10-301/601: Introduction to Machine Learning Lecture 22: Q-learning and Deep RL

Matt Gormley & Henry Chai

11/13/24

Front Matter

- Announcements
  - HW7 released 11/7, due 11/17 at 11:59 PM
    - Please be mindful of your grace day usage (see the course syllabus for the policy)
  - Exam 2 viewings happening this week on Tuesday, Wednesday and Thursday, after our regularly scheduled OH
    - Please check the OH calendar for exact times and locations

Q: How can I get one of those sweet hoodies you're wearing?

- Applications are due by Wednesday, November 20th (1 week from today)
- For more information and the application, see https://www.ml.cmu.edu/academics/ta.html

Recall: Synchronous vs. Asynchronous Value Iteration

Algorithm 1 Asynchronous Value Iteration

1: **procedure** ASYNCHRONOUSVALUEITERATION($R(s,a)$, $p(\cdot|s,a)$)
2: Initialize value function $V(s) = 0$ or randomly
3: **while** not converged **do**
4: **for** $\boxed{s \in \mathcal{S}}$ **do**
5: $V(s) = \max_a R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V(s')$
6: Let $\pi(s) = \text{argmax}_a R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V(s'), \ \forall s$
7: **return** π

asynchronous updates: compute and update V(s) for each state one at a time

Algorithm 1 Synchronous Value Iteration

1: **procedure** SYNCHRONOUSVALUEITERATION($R(s,a)$, $p(\cdot|s,a)$)
2: Initialize value function $V(s)^{(0)} = 0$ or randomly
3: $t = 0$
4: **while** not converged **do**
5: **for** $s \in \mathcal{S}$ **do**
6: $V(s)^{(t+1)} = \max_a R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V(s')^{(t)}$
7: $t = t + 1$
8: Let $\pi(s) = \text{argmax}_a R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V(s'), \ \forall s$
9: **return** π

synchronous updates: compute all the fresh values of V(s) from all the stale values of V(s), then update V(s) with fresh values

Recall:

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

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

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

- **Theorem 3**: Policy convergence

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

Q: What happens when the rewards are stochastic?

- **Theorem 1**: Value function convergence

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

- **Theorem 2**: Convergence criterion

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

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

- **Theorem 3**: Policy convergence

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

Stochastic Rewards

- Insight: one way of representing stochastic rewards is with *deterministic* rewards that depend on the next state, s_{t+1}, assuming transitions are stochastic

From the Environment (i.e. the MDP)

- State space, $\mathcal{S}$
- Action space, $\mathcal{A}$
- $\boxed{\text{Reward function, } R(s, a, s'),\ R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}}$
- 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

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

Stochastic Rewards

- Insight: one way of representing stochastic rewards is with *deterministic* rewards that depend on the next state, s_{t+1}, assuming transitions are stochastic

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

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

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

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

Stochastic Rewards

- Insight: one way of representing stochastic rewards is with *deterministic* rewards that depend on the next state, s_{t+1}, assuming transitions are stochastic

Algorithm 1 Value Iteration (stochastic transitions, 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**
      $\boxed{V(s) = \max_a \sum_{s' \in \mathcal{S}} p(s'|s, a)(R(s, a, s') + \gamma V(s'))}$
  Let $\pi(s) = \text{argmax}_a \sum_{s' \in \mathcal{S}} p(s'|s, a)(R(s, a, s') + \gamma V(s'))$, $\forall s$
  return π

Q: If the thing we care about learning is the policy, why don't we just learn that directly?

- Insight: one way of representing stochastic rewards is with *deterministic* rewards that depend on the next state, s_{t+1}, assuming transitions are stochastic

Algorithm 1 Value Iteration (stochastic transitions, stochastic rewards)

1: **procedure** VALUEITERATION($R(s,a,s')$ reward function, $p(\cdot|s,a)$ transition probabilities)
2: Initialize value function $V(s) = 0$ or randomly
3: **while** not converged **do**
4: **for** $s \in \mathcal{S}$ **do**
5: $\boxed{V(s) = \max_a \sum_{s' \in \mathcal{S}} p(s'|s,a)(R(s,a,s') + \gamma V(s'))}$
6: Let $\pi(s) = \text{argmax}_a \sum_{s' \in \mathcal{S}} p(s'|s,a)(R(s,a,s') + \gamma V(s')), \forall s$
7: **return** π

Policy Iteration

- Inputs: $R(s,a), p(s' \mid s,a)$
- Initialize π randomly
- While not converged, do:
    - Solve the Bellman equations defined by policy π

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

    - Update π

$$\forall s,\ \pi(s) \leftarrow \underset{a \in \mathcal{A}}{\operatorname{argmax}} \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) V^\pi(s') \right]$$

- Return π

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

MDP and Value/Policy Iteration Learning Objectives

You should be able to…

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

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?

