

10-301/601: Introduction to Machine Learning

Lecture 23: Q-learning and Deep RL

Henry Chai

7/26/22

Front Matter

- Announcements
 - HW7 released 7/20, due 7/27 (tomorrow) at 1 PM
 - Please be mindful of your grace day usage (see [the course syllabus](#) for the policy)
- Recommended Readings
 - Mitchell, Chapter 13

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?

Recall: Value Iteration

- Inputs: $R(s, a)$, $p(s' | s, a)$, γ
- 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 $a \in \mathcal{A}$

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

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

- 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 π^*

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

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

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

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

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

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

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

$Q^*(s, a)$ 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 π^* !

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

Algorithm 1: Online learning (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 $|\mathcal{S}| \times |\mathcal{A}|$ 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: ϵ -greedy online learning (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 $|\mathcal{S}| \times |\mathcal{A}|$ 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 $r = R(s, a)$
 - 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: ϵ -greedy online learning (table form)

- Inputs: discount factor γ , 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 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 $r = R(s, a)$
 - Update the state: $s \leftarrow s'$ where $s' \sim p(s' | s, a)$
 - Update $Q(s, a)$:

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

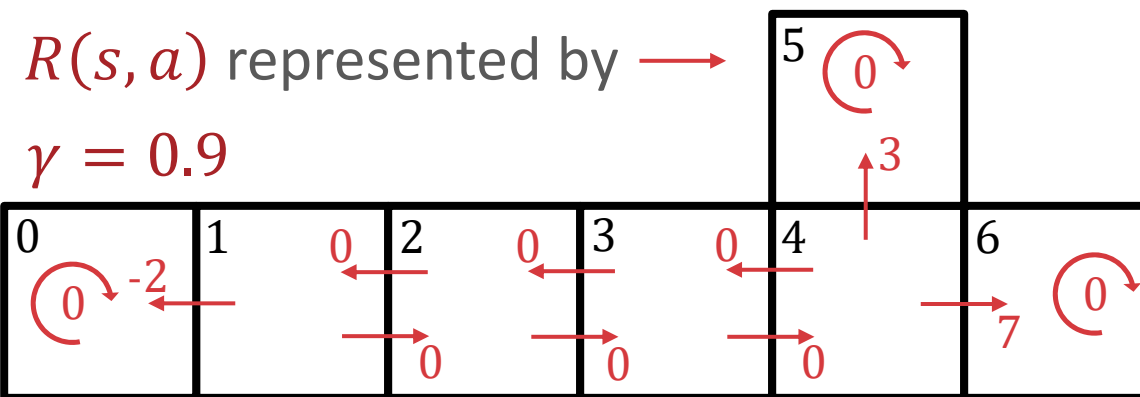
Learning $Q^*(s, a)$ w/ deterministic rewards

Algorithm 3: ϵ -greedy online learning (table form)

