CS 188: Artificial Intelligence

Reinforcement Learning I

Instructors: Stuart Russell and Patrick Virtue

University of California, Berkeley
What if the MDP is initially unknown? Lots of things change!

- **Exploration**: you have to *try unknown actions* to get information
- **Exploitation**: eventually, you have to use what you know
- **Sampling**: you may need to repeat many times to get good estimates
- **Generalization**: what you learn in one state may apply to others too
Basic idea:

- Learn how to maximize expected rewards based on observed samples of transitions.
Example: Learning to Walk

[Video: AIBO WALK – initial]

[Kohl and Stone, ICRA 2004]
Example: Learning to Walk

[Kohl and Stone, ICRA 2004]

[Video: AIBO WALK – finished]
Example: Sidewinding

[Video: SNAKE – climbStep+sidewinding]
Example: Toddler Robot

[Video: TODDLER – 40s]

[Tedrake, Zhang and Seung, 2005]
The Crawler!
Still assume a Markov decision process (MDP):

- A set of states $s \in S$
- A set of actions (per state) $A$
- A model $P(s'|a,s)$
- A reward function $R(s,a,s')$

Still looking for a policy $\pi(s)$

New twist: don’t know $P$ or $R$

- I.e. we don’t know which states are good or what the actions do
- Must actually try actions and explore new states -- to boldly go where no Pacman agent has been before
Offline (MDPs) vs. Online (RL)

Offline Solution

Online Learning
Approaches to reinforcement learning

1. Learn the model, solve it, execute the solution
2. Learn values from experiences, use to make decisions
   a. Direct utility estimation
   b. Temporal difference learning
   c. Q-learning
3. Learn policies directly
Model-Based 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 $P(s' | s, a)$
  - Discover each $R(s,a,s')$ when we experience the transition

- **Step 2: Solve the learned MDP**
  - Use, e.g., value or policy iteration, as before
Example: Model-Based Learning

Input Policy $\pi$

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

$P(s'|s,a)$
- $P(C|B, \text{east}) = 1.00$
- $P(D|C, \text{east}) = 0.75$
- $P(A|C, \text{east}) = 0.25$
- ...$

$R(s,a,s')$
- $R(B, \text{east, C}) = -1$
- $R(C, \text{east, D}) = -1$
- $R(D, \text{exit, x}) = +10$
- ...
Pros and cons

Pro:
- Makes efficient use of experiences

Con:
- May not scale to large state spaces
  - Learns model one state-action pair at a time (but this is fixable)
  - Cannot solve MDP for very large $|S|$
Model-Free Learning
Basic idea of model-free methods

- To approximate expectations with respect to a distribution, you can either
  - Estimate the distribution from samples, compute an expectation
  - Or, estimate the expectation from samples directly
Example: Expected Age

Goal: Compute expected age of cs188 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 $[a_1, a_2, \ldots, a_N]$

<table>
<thead>
<tr>
<th>Unknown P(A): “Model Based”</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\hat{P}(A) = N_a / N$</td>
</tr>
<tr>
<td>$E[A] \approx \sum_a \hat{P}(a) \cdot a$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Unknown P(A): “Model Free”</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E[A] \approx 1 / N \sum_i a_i$</td>
</tr>
</tbody>
</table>

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

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

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

- **In this case:**
  - Learner is “along for the ride”
  - No choice about what actions to take: just do it
  - This is NOT offline planning! Agent takes actions in the world.
Direct utility estimation

- **Goal:** Estimate $V^\pi(s)$, i.e., expected total discounted reward from $s$ onwards

- **Idea:**
  - Use the *actual* sum of discounted rewards from $s$
  - Average over multiple trials and visits to $s$

- This is called **direct utility estimation**
Assume: $\gamma = 1$

### Input Policy $\pi$

```
B   C   D
A
E
```

### Observed Episodes (Training)

<table>
<thead>
<tr>
<th>Episode</th>
<th>Start</th>
<th>Move</th>
<th>End</th>
<th>Reward</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>B</td>
<td>east</td>
<td>C</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>east</td>
<td>D</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>D</td>
<td>exit</td>
<td>x</td>
<td>+10</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>east</td>
<td>C</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>east</td>
<td>D</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>D</td>
<td>exit</td>
<td>x</td>
<td>+10</td>
</tr>
<tr>
<td>3</td>
<td>E</td>
<td>north</td>
<td>C</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>east</td>
<td>D</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>D</td>
<td>exit</td>
<td>x</td>
<td>+10</td>
</tr>
<tr>
<td>4</td>
<td>E</td>
<td>north</td>
<td>C</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>east</td>
<td>A</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>exit</td>
<td>x</td>
<td>-10</td>
</tr>
</tbody>
</table>

### Output Values

```
+8 +4 +10
-10
+10
-2
```
Problems with Direct Estimation

What’s good about direct estimation?
- It’s easy to understand
- It doesn’t require any knowledge of $P(s' | s, a)$ or $R(s, a, s')$
- It converges to the right answer in the limit

What’s bad about it?
- Each state must be learned separately (fixable)
- It ignores information about state connections
- 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?
Temporal difference (TD) learning
TD as approximate Bellman update

- Policy evaluation (version 1) improves the estimate of $V^\pi$ by computing Bellman update, i.e., expectation over next-state values:
  - $V^\pi_{k+1}(s) \leftarrow \sum_{s'} P(s' | \pi(s), s) [R(s, \pi(s), s') + \gamma V^\pi_k(s')]$

- Idea 1: Use actual samples to estimate the expectation:
  - $\text{sample}_1 = R(s, \pi(s), s_1') + \gamma V^\pi_k(s_1')$
  - $\text{sample}_2 = R(s, \pi(s), s_2') + \gamma V^\pi_k(s_2')$
  - ...
  - $\text{sample}_N = R(s, \pi(s), s_N') + \gamma V^\pi_k(s_N')$
  - $V^\pi_{k+1}(s) \leftarrow 1/N \sum_i \text{sample}_i$
TD as approximate Bellman update

- **Idea 2:** Update value of $s$ after each transition $s,a,s',r$:
  - Update $V^\pi([3,1])$ based on $R([3,1],up,[3,2])$ and $\gamma V^\pi([3,2])$
  - Update $V^\pi([3,2])$ based on $R([3,2],up,[3,3])$ and $\gamma V^\pi([3,3])$
  - Update $V^\pi([3,3])$ based on $R([3,3],right,[4,3])$ and $\gamma V^\pi([4,3])$
TD as approximate Bellman update

- Idea 3: Update values by maintaining a *running average*
Running averages

- How do you compute the average of 1, 4, 7?
- Method 1: add them up and divide by N
  - \(1 + 4 + 7 = 12\)
  - average = \(\frac{12}{N} = \frac{12}{3} = 4\)
- Method 2: keep a running average \(\mu_n\) and a running count \(n\)
  - \(n=0\) \(\mu_0 = 0\)
  - \(n=1\) \(\mu_1 = (0 \cdot \mu_0 + x_1)/1 = (0 \cdot 0 + 1)/1 = 1\)
  - \(n=2\) \(\mu_2 = (1 \cdot \mu_1 + x_2)/2 = (1 \cdot 1 + 4)/2 = 2.5\)
  - \(n=3\) \(\mu_3 = (2 \cdot \mu_2 + x_3)/3 = (2 \cdot 2.5 + 7)/3 = 4\)
  - General formula: \(\mu_n = ((n-1) \cdot \mu_{n-1} + x_n)/n\)
  - \(= \left[\frac{(n-1)}{n}\right] \mu_{n-1} + \left[\frac{1}{n}\right] x_n\) (weighted average of old mean, new sample)
What if we use a weighted average with a fixed weight?

- \( \mu_n = (1-\alpha) \mu_{n-1} + \alpha x_n \)
- \( n=1 \quad \mu_1 = x_1 \)
- \( n=2 \quad \mu_2 = (1-\alpha) \cdot \mu_1 + \alpha x_2 = (1-\alpha) \cdot x_1 + \alpha x_2 \)
- \( n=3 \quad \mu_3 = (1-\alpha) \cdot \mu_2 + \alpha x_3 = (1-\alpha)^2 \cdot x_1 + \alpha(1-\alpha)x_2 + \alpha x_3 \)
- \( n=4 \quad \mu_4 = (1-\alpha) \cdot \mu_3 + \alpha x_4 = (1-\alpha)^3 \cdot x_1 + \alpha(1-\alpha)^2x_2 + \alpha(1-\alpha)x_3 + \alpha x_4 \)

- I.e., *exponential forgetting* of old values

- \( \mu_n \) is a convex combination of sample values (weights sum to 1)

- \( E[\mu_n] \) is a convex combination of \( E[X_i] \) values, hence unbiased
TD as approximate Bellman update

- Idea 3: Update values by maintaining a *running average*
- $\text{sample} = R(s, \pi(s), s') + \gamma V^\pi(s')$
- $V^\pi(s) \leftarrow (1-\alpha) \cdot V^\pi(s) + \alpha \cdot \text{sample}$
- $V^\pi(s) \leftarrow V^\pi(s) + \alpha \cdot [\text{sample} - V^\pi(s)]$
- This is the temporal difference learning rule
- $[\text{sample} - V^\pi(s)]$ is the “TD error”
- I.e., observe a sample, move $V^\pi(s)$ a little bit to make it more consistent with its neighbor $V^\pi(s')$
Example: Temporal Difference Learning

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

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

- TD value learning is a model-free way to do policy evaluation, mimicking Bellman updates with running sample averages
- But we can’t use the value function, or improve the policy, without a transition model to do one-step greedy expectimax!
Active Reinforcement Learning
Active Reinforcement Learning

- Full reinforcement learning:
  - You don’t know the transition model
  - You don’t know the reward function
  - 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...
Value iteration and Q-iteration

- Approximate version of value iteration update gives TD learning:
  - $V_{\pi}^{k+1}(s) \leftarrow \sum_{s'} P(s' | \pi(s), s) [R(s, \pi(s), s') + \gamma V_{\pi}^k(s') ]$
  - $V^\pi(s) \leftarrow (1-\alpha) \cdot V^\pi(s) + \alpha \cdot [R(s, \pi(s), s') + \gamma V^\pi(s') ]$

- Approximate version of Q iteration update gives Q learning:
  - $Q_{k+1}(s,a) \leftarrow \sum_{s'} P(s' | a,s) [R(s,a,s') + \gamma \max_{a'} Q_k(s',a') ]$
  - $Q(s,a) \leftarrow (1-\alpha) \cdot Q(s,a) + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q(s',a') ]$

- We obtain a policy from learned Q, with no model!
Q-Learning

- 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:
    $$sample = R(s,a,s') + \gamma \max_{a'} Q_k(s',a')$$
  - Incorporate the new estimate into a running average:
    $$Q(s,a) \leftarrow (1-\alpha) Q(s,a) + \alpha \cdot [sample]$$

[Demo: Q-learning – gridworld (L10D2)]
[Demo: Q-learning – crawler (L10D3)]
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
Summary

- RL solves MDPs via direct experience of transitions and rewards
- There are several schemes:
  - Learn the MDP model and solve it
  - Learn $V$ directly from sums of rewards, or by TD local adjustments
    - Still need a model to make decisions by lookahead
  - Learn $Q$ by local Q-learning adjustments, use it directly to pick actions
    - (and about 100 other variations)
- Big missing piece: how can we scale this up to large state spaces like Tetris ($2^{200}$)?