Reinforcement Learning: Q-Learning
Reminders

• Homework 8: Reinforcement Learning
  – Out: Fri, Apr 10
  – Due: Wed, Apr 22 at 11:59pm

• Today’s In-Class Poll
  – http://poll.mlcourse.org
VALUE ITERATION
Value Iteration

Algorithm 1 Value Iteration

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

Variant 2: without $Q(s,a)$ table
Value Iteration

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:     $Q(s,a) = 0$
5:     **for** $s \in S$ do
6:         **for** $a \in A$ do
7:             $Q(s,a) = R(s,a) + \gamma \sum_{s' \in S} p(s'|s, a)V(s')$
8:         $V(s) = \max_a Q(s, a)$
9:     **let** $\pi(s) = \arg\max_a Q(s,a), \ \forall s$
10: **return** $\pi$

**Variant 1:** with $Q(s,a)$ table
Synchronous vs. Asynchronous Value Iteration

Algorithm 1 Asynchronous Value Iteration

1: procedure ASYNCHRONOUSVALUEITERATION($R(s, a), p(·|s, a)$)
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: end for
9: end while
10: Let $\pi(s) = \arg\max_a R(s, a) + \gamma \sum_{s' \in S} p(s'|s, a) V(s')$, \(\forall s\)
11: return $\pi$

Algorithm 1 Synchronous Value Iteration

1: procedure SYNCHRONOUSVALUEITERATION($R(s, a), p(·|s, a)$)
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: end for
9: end while
10: Let $\pi(s) = \arg\max_a R(s, a) + \gamma \sum_{s' \in S} p(s'|s, a) V(s')$, \(\forall s\)
11: 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** VALUE ITERATION($R(s, a)$ 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


**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 $\pi$

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

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

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

6:  **return** $\pi$
Policy Iteration

**Algorithm 1** Policy Iteration

1: **procedure** POLICYITERATION($R(s, a)$, $p(s'|s, a)$, $p(\cdot|s, a)$, 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
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
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? If yes, what is the bound? If no, why not?

\[ \text{\# iterations} \leq |A| \]
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** VALUE ITERATION($R(s, a)$ 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 $a \in \mathcal{A}$ 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 $\pi(s) = \text{ argmax}_a Q(s, a)$, $\forall s$
9. **return** $\pi$

### Algorithm 1 Policy Iteration

1. **procedure** POLICY ITERATION($R(s, a)$ 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.     $\pi(s) = \max_a R(s, a) + \gamma \sum_{s' \in S} p(s'|s, a)V^\pi(s')$
8. **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
Today’s lecture is brought you by the letter....
Q-LEARNING
Q-Learning

Whiteboard

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

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

$Q^*(s,a)$ values

$V^*(s)$ values

One optimal policy

Figure from Tom Mitchell
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
Reordering Experiences

Ex: Easiest Maze Ever!

\[ y = 0.9 \]
\[ S = \{A, B, C, D\} \]
\[ A = \{E, W\} \]
\[ Q(s,a) = 0 \text{ at start} \]

1) Suppose we visit:

<table>
<thead>
<tr>
<th>S</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>E</td>
<td>0</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>E</td>
<td>0</td>
<td>C</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>E</td>
<td>100</td>
<td>D</td>
</tr>
</tbody>
</table>

\[ Q(A,E) = 0 \]
\[ Q(B,E) = 0 \]
\[ Q(C,E) = 100 \]

2) Suppose we visit in reverse order:

<table>
<thead>
<tr>
<th>S</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>C</td>
<td>E</td>
<td>100</td>
<td>D</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>E</td>
<td>0</td>
<td>C</td>
</tr>
<tr>
<td>3</td>
<td>A</td>
<td>E</td>
<td>0</td>
<td>B</td>
</tr>
</tbody>
</table>

\[ Q(C,E) = 100 \]
\[ Q(B,E) = 90 \]
\[ Q(A,E) = 81 \]

\[ \text{diff updates} \]
**Designing State Spaces**

**Q:** Do we have to retrain our RL agent every time we change our state space?

**A:** Yes. But whether your state space changes from one setting to another is determined by your design of the state representation.

Two examples:

- State Space A: \(<x, y>\) position on map
e.g. \(s_t = <74, 152>\)
- State Space B: window of pixel colors centered at current Pac Man location
e.g. \(s_t = \begin{array}{ccc} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{array} \)
DEEP RL EXAMPLES
TD Gammon → Alpha Go

Learning to beat the masters at board games

| THEN | NOW |
| "...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=ePv0Fs9cGgU](https://www.youtube.com/watch?v=ePv0Fs9cGgU)

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>HNet 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>HNet 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 HNet and DQN. HNet 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

**Key Idea:**
1. Use a neural network $Q(s,a; \theta)$ to approximate $Q^*(s,a)$
2. Learn the parameters $\theta$ via SGD with training examples $< s_t, a_t, r_t, s_{t+1} >$
Deep Q-Learning

**Whiteboard**

- Strawman loss function (i.e. what we cannot compute)
- Approximating the Q function with a neural network
- Approximating the Q function with a linear model
- Deep Q-Learning
- Function approximators
  $\langle \text{state, action}_i \rangle \rightarrow q$-value
  **vs.**
  state $\rightarrow$ all action q-values
not-so-Deep Q-Learning

Approx w/NN:
\[ Q(s, a; \theta) \]

Approx w/Linear Regression:
Represent state as a feature vector:
\[ \tilde{s} \in \{0, 1, 0, 1, \ldots, 1\}^T \]

\[ |\tilde{x}| \in \mathbb{R}^M \]
\[ M = \text{# features} \]

\[ Q(\tilde{s}, a; \theta) = \tilde{s}^T \Theta_a \]
\[ \Theta = \left[ \begin{array}{c} \tilde{\theta}_1^T \\ \tilde{\theta}_2^T \\ \vdots \\ \tilde{\theta}_K^T \end{array} \right] \quad \Theta \in \mathbb{R}^{K \times M} \]

\[ \nabla_{\theta} Q(\tilde{s}, a; \theta) = ? \]
\[ = \tilde{s} \text{ if } a = b \]
\[ = 0 \text{ if } a \neq b \]
\[ = \frac{1}{2} \text{ if } a = b \]

\[ \nabla Q(s, a; \theta) = \begin{bmatrix} 0 \cdots 0 \\ \vdots \\ \tilde{s}^T \\ \vdots \\ 0 \cdots 0 \end{bmatrix} \]
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

Figure from Silver et al. (2016)
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 from Silver et al. (2016)
• Results of a tournament
• From Silver et al. (2016): “a 230 point gap corresponds to a 79% probability of winning”

Figure from Silver et al. (2016)
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
Q-Learning

Question:
For the $R(s,a)$ values shown on the arrows below, which are the corresponding $Q^*(s,a)$ values? Assume discount factor = 0.5.

Answer:
ML Big Picture

Learning Paradigms:
What data is available and when? What form of prediction?
- supervised learning
- unsupervised learning
- semi-supervised learning
- reinforcement learning
- active learning
- imitation learning
- domain adaptation
- online learning
- density estimation
- recommender systems
- feature learning
- manifold learning
- dimensionality reduction
- ensemble learning
- distant supervision
- hyperparameter optimization

Problem Formulation:
What is the structure of our output prediction?
- boolean Binary Classification
- categorical Multiclass Classification
- ordinal Ordinal Classification
- real Regression
- ordering Ranking
- multiple discrete Structured Prediction
- multiple continuous (e.g. dynamical systems)
- both discrete & (e.g. mixed graphical models) cont.

Facets of Building ML Systems:
How to build systems that are robust, efficient, adaptive, effective?
1. Data prep
2. Model selection
3. Training (optimization / search)
4. Hyperparameter tuning on validation data
5. (Blind) Assessment on test data

Theoretical Foundations:
What principles guide learning?
- probabilistic
- information theoretic
- evolutionary search
- ML as optimization

Big Ideas in ML:
Which are the ideas driving development of the field?
- inductive bias
- generalization / overfitting
- bias-variance decomposition
- generative vs. discriminative
- deep nets, graphical models
- PAC learning
- distant rewards

Application Areas
- Key challenges?
- NLP, Speech, Computer Vision, Robotics, Medicine, Search

- Big Picture
- Problem Formulation
- Learning Paradigms
- Theoretical Foundations
- Facets of Building ML Systems
- Big Ideas in ML
- Application Areas
# Learning Paradigms

<table>
<thead>
<tr>
<th>Paradigm</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supervised</td>
<td>$\mathcal{D} = {\mathbf{x}^{(i)}, y^{(i)}}_{i=1}^N$  $\mathbf{x} \sim p^<em>(\cdot)$ and $y = c^</em>(\cdot)$</td>
</tr>
<tr>
<td>← Regression</td>
<td>$y^{(i)} \in \mathbb{R}$</td>
</tr>
<tr>
<td>← Classification</td>
<td>$y^{(i)} \in {1, \ldots, K}$</td>
</tr>
<tr>
<td>← Binary classification</td>
<td>$y^{(i)} \in {+1, -1}$</td>
</tr>
<tr>
<td>← Structured Prediction</td>
<td>$\mathbf{y}^{(i)}$ is a vector</td>
</tr>
<tr>
<td>Unsupervised</td>
<td>$\mathcal{D} = {\mathbf{x}^{(i)}}_{i=1}^N$  $\mathbf{x} \sim p^*(\cdot)$</td>
</tr>
<tr>
<td>Semi-supervised</td>
<td>$\mathcal{D} = {\mathbf{x}^{(i)}, y^{(i)}}<em>{i=1}^{N_1} \cup {\mathbf{x}^{(j)}}</em>{j=1}^{N_2}$</td>
</tr>
<tr>
<td>Online</td>
<td>$\mathcal{D} = {(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), (\mathbf{x}^{(3)}, y^{(3)}), \ldots}$</td>
</tr>
<tr>
<td>Active Learning</td>
<td>$\mathcal{D} = {\mathbf{x}^{(i)}}_{i=1}^N$ and can query $y^{(i)} = c^*(\cdot)$ at a cost</td>
</tr>
<tr>
<td>Imitation Learning</td>
<td>$\mathcal{D} = {(s^{(1)}, a^{(1)}), (s^{(2)}, a^{(2)}), \ldots}$</td>
</tr>
<tr>
<td>Reinforcement Learning</td>
<td>$\mathcal{D} = {(s^{(1)}, a^{(1)}, r^{(1)}), (s^{(2)}, a^{(2)}, r^{(2)}), \ldots}$</td>
</tr>
</tbody>
</table>