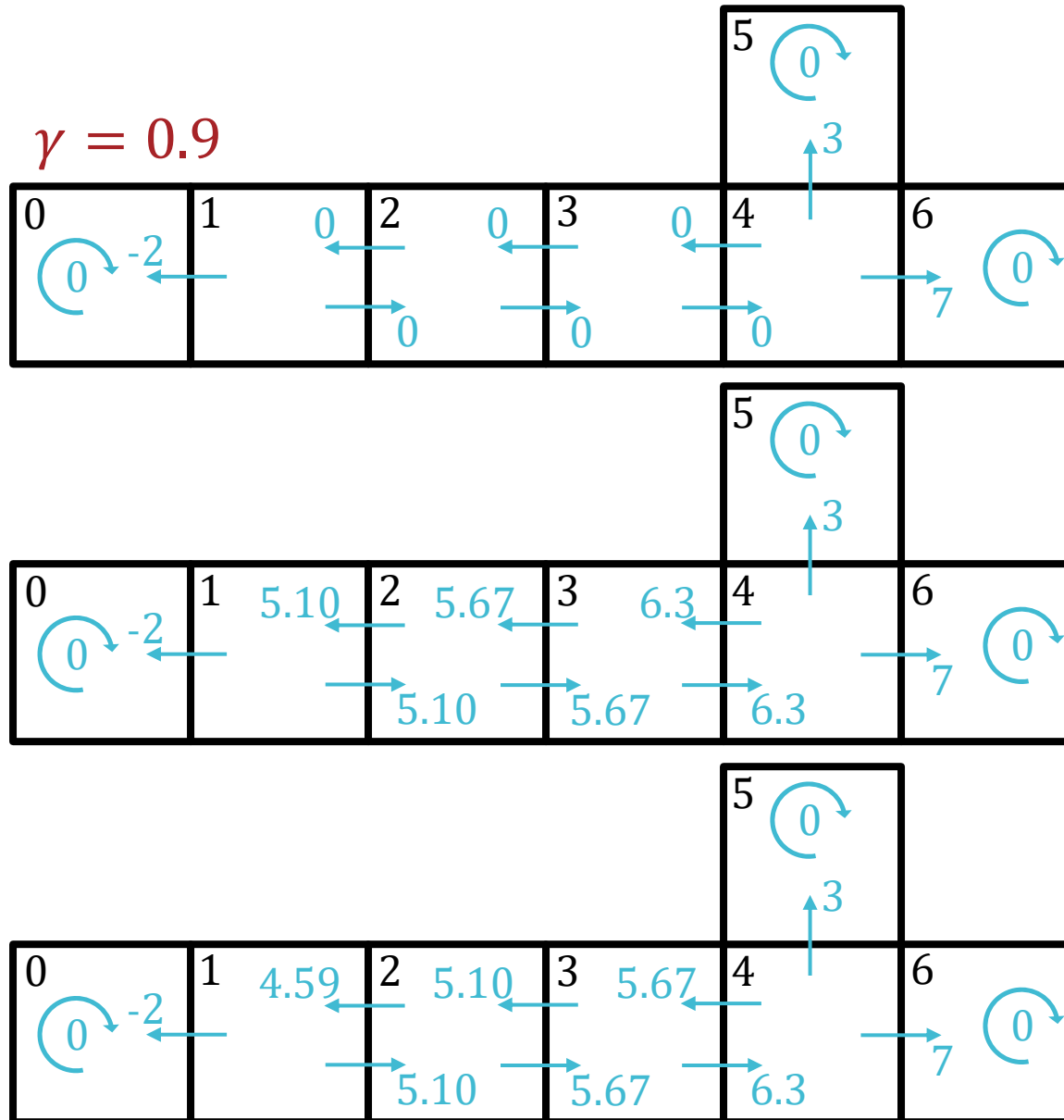
- Inputs: discount factor γ , an initial state s , greediness parameter $\epsilon \in [0, 1]$, learning rate $\alpha \in [0, 1]$ (“trust parameter”)
- Initialize $Q(s, a) = 0 \forall s \in \mathcal{S}, a \in \mathcal{A}$ (Q is a $|\mathcal{S}| \times |\mathcal{A}|$ array)
- While TRUE, do
 - With probability 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 $r = R(s, a)$
 - Update the state: $s \leftarrow s'$ where $s' \sim p(s' | s, a)$
 - Update $Q(s, a)$: Temporal difference

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

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

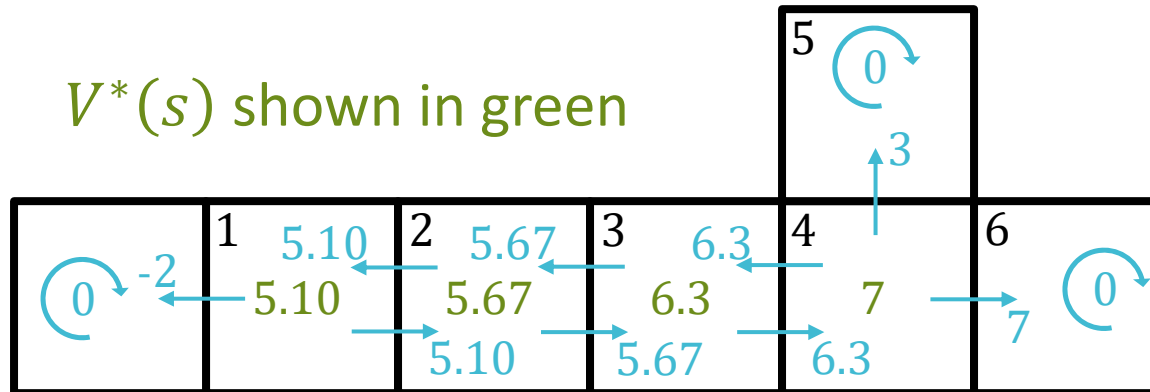


Which set of blue arrows (roughly) corresponds to $Q^*(s, a)$?



