10-301/601 Introduction to Machine Learning

Machine Learning Department
School of Computer Science
Carnegie Mellon University

Reinforcement Learning:
Q-Learning

Matt Gormley & Henry Chai
Lecture 23
Nov. 12, 2021

Reminders

- Homework 7: HMMs
  – Out: Wed, Nov. 03
  – Due: Fri, Nov. 12 at 11:59pm
- Homework 8: RL
  – Out: Fri, Nov. 12
  – Due: Sun, Nov. 21 at 11:59pm

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://en.wikipedia.org/wiki/San_Jose_Earthquakes#/media/File:Q_at_Galaxy_at_Earthquakes_2010-08-21_1.JPG

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

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

Key questions for today

1. What can we do if the reward function and/or transition probabilities are unknown?

2. How can we handle infinite (or just very large) state/action spaces?

$Q^*(s,a)$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

- Algorithm 1: Online learning of Q^* (table form)
  - Inputs: discount factor γ,

    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: ϵ-greedy online learning of Q^* (table form)
    - Inputs: discount factor γ,

      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 ϵ, take the greedy action → $a = \underset{a' \in \mathcal{A}}{\operatorname{argmax}}\; Q(s,a')$. Otherwise (with probability $1 - \epsilon$), take a random action a
        - Receive reward $r = R(s,a)$
        - Observe the new state $s' = \delta(s,a)$
        - Update Q and s

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

          $$s \leftarrow s'$$

Learning $Q^*(s,a)$

- Algorithm 3: ϵ-greedy online learning of Q^* (table form)
  - Inputs: discount factor γ,

    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 ϵ, take the greedy action $a = \operatorname*{argmax}_{\underline{a' \in \mathcal{A}}} Q(s,a')$. Otherwise (with probability $1-\epsilon$), take a random action a
    - Receive reward $\underline{r = R(s,a)} \approx \mathbb{E}[R(s,a)]$
    - 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 $Q^*(s,a)$

- Algorithm 3: ϵ-greedy online learning of Q^* (table form)
  - Inputs: discount factor γ,

    an initial state s,

    greediness parameter $\epsilon \in [0,1]$,

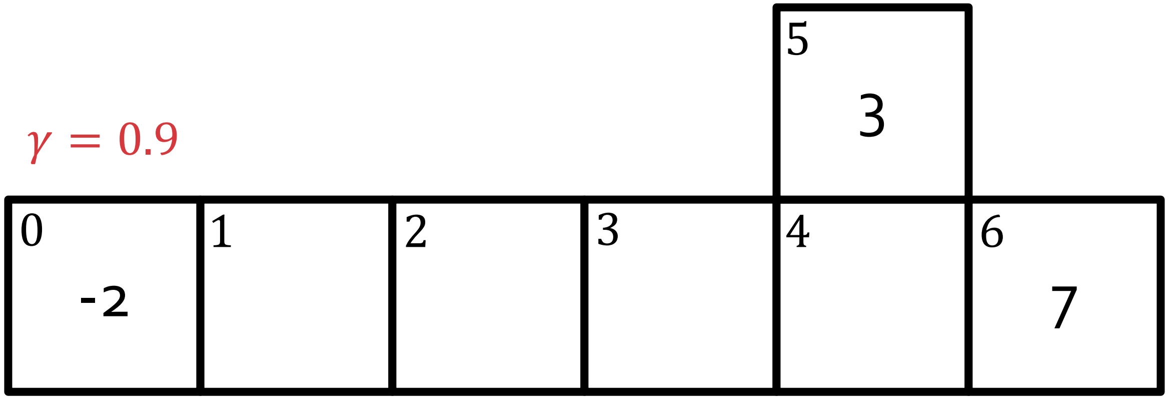
    learning rate $\alpha \in [0,1]$ ("mistrust parameter")
  - Initialize $Q(s,a) = 0 \ \forall\, s \in \mathcal{S}, a \in \mathcal{A}$
    (Q is a $|\mathcal{S}|\times|\mathcal{A}|$ table or array)
  - While TRUE, do
    - With probability ϵ, take the greedy action $a = \operatorname{argmax}_{a' \in \mathcal{A}} Q(s,a')$. Otherwise (with probability $1-\epsilon$), take a random action a
    - Receive reward $r = R(s,a)$
    - Observe the new state $s' \sim p(S' \mid s,a)$
    - Update Q and s ✓

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

$$s \leftarrow s'$$

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

$\gamma = 0.9$

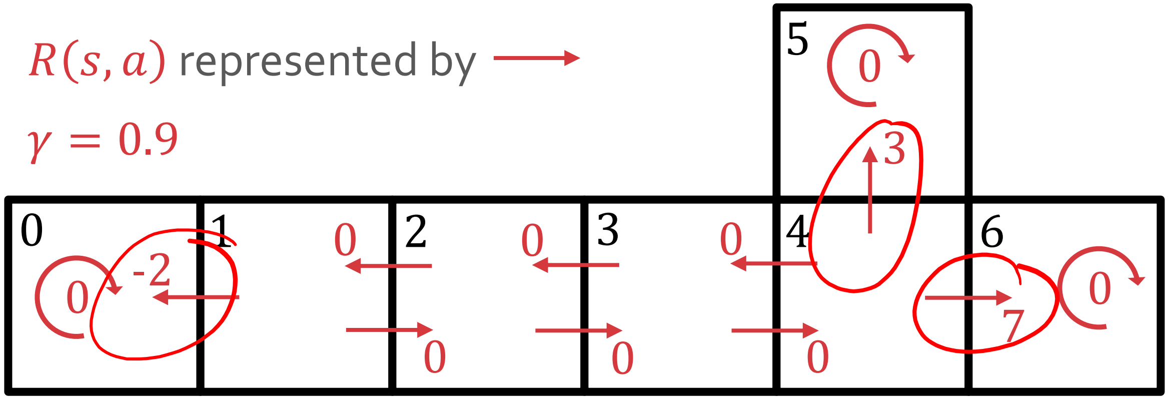


$$R(s, a) = \begin{cases} -2 \text{ if entering state } 0 \text{ (safety)} \\ 3 \text{ if entering state } 5 \text{ (field goal)} \\ 7 \text{ if entering state } 6 \text{ (touch down)} \\ 0 \text{ otherwise} \end{cases}$$

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

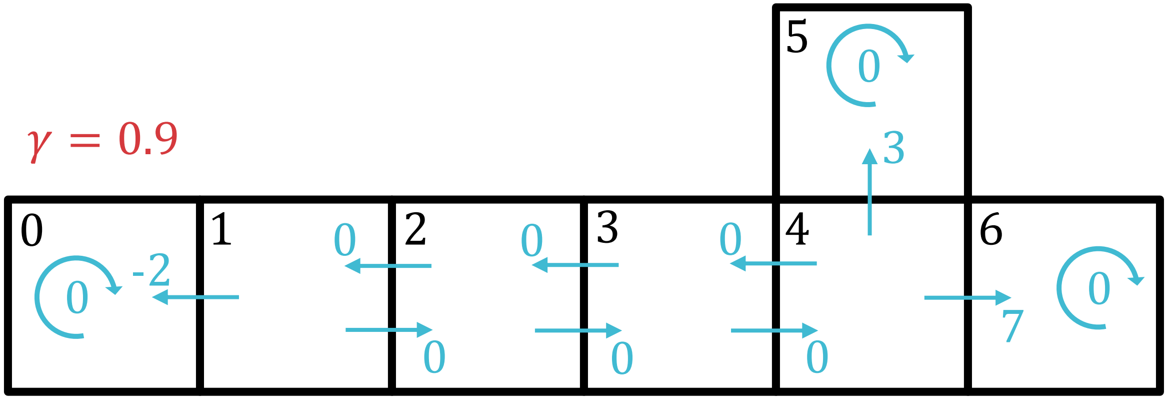
$R(s,a)$ represented by →

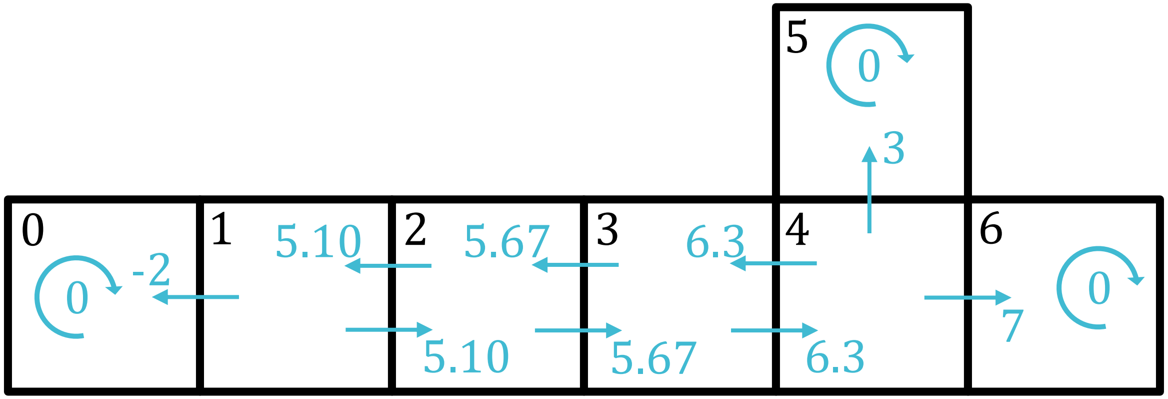
$\gamma = 0.9$

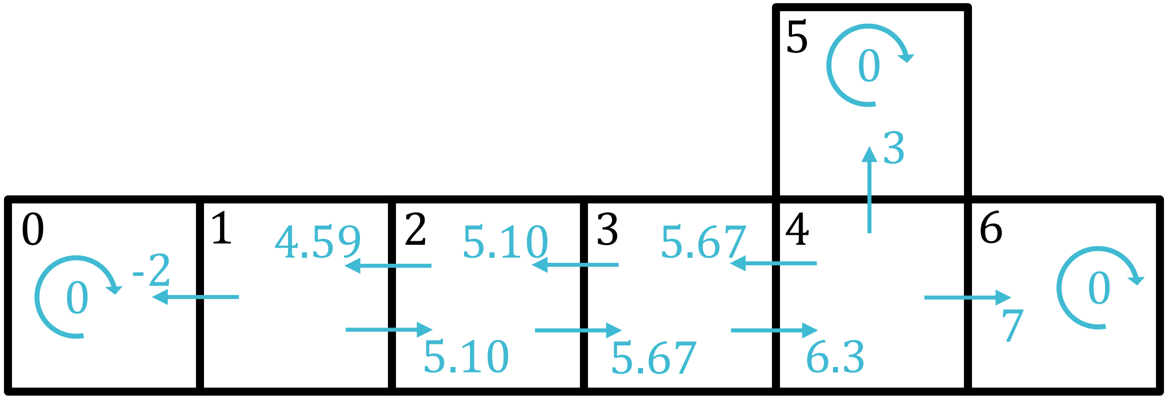


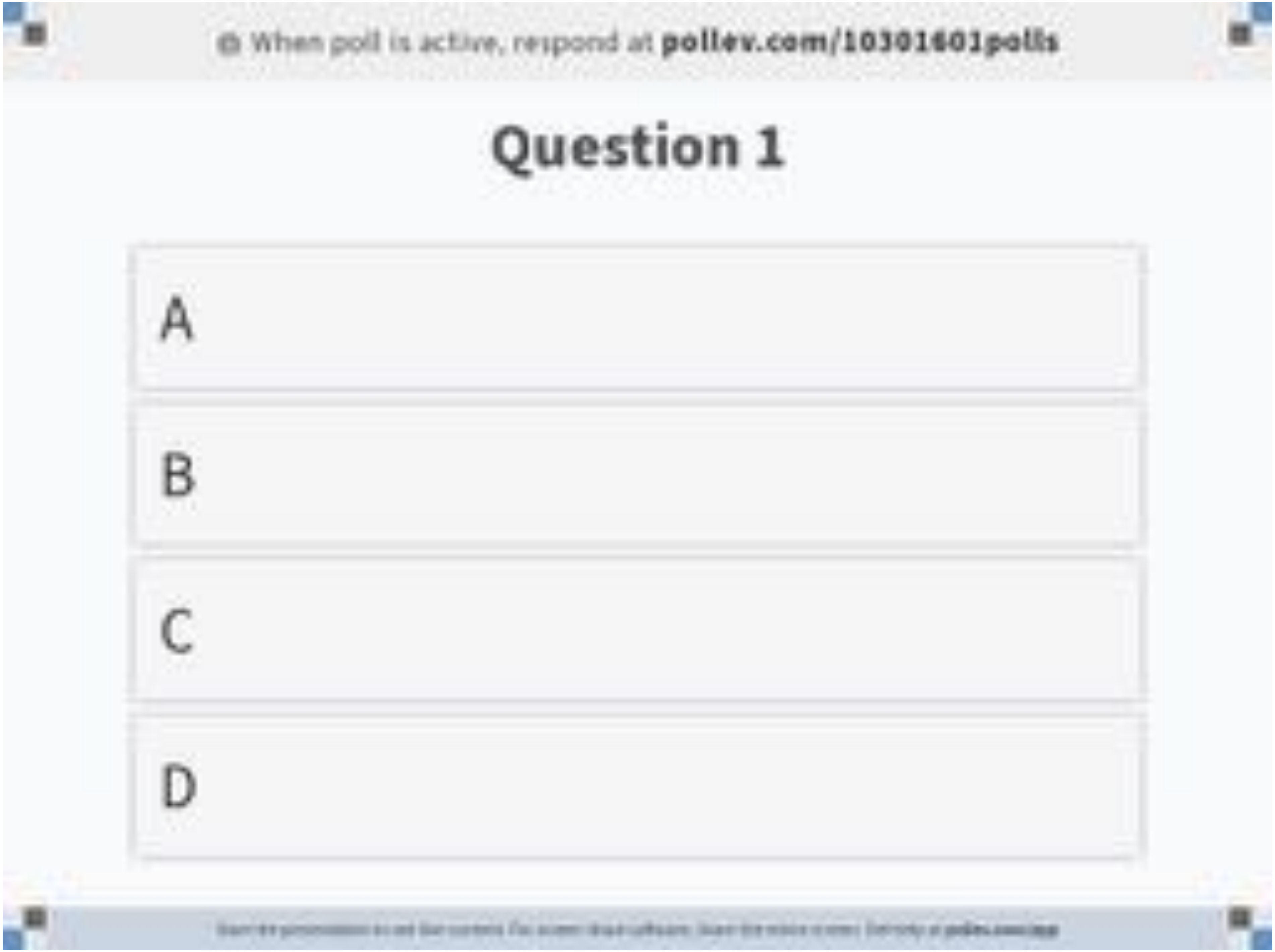
Poll Q1: Which set of blue arrows (roughly) corresponds to $Q^*(s,a)$?

