10-701
Machine Learning

Reinforcement learning (RL)
From MDPs to RL

- We still use the same Markov model with rewards and actions
- But there are a few differences:
  1. We do not assume we know the Markov model
  2. We adapt to new observations (online vs. offline)
- Examples:
  - Game playing
  - Robot interacting with environment
  - Agents
RL

- No actions
- With actions
Scenario

• You wonder the world
• At each time point you see a state and a reward
• Your goal is to compute the sum of discounted rewards for each state
Scenario

- You wonder the world
- At each time point you see a state and a reward
- Your goal is to compute the sum of discounted rewards for each state
- We will denote these by $J^\text{est}(S_i)$

\[
S_1, 4 \rightarrow S_2, 0 \rightarrow S_3, 2 \rightarrow S_2, 2 \rightarrow S_4, 0
\]
Discounted rewards: $\gamma = 0.9$

- Lets compute the discounted rewards for each time point:
  - $t1$: $4 + 0.9\times0 + 0.9^2\times2 + 0.9^3\times2 = 7.1$
  - $t2$: $0 + 0.9\times2 + 0.9^2\times2 = 3.4$
  - $t3$: $2 + 0.9\times2 = 3.8$
  - $t4$: $2 + 0 = 2$
  - $t5$: $0 = 0$

<table>
<thead>
<tr>
<th>State</th>
<th>Observations</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$</td>
<td>7.1</td>
<td>7.1</td>
</tr>
<tr>
<td>$S_2$</td>
<td>3.4, 2</td>
<td>2.7</td>
</tr>
<tr>
<td>$S_3$</td>
<td>3.8</td>
<td>3.8</td>
</tr>
<tr>
<td>$S_4$</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Supervised learning for RL

- Observe set of states and rewards: \((s(0),r(0)) \ldots (s(T),r(T))\)
- For \(t=0 \ldots T\) compute discounted sum:

\[
J[t] = \sum_{i=t}^{T} \gamma^{i-t} r_i
\]

- Compute \(J^{est}(s_i) = \text{(mean of } J(t) \text{ for } t \text{ such that } s(t) = s_i)\)

\[
J^{est}[s_i] = \frac{\sum J[t]}{\# s[t] = s_i}
\]

We assume that we observe each state frequently enough and that we have many observations so that the final observations do not have a big impact on our prediction
Algorithm for supervised learning

1. Initialize \( \text{Counts}(s_i) = J(s_i) = \text{Disc}(s_i) = 0 \)
2. Observe a state \( s_i \) and a reward \( r \)
3. \( \text{Counts}(s_i) = \text{Counts}(s_i) + 1 \)
4. \( \text{Disc}(s_i) = \text{Disc}(s_i) + 1 \)
5. For all states \( j \)
   \[ J(s_j) = J(s_j) + r \cdot \text{Disc}(s_j) \]
   \[ \text{Disc}(s_j) = \gamma \cdot \text{Disc}(s_j) \]
6. Go to 2

At any time we can estimate \( J^* \) by setting:
\[ J_{\text{est}}(s_i) = J(s_i) / \text{Counts}(s_i) \]
Running time and space

- Each update takes $O(n)$ where $n$ is the number of states, since we are updating vectors containing entries for all states.
- Space is also $O(n)$

1. Convergences to true $J^*$ can be proven
2. Can be more efficient by ignoring states for which Disc() is very low already.
Problems with supervised learning

- Takes a long time to converge
- Does not use all available data
  - We can learn transition probabilities as well!
Certainty-Equivalent (CE) Learning

- Lets try to learn the underlying Markov system’s parameters

\[ S_1, 4 \rightarrow S_2, 0 \rightarrow S_3, 2 \rightarrow S_2, 2 \rightarrow S_4, 0 \]
CE learning

- We keep track of three vectors:
  \( \text{Counts}(s) \): number of times we visited state \( s \)
  \( J(s) \): sum of rewards from state \( s \)
  \( \text{Trans}(i,j) \): number of times we transitioned from state \( s_i \) to state \( s_j \)

- When we visit state \( s_i \), receive reward \( r \) and move to state \( s_j \) we do the following:

  \[
  \begin{align*}
  \text{Counts}(s_i) &= \text{Counts}(s_i) + 1 \\
  J(s_i) &= J(s_i) + r \\
  \text{Trans}(i,j) &= \text{Trans}(i,j) + 1
  \end{align*}
  \]
CE learning