(Asynchronous) 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 $s \in \mathcal{S}$

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

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

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

- Return π^*

(Asynchronous) Value Iteration Revisited

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

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

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

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

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

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

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

- 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$]$

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

$s' = \delta(s,a)$

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

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

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

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

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

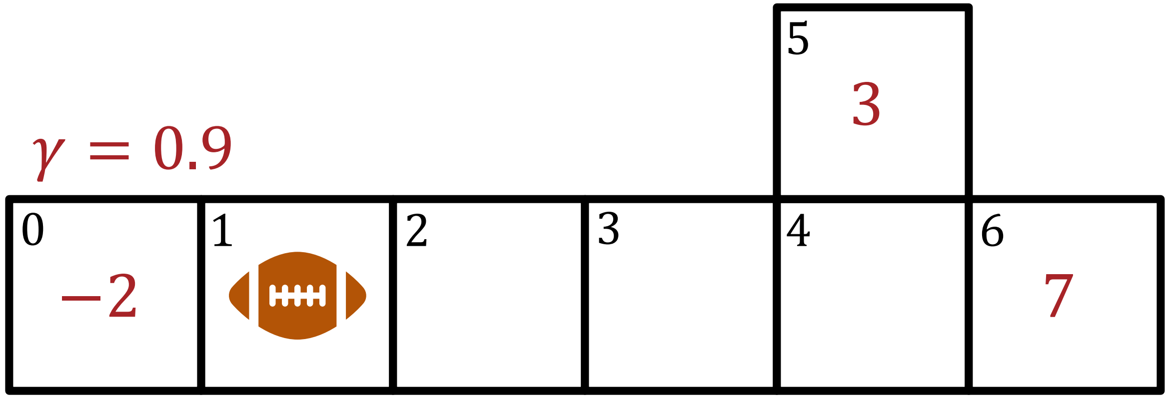
Learning $Q^*(s, a)$ w/ deterministic rewards

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 $Q(s, a)$:

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

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

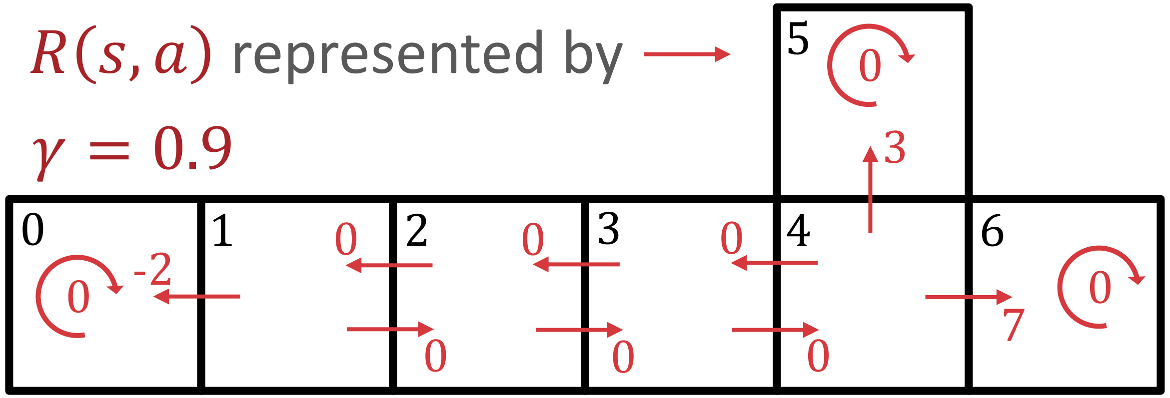


$$R(s,a) = \begin{cases} -2 \text{ if entering state 0 (safety)} \\ 3 \text{ if entering state 5 (field goal)} \\ 7 \text{ if entering state 6 (touch down)} \\ 0 \text{ otherwise} \end{cases}$$

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

$R(s, a)$ represented by →

$\gamma = 0.9$

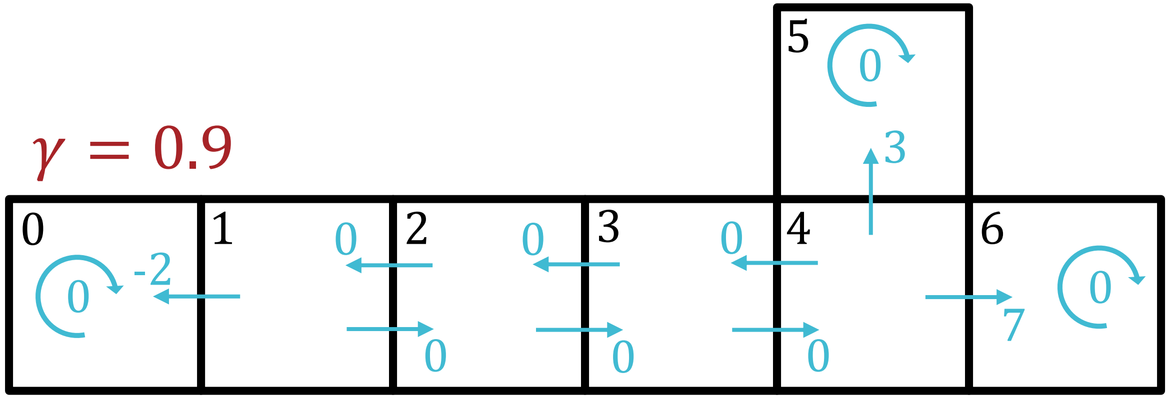


Poll Question 1:

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

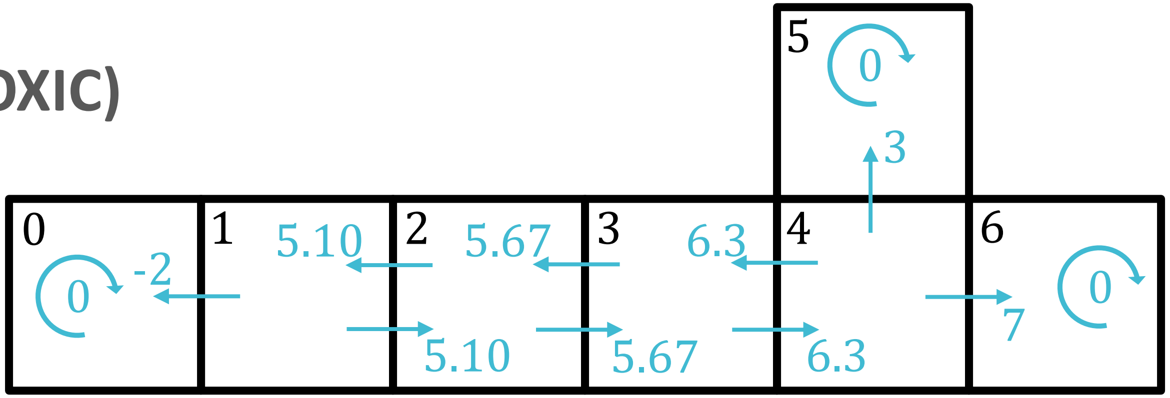
$\gamma = 0.9$

A.

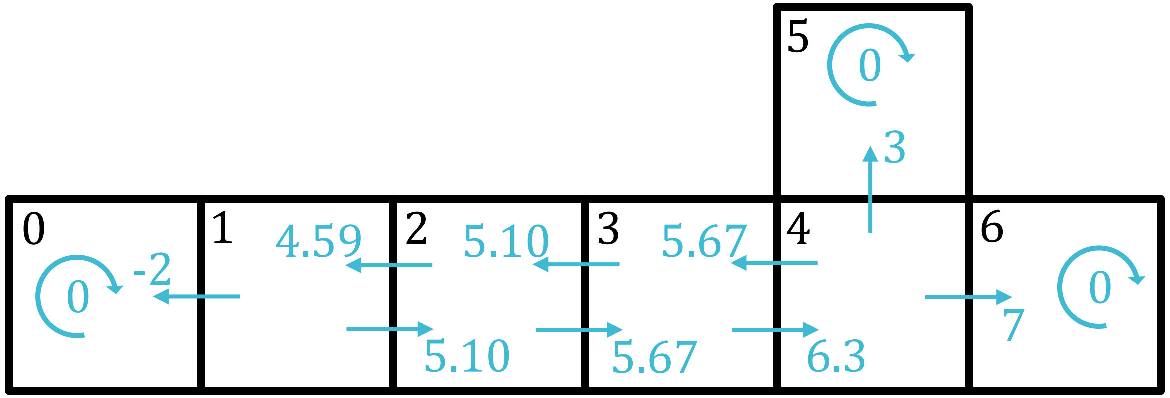


B. **(TOXIC)**

C.



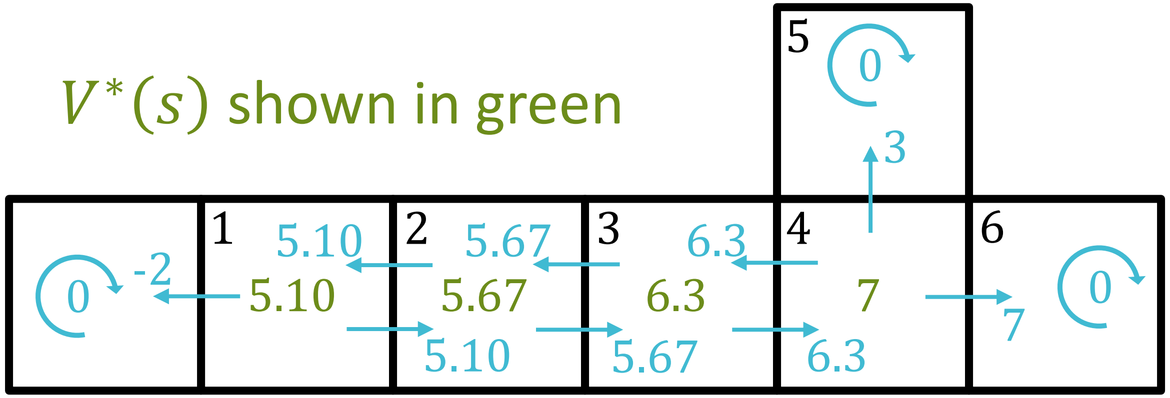
D.



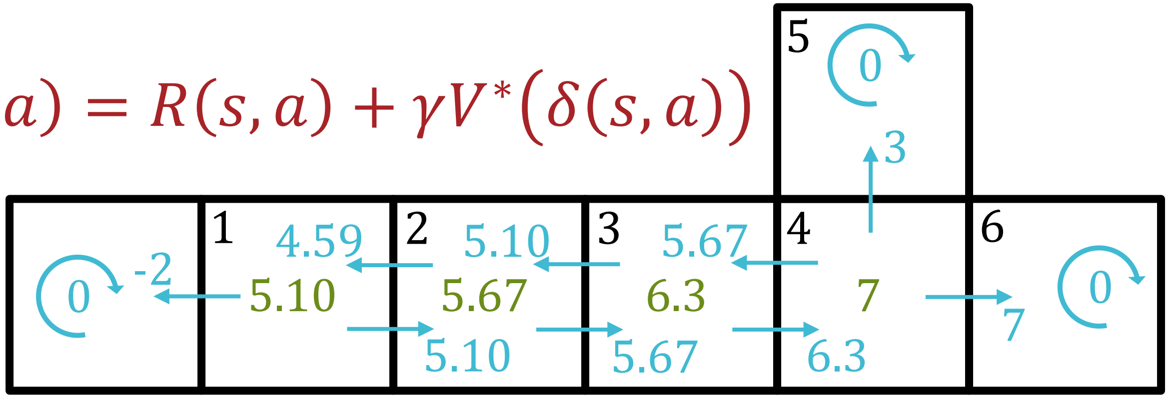
Poll Question 1:

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

$V^*(s)$ shown in green



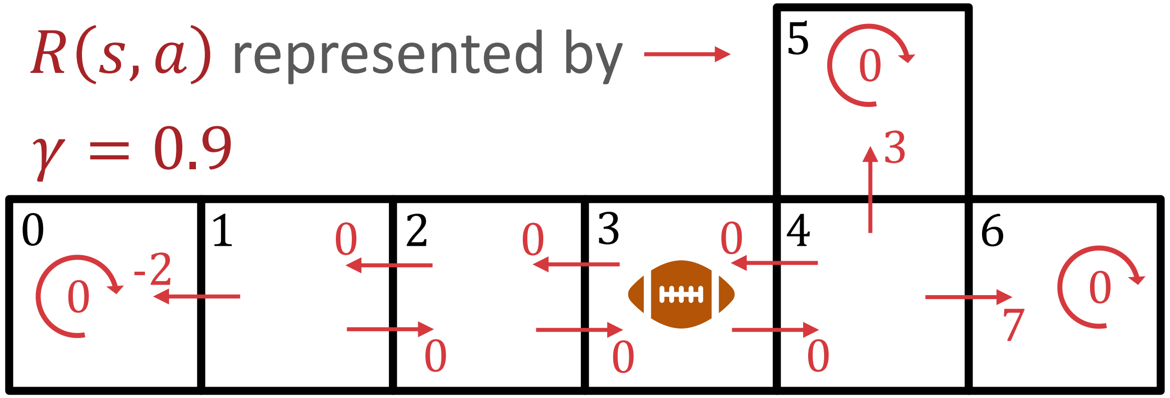
$$Q^*(s,a) = R(s,a) + \gamma V^*\big(\delta(s,a)\big)$$



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

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

$\gamma = 0.9$

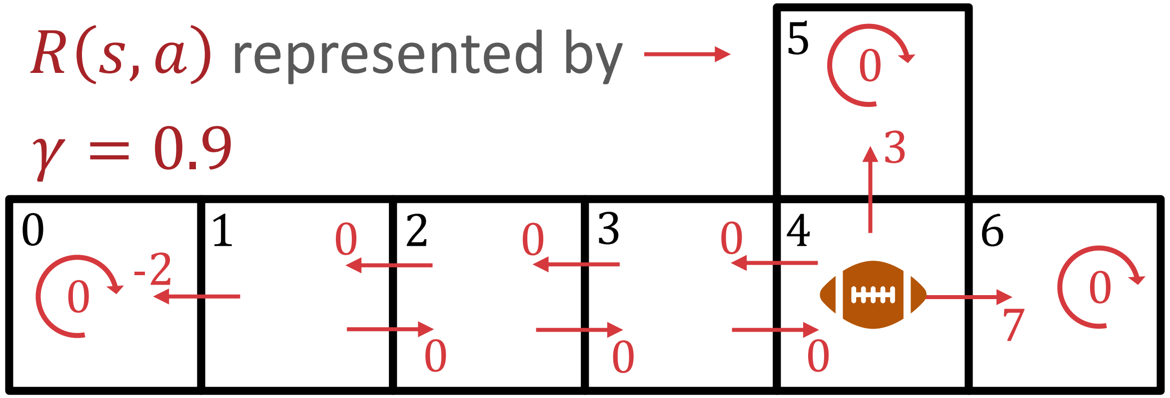


$Q(s, a)$	$\rightarrow$	$\leftarrow$	$\uparrow$	$\circlearrowright$
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0

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

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

$\gamma = 0.9$



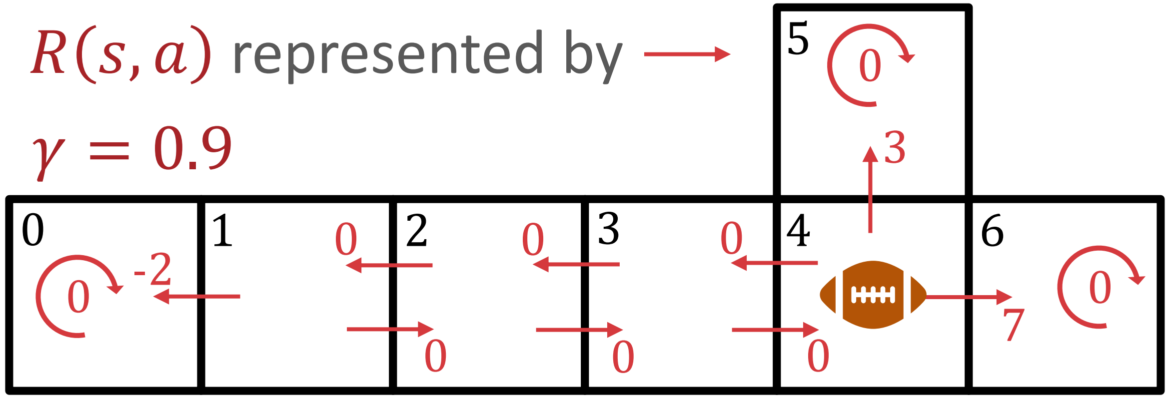
$$Q(3,\rightarrow) \leftarrow 0 + (0.9) \max_{a' \in \{\rightarrow,\leftarrow,\uparrow,\circlearrowright\}} Q(4,a') = 0$$

$Q(s,a)$	→	←	↑	↻
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 →

$\gamma = 0.9$

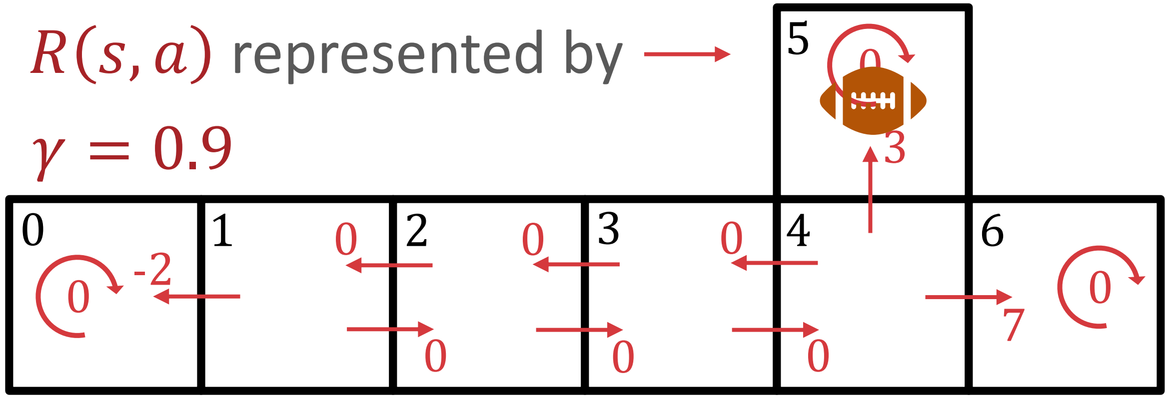


$Q(s,a)$	→	←	↑	↻
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0

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

$R(s,a)$ represented by →

$\gamma = 0.9$



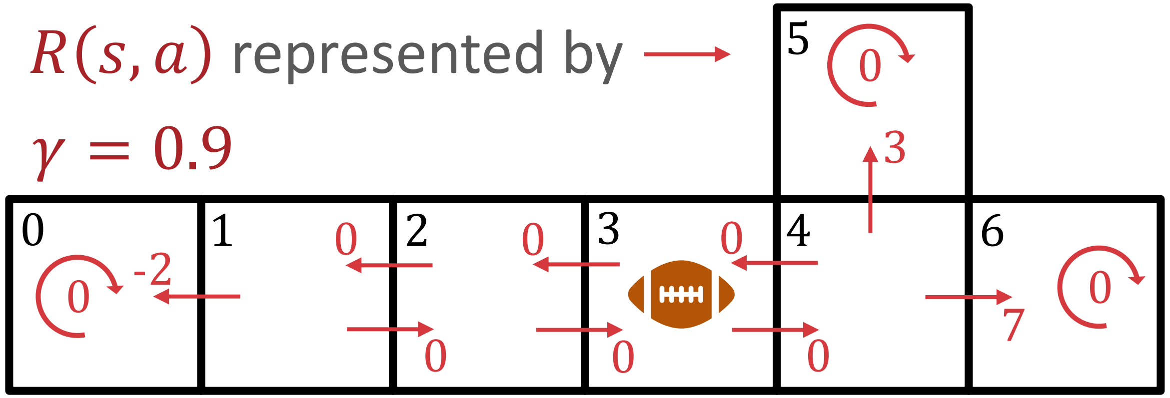
$$Q(4,\uparrow) \leftarrow 3 + (0.9) \max_{a' \in \{\rightarrow,\leftarrow,\uparrow,\circlearrowright\}} Q(5,a') = 3$$

