Deep RL
+
K-Means
Reminders

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

• Homework 9: Learning Paradigms
  – Out: Wed, Apr. 22
  – Due: Wed, Apr. 29 at 11:59pm
  – Can only be submitted up to 3 days late, so we can return grades before final exam

• Today’s In-Class Poll
  – http://poll.mlcourse.org
DEEP RL EXAMPLES
TD Gammon → Alpha Go

Learning to beat the masters at board games

THEN

“...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...”

NOW

(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

Videos:

- Atari Breakout: https://www.youtube.com/watch?v=V1eYniJoRnk
- Space Invaders: https://www.youtube.com/watch?v=ePvoFs9cGgU

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

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
  \(<\text{state, action}_i> \rightarrow \text{q-value}\)
  \textbf{vs.}
  \text{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 \( e_t = \langle s_t, a_t, r_t, s_{t+1} \rangle \)
  – 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

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)
Alpha Go

- 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
Q1

Q-Learning

\[ Q(s,a) \leftarrow r + (0.5) \max_{a'} Q(s',a') \]

**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:**

\[ F = \text{None of the above} \]
**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

**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

---

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

---

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

---

**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
# Learning Paradigms

<table>
<thead>
<tr>
<th>Paradigm</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supervised</td>
<td>$\mathcal{D} = {x^{(i)}, y^{(i)}}_{i=1}^{N}$ $x \sim p^<em>(\cdot)$ 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>$y^{(i)}$ is a vector</td>
</tr>
<tr>
<td>Unsupervised</td>
<td>$\mathcal{D} = {x^{(i)}}_{i=1}^{N}$ $x \sim p^*(\cdot)$</td>
</tr>
<tr>
<td>Semi-supervised</td>
<td>$\mathcal{D} = {x^{(i)}, y^{(i)}}<em>{i=1}^{N_1} \cup {x^{(j)}}</em>{j=1}^{N_2}$</td>
</tr>
<tr>
<td>Online</td>
<td>$\mathcal{D} = {(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), (x^{(3)}, y^{(3)}), \ldots}$</td>
</tr>
<tr>
<td>Active Learning</td>
<td>$\mathcal{D} = {x^{(i)}}_{i=1}^{N}$ 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>
K-Means Outline

- **Clustering: Motivation / Applications**
- **Optimization Background**
  - Coordinate Descent
  - Block Coordinate Descent
- **Clustering**
  - Inputs and Outputs
  - Objective-based Clustering
- **K-Means**
  - K-Means Objective
  - Computational Complexity
  - K-Means Algorithm / Lloyd’s Method
- **K-Means Initialization**
  - Random
  - Farthest Point
  - K-Means++
CLUSTERING
Clustering, Informal Goals

**Goal:** Automatically partition *unlabeled* data into groups of similar datapoints.

**Question:** When and why would we want to do this?

**Useful for:**

- Automatically organizing data.
- Understanding hidden structure in data.
- Preprocessing for further analysis.
  - Representing high-dimensional data in a low-dimensional space (e.g., for visualization purposes).

Slide courtesy of Nina Balcan
Applications (Clustering comes up everywhere...) 

- Cluster news articles or web pages or search results by topic.

- Cluster protein sequences by function or genes according to expression profile.

- Cluster users of social networks by interest (community detection).

Slide courtesy of Nina Balcan
Applications (Clustering comes up everywhere...)

- Cluster customers according to purchase history.

- Cluster galaxies or nearby stars (e.g. Sloan Digital Sky Survey)

- And many many more applications....

Slide courtesy of Nina Balcan
Optimization Background

Whiteboard:

– Coordinate Descent
– Block Coordinate Descent
Clustering

Question: Which of these partitions is “better”?
K-MEANS
K-Means Algorithm

• **Given** unlabeled feature vectors 
\[ D = \{x^{(1)}, x^{(2)}, \ldots, x^{(N)}\} \]

• **Initialize** cluster centers 
\[ c = \{c^{(1)}, \ldots, c^{(K)}\} \]
and cluster assignments 
\[ z = \{z^{(1)}, z^{(2)}, \ldots, z^{(N)}\} \]

• **Repeat** until convergence:

\[ \begin{align*}
& \text{for } j \text{ in } \{1, \ldots, K\} \\
& \quad c^{(j)} = \text{mean of all points assigned to cluster } j \\
& \text{for } i \text{ in } \{1, \ldots, N\} \\
& \quad z^{(i)} = \text{index } j \text{ of cluster center nearest to } x^{(i)}
\end{align*} \]
Example: Real-World Dataset
K-Means

Whiteboard:

– Clustering: Inputs and Outputs
– Objective-based Clustering
– K-Means Objective
– Computational Complexity
– K-Means Algorithm / Lloyd’s Method
K-Means Algorithm

• **Given** unlabeled feature vectors
  \[ D = \{ \mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(N)} \} \]

• **Initialize** cluster centers \[ \mathbf{c} = \{ \mathbf{c}^{(1)}, \ldots, \mathbf{c}^{(K)} \} \]
  and cluster assignments \[ \mathbf{z} = \{ \mathbf{z}^{(1)}, \mathbf{z}^{(2)}, \ldots, \mathbf{z}^{(N)} \} \]

• **Repeat** until convergence:
  – for \( j \) in \{1,...,K\}
    \[ \mathbf{c}^{(j)} = \text{mean of all} \text{ points assigned to cluster } j \]
  – for \( i \) in \{1,...,N\}
    \[ \mathbf{z}^{(i)} = \text{index } j \text{ of cluster center nearest to } \mathbf{x}^{(i)} \]
K-Means Initialization

Whiteboard:
  – Random
  – Furthest Traversal
  – K-Means++
K=3 cluster centers

K-MEANS EXAMPLE
Example: K-Means
Example: K-Means
Example: K-Means
Example: K-Means
Example: K-Means
Example: K-Means
Example: K-Means
Example: K-Means
K=2 cluster centers

K-MEANS EXAMPLE
Example: K-Means
Example: K-Means
Example: K-Means
Example: K-Means
Example: K-Means
Example: K-Means
Example: K-Means
Example: K-Means
Example: K-Means
Example: K-Means
Example: K-Means
K-MEANS PERFORMANCE
Example: Given a set of datapoints
Lloyd’s method: Random Initialization

Select initial centers at random
Lloyd’s method: Random Initialization

Assign each point to its nearest center
Lloyd’s method: Random Initialization

Recompute optimal centers given a fixed clustering
Lloyd’s method: Random Initialization

Assign each point to its nearest center
Lloyd’s method: Random Initialization

Recompute optimal centers given a fixed clustering

Slide courtesy of Nina Balcan
Lloyd’s method: Random Initialization

Assign each point to its nearest center
Lloyd’s method: Random Initialization

Recompute optimal centers given a fixed clustering

Get a good quality solution in this example.

Slide courtesy of Nina Balcan
Lloyd’s method: Performance

It always converges, but it may converge at a local optimum that is different from the global optimum, and in fact could be arbitrarily worse in terms of its score.
Lloyd’s method: Performance

Local optimum: every point is assigned to its nearest center and every center is the mean value of its points.

Slide courtesy of Nina Balcan
Lloyd’s method: Performance

It is arbitrarily worse than optimum solution....
Lloyd’s method: Performance

This bad performance, can happen even with well separated Gaussian clusters.

Slide courtesy of Nina Balcan
Lloyd’s method: Performance

This bad performance, can happen even with well separated Gaussian clusters.

Some Gaussian are combined.....

Slide courtesy of Nina Balcan
Learning Objectives

K-Means

You should be able to...

1. Distinguish between coordinate descent and block coordinate descent
2. Define an objective function that gives rise to a "good" clustering
3. Apply block coordinate descent to an objective function preferring each point to be close to its nearest objective function to obtain the K-Means algorithm
4. Implement the K-Means algorithm
5. Connect the nonconvexity of the K-Means objective function with the (possibly) poor performance of random initialization