10-301/601: Introduction to Machine Learning Lecture 23: Q-learning and Deep RL

Henry Chai & Matt Gormley

11/21/22

Front Matter

- Announcements
  - HW7 released 11/11, due 11/21 (today!) at 11:59 PM
  - HW8 released 11/21 (today!), due 12/2 at 11:59 PM
    - Please be mindful of your grace day usage

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?

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: $R(s, a)$, $p(s' \mid s, a)$, 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 $a \in \mathcal{A}$

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

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

- 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 π^*

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

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

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

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

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

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

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

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

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

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

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

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

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

- 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

    $$a = \underset{a' \in \mathcal{A}}{\operatorname{argmax}} \; Q(s,a')$$

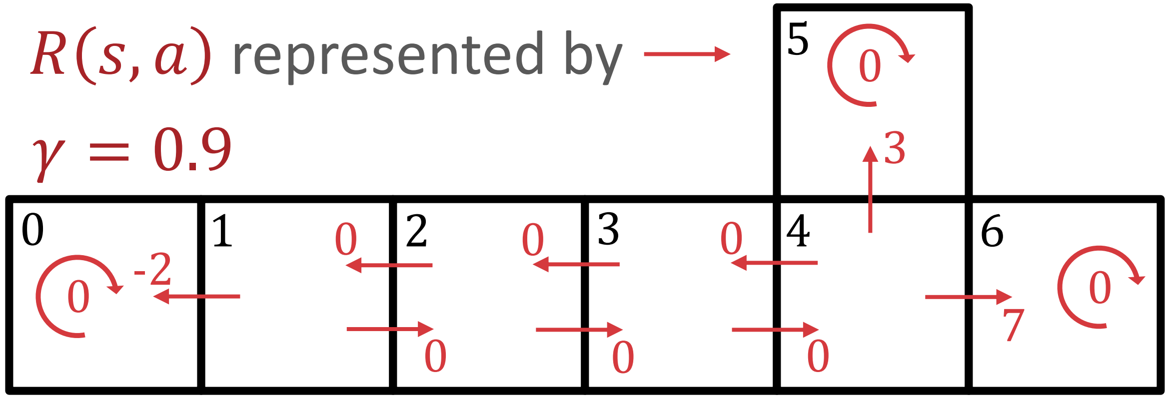
    Otherwise, with probability $1 - \epsilon$, take a random action a
  - Receive reward $r = R(s,a)$
  - Update the state: $s \leftarrow s'$ where $s' = \delta(s,a)$
  - Update $Q(s,a)$:

