Reinforcement Learning: Q-Learning
Reminders

• Homework 7: HMMs
  – Out: Fri, Mar 29
  – Due: Mon, Apr 15 at 11:59pm

• Homework 8: Reinforcement Learning
  – Out: Wed, Apr 10
  – Due: Wed, Apr 24 at 11:59pm

• Today’s In-Class Poll
VALUE ITERATION
Definitions for Value Iteration

Whiteboard

– State trajectory
– Value function
– Bellman equations
– Optimal policy
– Optimal value function
– Computing the optimal policy
– Ex: Path Planning
**RL Terminology**

**Question:** Match each term (on the left) to the corresponding statement or definition (on the right)

<table>
<thead>
<tr>
<th>Terms:</th>
<th>Statements:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. a reward function</td>
<td>1. gives the expected future discounted reward of a state</td>
</tr>
<tr>
<td>B. a transition probability</td>
<td>2. maps from states to actions</td>
</tr>
<tr>
<td>C. a policy</td>
<td>3. quantifies immediate success of agent</td>
</tr>
<tr>
<td>D. state/action/reward triples</td>
<td>4. is a deterministic map from state/action pairs to states</td>
</tr>
<tr>
<td>E. a value function</td>
<td>5. quantifies the likelihood of landing a new state, given a state/action pair</td>
</tr>
<tr>
<td>F. transition function</td>
<td>6. is the desired output of an RL algorithm</td>
</tr>
<tr>
<td>G. an optimal policy</td>
<td>7. can be influenced by trading off between exploitation/exploration</td>
</tr>
<tr>
<td>H. Matt’s favorite statement</td>
<td></td>
</tr>
</tbody>
</table>
Example: Path Planning
Example: Robot Localization

$r(s, a)$ (immediate reward) values

One optimal policy

$V^*(s)$ values

Figure from Tom Mitchell
Value Iteration

Whiteboard

– Value Iteration Algorithm
– Synchronous vs. Asynchronous Updates
Value Iteration

Algorithm 1 Value Iteration

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

Variant 1: with $Q(s,a)$ table
Algorithm 1 Value Iteration

1: procedure VALUEITERATION($R(s, a)$ 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:         $V(s) = \max_a R(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a)V(s')$
6:     Let $\pi(s) = \arg\max_a R(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a)V(s'), \forall s$
7: return $\pi$

Variant 2: without Q(s,a) table
Synchronous vs. Asynchronous Value Iteration

**Algorithm 1** Asynchronous Value Iteration

1. `procedure ASYNCHRONOUS_VALUE_ITERATION(R(s, a), p(·|s, a))`
2. Initialize value function $V(s)^{(0)} = 0$ or randomly
3. $t = 0$
4. `while` not converged `do`
5. `for` $s \in S$ `do`
6. $V(s)^{(t+1)} = \max_a R(s, a) + \gamma \sum_{s' \in S} p(s'|s, a)V(s')^{(t)}$
7. $t = t + 1$
8. Let $\pi(s) = \arg\max_a R(s, a) + \gamma \sum_{s' \in S} p(s'|s, a)V(s')$, $\forall s$
9. `return` $\pi$

**Algorithm 1** Synchronous Value Iteration

1. `procedure SYNCHRONOUS_VALUE_ITERATION(R(s, a), p(·|s, a))`
2. Initialize value function $V(s)^{(0)} = 0$ or randomly
3. $t = 0$
4. `while` not converged `do`
5. `for` $s \in S$ `do`
6. $V(s)^{(t+1)} = \max_a R(s, a) + \gamma \sum_{s' \in S} p(s'|s, a)V(s')^{(t)}$
7. $t = t + 1$
8. Let $\pi(s) = \arg\max_a R(s, a) + \gamma \sum_{s' \in S} p(s'|s, a)V(s')$, $\forall s$
9. `return` $\pi$

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

**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
Value Iteration Convergence

Theorem 1 (Bertsekas (1989))
$V$ converges to $V^*$, if each state is visited infinitely often

Theorem 2 (Williams & Baird (1993))
if $\max_s |V^{t+1}(s) - V^t(s)| < \epsilon$
then $\max_s |V^{t+1}(s) - V^*(s)| < \frac{2\epsilon \gamma}{1 - \gamma}$, $\forall s$

Theorem 3 (Bertsekas (1987))
greedy policy will be optimal in a finite number of steps (even if not converged to optimal value function!)
Value Iteration Variants

Question:
True or False: The value iteration algorithm shown below is an example of **synchronous** updates