$\gamma = 0.9$









When poll is active, respond at **pollev.com/10301601polls**

Question 1

A

B

C

D

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Poll Q1: Which set of blue arrows corresponds to $Q^*(s,a)$?

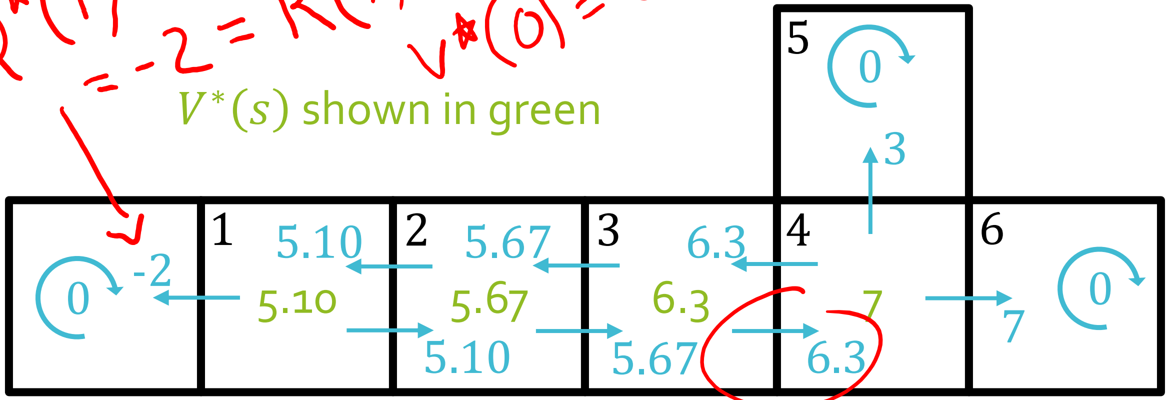
$Q(4,\leftarrow) = 0 + (0.9)(6.3) = 5.67$

$Q^*(1,\leftarrow) = -2 = R(1,\leftarrow)$

$V^*(0) = 0$

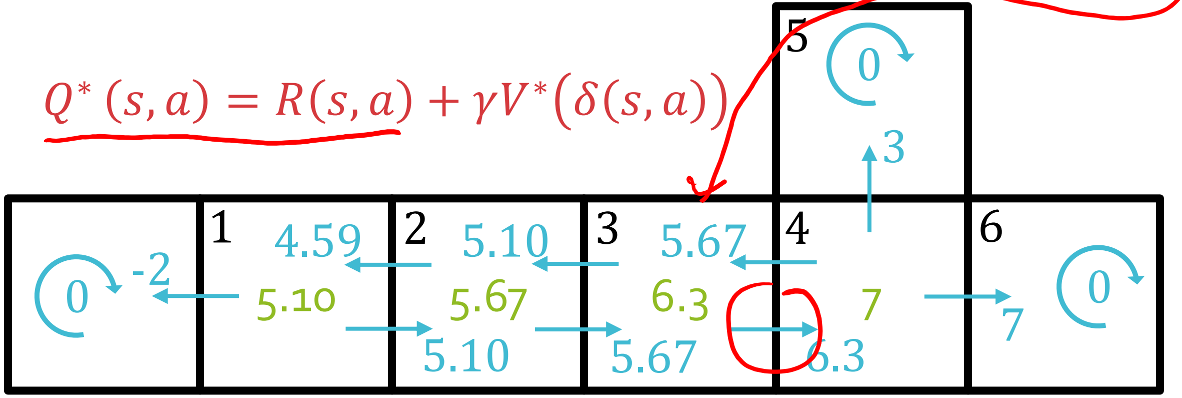
$V^*(s)$ shown in green

C.



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

D.

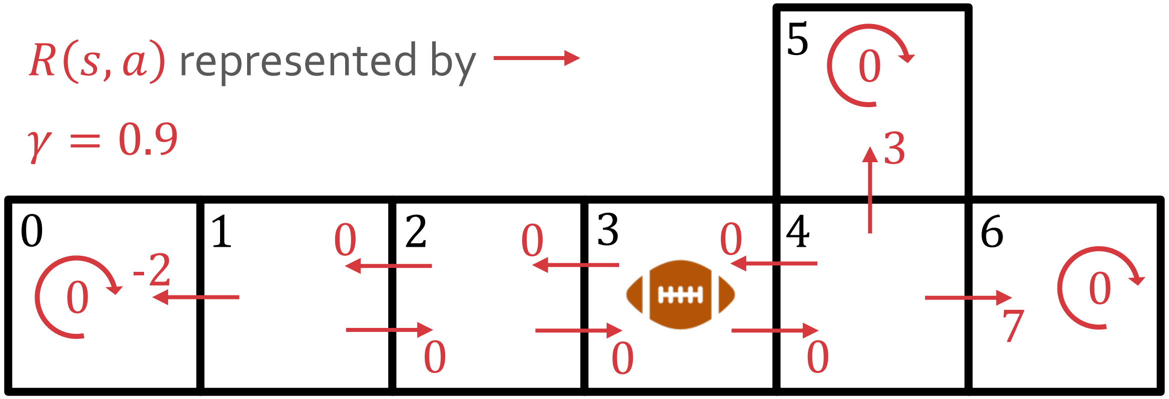


$Q^*(3,\rightarrow) = 0 + (0.9)(7) = 6.3$

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

$R(s,a)$ represented by →

$\gamma = 0.9$

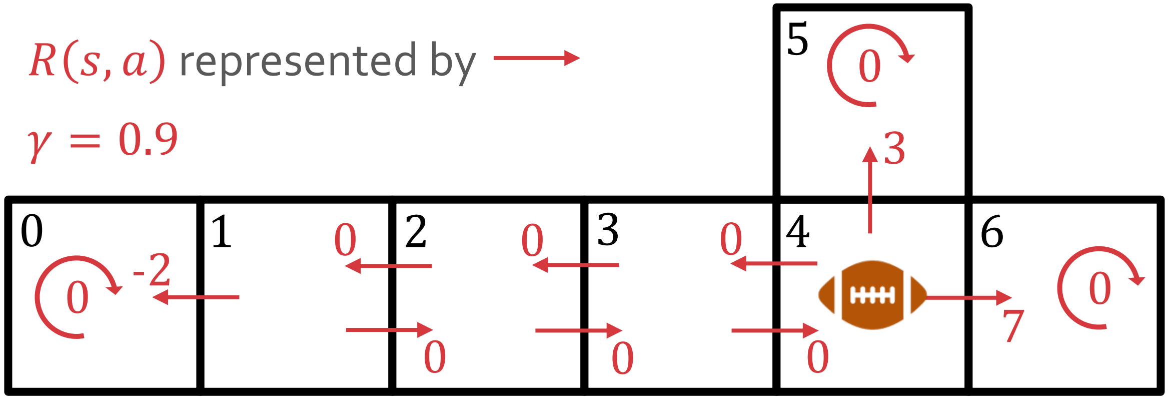


$Q(s,a)$	→	←	↑	↻
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0

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

$R(s,a)$ represented by →

$\gamma = 0.9$



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

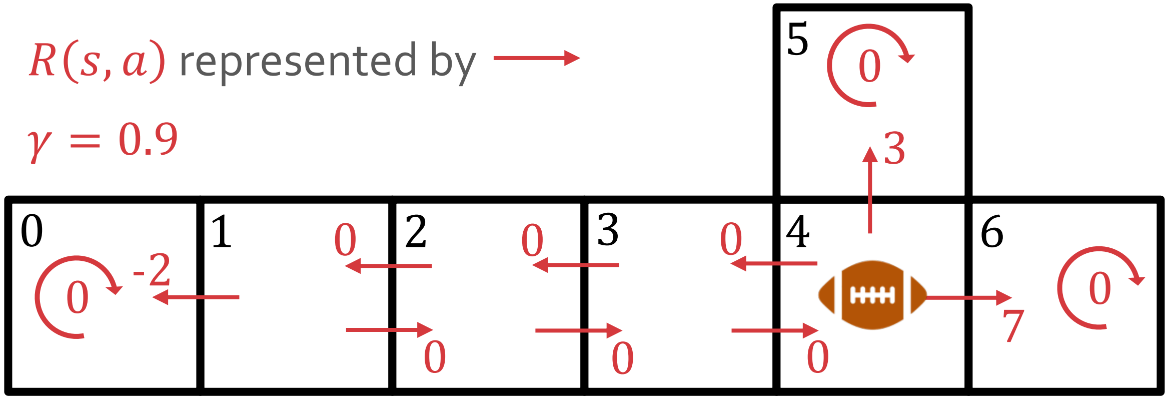
0

$Q(s,a)$	→	←	↑	↻
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

$R(s,a)$ represented by →

$\gamma = 0.9$

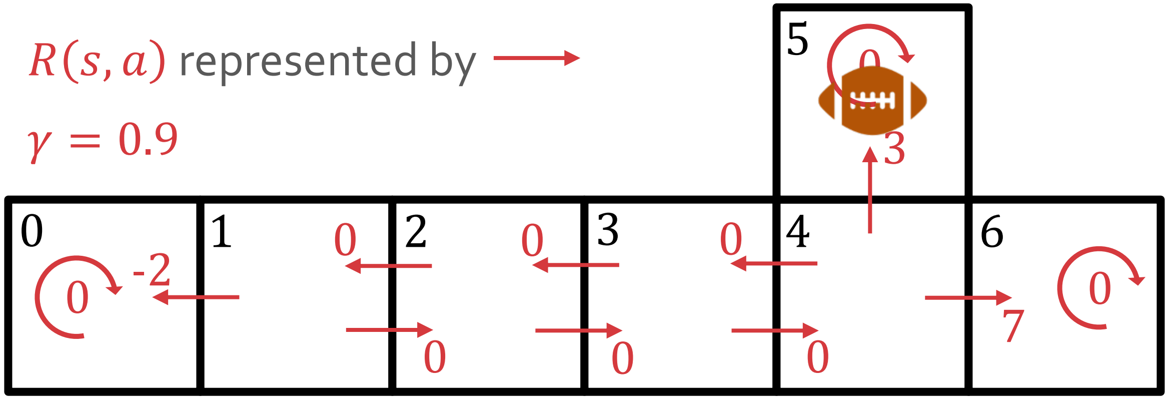


$Q(s,a)$	→	←	↑	↻
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0

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

$R(s,a)$ represented by →

$\gamma = 0.9$



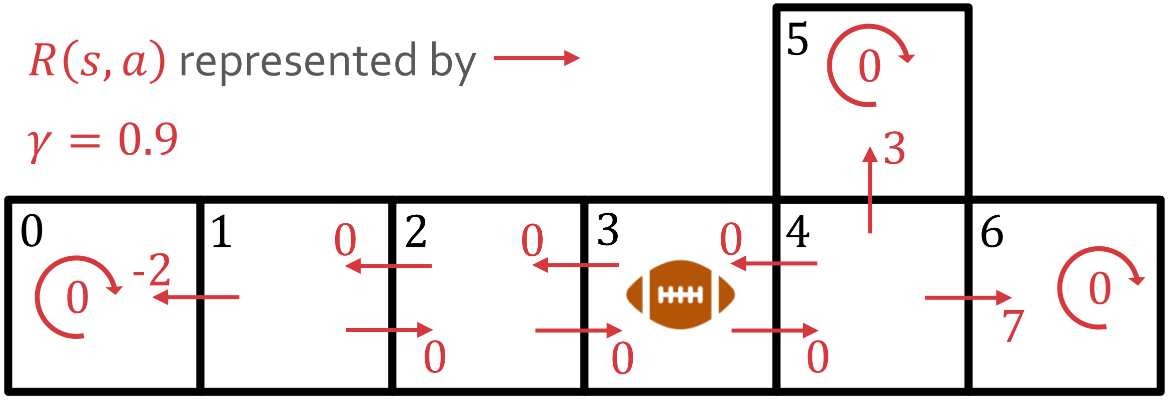
$$Q(4,\uparrow) \leftarrow 3 + (0.9) \max_{a' \in \{\rightarrow,\leftarrow,\uparrow,\circlearrowright\}} Q(5,a') = 3$$

$Q(s,a)$	→	←	↑	↻
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	$\underline{0}$	0
5	0	0	0	0
6	0	0	0	0

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

$R(s,a)$ represented by $\longrightarrow$

$\gamma = 0.9$



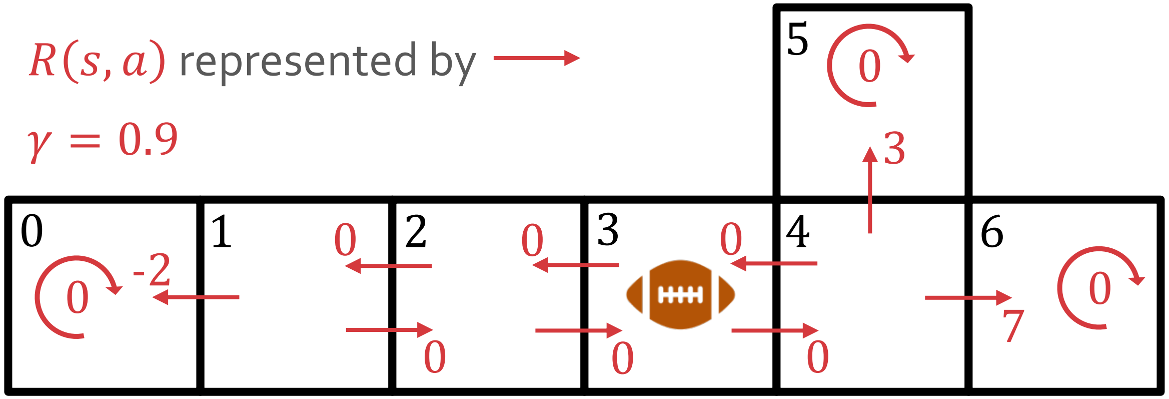
$$Q(3,\rightarrow) \leftarrow 0 + (0.9) \max_{a' \in \{\rightarrow,\leftarrow,\uparrow,\circlearrowright\}} Q(4,a') = 2.7$$

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

$R(s,a)$ represented by →

$\gamma = 0.9$



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

$Q(s,a)$	→	←	↑	↻
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

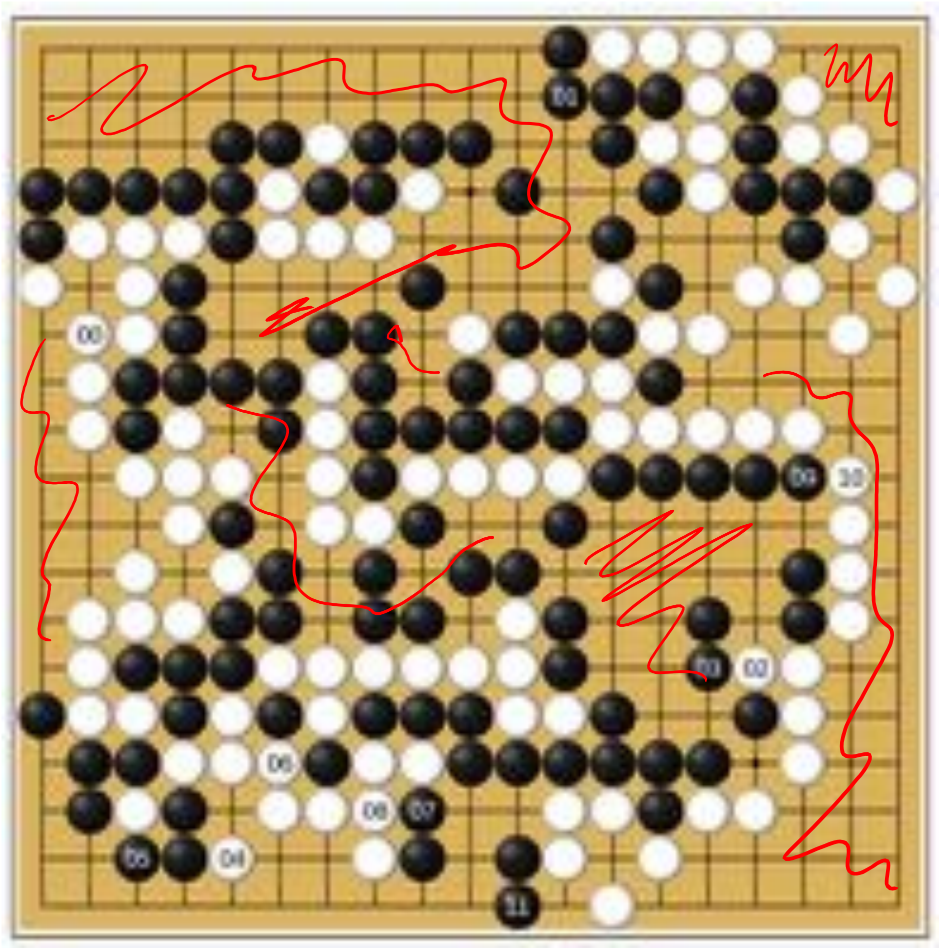
- For Algorithms 1 & 2 (deterministic transitions), Q converges to Q^* if
  1. Every valid state-action pair is visited infinitely often
     - Q-learning is exploration-insensitive: any visitation strategy that satisfies this property will work!
  2. $0 \leq \gamma < 1$
  3. $|R(s,a)| < \beta \; \forall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