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

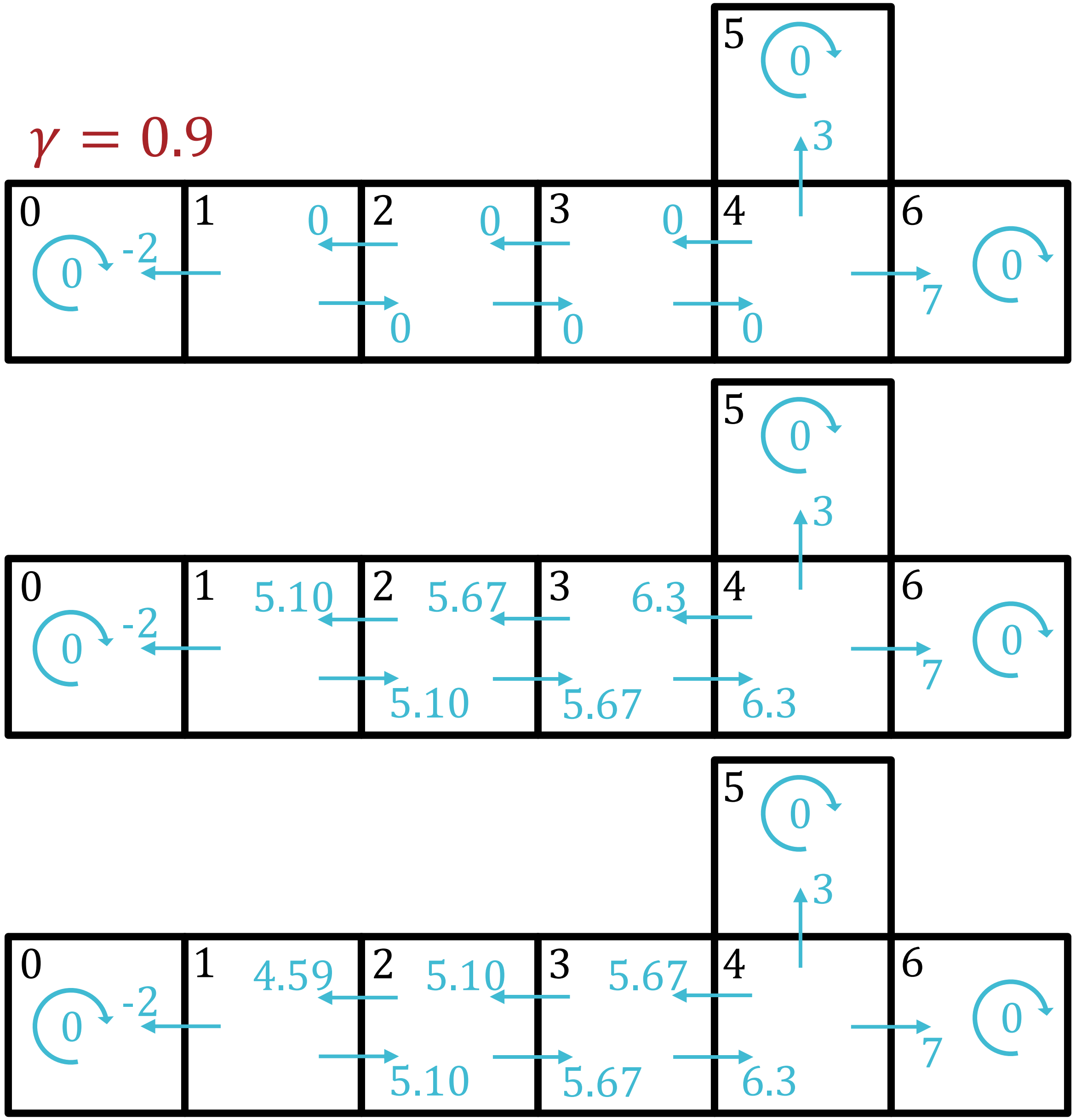
Learning $Q^*(s,a)$: Example

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

$\gamma = 0.9$



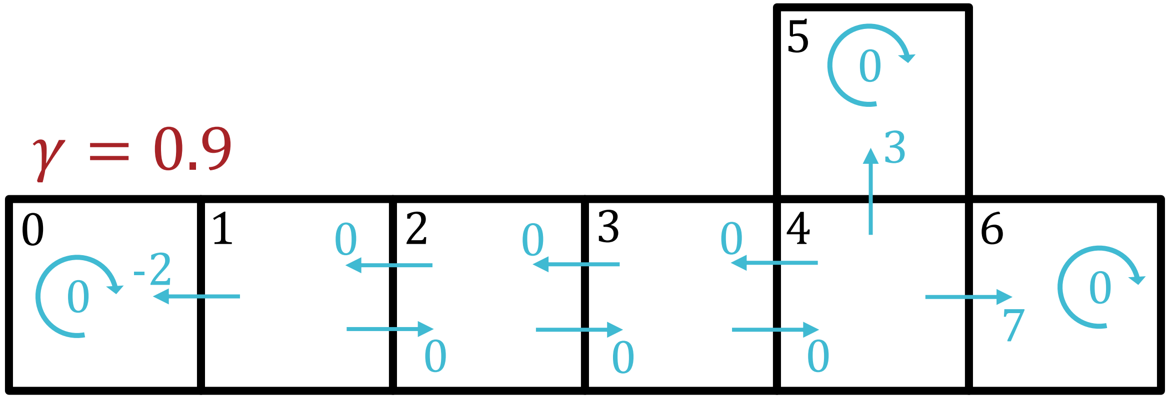
Which set of blue arrows (roughly) corresponds to $Q^*(s,a)$?



Poll Question 1:

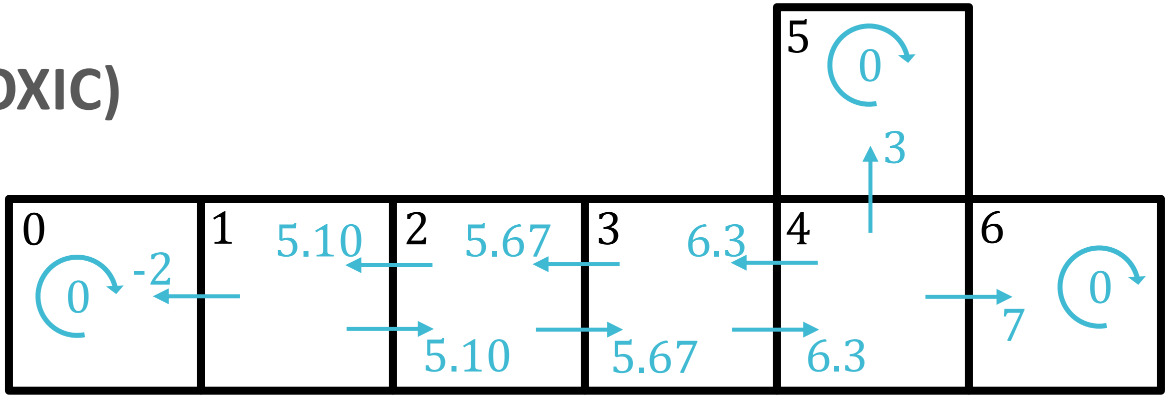
Which set of blue arrows (roughly) corresponds to $Q^*(s,a)$?

A.