- When we visit state $s_i$, receive reward $r$ and move to state $s_j$ we do the following:
  
  $$\text{Counts}(s_i) = \text{Counts}(s_i) + 1$$
  
  $$J(s_i) = J(s_i) + r$$
  
  $$\text{Trans}(i,j) = \text{Trans}(i,j) + 1$$

  Using this we can estimate at any time the following parameters:

  $$R^{est}(s_i) = J(s_i) / \text{Counts}(s_i)$$
  
  $$P^{est}(j|i) = \text{Trans}(i,j) / \text{Counts}(s_i)$$
Example: CE learning

\[ R_{est}(s_i) \]

<table>
<thead>
<tr>
<th>State</th>
<th>Mean reward</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S_1 )</td>
<td>4</td>
</tr>
<tr>
<td>( S_2 )</td>
<td>1</td>
</tr>
<tr>
<td>( S_3 )</td>
<td>2</td>
</tr>
<tr>
<td>( S_4 )</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ P_{est}(j|i) \]

<table>
<thead>
<tr>
<th></th>
<th>s1</th>
<th>s2</th>
<th>s3</th>
<th>s4</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>s2</td>
<td>0</td>
<td>0</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>s3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>s4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
S_1, 4 \rightarrow S_2, 0 \rightarrow S_3, 2 \rightarrow S_2, 2 \rightarrow S_4, 0
\]
Certainty Equivalent learning

We can estimate at any time the following parameters:

\[ R^{est}(s_i) = \frac{J(s_i)}{\text{Counts}(s_i)} \]

\[ P^{est}(j|i) = \frac{\text{Trans}(i,j)}{\text{Counts}(s_i)} \]

We now can solve the MDP by setting, for all states \( s_k \):

\[ J^{est}(s_k) = r^{est}(s_k) + \gamma \sum_j p^{est}(s_j | s_k) J^{est}(s_j) \]
CE: Running time and space

Running time

• Updates: $O(1)$
• Solving MDP:
  - $O(n^3)$ using matrix inversion
  - $O(n^2 \times \#it)$ when using value iteration

Space

• $O(n^2)$ for transition probabilities
Improving CE: One backup

- We do the same updates and estimates as the original CE:
  \[
  \begin{align*}
  \text{Counts}(s_i) &= \text{Counts}(s_i) + 1 \\
  J(s_i) &= J(s_i) + r \\
  \text{Trans}(i,j) &= \text{Trans}(i,j) + 1
  \end{align*}
  \]
  \[
  R_{est}(s_i) = \frac{J(s_i)}{\text{Counts}(s_i)} \\
  P_{est}(j|i) = \frac{\text{Trans}(i,j)}{\text{Counts}(s_i)}
  \]

- But we do not carry out the full value iteration
- Instead, we \textbf{only} update $J_{est}(s_i)$ for the current state:

\[
J_{est}(s_i) = r_{est}(s_i) + \gamma \sum_j p_{est}(s_j | s_i) J_{est}(s_j)
\]
CE one backup: Running time and space

Running time
• Updates: $O(1)$
• Solving MDP:
  - $O(1)$ just update current state

Space
• $O(n^2)$ for transition probabilities
  - Still a lot of memory, but much more efficient
  - Can prove convergence to optimal solution (but slower than CE)
Summary so far

- Three methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supervised learning</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>CE learning</td>
<td>$O(n^2 \times \text{#it})$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>One backup CE</td>
<td>$O(1)$</td>
<td>$O(n^2)$</td>
</tr>
</tbody>
</table>
Temporal difference (TD) learning

• Goal: Same efficiency as one backup CE while much less space
• We only maintain the $J_{est}$ array.
• Assume we have $J_{est}(s_1) \ldots J_{est}(s_n)$. If we observe a transition from state $s_i$ to state $s_j$ and a reward $r$, we update using the following rule:

$$J_{est}^i(s_i) = (1 - \alpha)J_{est}^i(s_i) + \alpha(r + \gamma J_{est}^j(s_j))$$
Temporal difference (TD) learning