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

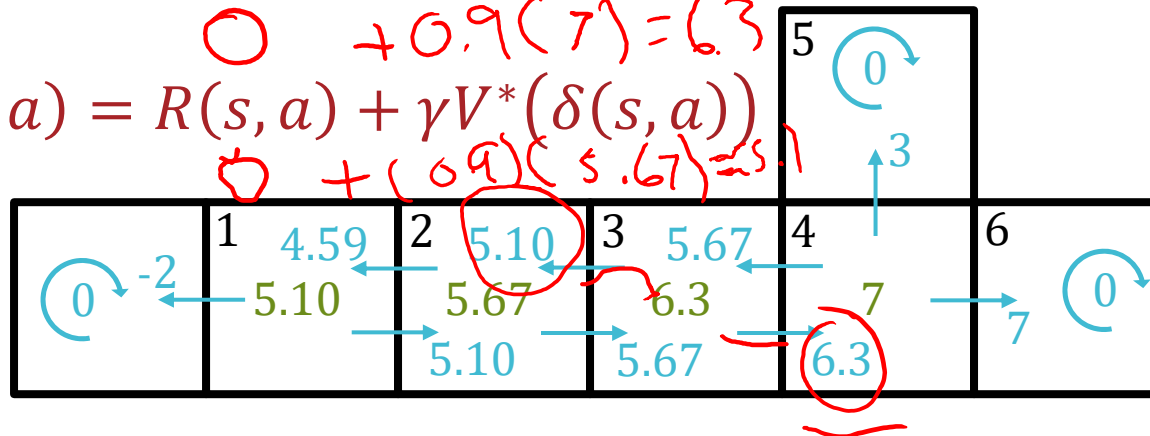
$V^*(s)$ shown in green



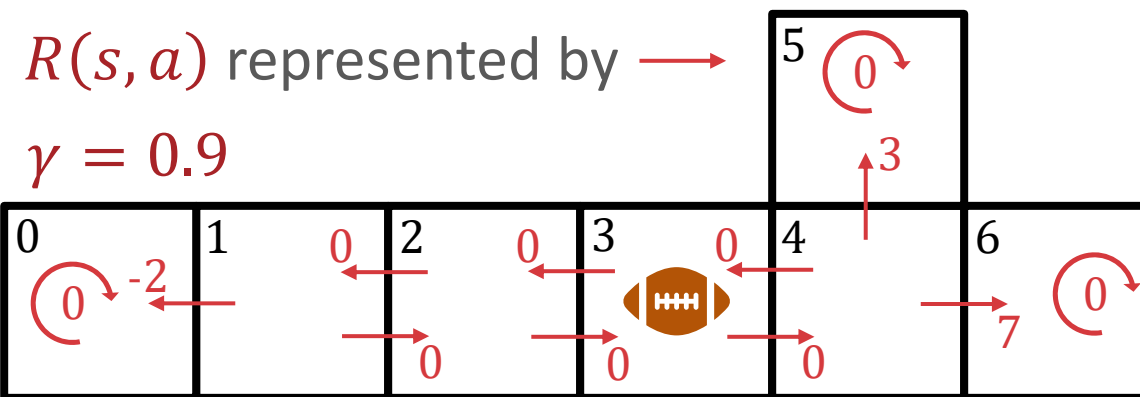
$$Q^*(s, a) = R(s, a) + \gamma V^*(\delta(s, a))$$

Handwritten calculations in red:

- $0 + 0.9(7) = 6.3$
- $0 + 0.9(5.67) = 5.10$

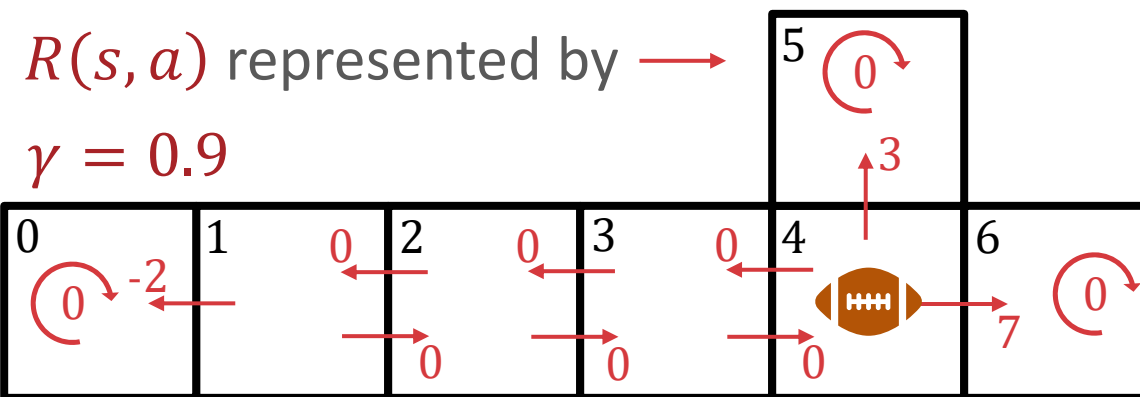


Learning $Q^*(s, a)$: Example



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

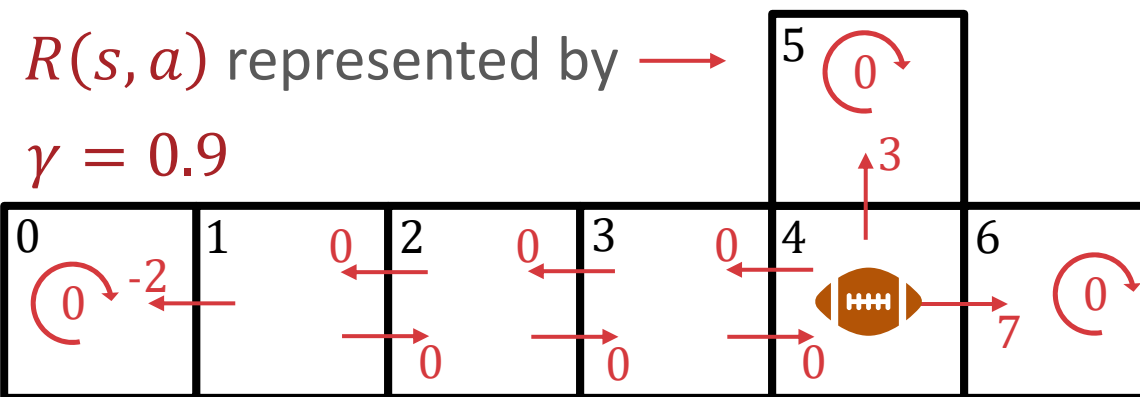


$$Q(3, \rightarrow) \leftarrow 0 + (0.9) \max_{a' \in \{\rightarrow, \leftarrow, \uparrow, \cup\}} Q(4, a') = 0$$

