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?

• Use online learning to gather data and learn $Q_*$

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: $R(s, a), p(s' | s, a), \gamma$
- 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 $a \in \mathcal{A}$
      - $Q(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s' | s, a)V(s')$
      - $V(s) \leftarrow \max_{a \in \mathcal{A}} Q(s, a)$
  - For $s \in \mathcal{S}$
    - $\pi^*(s) \leftarrow \arg\max_{a \in \mathcal{A}} R(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s' | s, a)V(s')$
- Return $\pi^*$
\( Q^*(s, a) \) w/ deterministic rewards

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

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

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

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

\[ \pi^*(s) = \arg\max_{a \in \mathcal{A}} Q^*(s, a) \]

\[ \cdot\text{Insight: if we know } Q^*, \text{ we can compute an optimal policy } \pi^*! \]
\( Q^*(s, a) \) w/ deterministic rewards and transitions

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

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

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

- \( \pi^*(s) = \arg\max_{a \in \mathcal{A}} Q^*(s, a) \)

  Insight: if we know \( Q^* \), we can compute an optimal policy \( \pi^* \)!
Learning $Q^*(s, a)$ w/ deterministic rewards and transitions

Algorithm 1: Online learning (table form)

- Inputs: discount factor $\gamma$, an initial state $s$

- Initialize $Q(s, a) = 0 \forall s \in S, a \in A$ ($Q$ is a $|S| \times |A|$ 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: $\varepsilon$-greedy online learning (table form)

- **Inputs:** discount factor $\gamma$, an initial state $s$, greediness parameter $\varepsilon \in [0, 1]$

- **Initialize** $Q(s, a) = 0 \ \forall \ s \in S, a \in A$ ($Q$ is a $|S| \times |A|$ array)

- **While** TRUE, do
  - With probability $\varepsilon$, take the greedy action
    \[ a = \text{argmax}_{a' \in A} Q(s, a') \]
  - Otherwise, with probability $1 - \varepsilon$, 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 $\gamma = 0.9$
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.  

$\gamma = 0.9$
Poll Question 1:
Which set of blue arrows (roughly) corresponds to $Q^*(s, a)$?

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

$V^*(s)$ shown in green
Learning $Q^*(s, a)$:
Example

$R(s, a)$ represented by $\gamma = 0.9$

\[
\begin{array}{c|c|c|c|c|c|c|c|c}
 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]
Learning $Q^*(s, a)$: Example

$\gamma = 0.9$

<table>
<thead>
<tr>
<th>$Q(s, a)$</th>
<th>$\rightarrow$</th>
<th>$\leftarrow$</th>
<th>$\uparrow$</th>
<th>$\bigcup$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Learning $Q^*(s,a)$: Example

\[ \gamma = 0.9 \]

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

<table>
<thead>
<tr>
<th>$Q(s,a)$</th>
<th>$\rightarrow$</th>
<th>$\leftarrow$</th>
<th>$\uparrow$</th>
<th>$\downarrow$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Learning $Q^*(s, a)$: Example

$\gamma = 0.9$

<table>
<thead>
<tr>
<th>$Q(s, a)$</th>
<th>$\rightarrow$</th>
<th>$\leftarrow$</th>
<th>$\uparrow$</th>
<th>$\leftarrow$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Learning $Q^*(s, a)$: Example

\[ \gamma = 0.9 \]

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

<table>
<thead>
<tr>
<th>$Q(s, a)$</th>
<th>$\rightarrow$</th>
<th>$\leftarrow$</th>
<th>$\uparrow$</th>
<th>$\downarrow$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
\[ \gamma = 0.9 \]

\[ Q(3, \rightarrow) \leftarrow 0 + (0.9) \max_{a' \in \{\rightarrow, \leftarrow, \uparrow, \text{↻}\}} Q(4, a') = 2.7 \]

Learning \( Q^*(s, a) \):

Example

<table>
<thead>
<tr>
<th>( Q(s, a) )</th>
<th>( \rightarrow )</th>
<th>( \leftarrow )</th>
<th>( \uparrow )</th>
<th>( \text{↻} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>(\mathbf{3})</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Learning $Q^*(s, a)$: Example

$$\gamma = 0.9$$

$$Q(3, \rightarrow) \leftarrow 0 + (0.9 \max_{a' \in \{\rightarrow, \leftarrow, \uparrow, \downarrow\}} Q(4, a')) = 2.7$$
Learning $Q^*(s, a)$ w/ deterministic rewards and transitions

**Algorithm 2:** $\epsilon$-greedy online learning (table form)

- Inputs: discount factor $\gamma$, an initial state $s$, greediness parameter $\epsilon \in [0, 1]$