Algorithm 1 Value Iteration

1: **procedure** VALUEITERATION($R(s, a)$ reward function, $p(\cdot|s, a)$ transition probabilities)
2: Initialize value function $V(s) = 0$ or randomly
3: **while** not converged **do**
4:  **for** $s \in S$ **do**
5:   **for** $a \in A$ **do**
6:     $Q(s, a) = R(s, a) + \gamma \sum_{s' \in S} p(s'|s, a)V(s')$
7:     $V(s) = \max_a Q(s, a)$
8:  **Let** $\pi(s) = \arg\max_a Q(s, a), \ \forall s$
9: **return** $\pi$
POLICY ITERATION
Algorithm 1 Policy Iteration

1: procedure POLICYITERATION($R(s,a)$ reward function, $p(\cdot|s,a)$ transition probabilities)
2: Initialize policy $\pi$ randomly
3: while not converged do
4:    Solve Bellman equations for fixed policy $\pi$
\[
V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} p(s'|s, \pi(s))V^\pi(s'), \forall s
\]
5:    Improve policy $\pi$ using new value function
\[
\pi(s) = \arg\max_a R(s, a) + \gamma \sum_{s' \in S} p(s'|s, a)V^\pi(s')
\]
6: return $\pi$
### Policy Iteration

**Algorithm 1** Policy Iteration

1: **procedure** POLICYITERATION($R(s, a)$, transition probabilities)

2: Initialize policy $\pi$ randomly

3: **while** not converged **do**

4: Solve Bellman equations for fixed policy $\pi$

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

5: Improve policy $\pi$ using new value function

$$\pi(s) = \arg\max_a R(s, a) + \gamma \sum_{s' \in S} p(s'|s, a)V^\pi(s')$$

6: **return** $\pi$

- **Compute value function for fixed policy is easy**
- **System of $|S|$ equations and $|S|$ variables**
- Greedy policy w.r.t. current value function
- Greedy policy might remain the same for a particular state if there is no better action
Policy Iteration Convergence

**In-Class Exercise:**
How many policies are there for a finite sized state and action space?

**In-Class Exercise:**
Suppose policy iteration is shown to improve the policy at every iteration. Can you bound the number of iterations it will take to converge?
Value Iteration vs. Policy Iteration

• Value iteration requires $O(|A| |S|^2)$ computation per iteration

• Policy iteration requires $O(|A| |S|^2 + |S|^3)$ computation per iteration

• In practice, policy iteration converges in fewer iterations

---

**Algorithm 1: Value Iteration**

1: `procedure VALUEITERATION(R(s, a) reward function, p(·|s, a) transition probabilities)`
2:   Initialize value function $V(s) = 0$ or randomly
3:   `while` not converged `do`
4:     `for` $s \in S$ `do`
5:       `for` $a \in A$ `do`
6:         $Q(s, a) = R(s, a) + \gamma \sum_{s' \in S} p(s'|s, a)V(s')$
7:       $V(s) = \max_{a} Q(s, a)$
8:   `end for`
9: `end for`
10: `return` $\pi$

**Algorithm 1: Policy Iteration**

1: `procedure POLICYITERATION(R(s, a) reward function, p(·|s, a) transition probabilities)`
2:   Initialize policy $\pi$ randomly
3:   `while` not converged `do`
4:     Solve Bellman equations for fixed policy $\pi$
5:     $V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} p(s'|s, \pi(s))V^\pi(s'), \forall s$
6:     Improve policy $\pi$ using new value function
7:     $\pi(s) = \arg\max_{a} R(s, a) + \gamma \sum_{s' \in S} p(s'|s, a)V^\pi(s')$
8: `end while`
9: `return` $\pi$
Learning Objectives

Reinforcement Learning: Value and Policy Iteration

You should be able to...

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

Whiteboard

– Motivation: What if we have zero knowledge of the environment?
– Q-Function: Expected Discounted Reward
Example: Robot Localization

Immediate rewards $r(s,a)$
State values $V^*(s)$

Consider first the case where $P(s'|s,a)$ is deterministic
Q-Learning

Whiteboard

– Q-Learning Algorithm
  • Case 1: Deterministic Environment
  • Case 2: Nondeterministic Environment
– Convergence Properties
– Exploration Insensitivity
– Ex: Re-ordering Experiences
– $\epsilon$-greedy Strategy
DEEP RL EXAMPLES
**TD Gammon ➔ Alpha Go**

**Learning to beat the masters at board games**

<table>
<thead>
<tr>
<th>THEN</th>
<th>NOW</th>
</tr>
</thead>
</table>

“…the world’s top computer program for backgammon, TD-GAMMON (Tesauro, 1992, 1995), learned its strategy by playing over one million practice games against itself…”

(Mitchell, 1997)
Playing Atari with Deep RL

- Setup: RL system observes the pixels on the screen.
- It receives rewards as the game score.
- Actions decide how to move the joystick / buttons.

Figures from David Silver (Intro RL lecture)
Playing Atari with Deep RL

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

Videos:

- Atari Breakout:  
  [https://www.youtube.com/watch?v=V1eYniJoRnk](https://www.youtube.com/watch?v=V1eYniJoRnk)

- Space Invaders:  
  [https://www.youtube.com/watch?v=ePvoFs9cGgU](https://www.youtube.com/watch?v=ePvoFs9cGgU)

Figures from Mnih et al. (2013)
Playing Atari with Deep RL

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

<table>
<thead>
<tr>
<th></th>
<th>B. Rider</th>
<th>Breakout</th>
<th>Enduro</th>
<th>Pong</th>
<th>Q*bert</th>
<th>Seaquest</th>
<th>S. Invaders</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>354</td>
<td>1.2</td>
<td>0</td>
<td>−20.4</td>
<td>157</td>
<td>110</td>
<td>179</td>
</tr>
<tr>
<td>Contingency [4]</td>
<td>1743</td>
<td>6</td>
<td>159</td>
<td>−17</td>
<td>960</td>
<td>723</td>
<td>268</td>
</tr>
<tr>
<td>DQN</td>
<td>4092</td>
<td>168</td>
<td>470</td>
<td>20</td>
<td>1952</td>
<td>1705</td>
<td>581</td>
</tr>
<tr>
<td>Human</td>
<td>7456</td>
<td>31</td>
<td>368</td>
<td>−3</td>
<td>18900</td>
<td>28010</td>
<td>3690</td>
</tr>
<tr>
<td>HNeat Best [8]</td>
<td>3616</td>
<td>52</td>
<td>106</td>
<td>19</td>
<td>1800</td>
<td>920</td>
<td>1720</td>
</tr>
<tr>
<td>HNeat Pixel [8]</td>
<td>1332</td>
<td>4</td>
<td>91</td>
<td>−16</td>
<td>1325</td>
<td>800</td>
<td>1145</td>
</tr>
<tr>
<td>DQN Best</td>
<td>5184</td>
<td>225</td>
<td>661</td>
<td>21</td>
<td>4500</td>
<td>1740</td>
<td>1075</td>
</tr>
</tbody>
</table>

Table 1: The upper table compares average total reward for various learning methods by running an $\epsilon$-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 $\epsilon$-greedy policy with $\epsilon = 0.05$. 

Figures from Mnih et al. (2013)
Deep Q-Learning

**Question:** What if our state space $S$ is too large to represent with a table?

**Examples:**
- $s_t =$ pixels of a video game
- $s_t =$ continuous values of a sensors in a manufacturing robot
- $s_t =$ sensor output from a self-driving car

**Answer:** Use a parametric function to approximate the table entries
Deep Q-Learning

*Whiteboard*

- Approximating the Q function with a neural network
- Deep Q-Learning
- Experience Replay
- function approximators
  \(<\text{state}, \text{action}_i> \rightarrow \text{q-value}\)
  \text{vs.}
  \state \rightarrow \text{all action q-values}
Experience Replay

• **Problems** with online updates for Deep Q-learning:
  – not i.i.d. as SGD would assume
  – quickly forget rare experiences that might later be useful to learn from

• **Uniform Experience Replay** (Lin, 1992):
  – Keep a *replay memory* $D = \{e_1, e_2, \ldots, e_N\}$ of $N$ most recent experiences $e_t = <s_t, a_t, r_t, s_{t+1}>$
  – Alternate two steps:
    1. Repeat $T$ times: randomly sample $e_i$ from $D$ and apply a Q-Learning update to $e_i$
    2. Agent selects an action using epsilon greedy policy to receive new experience that is added to $D$

• **Prioritized Experience Replay** (Schaul et al, 2016)
  – similar to Uniform ER, but sample so as to prioritize experiences with high error
Alpha Go

Game of Go (圍棋)

- **19x19 board**
- Players alternately play black/white stones
- **Goal** is to fully encircle the largest region on the board
- **Simple** rules, but extremely complex game play

Game 1
Fan Hui (Black), AlphaGo (White)
AlphaGo wins by 2.5 points
Alpha Go

- State space is too large to represent explicitly since 
  # of sequences of moves is $O(b^d)$
  - Go: $b=250$ and $d=150$
  - Chess: $b=35$ and $d=80$
- Key idea:
  - Define a neural network to approximate the value function
  - Train by policy gradient

Figure 2a
Policy network

\[ \pi \]
\[ p_{\pi} \]
\[ \rho \]
\[ p_{\rho} \]
\[ v_\theta \]
\[ p_{\sigma} \]
\[ \sigma \]
\[ v_\sigma \]

Rollout policy
SL policy network
RL policy network
Value network
Policy network
Value network

Policy network

Neural network

Data

Human expert positions
Self-play positions

© 2016 Macmillan Publishers Limited. All rights reserved
• Results of a tournament
• From Silver et al. (2016): “a 230 point gap corresponds to a 79% probability of winning”
Learning Objectives

Reinforcement Learning: Q-Learning

You should be able to...

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