$Q(s, a)$	\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

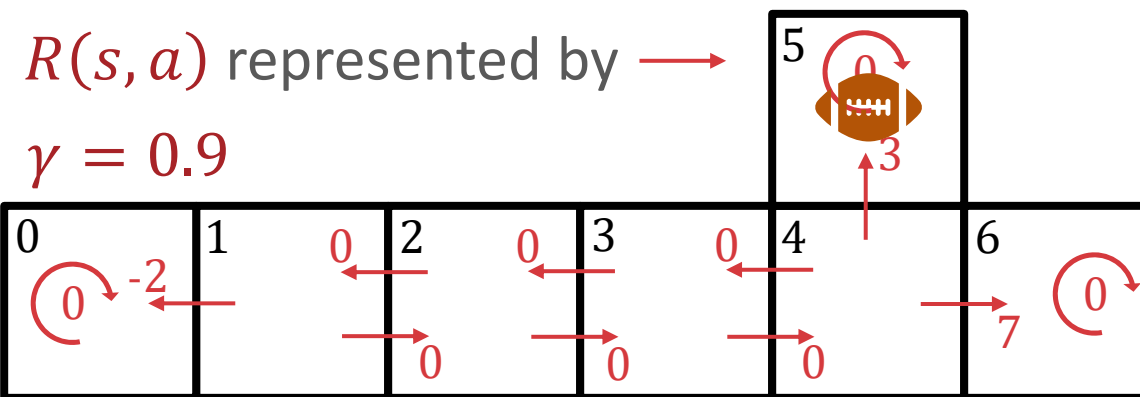
0

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



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

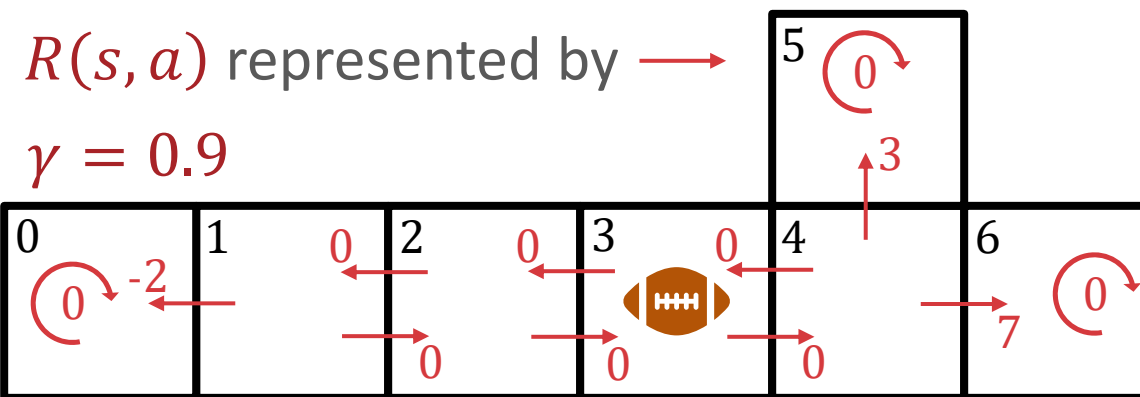
Learning $Q^*(s, a)$: Example



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

$Q(s, a)$	\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

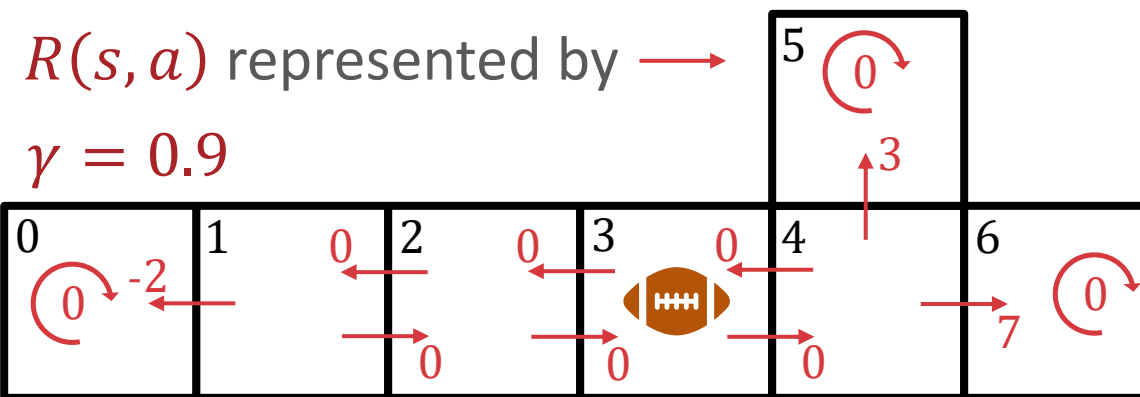
Learning $Q^*(s, a)$: Example



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