• Assume we have $J^{est}(s_1) \ldots J^{est}(s_n)$. If we observe a transition from state $s_i$ to state $s_j$ and a reward $r$, we update using the following rule:

$$J^{est}(s_i) = (1 - \alpha)J^{est}(s_i) + \alpha(r + \gamma J^{est}(s_j))$$

parameter to determine how much weight we place on current observation

We have seen similar update rule before, as always, choosing $\alpha$ is an issue
Convergence

• TD learning is guaranteed to converge if:
  • All states are visited often
  • And: \[ \sum_t \alpha_t = \infty \]
    \[ \sum_t \alpha_t^2 < \infty \]

For example, \( \alpha_t = C/t \) for some constant C would satisfy both requirements
### TD: Complexity and space

- Time to update: $O(1)$
- Space: $O(n)$

<table>
<thead>
<tr>
<th>Method</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supervised learning</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>CE learning</td>
<td>$O(n^2 \times \text{#it})$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>One backup CE</td>
<td>$O(1)$</td>
<td>$O(n^2)$</td>
</tr>
</tbody>
</table>
• No actions  ✓
• With actions
So far we assumed that we cannot impact the transition after visiting a state.

In real world situations we often have a choice of actions we take (as we discussed for MDPs).

How can we learn the best policy for such cases?
Policy learning using CE: Example

**Rest(s_i)**

<table>
<thead>
<tr>
<th>State</th>
<th>Mean reward</th>
</tr>
</thead>
<tbody>
<tr>
<td>S₁</td>
<td>4</td>
</tr>
<tr>
<td>S₂</td>
<td>4/3</td>
</tr>
<tr>
<td>S₃</td>
<td>2.5</td>
</tr>
<tr>
<td>S₄</td>
<td>0</td>
</tr>
</tbody>
</table>

**Rest(s_j|i,a)**

<table>
<thead>
<tr>
<th>s₁,A</th>
<th>s₁,B</th>
<th>s₂</th>
<th>s₃</th>
<th>s₄</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>2/3</td>
<td>1/3</td>
<td></td>
</tr>
<tr>
<td>1/3</td>
<td>1/3</td>
<td>0</td>
<td>1/3</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Diagram**

- **Action A**
  - S₁, 4 → S₃, 3 → S₄, 0 → S₂, 2 → S₃, 2
- **Action B**
  - S₁, 4 → S₂, 0 → S₃, 2 → S₂, 2 → S₄, 0
Policy learning using CE

We can easily update CE by setting:

\[ J^{est}(s_k) = r^{est}(s_k) + \max_a \left( \gamma \sum_j p^{est}(s_j | s_k, a) J^{est}(s_j) \right) \]

Should we always choose the best action as our next step?

We revise our transition model to include actions.
Policy learning for TD

• TD is model free
• We can adjust TD to learn policies by defining the Q function:
  \[ Q^*(s_i, a) = \text{expected sum of future (discounted) rewards if we start at state } s_i \text{ and take action } a \]
• When we take a specific action \( a \) in state \( s_i \) and then transition to state \( s_j \) we can update the Q function directly by setting:

\[
Q^{est}(S_i, a) = (1 - \alpha)Q^{est}(S_i, a) + \alpha (r_i + \gamma \max_a Q^{est}(S_j, a'))
\]

Instead of the \( J^{est} \) vector we maintain the \( Q^{est} \) matrix, which is a rather sparse \( n \) by \( m \) matrix (\( n \) states and \( m \) actions)
Choosing the next action

- We can select the action that results in the highest expected sum of future rewards.
- But that may not be the best action. Remember, we are only sampling from the distribution of possible outcomes. We do not want to avoid potentially beneficial actions.
- Instead, we can take a more probabilistic approach:

\[
p(a) = \frac{1}{Z} \exp\left( - \frac{Q^{est}(s_i, a)}{f(t)} \right)
\]

The probability we will use action \( a \) decreases as time goes by and we are more confident in the model we learned.

Normalizing constant
Choosing the next action

- Instead, we can take a more probabilistic approach:

\[ p(a) \propto \exp\left( -\frac{Q_{est}(s_i, a)}{f(t)} \right) \]

- We can initialize Q values to be high to increase the likelihood that we will explore more options.
- It can be shown that Q learning converges to optimal policy.
Demo
What you should know

• Differences between MDP and RL
• Strategies for computing with expected rewards
• Strategies for computing rewards and actions
• Q learning