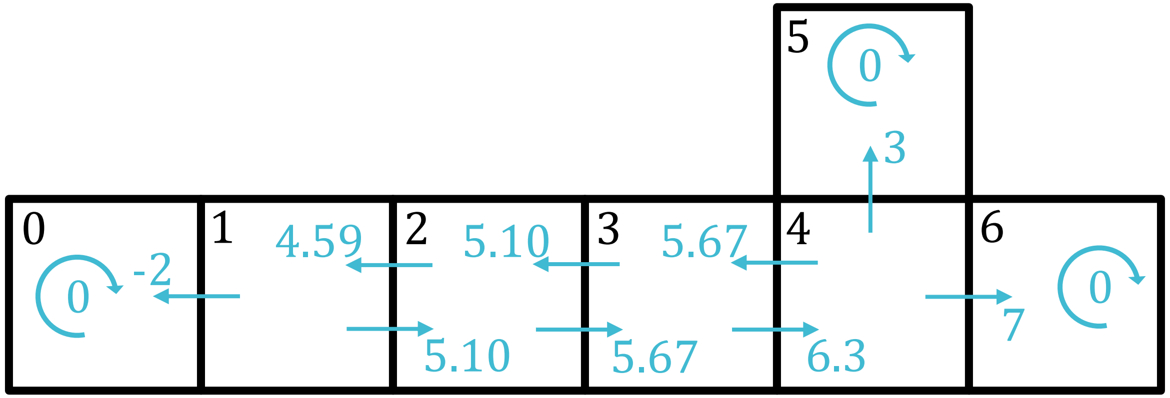


B. **(TOXIC)**

C.



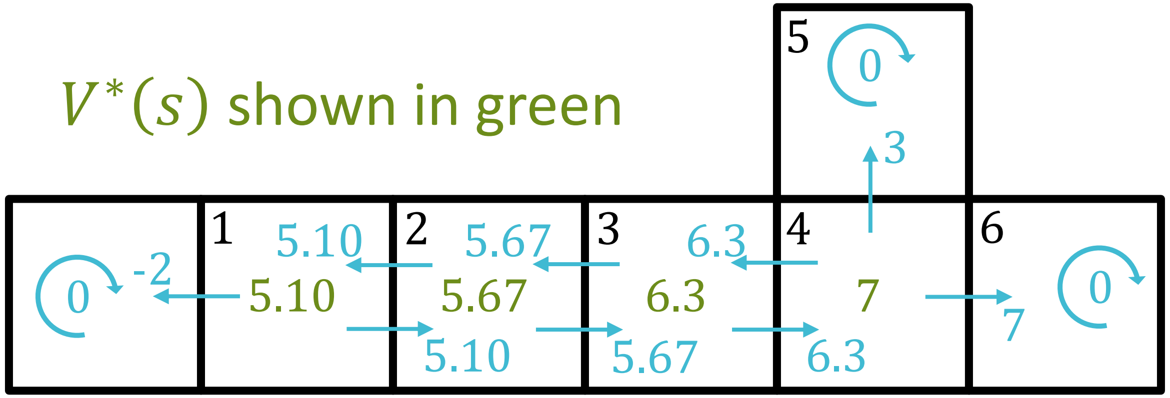
D.



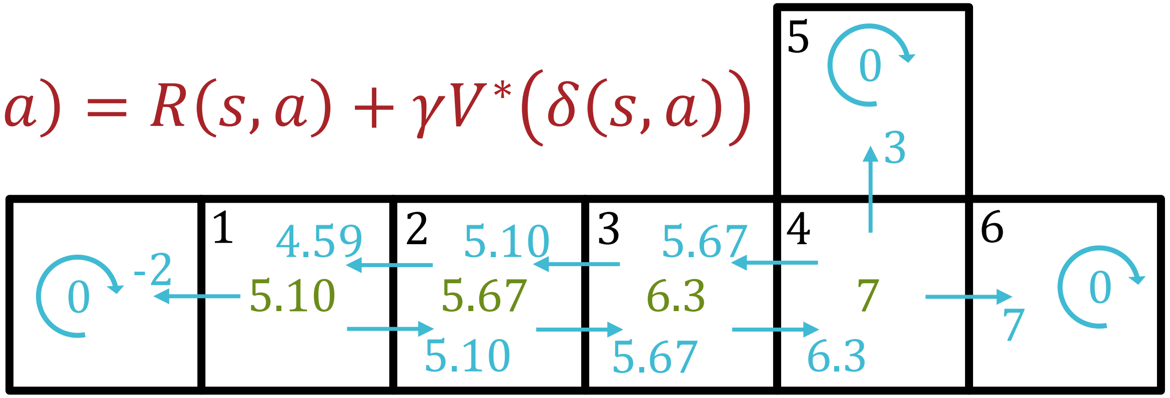
Poll Question 1:

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

$V^*(s)$ shown in green



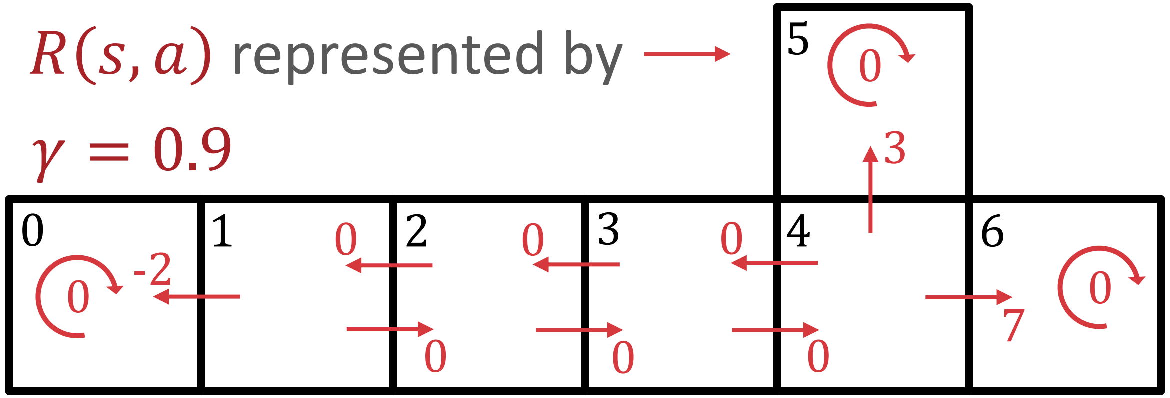
$$Q^*(s, a) = R(s, a) + \gamma V^*(\delta(s, a))$$



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

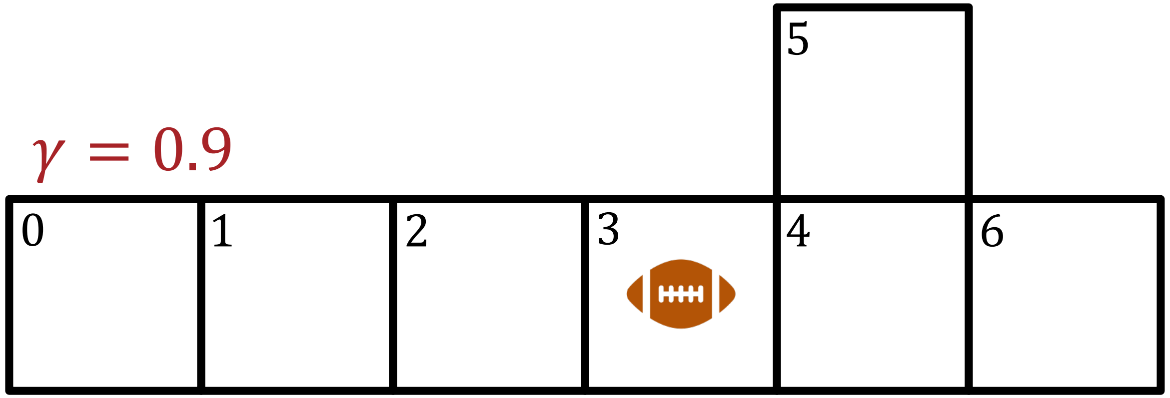
$R(s,a)$ represented by →

$\gamma = 0.9$



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

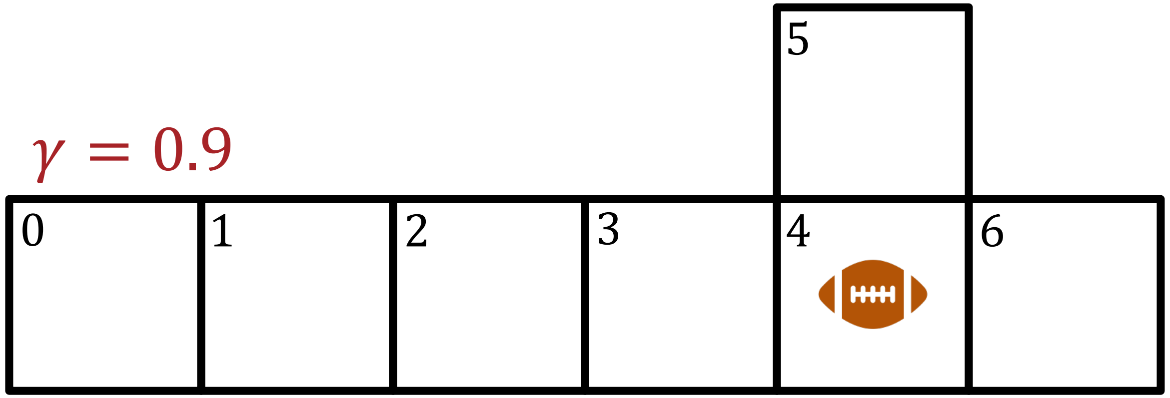
$\gamma = 0.9$



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

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

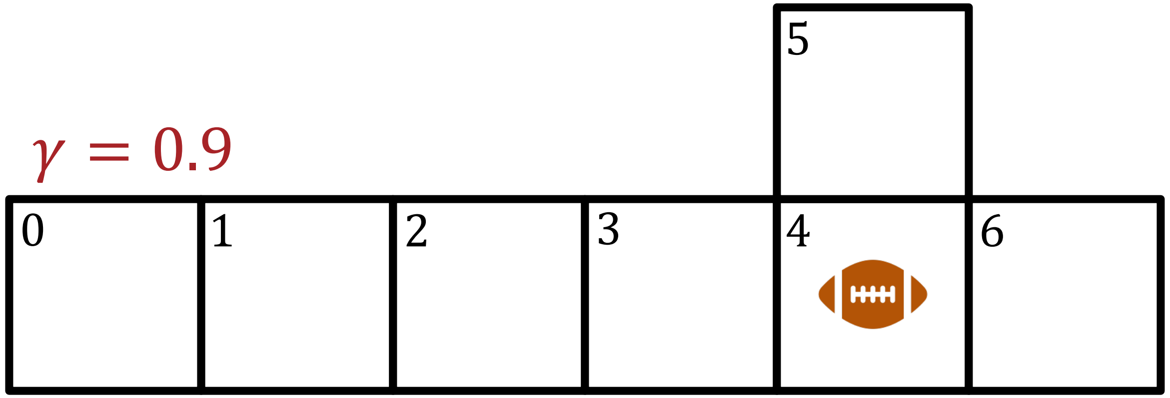
$\gamma = 0.9$



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

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

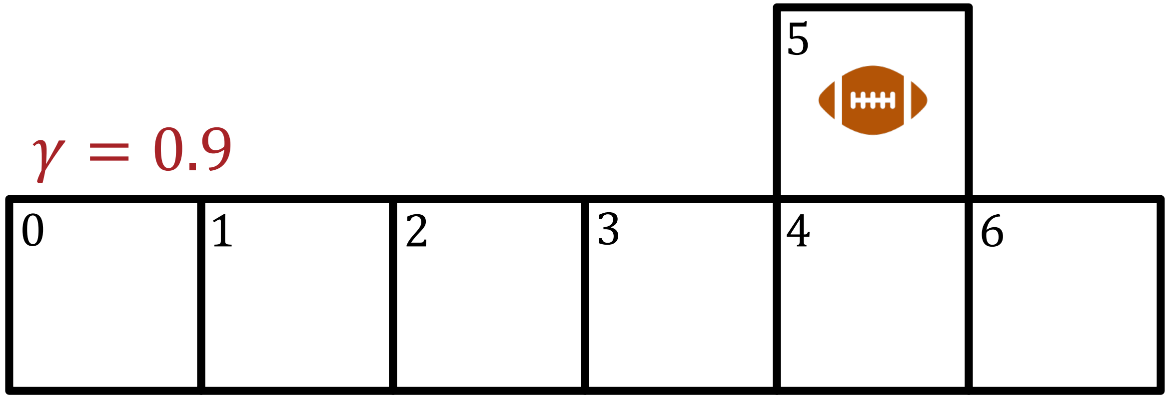
Learning $Q^*(s,a)$: Example



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

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

$\gamma = 0.9$

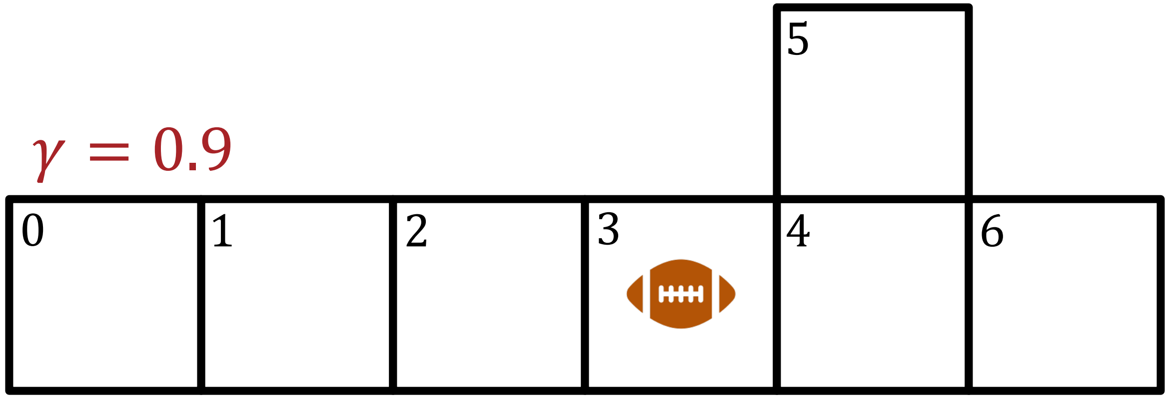


$$Q(4,\uparrow) \leftarrow 3 + (0.9) \max_{a' \in \{\rightarrow,\leftarrow,\uparrow,\circlearrowright\}} Q(5,a') = 3$$

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

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

$\gamma = 0.9$

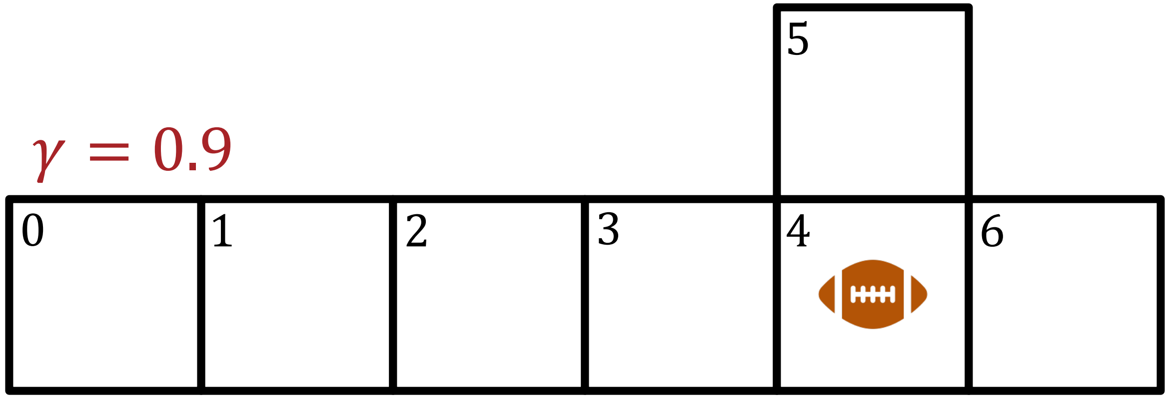


$$Q(3,\rightarrow) \leftarrow 0 + (0.9) \max_{a' \in \{\rightarrow,\leftarrow,\uparrow,\circlearrowright\}} Q(4,a') = 2.7$$

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

$\gamma = 0.9$



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

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

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
    $$a = \underset{a' \in \mathcal{A}}{\operatorname{argmax}} \; Q(s, a')$$
    Otherwise, with probability $1 - \epsilon$, take a random action a
  - Receive reward $r = R(s, a)$
  - 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
    $$a = \underset{a' \in \mathcal{A}}{\operatorname{argmax}} \; Q(s, a')$$
    Otherwise, with probability $1 - \epsilon$, take a random action a
  - Receive reward $r = R(s, a)$
  - Update the state: $s \leftarrow s'$ where $s' \sim p(s' \mid s, a)$
  - Update $Q(s, a)$:
    $$Q(s, a) \leftarrow (1 - \alpha) \underbrace{Q(s, a)}_{\text{Current value}} + \alpha \underbrace{\left(r + \gamma \max_{a'} Q(s', a') \right)}_{\text{Update w/ deterministic transitions}}$$

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
    $$a = \underset{a' \in \mathcal{A}}{\operatorname{argmax}} \; Q(s, a')$$
    Otherwise, with probability $1 - \epsilon$, take a random action a
  - Receive reward $r = R(s, a)$
  - Update the state: $s \leftarrow s'$ where $s' \sim p(s' \mid s, a)$
  - Update $Q(s, a)$:

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

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., $\alpha_t = 1/_{t+1}$

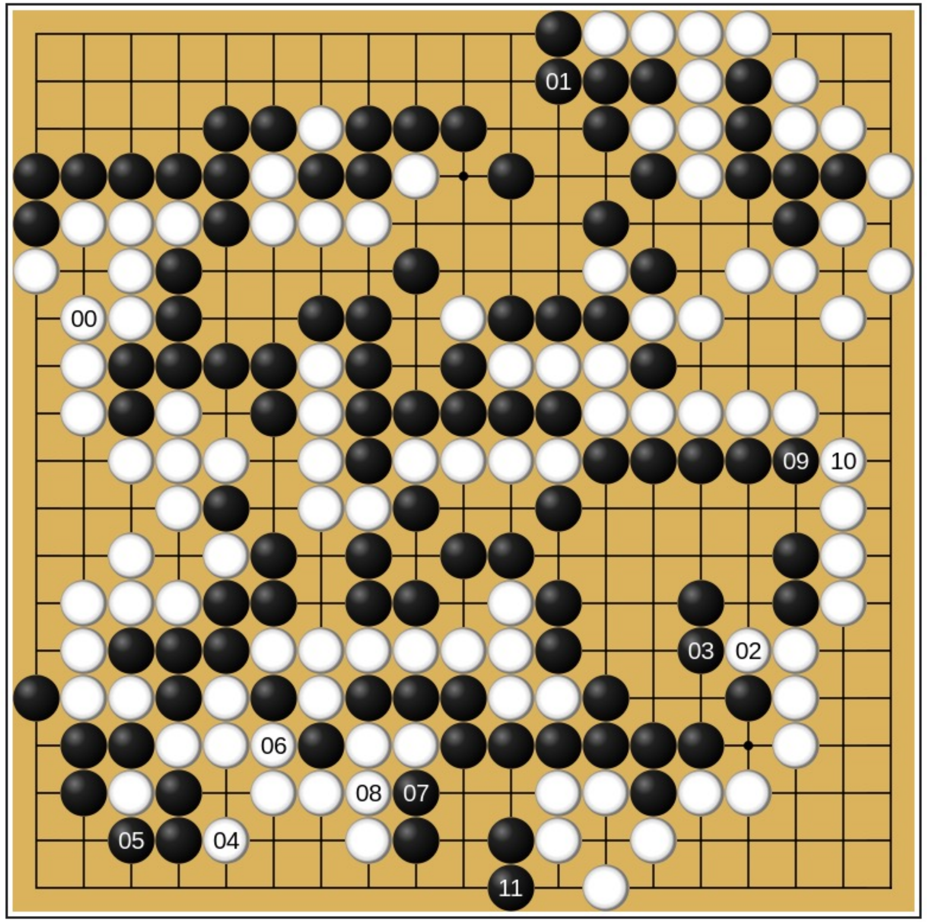
Two big Q's

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?

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)



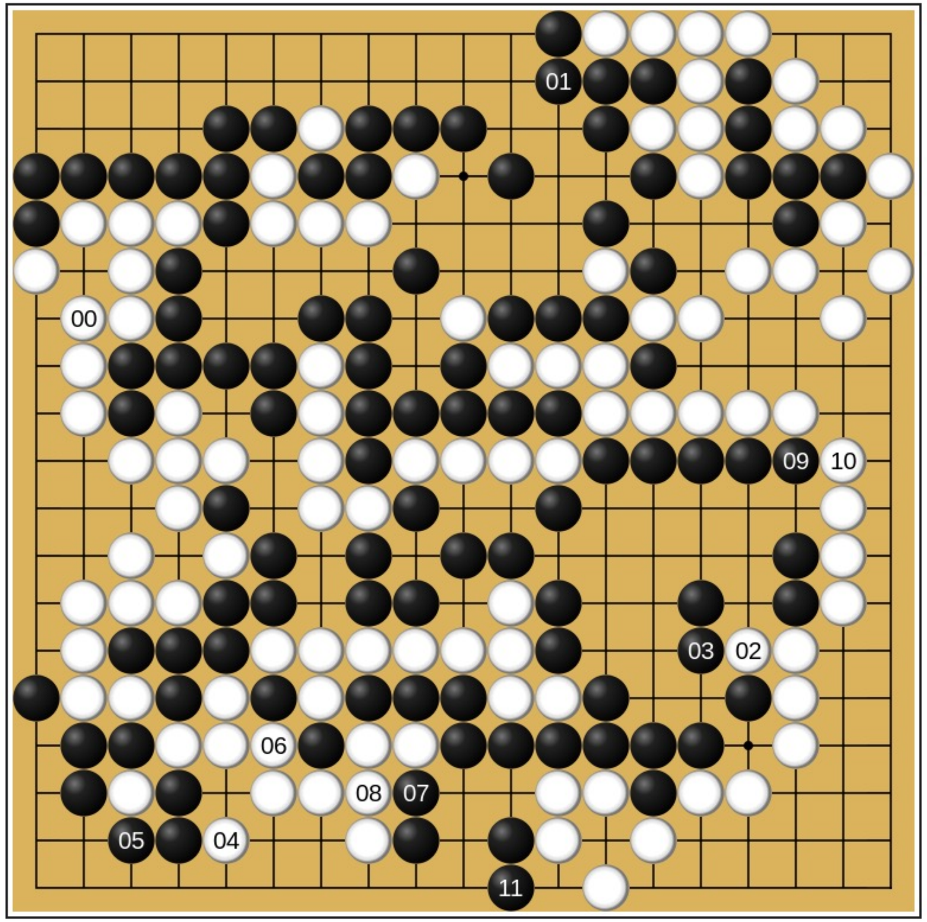
Playing Go

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

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

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

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



Poll Question 2:

Which is the best approximation to the number of legal board states in Go?

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

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

C. A googol $= 10^{100}$

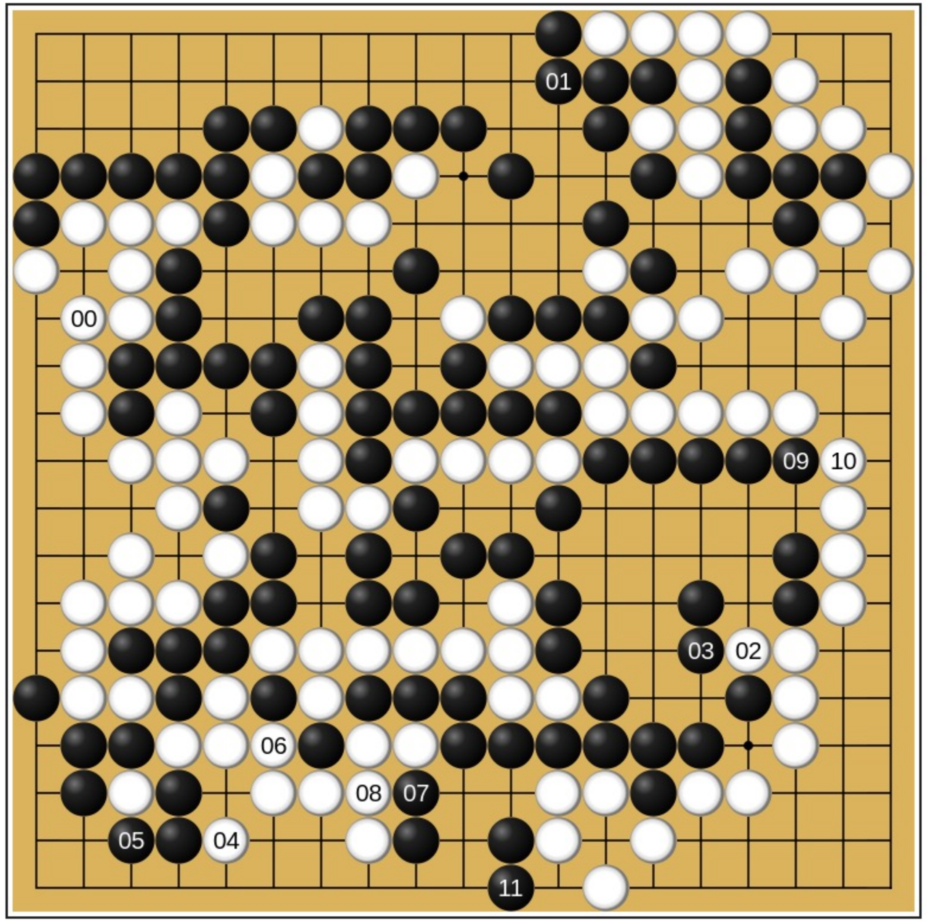
D. The number of possible *games* of chess $\sim 10^{120}$

E. A googolplex $= 10^{\text{googol}}$

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

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

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



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

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

Playing Go

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

Two big Q's

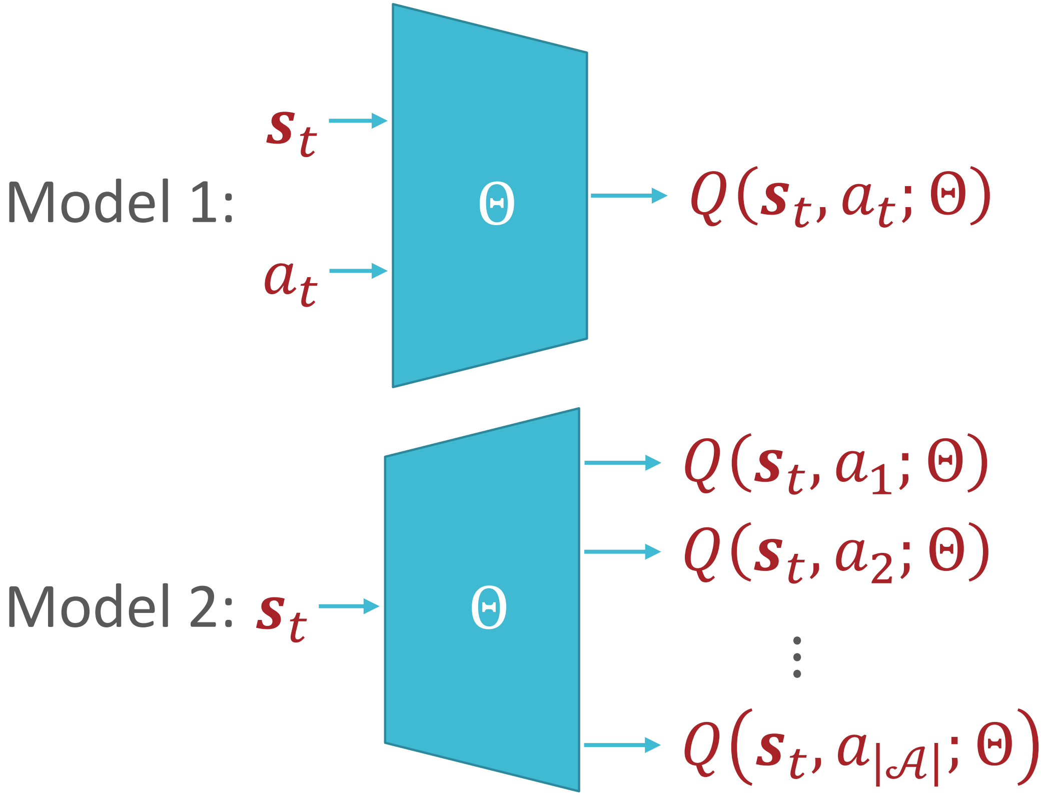
1. What can we do if the reward and/or transition functions/distributions are unknown?
   - Use online learning to gather data and learn $Q^*(s, a)$
2. How can we handle infinite (or just very large) state/action spaces?
   - Throw a neural network at it!

Deep Q-learning

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

Deep Q-learning: Model

- Represent states using some feature vector $\boldsymbol{s}_t \in \mathbb{R}^M$ e.g. for Go, $\boldsymbol{s}_t = [1, 0, -1, \dots, 1]^T$
- Define a neural network architecture



Deep Q-learning: Loss Function

- "True" loss

2. Don't know Q^*

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

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

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

   - 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 (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 $(\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}_{t+1})$
  - Update $\Theta^{(t)}$ by taking a step opposite the gradient

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

    where

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

Deep Q-learning: Experience Replay

- Issue: 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 $e_t = (\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}_{t+1})$ (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)

Q-learning and Deep RL Learning Objectives

You should be able to…

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