$Q(s, a)$	\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

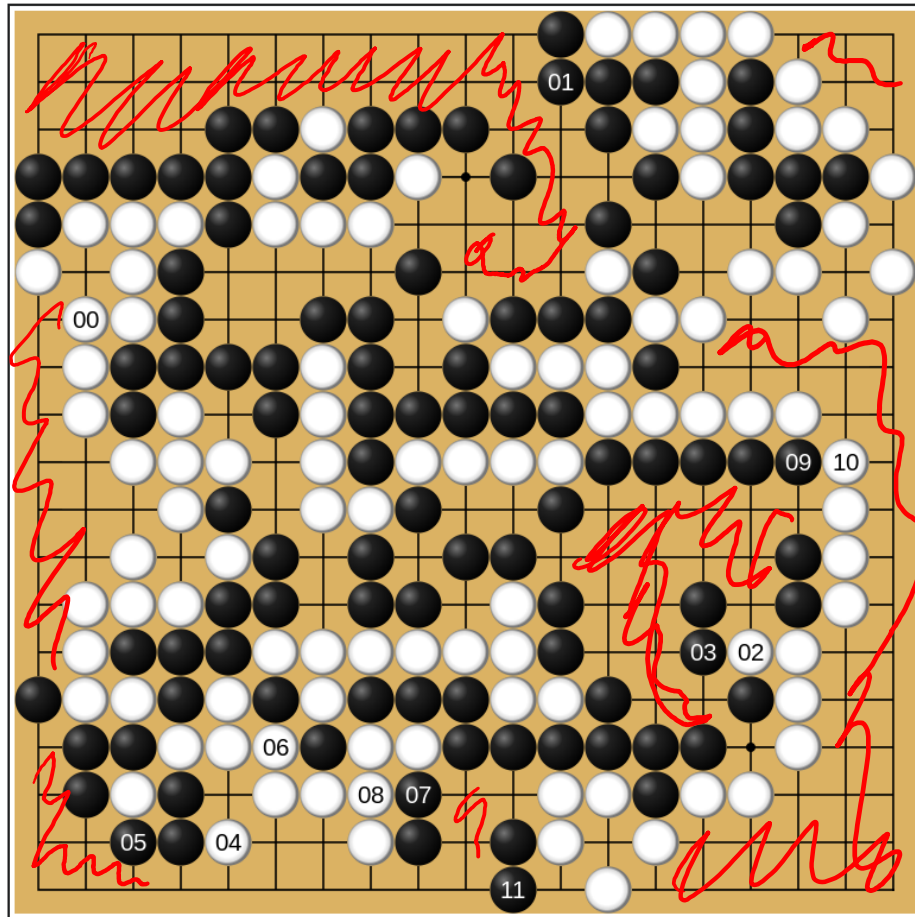
Learning $Q^*(s, a)$: Convergence

- 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 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., $\alpha_t = 1/t+1$

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

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

The number of stars in the universe $\sim 10^{24}$

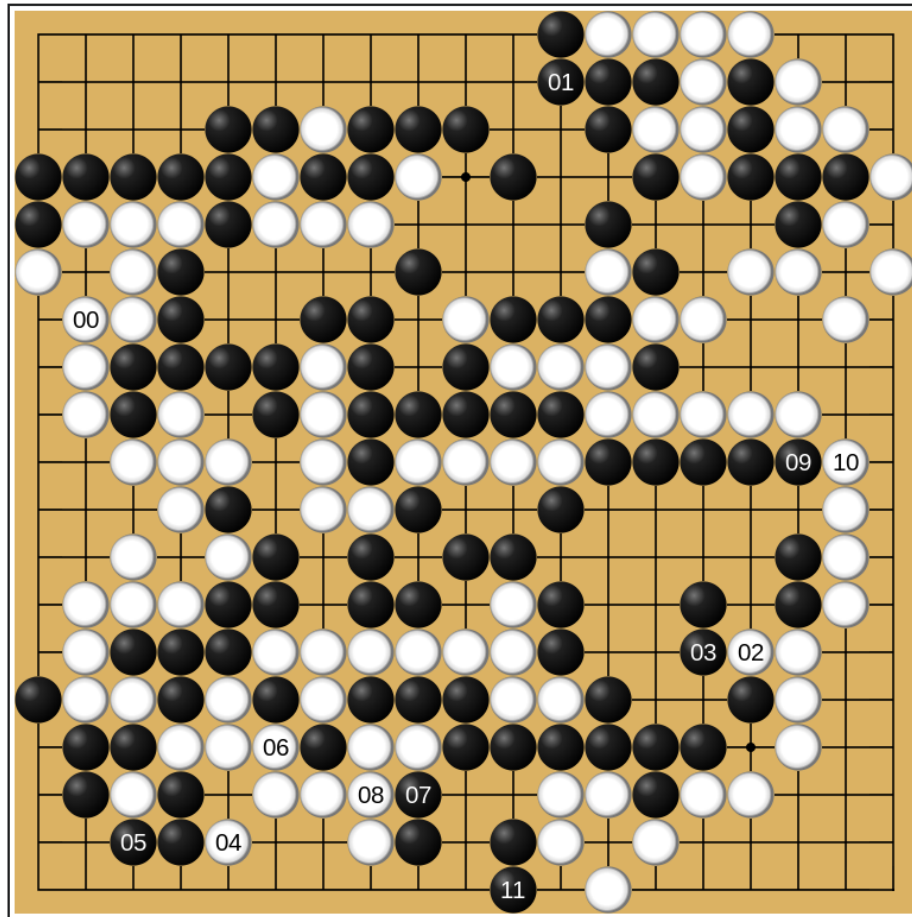
The number of atoms in the universe $\sim 10^{80}$

A googol = 10^{100}

The number of possible *games* of chess $\sim 10^{120}$

A googolplex = 10^{googol}

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

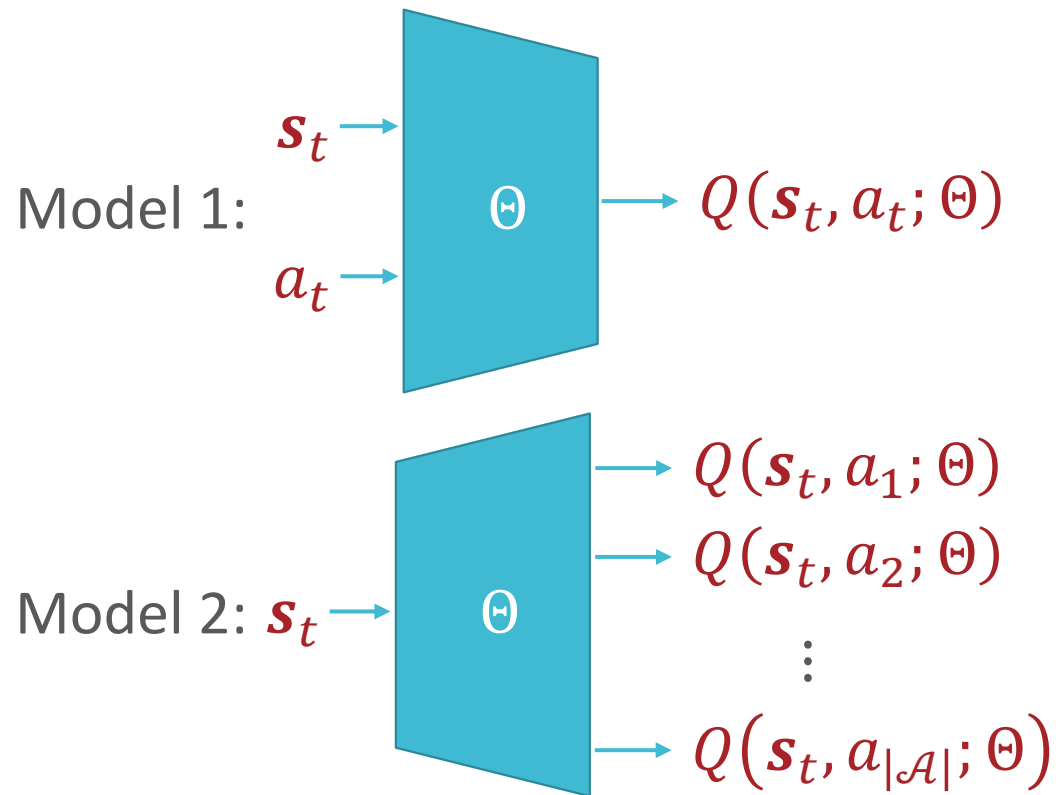
1. What can we do if the reward and/or transition functions/distributions are unknown?
 - Use online learning to gather data and learn $Q^*(s, a)$
2. How can we handle infinite (or just very large) state/action spaces?
 - Throw a neural network at it!

Deep Q-learning

- Use a parametric function, $Q(s, a; \Theta)$, to approximate $Q^*(s, a)$
 - Learn the parameters using SGD
 - Training data $(\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1})$ 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 neural network



Deep Q-learning: Loss Function

- “True” loss

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

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

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

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

$$\ell(\Theta^{(t)}, \Theta^{(t+1)}) = \left(\underbrace{y - Q(s, a; \Theta^{(t+1)})}_{\approx 0} \right)^2$$

Deep Q-learning

Algorithm 4: Online learning (parametric form)

- Inputs: discount factor γ , an initial state s_0 , learning rate α
- 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
$$\Theta^{(t+1)} \leftarrow \Theta^{(t)} - \alpha \nabla_{\Theta^{(t+1)}} \ell(\Theta^{(t)}, \Theta^{(t+1)})$$

where

$$\begin{aligned} & \nabla_{\Theta^{(t+1)}} \ell(\Theta^{(t)}, \Theta^{(t+1)}) \\ &= 2 \left(\gamma - Q(s, a; \Theta^{(t+1)}) \right) \nabla_{\Theta^{(t+1)}} Q(s, a; \Theta^{(t+1)}) \end{aligned}$$

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

- 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
 - Both are guaranteed to converge under some assumptions
- 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