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

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

$R(s,a)$ represented by →

$\gamma = 0.9$

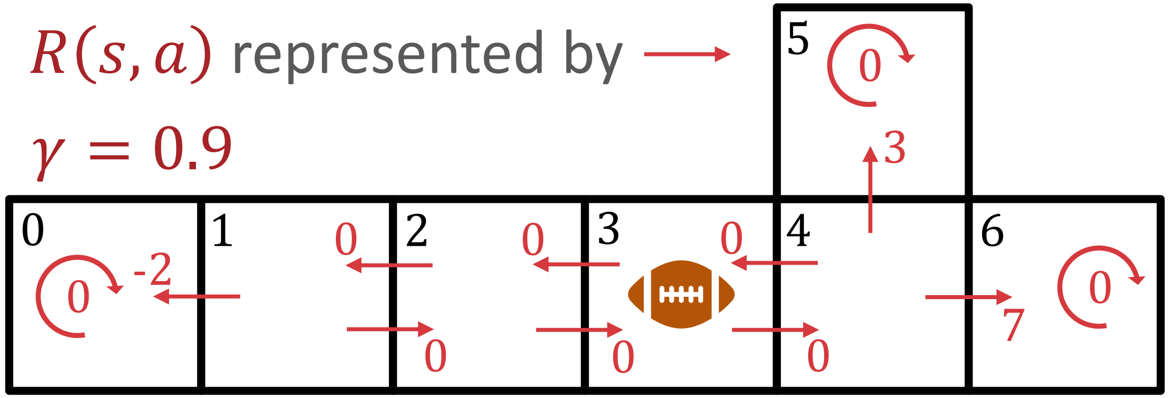


$$Q(3,\rightarrow) \leftarrow 0 + (0.9) \max_{a' \in \{\rightarrow,\leftarrow,\uparrow,\circlearrowright\}} Q(4,a') = 2.7$$

$Q(s,a)$	→	←	↑	↻
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~~ 7	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

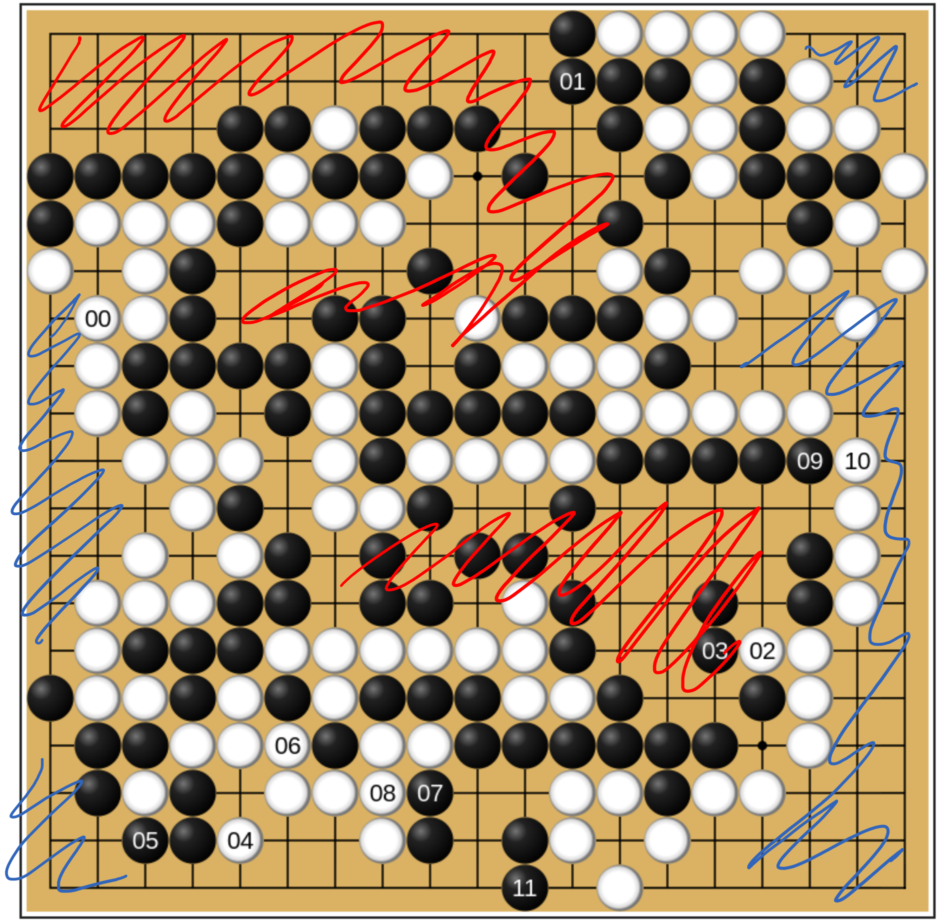
Learning $Q^*(s,a)$: Convergence

- 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?
   - A: Use online learning to gather data and learn $Q^*(s,a)$
2. How can we handle infinite (or just very large) state/action spaces?

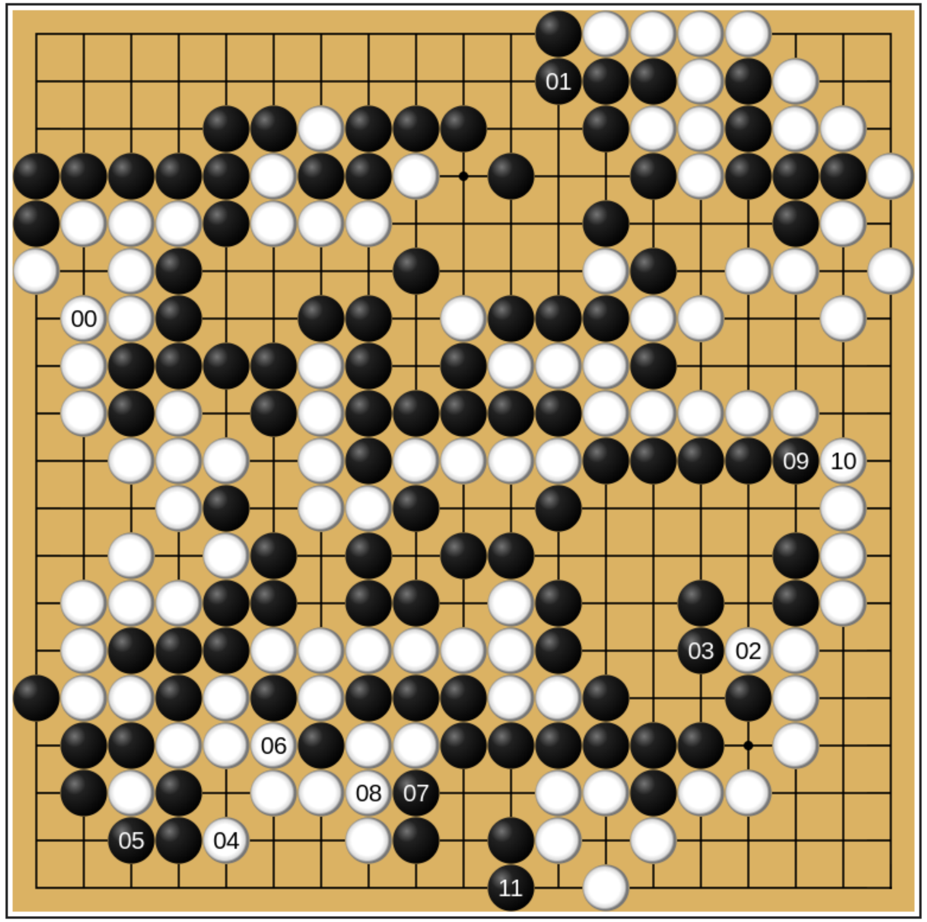
AlphaGo (Black) vs. Lee Sedol (White) Game 2 final position (AlphaGo wins)



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

Playing Go

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



Poll Question 2: Which is the best approximation to the number of legal board states in Go?

A. 42 **(TOXIC)**

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

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

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

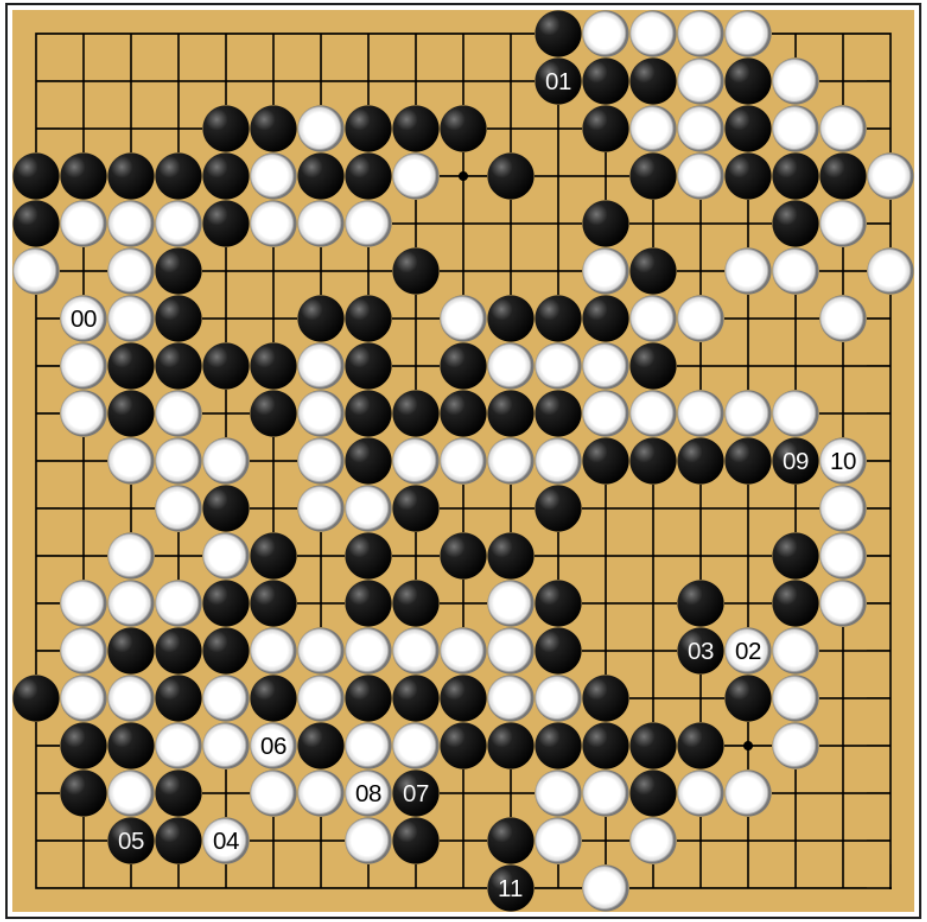
E. The number of possible *games* of chess $\sim 10^{120}$

F. 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)



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!

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

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

Two big Q's

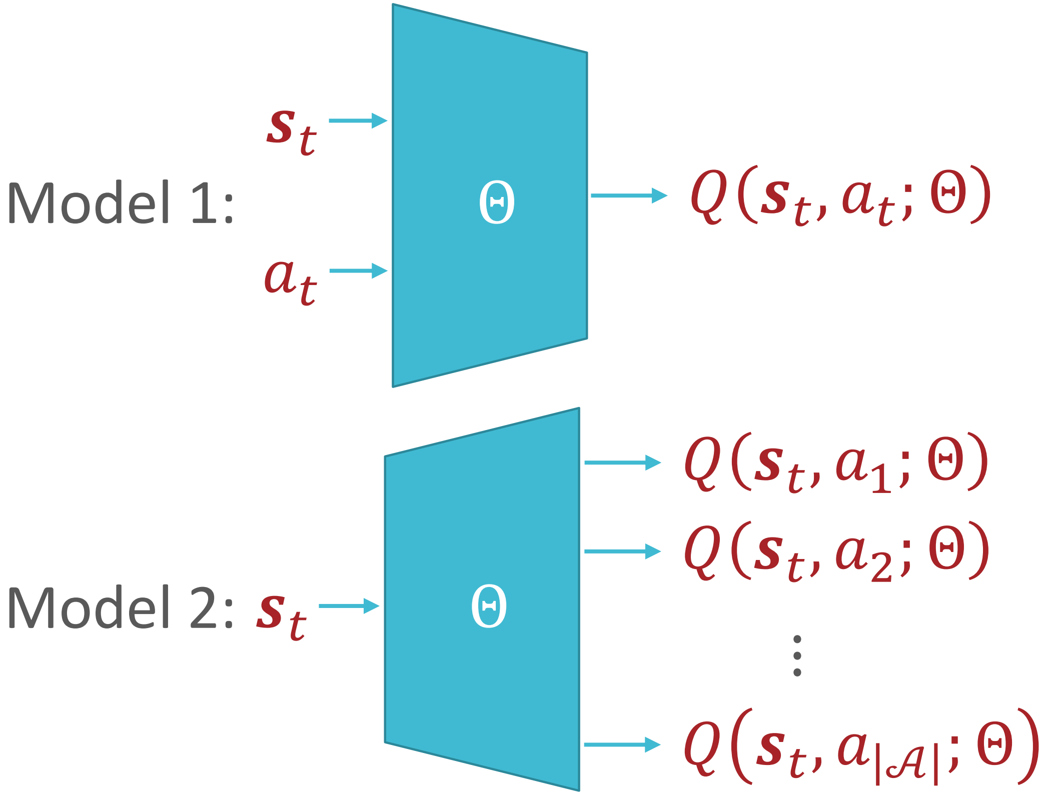
1. What can we do if the reward and/or transition functions/distributions are unknown?
   - A: Use online learning to gather data and learn $Q^*(s,a)$
2. How can we handle infinite (or just very large) state/action spaces?
   - A: Throw a neural network at it!

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



Deep Q-learning: Loss Function

- "True" loss

$$\ell(\Theta) = \underbrace{\sum_{s \in \mathcal{S}}}_{\text{1. } \mathcal{S} \text{ too big to compute this sum}} \sum_{a \in \mathcal{A}} \Big(\overbrace{Q^*(s,a)}^{\text{2. Don't know } Q^*} - Q(s,a;\Theta)\Big)^2$$

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

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

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

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

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

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

where

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

Deep Q-learning: Experience Replay

- SGD assumes iid training samples but in RL, samples are *highly* correlated
- Idea: maintain a "replay buffer" $\mathcal{D} = \{e_1, e_2, \dots, e_N\}$ of the N most recent experiences $e_t = (\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}_{t+1})$ (Lin, 1992)
  - Keeps the agent from "forgetting" recent experiences
- In each iteration, we:
  1. Sample some experience e_i (or a mini-batch of experiences $E = \{e_1, \dots, e_T\}$) uniformly at random from $\mathcal{D}$ and apply the Q-learning update
  2. Add 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