Reinforcement Learning

- We still assume an 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 $R$, so must try out actions
- Big idea: Compute all averages over $T$ using sample outcomes
### The Story So Far: MDPs and RL

#### Known MDP: Offline Solution

<table>
<thead>
<tr>
<th>Goal</th>
<th>Technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compute $V^<em>$, $Q^</em>$, $\pi^*$</td>
<td>Value / policy iteration</td>
</tr>
<tr>
<td>Evaluate a fixed policy $\pi$</td>
<td>Policy evaluation</td>
</tr>
</tbody>
</table>

#### Unknown MDP: Model-Based

<table>
<thead>
<tr>
<th>Goal</th>
<th>Technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compute $V^<em>$, $Q^</em>$, $\pi^*$</td>
<td>VI/PI on approx. MDP</td>
</tr>
<tr>
<td>Evaluate a fixed policy $\pi$</td>
<td>PE on approx. MDP</td>
</tr>
</tbody>
</table>

#### Unknown MDP: Model-Free

<table>
<thead>
<tr>
<th>Goal</th>
<th>Technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compute $V^<em>$, $Q^</em>$, $\pi^*$</td>
<td>Q-learning</td>
</tr>
<tr>
<td>Evaluate a fixed policy $\pi$</td>
<td>Value Learning</td>
</tr>
</tbody>
</table>
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: 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
  - Basically, in the limit, it doesn’t matter how you select actions (!)
Video of Demo Q-Learning -- Gridworld
Video of Demo Q-Learning -- Crawler
Exploration vs. Exploitation
Video of Demo Q-learning – Manual Exploration – Bridge Grid
Video of Demo Q-learning – Epsilon-Greedy – Crawler
Video of Demo Q-learning – Exploration Function – Crawler
Regret

Even if you learn the optimal policy, you still make mistakes along the way!

Regret is a measure of your total mistake cost: the difference between your (expected) rewards, including youthful suboptimality, and optimal (expected) rewards.

Minimizing regret goes beyond learning to be optimal – it requires optimally learning to be optimal.

Example: random exploration and exploration functions both end up optimal, but random exploration has higher regret.
Approximate Q-Learning
Generalizing Across States

- Basic Q-Learning keeps a table of all q-values.

- In realistic situations, we cannot possibly learn about every single state!
  - Too many states to visit them all in training
  - Too many states to hold the q-tables in memory

- Instead, we want to generalize:
  - Learn about some small number of training states from experience
  - Generalize that experience to new, similar situations
  - This is a fundamental idea in machine learning, and we’ll see it over and over again.

[demo - RL pacman]
Example: Pacman

Let’s say we discover through experience that this state is bad:

In naïve q-learning, we know nothing about this state:

Or even this one!

[Demo: Q-learning – pacman – tiny – watch all (L11D5)]
[Demo: Q-learning – pacman – tiny – silent train (L11D6)]
[Demo: Q-learning – pacman – tricky – watch all (L11D7)]
Video of Demo Q-Learning Pacman – Tiny – Watch All
Video of Demo Q-Learning Pacman – Tiny – Silent Train
Video of Demo Q-Learning Pacman – Tricky – Watch All
Feature-Based Representations

- Solution: describe a state using a vector of features (properties)
  - Features are functions from states to real numbers (often 0/1) that capture important properties of the state
  - Example features:
    - Distance to closest ghost
    - Distance to closest dot
    - Number of ghosts
    - $1 / (\text{dist to dot})^2$
    - Is Pacman in a tunnel? (0/1)
    - ...... etc.
    - Is it the exact state on this slide?
  - Can also describe a q-state $(s, a)$ with features (e.g. action moves closer to food)
Linear Value Functions

- Using a feature representation, we can write a q function (or value function) for any state using a few weights:

\[ V(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s) \]

\[ Q(s, a) = w_1 f_1(s, a) + w_2 f_2(s, a) + \ldots + w_n f_n(s, a) \]

- Advantage: our experience is summed up in a few powerful numbers

- Disadvantage: states may share features but actually be very different in value!
Approximate Q-Learning

\[ Q(s, a) = w_1 f_1(s, a) + w_2 f_2(s, a) + \ldots + w_n f_n(s, a) \]

- Q-learning with linear Q-functions:
  - Transition: \( (s, a, r, s') \)
  - Difference: \( r + \gamma \max_{a'} Q(s', a') - Q(s, a) \)
  - \( Q(s, a) \leftarrow Q(s, a) + \alpha \text{[difference]} \)
  - \( w_i \leftarrow w_i + \alpha \text{[difference]} f_i(s, a) \)

- Intuitive interpretation:
  - Adjust weights of active features
  - E.g., if something unexpectedly bad happens, blame the features that were on: disprefer all states with that state’s features

- Formal justification: online least squares
Example: Q-Pacman

\[ Q(s, a) = 4.0 f_{DOT}(s, a) - 1.0 f_{GST}(s, a) \]

\[ f_{DOT}(s, \text{NORTH}) = 0.5 \]
\[ f_{GST}(s, \text{NORTH}) = 1.0 \]

\[ Q(s, \text{NORTH}) = +1 \]
\[ r + \gamma \max_{a'} Q(s', a') = -500 + 0 \]

\[ \text{difference} = -501 \]
\[ w_{DOT} \leftarrow 4.0 + \alpha [-501] 0.5 \]
\[ w_{GST} \leftarrow -1.0 + \alpha [-501] 1.0 \]

\[ Q(s, a) = 3.0 f_{DOT}(s, a) - 3.0 f_{GST}(s, a) \]
Video of Demo Approximate Q-Learning -- Pacman
DeepMind Atari (©Two Minute Lectures) approximate Q-learning with neural nets

Google DeepMind's Deep Q-learning

The algorithm will play Atari breakout.
Q-Learning and Least Squares
Linear Approximation: Regression

Prediction:
\[ \hat{y} = w_0 + w_1 f_1(x) \]

Prediction:
\[ \hat{y}_i = w_0 + w_1 f_1(x) + w_2 f_2(x) \]
Optimization: Least Squares

\[
\text{total error} = \sum_i (y_i - \hat{y}_i)^2 = \sum_i \left( y_i - \sum_k w_k f_k(x_i) \right)^2
\]
Minimizing Error

Imagine we had only one point \( x \), with features \( f(x) \), target value \( y \), and weights \( w \):

\[
\text{error}(w) = \frac{1}{2} \left( y - \sum_k w_k f_k(x) \right)^2 \\
\frac{\partial \text{error}(w)}{\partial w_m} = -\left( y - \sum_k w_k f_k(x) \right) f_m(x) \\
w_m \leftarrow w_m + \alpha \left( y - \sum_k w_k f_k(x) \right) f_m(x)
\]

Approximate q update explained:

\[
w_m \leftarrow w_m + \alpha \left[ r + \gamma \max_{\alpha'} Q(s', \alpha') - Q(s, \alpha) \right] f_m(s, \alpha)
\]

“target” “prediction”
Overfitting: Why Limiting Capacity Can Help
New in Model-Free RL

Google DeepMind's
Deep Q-learning

The algorithm will play Atari breakout.
Policy Search
Problem: often the feature-based policies that work well (win games, maximize utilities) aren’t the ones that approximate $V / Q$ best
- E.g. your value functions from project 2 were probably horrible estimates of future rewards, but they still produced good decisions
- Q-learning’s priority: get Q-values close (modeling)
- