CS 343: Artificial Intelligence

Reinforcement Learning

Prof. Yuke Zhu

The University of Texas at Austin

[These slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at http://ai.berkeley.edu.]
Announcements

Homework 3: Games
Due Monday, 2/28 at 11:59 pm

Project 2: Multi-Agent Pacman
Due Wednesday, 3/2 at 11:59 pm

Homework 4: MDPs
Due Monday, 3/7 at 11:59 pm
Reinforcement Learning
Reinforcement Learning

- Still assume a Markov decision process (MDP):
  - A set of states \( s \in S \)
  - A set of actions (per state) \( A \)
  - A model \( T(s,a,s') \)
  - A reward function \( R(s,a,s') \)
- Still looking for a policy \( \pi(s) \)
- New twist: don’t know \( T \) or \( R \)
  - I.e. we don’t know which states are good or what the actions do
  - Must actually try actions and states out to learn
Value Iteration

- Bellman equations characterize the optimal values:

\[ V^*(s) = \max_a \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V^*(s') \right] \]

- Value iteration computes them:

\[ V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V_k(s') \right] \]

- Value iteration is just a fixed point solution method
  - ... though the \( V_k \) vectors are also interpretable as time-limited values
Reinforcement Learning

Basic idea:
- Receive feedback in the form of rewards
- Agent’s utility is defined by the reward function
- Must (learn to) act so as to maximize expected rewards
- All learning is based on observed samples of outcomes!
Example: Learning to Walk

Initial Hand Coded Walk

Initial (19.5 cm/s)

Learned Walk

After learning (28 cm/s)
Example: Atari from raw pixels
Example: Robot manipulation
Offline (MDPs) vs. Online (RL)

Offline Solution

Online Learning
Model-Based Learning

- **Model-Based Idea:**
  - Learn an approximate model based on experiences
  - Solve for values as if the learned model were correct

- **Step 1: Learn empirical MDP model**
  - Count outcomes $s'$ for each $s, a$
  - Normalize to give an estimate of $\hat{P}(s, a, s')$
  - Discover each $\hat{R}(s, a, s')$ when we experience $(s, a, s')$

- **Step 2: Solve the learned MDP**
  - For example, use value iteration, as before
Example: Model-Based Learning

Input Policy $\pi$

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>

Assume: $\gamma = 1$

Observed Episodes (Training)

**Episode 1**
- B, east, C, -1
- C, east, D, -1
- D, exit, x, +10

**Episode 2**
- B, east, C, -1
- C, east, D, -1
- D, exit, x, +10

**Episode 3**
- E, north, C, -1
- C, east, D, -1
- D, exit, x, +10

**Episode 4**
- E, north, C, -1
- C, east, A, -1
- A, exit, x, -10

Learned Model

$\hat{T}(s, a, s')$

- $T(B, \text{east}, C) = \ldots$
- $T(C, \text{east}, D) = \ldots$
- $T(C, \text{east}, A) = \ldots$

$\hat{R}(s, a, s')$

- $R(B, \text{east}, C) = \ldots$
- $R(C, \text{east}, D) = \ldots$
- $R(D, \text{exit}, x) = \ldots$
Example: Model-Based Learning

**Input Policy** $\pi$

**Observed Episodes (Training)**

**Episode 1**
- B, east, C, -1
- C, east, D, -1
- D, exit, x, +10

**Episode 2**
- B, east, C, -1
- C, east, D, -1
- D, exit, x, +10

**Episode 3**
- E, north, C, -1
- C, east, D, -1
- D, exit, x, +10

**Episode 4**
- E, north, C, -1
- C, east, A, -1
- A, exit, x, -10

**Learned Model**

$\hat{T}(s, a, s')$

- $T(B, \text{east}, C) = 1.00$
- $T(C, \text{east}, D) = 0.75$
- $T(C, \text{east}, A) = 0.25$

$\hat{R}(s, a, s')$

- $R(B, \text{east}, C) = -1$
- $R(C, \text{east}, D) = -1$
- $R(D, \text{exit}, x) = +10$

Assume: $\gamma = 1$
### Example: Expected Age

**Goal:** Compute expected age of CS 343 students

<table>
<thead>
<tr>
<th>Known P(A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ E[A] = \sum_{a} P(a) \cdot a = 0.35 \times 20 + \ldots ]</td>
</tr>
</tbody>
</table>

Without P(A), instead collect samples \([a_1, a_2, \ldots, a_N]\)

#### Unknown P(A): “Model Based”

\[
\hat{P}(a) = \frac{\text{num}(a)}{N} \\
E[A] \approx \sum_{a} \hat{P}(a) \cdot a
\]

**Why does this work?** Because eventually you learn the right model.

#### Unknown P(A): “Model Free”

\[
E[A] \approx \frac{1}{N} \sum_{i} a_i
\]

**Why does this work?** Because samples appear with the right frequencies.
Model-Free Learning
Passive Reinforcement Learning
Passive Reinforcement Learning

- **Simplified task: policy evaluation**
  - Input: a fixed policy $\pi(s)$
  - You don’t know the transitions $T(s,a,s')$
  - You don’t know the rewards $R(s,a,s')$
  - Goal: learn the state values

- **In this case:**
  - Learner is “along for the ride”
  - No choice about what actions to take
  - Just execute the policy and learn from experience
  - This is NOT offline planning! You actually take actions in the world.
Direct Evaluation

- **Goal:** Compute values for each state under $\pi$

- **Idea:** Average together observed sample values
  - Act according to $\pi$
  - Every time you visit a state, write down what the sum of discounted rewards turned out to be
  - Average those samples

- This is called direct evaluation
Example: Direct Evaluation

Assume: $\gamma = 1$

Input Policy $\pi$

Observed Episodes (Training)

Episode 1
- B, east, C, -1
- C, east, D, -1
- D, exit, x, +10

Episode 2
- B, east, C, -1
- C, east, D, -1
- D, exit, x, +10

Episode 3
- E, north, C, -1
- C, east, D, -1
- D, exit, x, +10

Episode 4
- E, north, C, -1
- C, east, A, -1
- A, exit, x, -10

Output Values
Problems with Direct Evaluation

- What’s good about direct evaluation?
  - It’s easy to understand
  - It doesn’t require any knowledge of $T, R$
  - It eventually computes the correct average values, using just sample transitions

- What bad about it?
  - It wastes information about state connections
  - Each state must be learned separately
  - So, it takes a long time to learn

Output Values

If $B$ and $E$ both go to $C$ under this policy, how can their values be different?
Why Not Use Policy Evaluation?

- Simplified Bellman updates calculate $V$ for a fixed policy:
  - Each round, replace $V$ with a one-step-look-ahead layer over $V$

  $$V_0^\pi(s) = 0$$

  $$V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V_k^\pi(s')]$$

  - This approach fully exploited the connections between the states
  - Unfortunately, we need $T$ and $R$ to do it!

- Key question: how can we do this update to $V$ without knowing $T$ and $R$?
  - In other words, how to we take a weighted average without knowing the weights?
We want to improve our estimate of $V$ by computing these averages:

$$V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s, \pi(s), s') \left[ R(s, \pi(s), s') + \gamma V_k^\pi(s') \right]$$

**Idea:** Take samples of outcomes $s'$ (by doing the action!) and average

$$sample_1 = R(s, \pi(s), s'_1) + \gamma V_k^\pi(s'_1)$$

$$sample_2 = R(s, \pi(s), s'_2) + \gamma V_k^\pi(s'_2)$$

$$\ldots$$

$$sample_n = R(s, \pi(s), s'_n) + \gamma V_k^\pi(s'_n)$$

$$V_{k+1}^\pi(s) \leftarrow \frac{1}{n} \sum_i sample_i$$

Almost! But we can’t rewind time to get sample after sample from state $s$. 
Temporal Difference Learning
Temporal Difference Learning

- Big idea: learn from every experience!
  - Update $V(s)$ each time we experience a transition $(s, a, s', r)$
  - Likely outcomes $s'$ will contribute updates more often

- Temporal difference learning of values
  - Policy still fixed, still doing evaluation!
  - Move values toward value of whatever successor occurs: running average

Sample of $V(s)$: \[ \text{sample} = R(s, \pi(s), s') + \gamma V^\pi(s') \]

Update to $V(s)$: \[ V^\pi(s) \leftarrow (1 - \alpha)V^\pi(s) + (\alpha)\text{sample} \]

Same update: \[ V^\pi(s) \leftarrow V^\pi(s) + \alpha(\text{sample} - V^\pi(s)) \]
Exponential Moving Average

- Exponential moving average
  - The running interpolation update: \( \bar{x}_n = (1 - \alpha) \cdot \bar{x}_{n-1} + \alpha \cdot x_n \)
  - Makes recent samples more important:
    \[
    \bar{x}_n = \frac{x_n + (1 - \alpha) \cdot x_{n-1} + (1 - \alpha)^2 \cdot x_{n-2} + \ldots}{1 + (1 - \alpha) + (1 - \alpha)^2 + \ldots}
    \]
  - Forgets about the past (distant past values were wrong anyway)
- Decreasing learning rate (alpha) can give converging averages
Example: Temporal Difference Learning

Assume: $\gamma = 1$, $\alpha = 1/2$

$$V^\pi(s) \leftarrow (1 - \alpha)V^\pi(s) + \alpha \left[ R(s, \pi(s), s') + \gamma V^\pi(s') \right]$$
Problems with TD Value Learning

- TD value leaning is a model-free way to do policy evaluation, mimicking Bellman updates with running sample averages.

- However, if we want to turn values into a (new) policy, we’re sunk:

\[ \pi(s) = \arg \max_a Q(s, a) \]

\[ Q(s, a) = \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V(s') \right] \]

- Idea: learn Q-values, not values.

- Makes action selection model-free too!
Active Reinforcement Learning
Active Reinforcement Learning

- Full reinforcement learning: optimal policies (like value iteration)
  - You don’t know the transitions $T(s,a,s')$
  - You don’t know the rewards $R(s,a,s')$
  - You choose the actions now
  - Goal: learn the optimal policy / values

- In this case:
  - Learner makes choices!
  - Fundamental tradeoff: exploration vs. exploitation
  - This is NOT offline planning! You actually take actions in the world and find out what happens...
Detour: Q-Value Iteration

- **Value iteration**: find successive (depth-limited) values
  - Start with $V_0(s) = 0$, which we know is right
  - Given $V_k$, calculate the depth $k+1$ values for all states:
    \[
    V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V_k(s') \right]
    \]

- **But Q-values are more useful**, so compute them instead
  - Start with $Q_0(s,a) = 0$, which we know is right
  - Given $Q_k$, calculate the depth $k+1$ q-values for all q-states:
    \[
    Q_{k+1}(s, a) \leftarrow \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma \max_{a'} Q_k(s', a') \right]
    \]
Q-Learning

- **Q-Learning**: sample-based Q-value iteration
  
  \[ Q_{k+1}(s, a) \leftarrow \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma \max_{a'} Q_k(s', a') \right] \]

- **Learn Q(s,a) values as you go**
  - Receive a sample \((s,a,s',r)\)
  - Consider your old estimate: \(Q(s, a)\)
  - Consider your new sample estimate:
    
    \[ \text{sample} = R(s, a, s') + \gamma \max_{a'} Q(s', a') \]
  - Incorporate the new estimate into a running average:
    
    \[ Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + (\alpha) \text{[sample]} \]
Demo of Q-Learning -- Gridworld
Demo of Q-Learning -- Crawler
Q-Learning Properties

- Amazing result: Q-learning converges to optimal policy -- even if you’re acting suboptimally!

- This is called off-policy learning

- Caveats:
  - You have to explore enough
  - You have to eventually make the learning rate small enough
  - ... but not decrease it too quickly
  - Basically, in the limit, it doesn’t matter how you select actions (!)