$$Q(s,a) = R(s,a) + \gamma \max_{a'} Q(s',a')$$

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



Playing Go

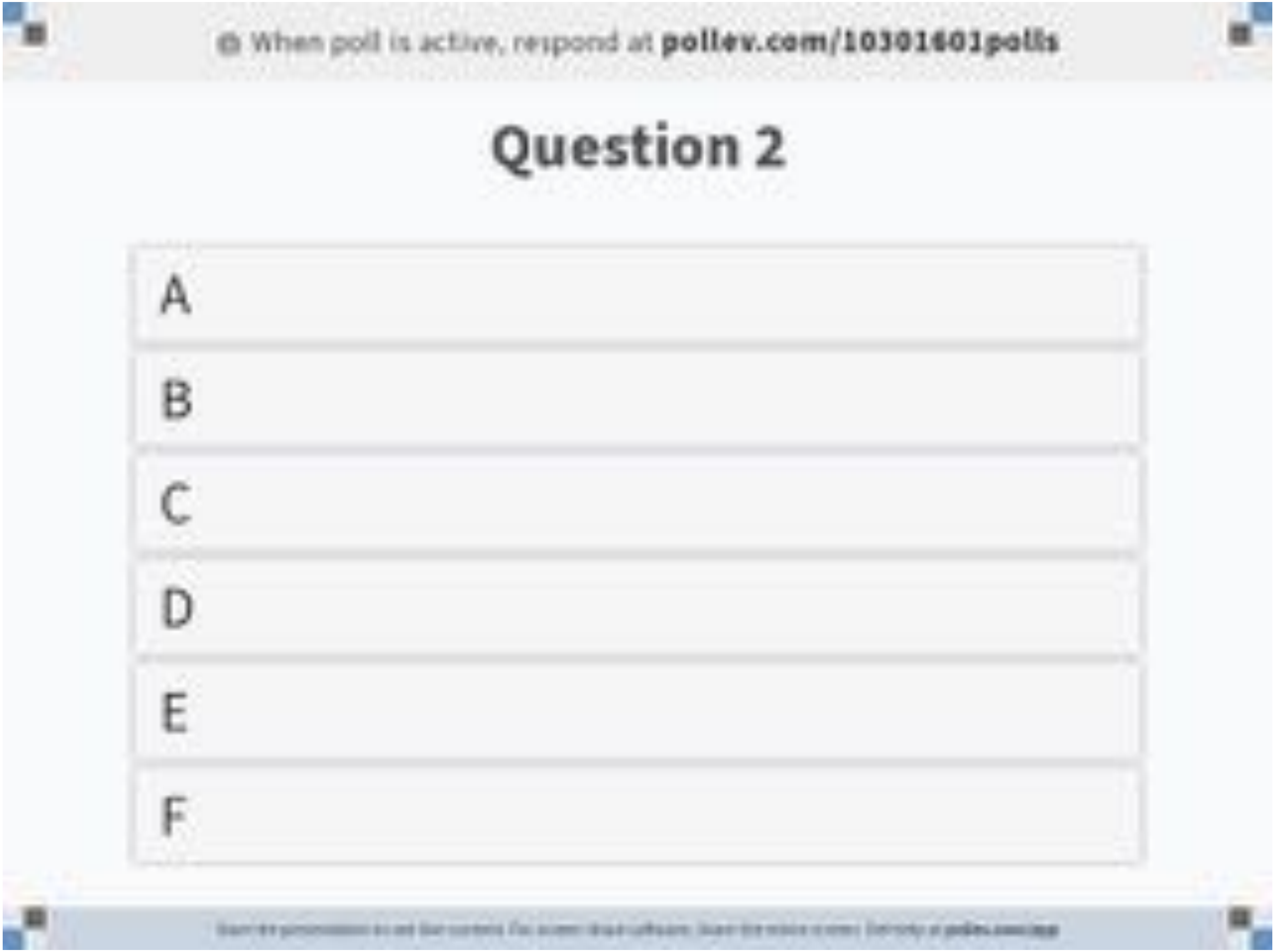
- 19-by-19 board
- Players alternate placing black and white stones
- The goal is claim more territory than the opponent

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

Poll Q2: Which of the following is the closest approximation to the number of legal board states in a game of Go?

Playing Go

- 19-by-19 board
- Players alternate placing black and white stones
- The goal is claim more territory than the opponent



When poll is active, respond at **pollev.com/10301601polls**

Question 2

A

B

C

D

E

F

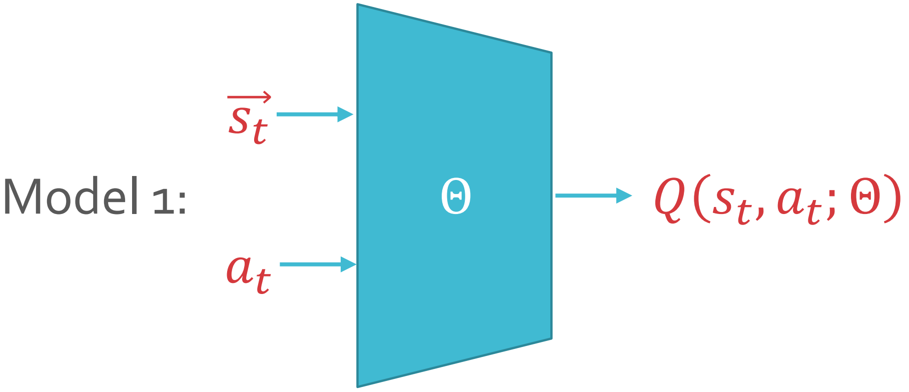
Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

