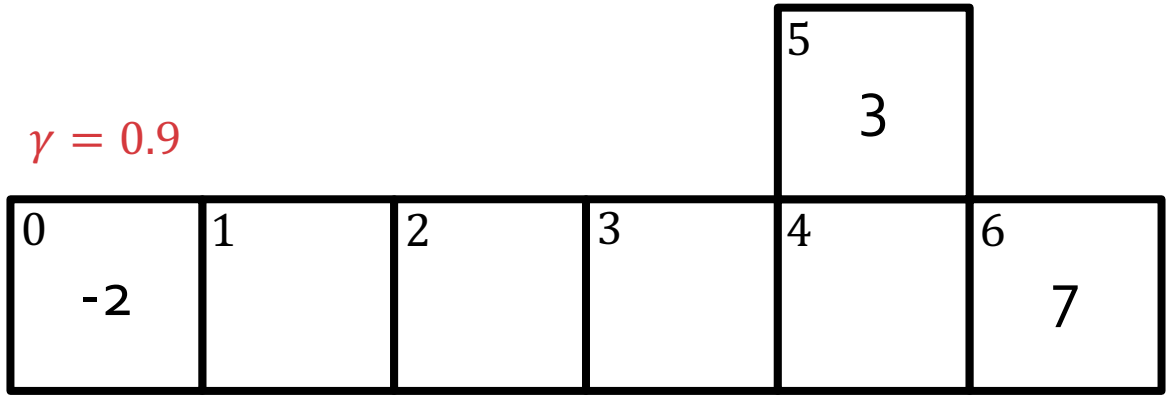



$$R(s,a) = \begin{cases} -2 \text{ if entering state 0 (safety)} \\ 3 \text{ if entering state 5 (field goal)} \\ 7 \text{ if entering state 6 (touch down)} \\ 0 \text{ otherwise} \end{cases}$$

$\gamma = 0.9$



# Example: Stochastic Transitions and Rewards

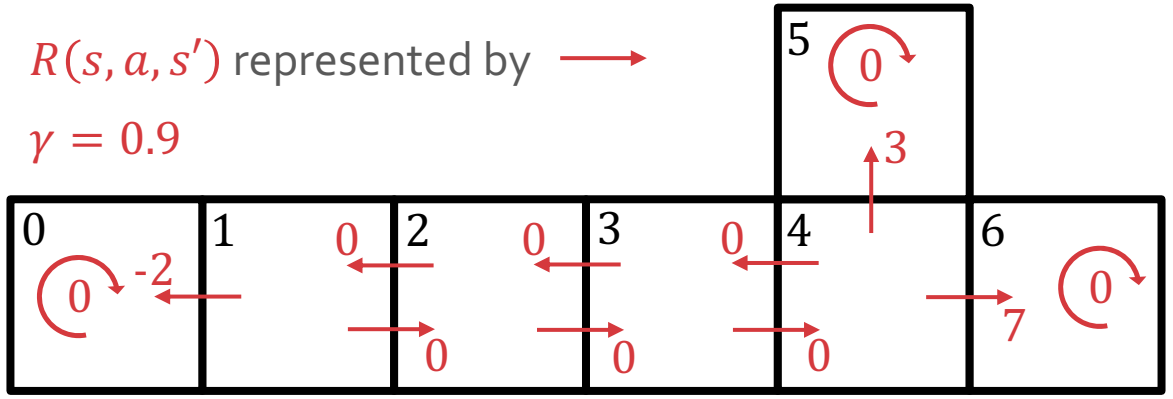



$$R(s, a, s') = \begin{cases} -2 \text{ if entering state 0 (safety)} \\ 3 \text{ if entering state 5 (field goal)} \\ 7 \text{ if entering state 6 (touch down)} \\ 0 \text{ otherwise} \end{cases}$$

# Example: Stochastic Transitions and Rewards

$R(s, a, s')$ represented by →

$\gamma = 0.9$

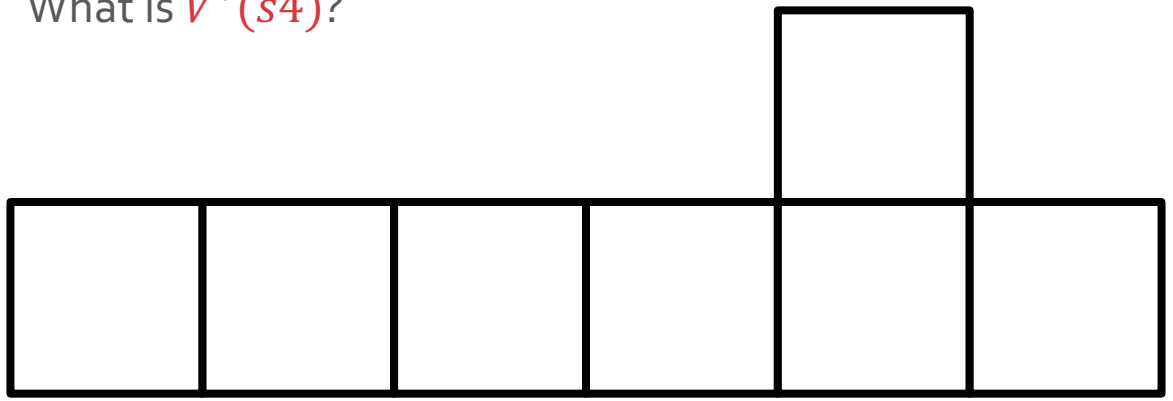



Suppose

- $p(s6 \mid s4,\ a) = 0.5$
- $p(s5 \mid s4,\ a) = 0.5$

$\gamma = 0.9$

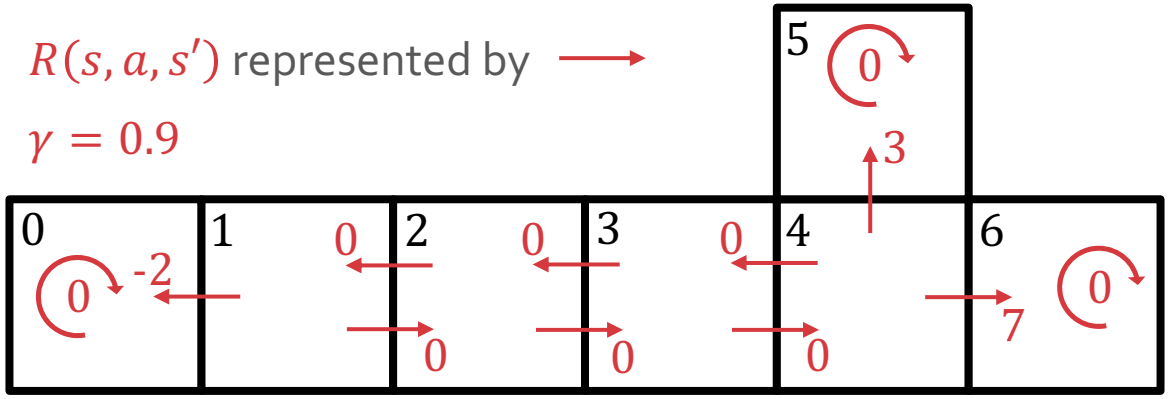
What is $V^*(s4)$?



# Example: Stochastic Transitions and Rewards

$R(s, a, s')$ represented by →

$\gamma = 0.9$

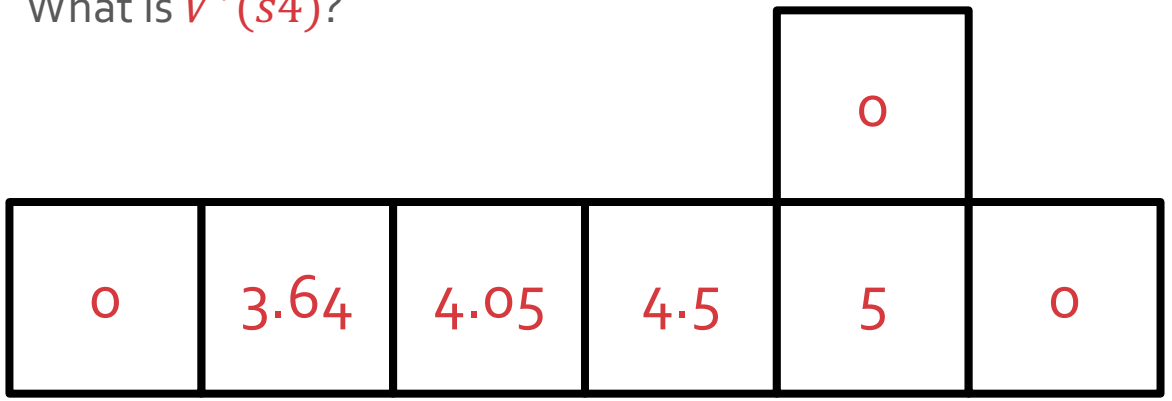



Suppose

- $p(s6 \mid s4,\ a) = 0.5$
- $p(s5 \mid s4,\ a) = 0.5$

$\gamma = 0.9$

What is $V^*(s4)$?



# Learning Objectives

**Reinforcement Learning: Value and Policy Iteration**

*You should be able to…*

1. Compare the reinforcement learning paradigm to other learning paradigms
2. Cast a real-world problem as a Markov Decision Process
3. Depict the exploration vs. exploitation tradeoff via MDP examples
4. Explain how to solve a system of equations using fixed point iteration
5. Define the Bellman Equations
6. Show how to compute the optimal policy in terms of the optimal value function
7. Explain the relationship between a value function mapping states to expected rewards and a value function mapping state-action pairs to expected rewards
8. Implement value iteration
9. Implement policy iteration
10. Contrast the computational complexity and empirical convergence of value iteration vs. policy iteration
11. Identify the conditions under which the value iteration algorithm will converge to the true value function
12. Describe properties of the policy iteration algorithm

# Q-LEARNING

What can we do if we don’t know the reward function / transition probabilities?

# Today's lecture is brought to you by the letter Q


Source: https://en.wikipedia.org/wiki/Avenue_Q#/media/File:Image-AvenueQlogo.png

# Today's lecture is brought to you by the letter Q

Source: https://vignette1.wikia.nocookie.net/jamesbond/images/9/9a/The_Four_Qs_-_Profile_(2).png/revision/latest?cb=20121102195112

# Today's lecture is brought to you by the letter Q

Source: https://www.npr.org/2017/06/03/531044118/there-may-not-be-flying-but-quidditch-still-creates-magic

# Value Iteration

**Algorithm 1** Value Iteration

**procedure** VALUEITERATION($R(s,a)$ reward function, $p(\cdot|s,a)$ transition probabilities)
    Initialize value function $V(s) = 0$ or randomly
    **while** not converged **do**
        **for** $s \in \mathcal{S}$ **do**
            **for** $a \in \mathcal{A}$ **do**
                $Q(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V(s')$
            $V(s) = \max_a Q(s,a)$
    Let $\pi(s) = \text{argmax}_a \, Q(s,a), \ \forall s$
    **return** $\pi$

Variant 1: with Q(s,a) table

# $Q^*(s, a)$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

- Algorithm 1: Online learning of $Q^*$ (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 $|\mathcal{S}| \times |\mathcal{A}|$ table or array)
  - While TRUE, do
    - Take a random action $a$
    - Receive some reward $r = R(s,a)$
    - Observe the new state $s' = \delta(s,a)$
    - Update $Q$ and $s$

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

      $s \leftarrow s'$

Online gathering of training sample $(s,a,r,s')$

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

- Algorithm 2: $\epsilon$-greedy online learning of $Q^*$ (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 $|\mathcal{S}| \times |\mathcal{A}|$ table or array)
  - While TRUE, do
    - With probability $1-\epsilon$, take the greedy action $a = \underset{a' \in \mathcal{A}}{\text{argmax}} \; Q(s,a')$. Otherwise (with probability $\epsilon$), take a random action $a$
    - Receive reward $r = R(s,a)$
    - Observe the new state $s' = \delta(s,a)$
    - Update $Q$ and $s$
      $Q(s,a) \leftarrow r + \gamma \max_{a'} Q(s',a')$
      $s \leftarrow s'$

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

- Algorithm 3: $\epsilon$-greedy online learning of $Q^*$ (table form)
  - Inputs: discount factor $\gamma$,
    an initial state $s$,
    greediness parameter $\epsilon \in [0,1]$,
    learning rate $\alpha \in [0,1]$ ("mistrust parameter")
  - Initialize $Q(s,a) = 0 \; \forall \; s \in \mathcal{S}, a \in \mathcal{A}$
    ($Q$ is a $|\mathcal{S}| \times |\mathcal{A}|$ table or array)
  - While TRUE, do
    - With probability $1-\epsilon$, take the greedy action $a = \underset{a' \in \mathcal{A}}{\text{argmax}} \; Q(s,a')$. Otherwise (with probability $\epsilon$), take a random action $a$
    - Receive reward $r = R(s,a)$
    - Observe the new state $s' \sim p(S' \mid s,a)$
    - Update $Q$ and $s$

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

$$s \leftarrow s'$$

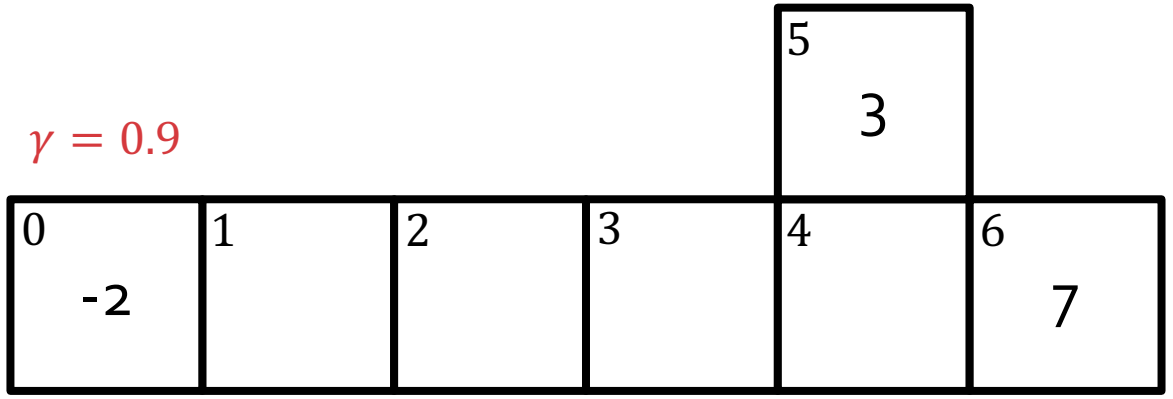
# Learning $Q^*(s,a)$ via temporal difference learning

- Algorithm 3: $\epsilon$-greedy online learning of $Q^*$ (table form)
  - Inputs: discount factor $\gamma$,
    an initial state $s$,
    greediness parameter $\epsilon \in [0,1]$,
    learning rate $\alpha \in [0,1]$ ("mistrust parameter")
  - Initialize $Q(s,a) = 0 \; \forall \; s \in \mathcal{S}, a \in \mathcal{A}$
    ($Q$ is a $|\mathcal{S}| \times |\mathcal{A}|$ table or array)
  - While TRUE, do
    - With probability $1-\epsilon$, take the greedy action $a = \underset{a' \in \mathcal{A}}{\text{argmax}} \; Q(s,a')$. Otherwise (with probability $\epsilon$), take a random action $a$
    - Receive reward $r = R(s,a)$
    - Observe the new state $s' \sim p(S' \mid s,a)$
    - Update $Q$ and $s$

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

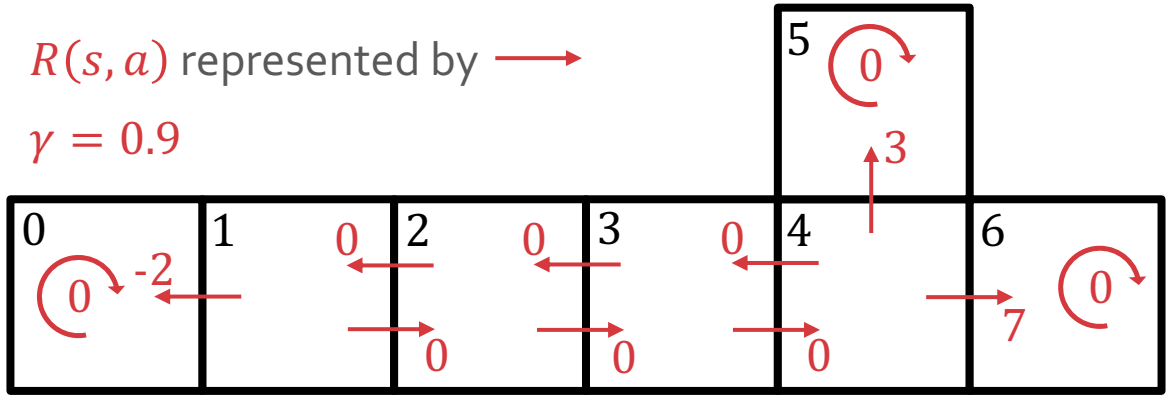
$$s \leftarrow s'$$

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

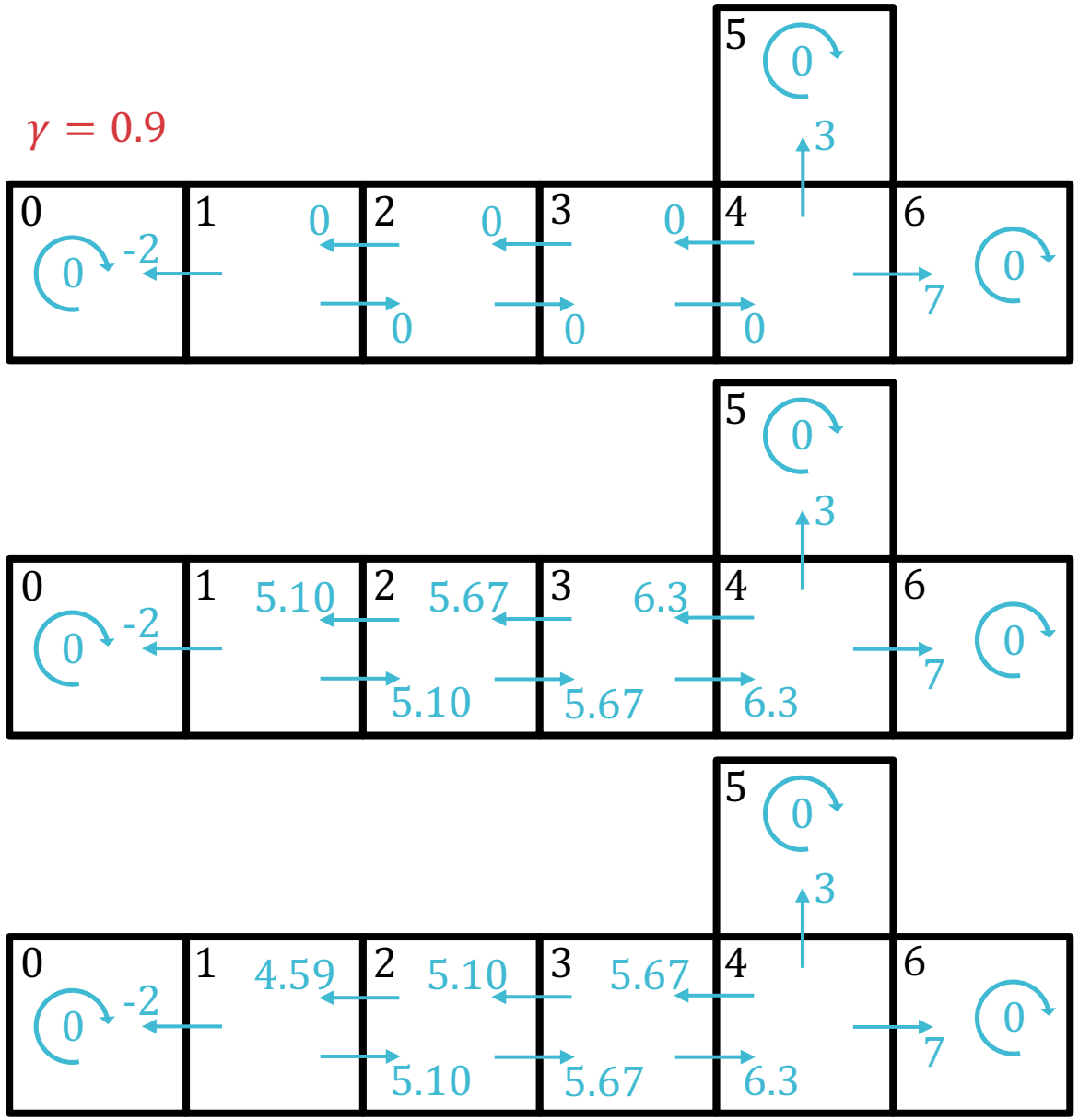



$$R(s,a) = \begin{cases} -2 \text{ if entering state 0 (safety)} \\ 3 \text{ if entering state 5 (field goal)} \\ 7 \text{ if entering state 6 (touch down)} \\ 0 \text{ otherwise} \end{cases}$$

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

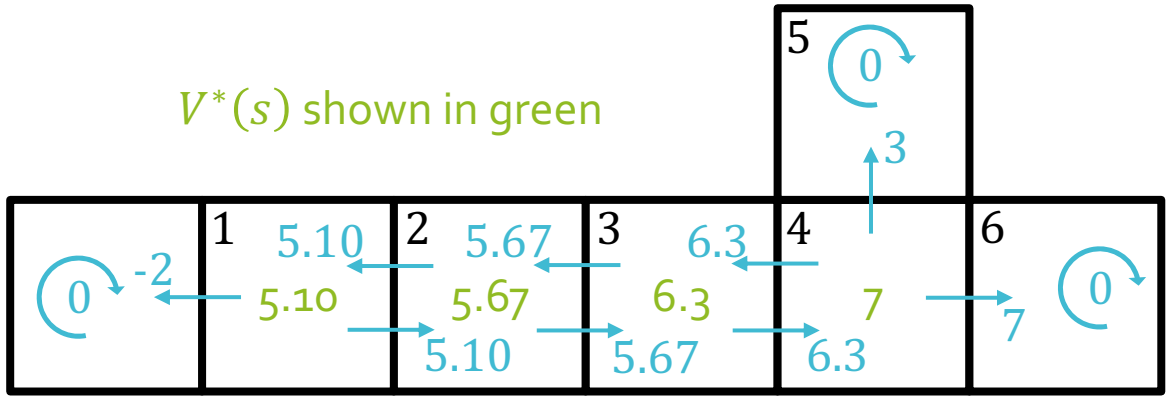


# Poll: Which set of blue arrows (roughly) corresponds to $Q^*(s, a)$?



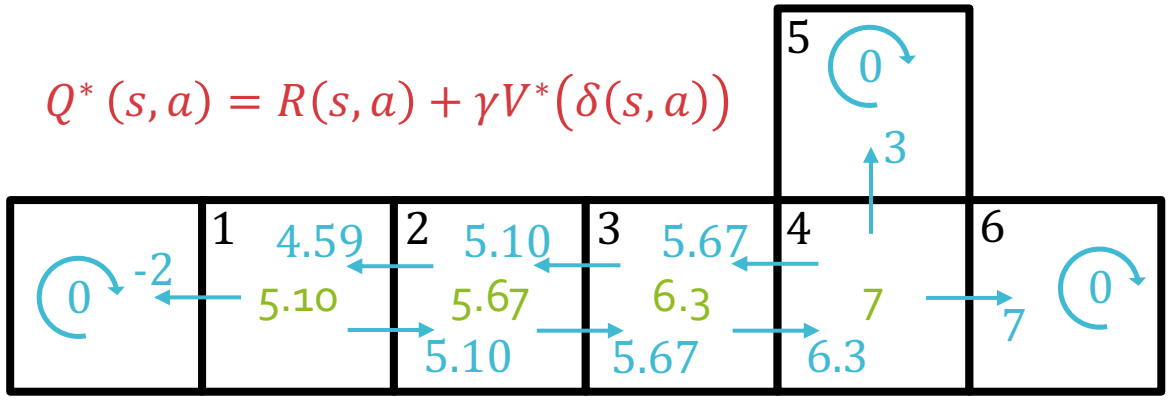
# Poll: Which set of blue arrows corresponds to $Q^*(s,a)$?

C.

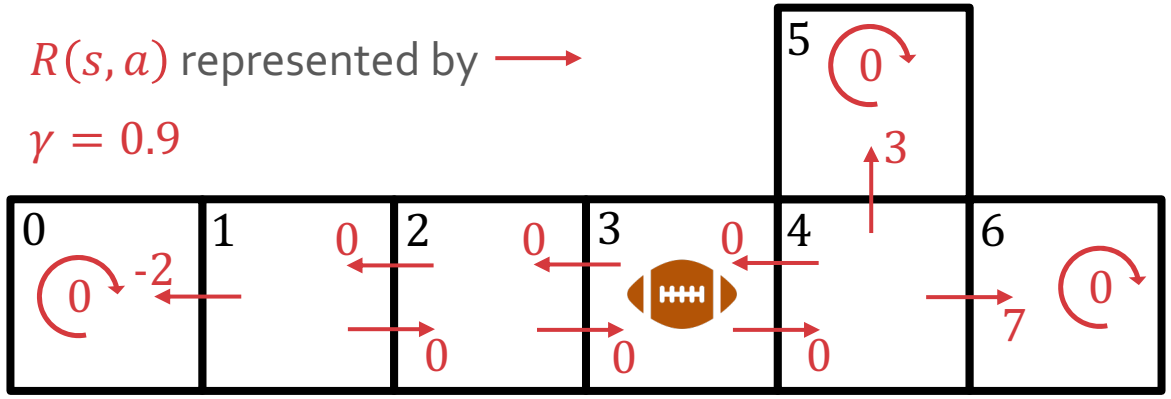



$Q^*(s,a) = R(s,a) + \gamma V^*(\delta(s,a))$

D.



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

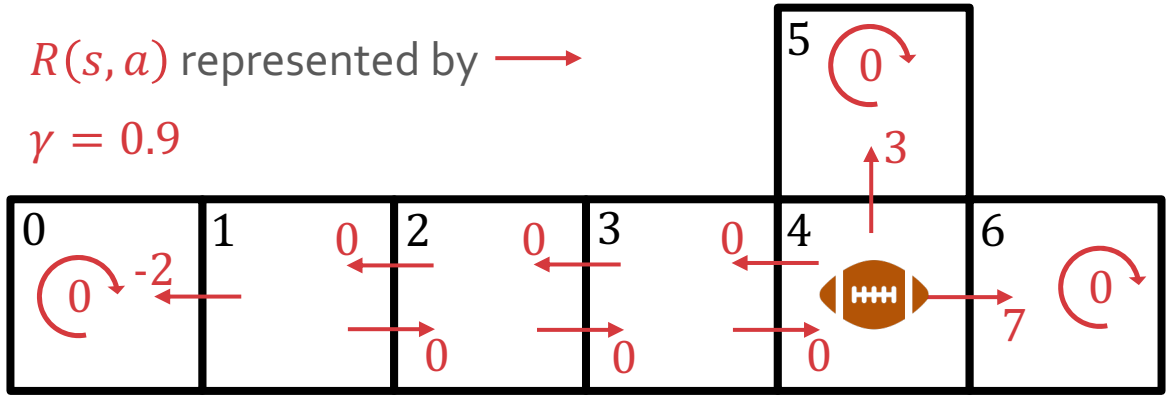



| $Q(s,a)$ | → | ← | ↑ | ↻ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 |

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

$R(s, a)$ represented by →

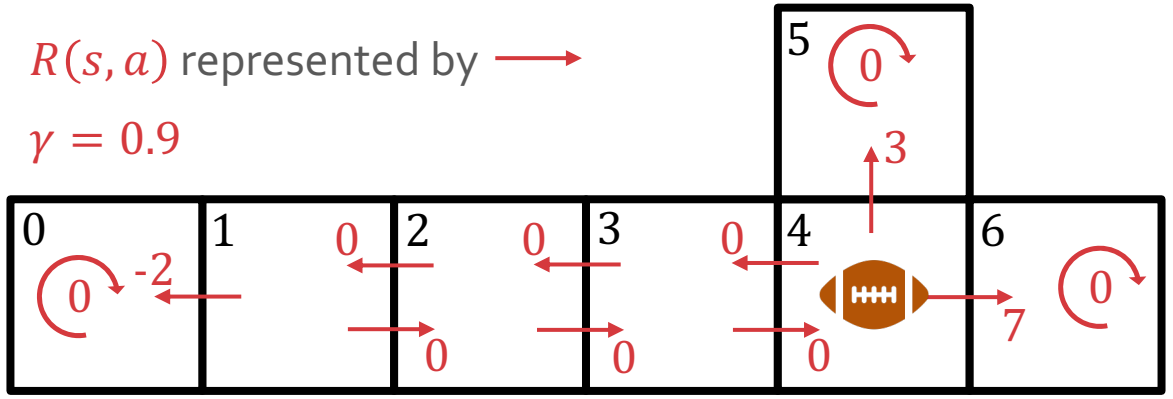
$\gamma = 0.9$




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

| $Q(s, a)$ | → | ← | ↑ | ↻ |
|---|---|---|---|---|
| 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

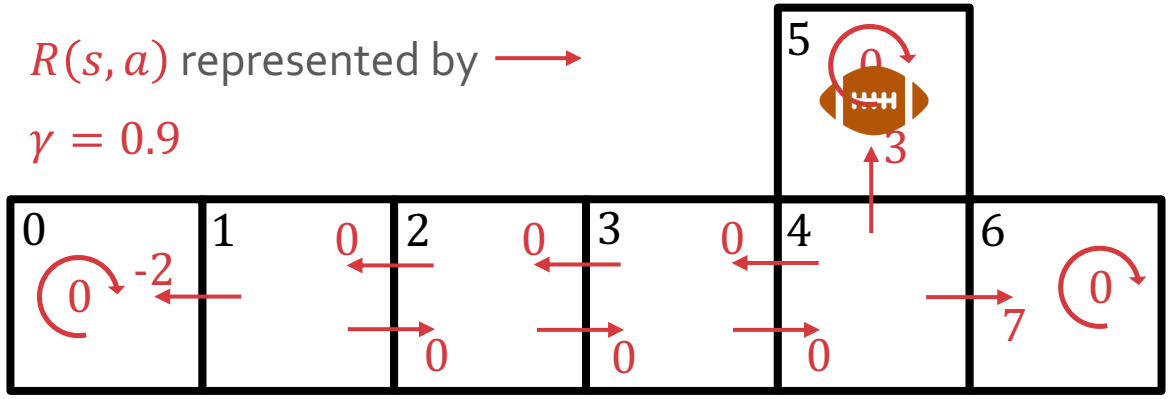



| $Q(s,a)$ | → | ← | ↑ | ↻ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 |

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

$R(s,a)$ represented by →

$\gamma = 0.9$



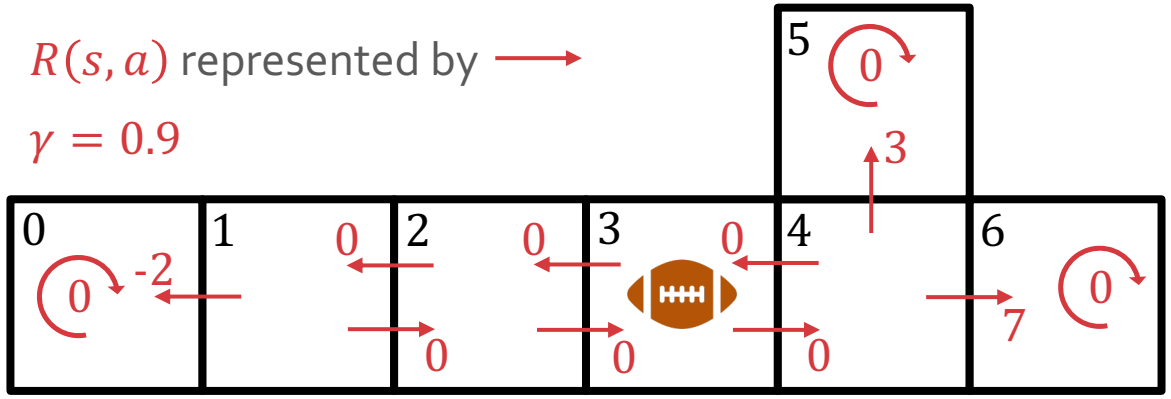

$$Q(4,\uparrow) \leftarrow 3 + (0.9) \max_{a' \in \{\rightarrow,\leftarrow,\uparrow,\circlearrowright\}} Q(5,a') = 3$$

| $Q(s,a)$ | → | ← | ↑ | ↻ |
|---|---|---|---|---|
| 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

$R(s,a)$ represented by ⟶

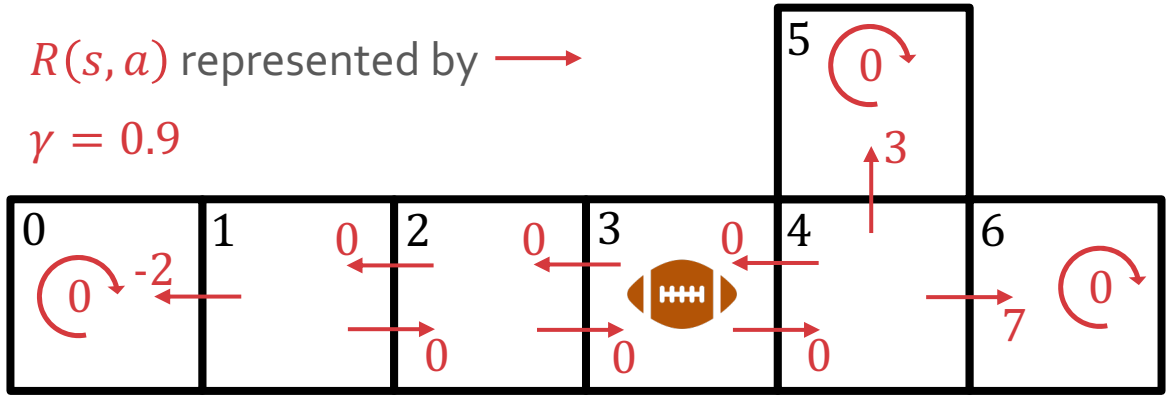
$\gamma = 0.9$




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

| $Q(s,a)$ | $\rightarrow$ | $\leftarrow$ | $\uparrow$ | $\circlearrowright$ |
|---|---|---|---|---|
| 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




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

| $Q(s,a)$ | $\rightarrow$ | $\leftarrow$ | $\uparrow$ | $\circlearrowright$ |
|---|---|---|---|---|
| 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 |

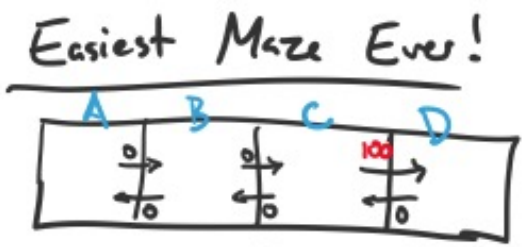
# Q-Learning Convergence

**Remarks**

- **Q converges to Q*** with probability 1.0, assuming…
  1. each <s, a> is visited infinitely often
  2. $0 \leq \gamma < 1$
  3. rewards are bounded $|R(s,a)| < \beta$, for all <s,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 $\sum_{t=0}^{\infty} \alpha_t^2 = 0$, e.g., $\alpha_t = {}^{1}/_{t+1}$
- Q-Learning is **exploration insensitive**
  ⇒ visiting the states in any order will work assuming point 1 is satisfied
- May take **many iterations** to converge in practice

# Reordering Experiences

**Ex: Easiest Maze Ever!**




arrows are $R(s,a)$

$\gamma = 0.9$

$S = \{A, B, C, D\}$

$A = \{E, W\}$

$Q(s,a) = 0$ at start

① suppose we visit

| i | s | a | r | s' |
|---|---|---|---|---|
| 1 | A | E | 0 | B |
| 2 | B | E | 0 | C |
| 3 | C | E | 100 | D |

$Q(A,E) = 0$

$Q(B,E) = 0$

$Q(C,E) = 100$

② suppose we visit in reverse order

| i | s | a | r | s' |
|---|---|---|---|---|
| 1 | C | E | 100 | D |
| 2 | B | E | 0 | C |
| 3 | A | E | 0 | B |

$Q(C,E) = 100$

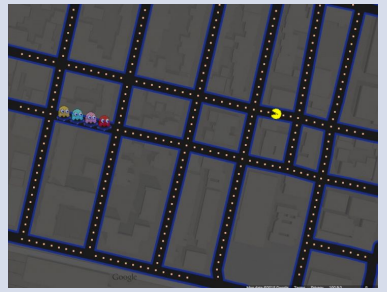
$Q(B,E) = 90$

$Q(A,E) = 81$

# Designing State Spaces

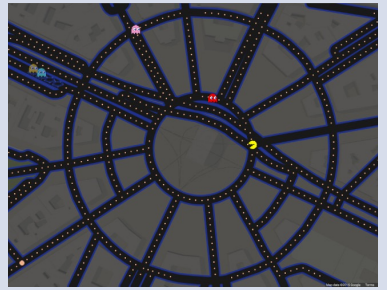
**Q:** Do we have to retrain our RL agent every time we change our state space?

**A:** Yes. But whether your state space changes from one setting to another is determined by your design of the state representation.



Two examples:

- State Space A: <x,y> position on map
  e.g. $s_t$ = <74, 152>
- State Space B: window of pixel colors centered at current Pac Man location
  e.g. $s_t$ =



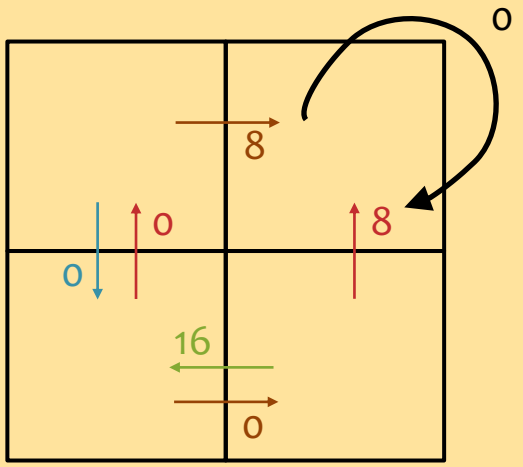
| | | |
|---|---|---|
| 0 | 1 | 0 |
| 0 | 0 | 0 |
| 1 | 1 | 1 |

# Poll: Q-Learning

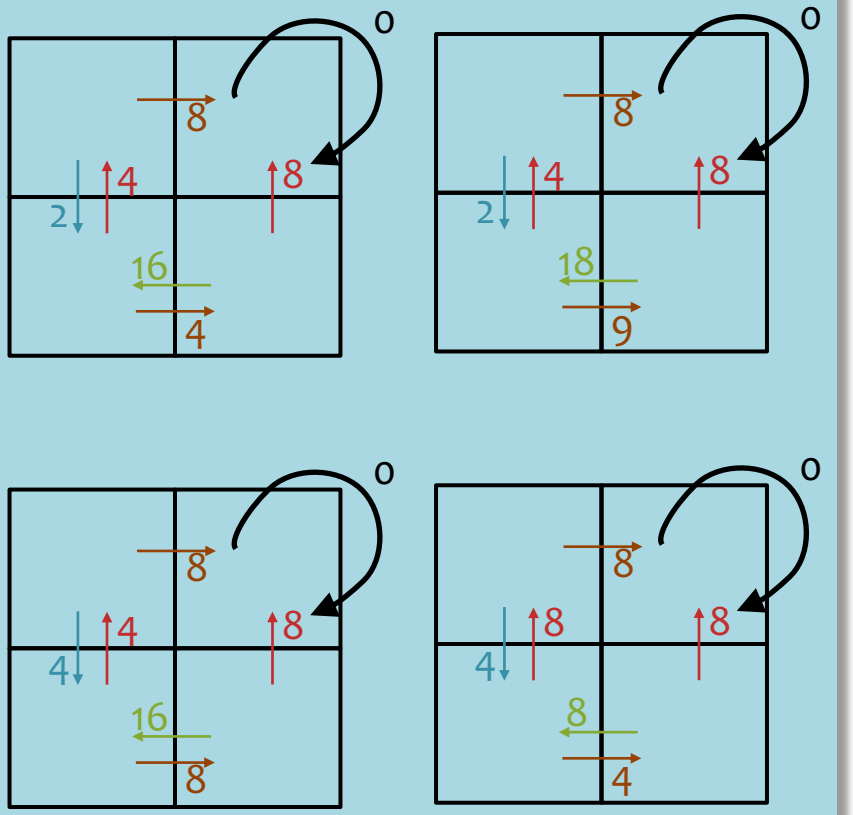
**Question:**

For the R(s,a) values shown on the arrows below, which are the corresponding Q*(s,a) values?

Assume discount factor = 0.5.



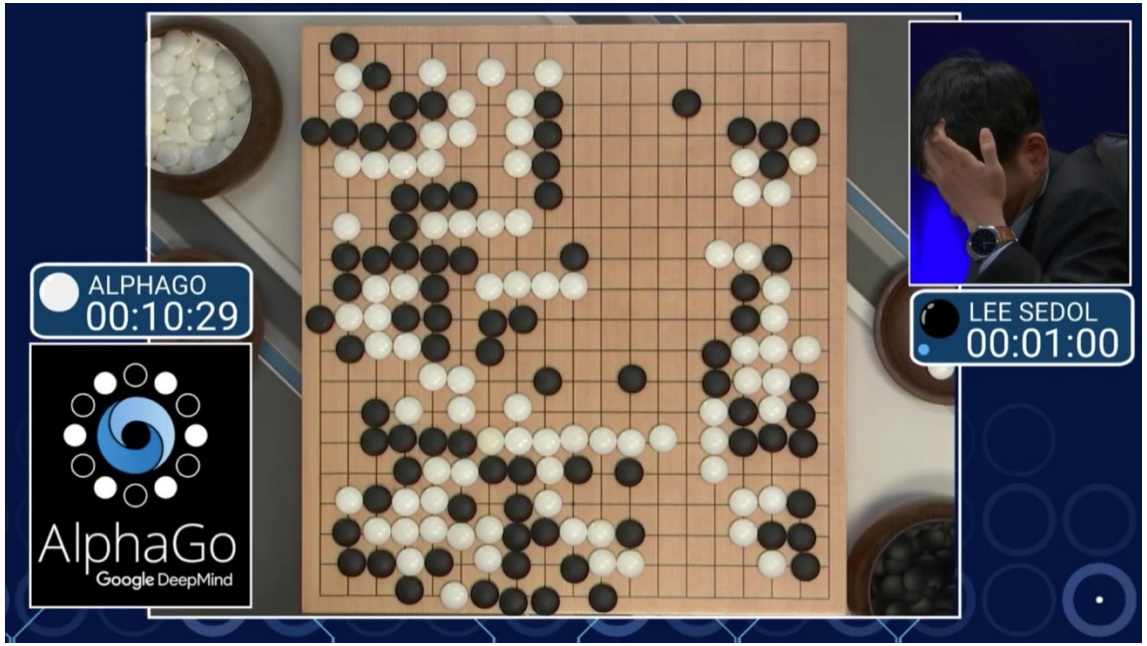

**Answer:**

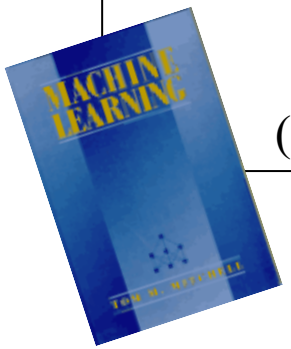


# DEEP RL FOR GAME OF GO

# TD Gammon → Alpha Go

## Learning to beat the masters at board games

| THEN | NOW |
| --- | --- |
| "…the world's top computer program for backgammon, TD-GAMMON (Tesauro, 1992, 1995), learned its strategy by playing over one million practice games against itself…" |   |



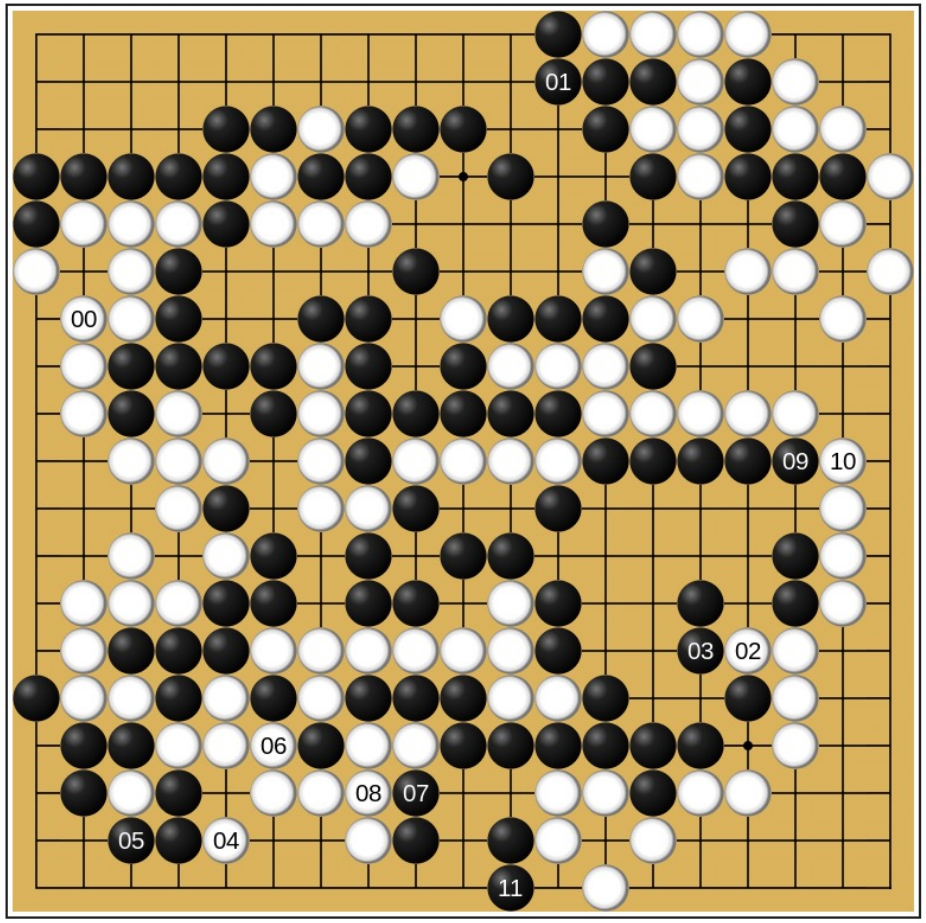

(Mitchell, 1997)

# Alpha Go

**Game of Go (圍棋)**

- 19x19 **board**
- Players alternately play black/white **stones**
- **Goal** is to fully encircle the largest region on the board
- **Simple** rules, but **extremely complex** game play

AlphaGo (Black) vs. Lee Sedol (White) - Game 2
Final position (AlphaGo wins in 211 moves)




Source: https://en.wikipedia.org/wiki/AlphaGo_versus_Lee_Sedol

# Alpha Go

- State space is too large to represent explicitly since # of sequences of moves is $O(b^d)$
  - Go: $b=250$ and $d=150$
  - Chess: $b=35$ and $d=80$
- Key idea:
  - Define a neural network to approximate the value function
  - Train by policy gradient

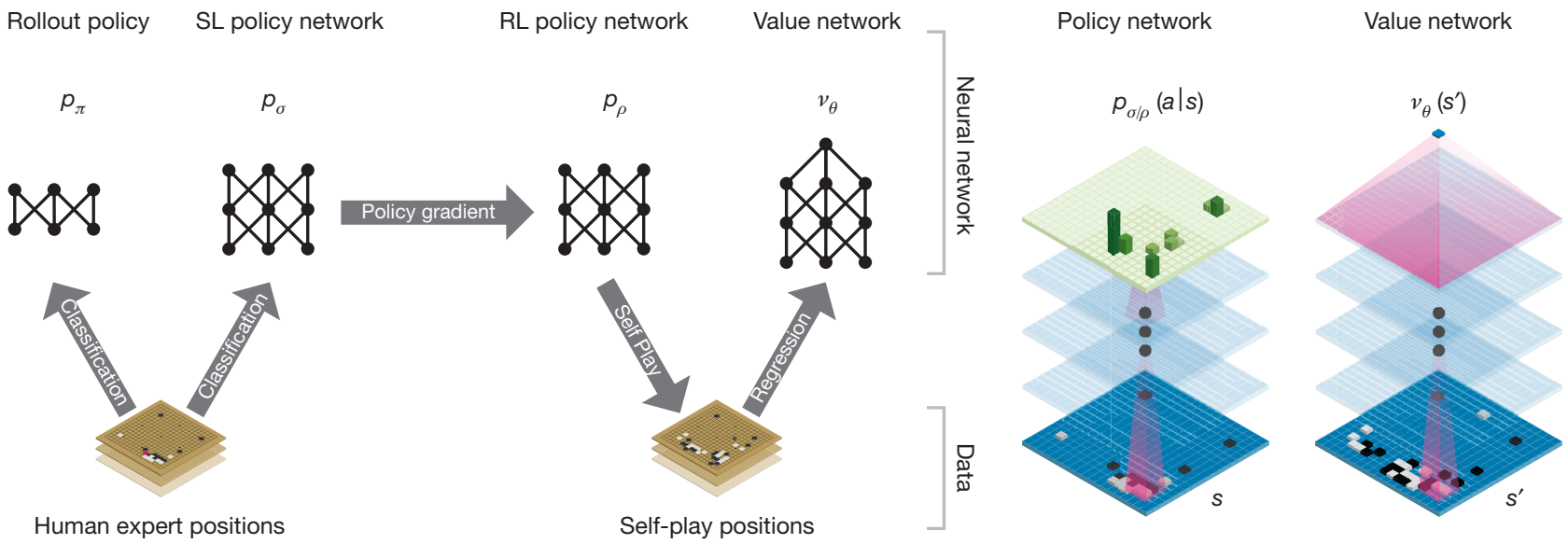



Figure from Silver et al. (2016)

# Alpha Go

- Results of a tournament
- From Silver et al. (2016): "a 230 point gap corresponds to a 79% probability of winning"

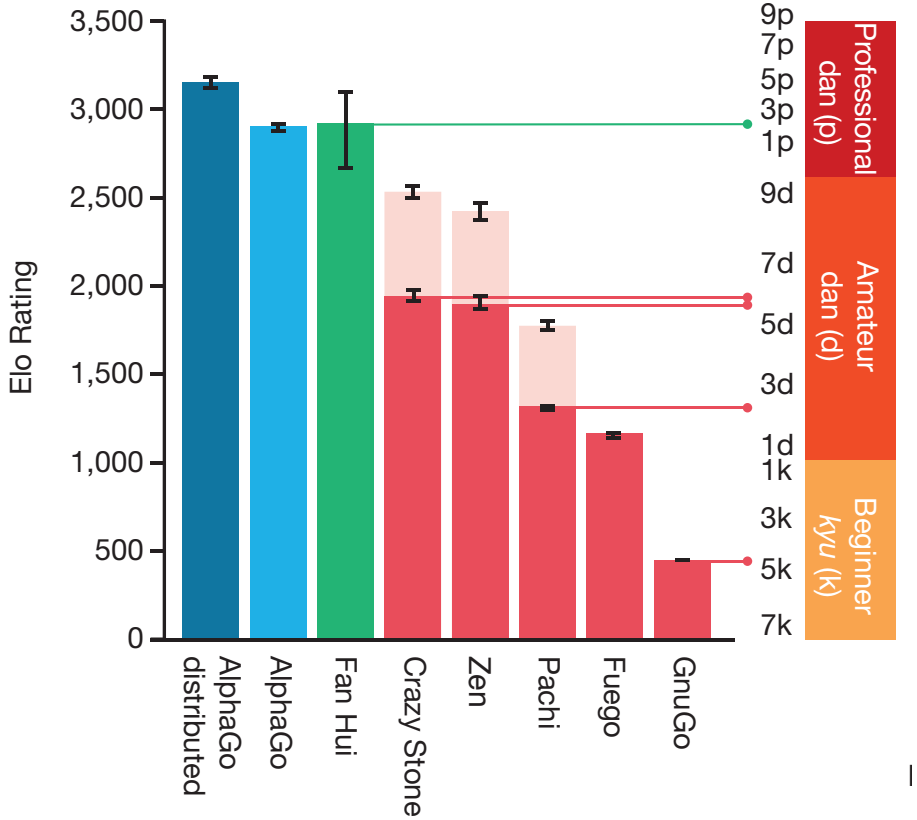



Figure from Silver et al. (2016)

# DEEP Q-LEARNING

# Deep Q-Learning

**Question:** *What if our state space S is too large to represent with a table?*

**Examples:**

- $s_t$ = pixels of a video game
- $s_t$ = continuous values of a sensors in a manufacturing robot
- $s_t$ = sensor output from a self-driving car

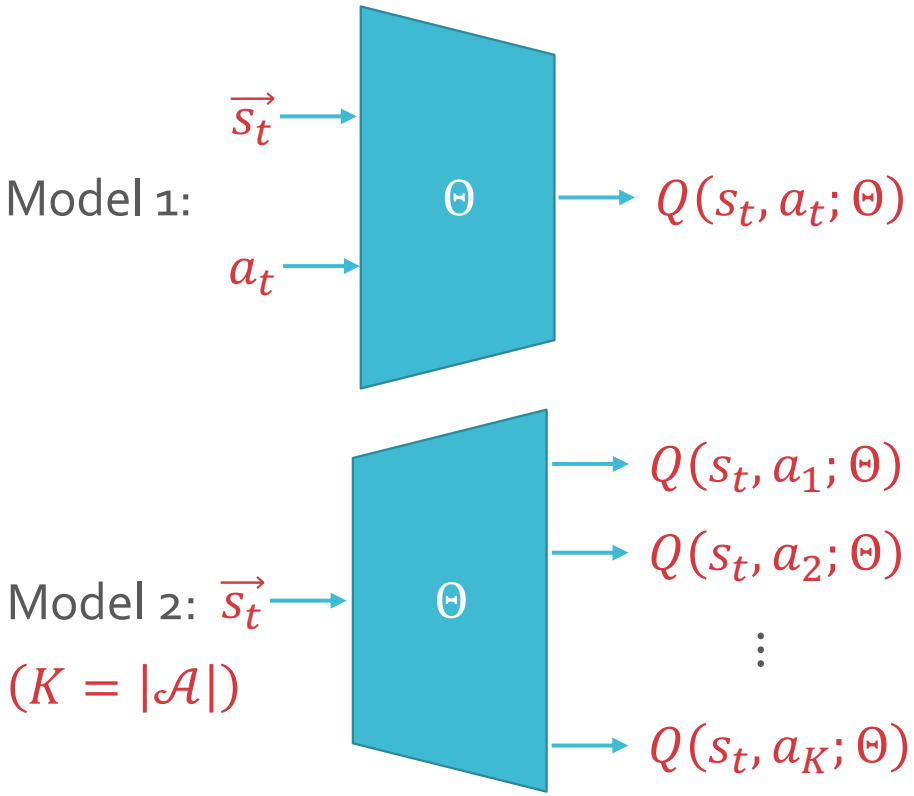
**Answer:** Use a parametric function to approximate the table entries

**Key Idea:**

1. Use a neural network $Q(s,a; \theta)$ to approximate $Q^*(s,a)$
2. Learn the parameters $\theta$ via SGD with training examples $< s_t, a_t, r_t, s_{t+1} >$

# Deep Q-learning: Model

- Represent states using some feature vector $\overrightarrow{s_t} \in \mathbb{R}^M$
  e.g., $\overrightarrow{s_t} = [1, 0, 0, \ldots, 1]^T$
- Define a neural network



# Deep Q-learning: Model

- Represent states using some feature vector $\overrightarrow{s_t} \in \mathbb{R}^M$ e.g., $\overrightarrow{s_t} = [1, 0, 0, \dots, 1]^T$
- Define ~~a neural network~~ a bunch of linear regressors (technically still neural networks…), one for each action (let $K = |\mathcal{A}|$)

$$Q(\vec{s}, a_k; \Theta) = \overrightarrow{\theta_k}^T \vec{s} \text{ where } \Theta = \begin{bmatrix} \overrightarrow{\theta_1} \\ \overrightarrow{\theta_2} \\ \vdots \\ \overrightarrow{\theta_K} \end{bmatrix} \in \mathbb{R}^{K \times M}$$

- Goal: $K \times M \ll |\mathcal{S}| \rightarrow$ computational tractability!

- Gradients are easy: $\nabla_{\overrightarrow{\theta_j}} Q(\vec{s}, a_k; \Theta) = \begin{cases} \vec{0} & \text{if } j \neq k \\ \vec{s} & \text{if } j = k \end{cases}$

# Deep Q-learning: Model

- Represent states using some feature vector $\overrightarrow{s_t} \in \mathbb{R}^M$
  e.g., $\overrightarrow{s_t} = [1, 0, 0, \ldots, 1]^T$
- Define ~~a neural network~~ a bunch of linear regressors (technically still neural networks…), one for each action (let $K = |\mathcal{A}|$)

$$Q(\vec{s}, a_k; \Theta) = \overrightarrow{\theta_k}^T \vec{s} \text{ where } \Theta = \begin{bmatrix} \overrightarrow{\theta_1} \\ \overrightarrow{\theta_2} \\ \vdots \\ \overrightarrow{\theta_K} \end{bmatrix} \in \mathbb{R}^{K \times M}$$

- Goal: $K \times M \ll |\mathcal{S}| \rightarrow$ computational tractability!
- Gradients are easy: $\nabla_\Theta Q(\vec{s}, a_k; \Theta) = \begin{bmatrix} \vec{0} \\ \vec{0} \\ \vdots \\ \vec{s} \\ \vdots \\ \vec{0} \end{bmatrix}$ ← Row $k$

# Deep Q-learning: Loss Function

2. Don't know $Q^*$

- "True" loss

$$\ell(\Theta) = \sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} \left(Q^*(s,a) - Q(s,a;\Theta)\right)^2$$

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

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

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

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

# Deep Q-learning

- Algorithm 4: Online learning of $Q^*$ (parametric form)
  - Inputs: discount factor $\gamma$,

    an initial state $s_0$,

    learning rate $\alpha$
  - Initialize parameters $\Theta^{(0)}$
  - For $t = 0, 1, 2, \ldots$
    - Gather training sample $(s_t, a_t, r_t, s_{t+1})$
    - Update $\Theta^{(t)}$ by taking a step opposite the gradient
    - $\Theta^{(const)} \leftarrow \Theta^{(t)}$

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

where

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

$$= 2\left(r + \gamma \max_{a'} Q\left(s', a'; \Theta^{(const)}\right) - Q\left(s, a; \Theta^{(t)}\right)\right) \nabla_{\Theta^{(t)}} Q\left(s, a; \Theta^{(t)}\right)$$

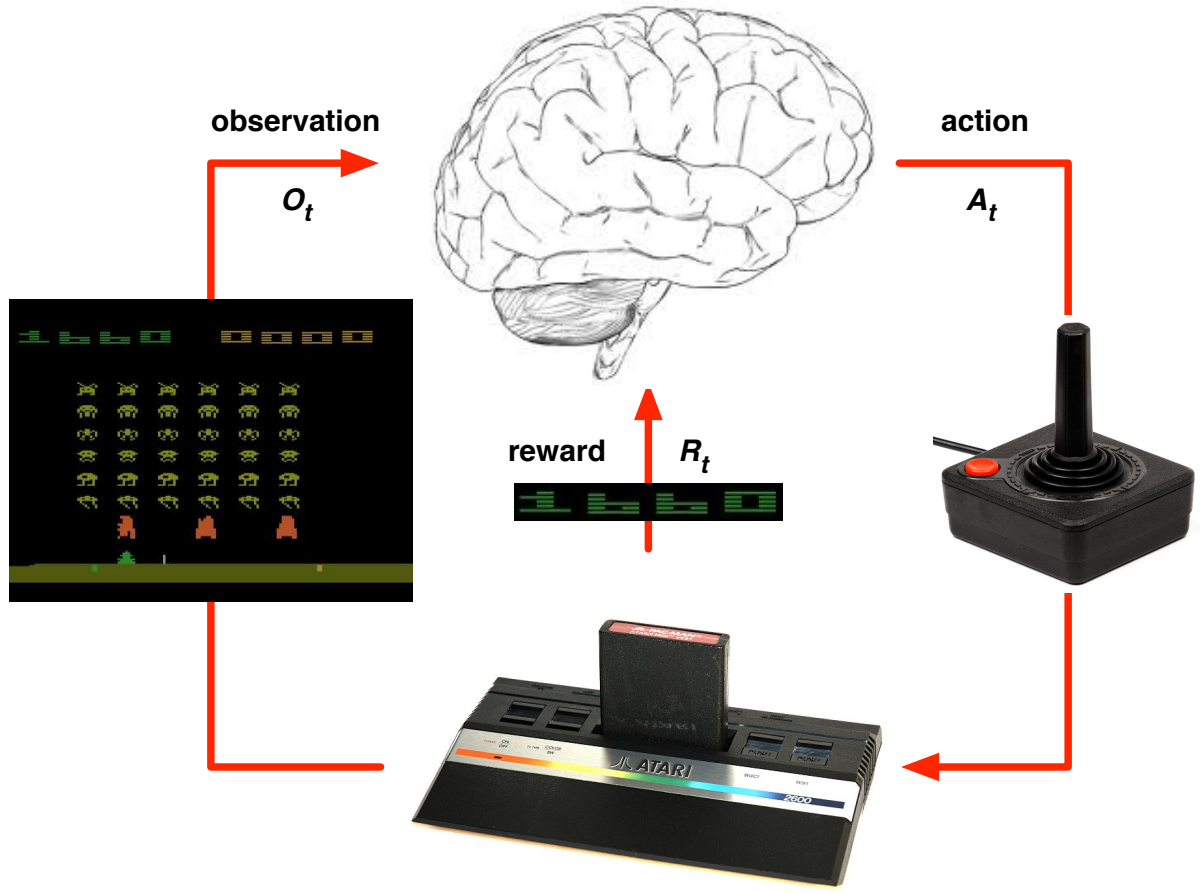
# Experience Replay

- **Problems** with online updates for Deep Q-learning:
  - not i.i.d. as SGD would assume
  - quickly forget rare experiences that might later be useful to learn from
- **Uniform Experience Replay** (Lin, 1992):
  - Keep a *replay memory* D = $\{e_1, e_2, \ldots, e_N\}$ of N most recent experiences $e_t = \langle s_t, a_t, r_t, s_{t+1} \rangle$
  - Alternate two steps:
    1. Repeat T times: randomly sample $e_i$ from D and apply a Q-Learning update to $e_i$
    2. Agent selects an action using epsilon greedy policy to receive new experience that is added to D
- **Prioritized Experience Replay** (Schaul et al, 2016)
  - similar to Uniform ER, but sample so as to prioritize experiences with high error

# DEEP RL FOR ATARI GAMES

# Playing Atari with Deep RL

- Setup: RL system observes the pixels on the screen
- It receives rewards as the game score
- Actions decide how to move the joystick / buttons




Figures from David Silver (Intro RL lecture)

# Playing Atari games with Deep RL


Source: https://www.youtube.com/watch?v=V1eYniJ0Rnk&t=2s&ab_channel=TwoMinutePapers

# Playing Atari games with Deep RL

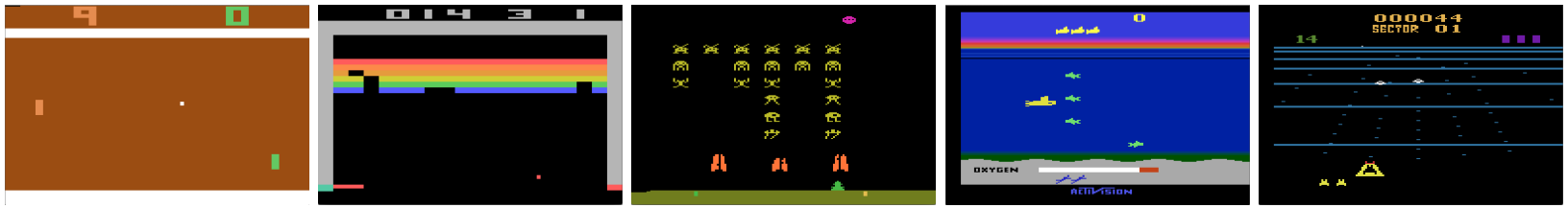


Figure 1: Screen shots from five Atari 2600 Games: (*Left-to-right*) Pong, Breakout, Space Invaders, Seaquest, Beam Rider

| | B. Rider | Breakout | Enduro | Pong | Q*bert | Seaquest | S. Invaders |
|---|---|---|---|---|---|---|---|
| **Random** | 354 | 1.2 | 0 | −20.4 | 157 | 110 | 179 |
| **Sarsa** [3] | 996 | 5.2 | 129 | −19 | 614 | 665 | 271 |
| **Contingency** [4] | 1743 | 6 | 159 | −17 | 960 | 723 | 268 |
| **DQN** | **4092** | **168** | **470** | **20** | **1952** | **1705** | **581** |
| **Human** | 7456 | 31 | 368 | −3 | 18900 | 28010 | 3690 |
| **HNeat Best** [8] | 3616 | 52 | 106 | 19 | 1800 | 920 | **1720** |
| **HNeat Pixel** [8] | 1332 | 4 | 91 | −16 | 1325 | 800 | 1145 |
| **DQN Best** | **5184** | **225** | **661** | **21** | **4500** | **1740** | 1075 |

Table 1: The upper table compares average total reward for various learning methods by running an $\epsilon$-greedy policy with $\epsilon = 0.05$ for a fixed number of steps. The lower table reports results of the single best performing episode for HNeat and DQN. HNeat produces deterministic policies that always get the same score while DQN used an $\epsilon$-greedy policy with $\epsilon = 0.05$.

Figures from Mnih et al. (2013)

# Learning Objectives

**Reinforcement Learning: Q-Learning**

*You should be able to…*

1. Apply Q-Learning to a real-world environment
2. Implement Q-learning
3. Identify the conditions under which the Q-learning algorithm will converge to the true value function
4. Adapt Q-learning to Deep Q-learning by employing a neural network approximation to the Q function
5. Describe the connection between Deep Q-Learning and regression

# BIG PICTURE

# ML Big Picture

**Learning Paradigms:**

*What data is available and when? What form of prediction?*

- supervised learning
- unsupervised learning
- semi-supervised learning
- reinforcement learning
- active learning
- imitation learning
- domain adaptation
- online learning
- density estimation
- recommender systems
- feature learning
- manifold learning
- dimensionality reduction
- ensemble learning
- distant supervision
- hyperparameter optimization

**Problem Formulation:**

*What is the structure of our output prediction?*

| | |
|---|---|
| boolean | Binary Classification |
| categorical | Multiclass Classification |
| ordinal | Ordinal Classification |
| real | Regression |
| ordering | Ranking |
| multiple discrete | Structured Prediction |
| multiple continuous | (e.g. dynamical systems) |
| both discrete & cont. | (e.g. mixed graphical models) |

**Application Areas**

*Key challenges?*

NLP, Speech, Computer Vision, Robotics, Medicine, Search

**Theoretical Foundations:**

*What principles guide learning?*

- ❑ probabilistic
- ❑ information theoretic
- ❑ evolutionary search
- ❑ ML as optimization

**Facets of Building ML Systems:**

*How to build systems that are robust, efficient, adaptive, effective?*

1. *Data prep*
2. *Model selection*
3. *Training (optimization / search)*
4. *Hyperparameter tuning on validation data*
5. *(Blind) Assessment on test data*

**Big Ideas in ML:**

*Which are the ideas driving development of the field?*

- *inductive bias*
- *generalization / overfitting*
- *bias-variance decomposition*
- *generative vs. discriminative*
- *deep nets, graphical models*
- *PAC learning*
- *distant rewards*

# Learning Paradigms

| Paradigm | Data |
|---|---|
| Supervised | $\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^{N} \qquad \mathbf{x} \sim p^*(\cdot)$ and $y = c^*(\cdot)$ |
| $\hookrightarrow$ Regression | $y^{(i)} \in \mathbb{R}$ |
| $\hookrightarrow$ Classification | $y^{(i)} \in \{1, \ldots, K\}$ |
| $\hookrightarrow$ Binary classification | $y^{(i)} \in \{+1, -1\}$ |
| $\hookrightarrow$ Structured Prediction | $\mathbf{y}^{(i)}$ is a vector |
| Unsupervised | $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^{N} \qquad \mathbf{x} \sim p^*(\cdot)$ |
| Semi-supervised | $\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^{N_1} \cup \{\mathbf{x}^{(j)}\}_{j=1}^{N_2}$ |
| Online | $\mathcal{D} = \{(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), (\mathbf{x}^{(3)}, y^{(3)}), \ldots\}$ |
| Active Learning | $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^{N}$ and can query $y^{(i)} = c^*(\cdot)$ at a cost |
| Imitation Learning | $\mathcal{D} = \{(s^{(1)}, a^{(1)}), (s^{(2)}, a^{(2)}), \ldots\}$ |
| Reinforcement Learning | $\mathcal{D} = \{(s^{(1)}, a^{(1)}, r^{(1)}), (s^{(2)}, a^{(2)}, r^{(2)}), \ldots\}$ |