- Initialize $Q(s, a) = 0 \forall s \in S, a \in A$ ($Q$ is a $|S| \times |A|$ array)

- While TRUE, do
  - With probability $\epsilon$, take the greedy action
    $$a = \arg\max_{a' \in A} Q(s, a')$$
  
  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: $\epsilon$-greedy online learning (table form)

- Inputs: discount factor $\gamma$, an initial state $s$, greediness parameter $\epsilon \in [0, 1]$, learning rate $\alpha \in [0, 1]$ ("trust parameter")
- Initialize $Q(s, a) = 0 \forall s \in S, a \in A$ ($Q$ is a $|S| \times |A|$ array)
- While TRUE, do
  - With probability $\epsilon$, take the greedy action $a = \arg\max_{a' \in A} Q(s, a')$
  - 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 $Q(s, a)$:
    
    $Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha \left( r + \gamma \max_{a'} Q(s', a') \right)$

Current value

Update w/ deterministic transitions
Learning \( Q^*(s, a) \) w/ deterministic rewards

Algorithm 3: \( \varepsilon \)-greedy online learning (table form)

- Inputs: discount factor \( \gamma \), an initial state \( s \),
  greediness parameter \( \epsilon \in [0, 1] \),
  learning rate \( \alpha \in [0, 1] \) (“trust parameter”)

- Initialize \( Q(s, a) = 0 \) \( \forall s \in S, a \in A \) \( (Q \) is a \(|S| \times |A| \) array)

- While TRUE, do
  - With probability \( \epsilon \), take the greedy action
    \[ a = \text{argmax}_{a' \in A} Q(s, a') \]
  - 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 \( Q(s, a) \):
    \[ Q(s, a) \leftarrow Q(s, a) + \alpha \left( r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right) \]
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. $|R(s, a)| < \beta \ \forall \ s \in S, a \in A$
  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. $|R(s,a)| < \beta \ \forall \ s \in S, a \in A$
  4. Initial $Q$ values are finite
  5. Learning rate $\alpha_t$ follows some “schedule” s.t.
     $$\sum_{t=0}^{\infty} \alpha_t = \infty \text{ and } \sum_{t=0}^{\infty} \alpha_t^2 < \infty$$
     e.g., $\alpha_t = \frac{1}{t+1}$
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?
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](https://en.wikipedia.org/wiki/AlphaGo_versus_Lee_Sedol)
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 $= 10^{\text{googol}}$
F. TOXIC

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
- There are $\sim 10^{170}$ legal Go board states!

Source: [https://en.wikipedia.org/wiki/AlphaGo_versus_Lee_Sedol](https://en.wikipedia.org/wiki/AlphaGo_versus_Lee_Sedol)
Source: [https://en.wikipedia.org/wiki/Go_and_mathematics](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?
   • 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 $(s_t, a_t, r_t, s_{t+1})$ gathered online by the agent/learning algorithm
Represent states using some feature vector $s_t \in \mathbb{R}^M$
eq 1, ..., 1$^T$

Define a neural network architecture
Deep Q-learning: Loss Function

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
     \[ Q^*(s, a) \approx r + \gamma \max_{a'} Q(s', a'; \Theta^{(t)}) := y \]
   - Set the parameters in the next iteration \( \Theta^{(t+1)} \) such that
     \[ Q(s, a; \Theta^{(t+1)}) \approx y \] by minimizing the squared loss
     \[ \ell(\Theta^{(t)}, \Theta^{(t+1)}) = (y - Q(s, a; \Theta^{(t+1)}))^2 \]

- "True" loss

\[ \ell(\Theta) = \sum_{s \in S} \sum_{a \in A} (Q^*(s, a) - Q(s, a; \Theta))^2 \]

1. \( S \) too big to compute this sum

2. Don’t know \( Q^* \)
Deep Q-learning

Algorithm 4: Online learning (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
    $$\Theta^{(t+1)} \leftarrow \Theta^{(t)} - \alpha \nabla_{\Theta^{(t+1)}} \ell(\Theta^{(t)}, \Theta^{(t+1)})$$
  where
    $$\nabla_{\Theta^{(t+1)}} \ell(\Theta^{(t)}, \Theta^{(t+1)})$$
    $$= 2 \left( \gamma - Q(s, a; \Theta^{(t+1)}) \right) \nabla_{\Theta^{(t+1)}} Q(s, a; \Theta^{(t+1)})$$
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” $\mathcal{D} = \{e_1, e_2, \ldots, e_N\}$ 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)
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