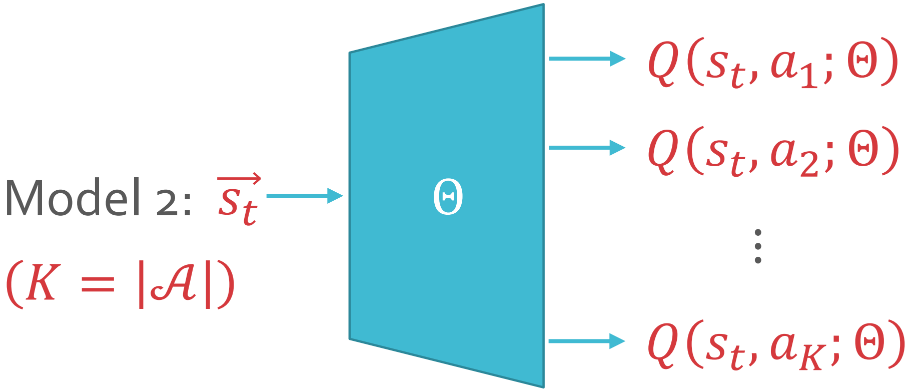
Deep Q-learning

- How can we handle infinite (or just very large) state/action spaces?
- ~~Just throw a neural network at it~~
- Use a parametric function $Q(s, a; \Theta)$ to approximate $Q^*(s, a)$
    - Learn the parameters using stochastic gradient descent
    - Training data (s_t, a_t, r_t, s_t) gathered online by the agent / learning algorithm

Deep Q-learning: Model

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

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

- Gradients are easy: $\nabla_\Theta Q(\vec{s}, a_k; \Theta) = \begin{array}{cl} \begin{bmatrix} \vec{0} \\ \vec{0} \\ \vdots \\ \vec{s} \\ \vdots \\ \vec{0} \end{bmatrix} & \begin{matrix} \\ \\ \\ \leftarrow \text{Row } k \\ \\ \\ \end{matrix} \end{array}$

Deep Q-learning: Loss Function

2. Don't know Q^*

- "True" loss

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

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 $\Theta^{(\mathrm{t})}$ the (temporal difference) target is

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

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

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

Deep Q-learning

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

    an initial state s_0,

    learning rate α
  - 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

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

      where

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

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 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)

RL Learning Goals: Q-Leaning and Deep RL

a. Apply Q-Learning to a real-world environment

b. Implement Q-learning

c. Identify the conditions under which the Q-learning algorithm will converge to the true value function

d. Adapt Q-learning to Deep Q-learning by employing a neural network approximation to the Q function

e. Describe the connection between Deep Q-Learning and regression

Bonus: Playing Atari games

Source: https://www.youtube.com/watch?v=V1eYniJoRnk&t=2s&ab_channel=TwoMinutePapers

Bonus: Playing Atari games

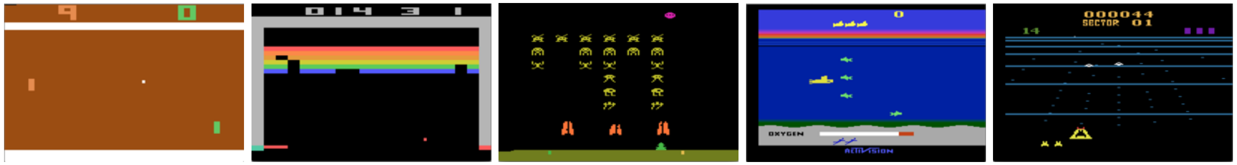
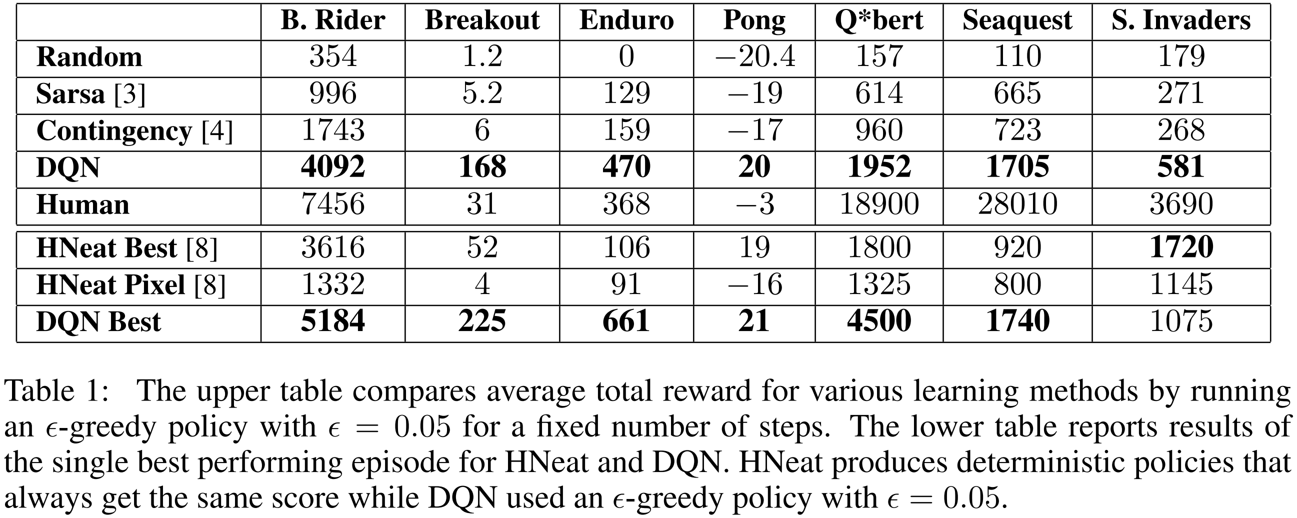


Figure 1: Screen shots from five Atari 2600 Games: (*Left-to-right*) Pong, Breakout, Space Invaders, Seaquest, Beam Rider



	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 ϵ-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 ϵ-greedy policy with $\epsilon = 0.05$.

Source: https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf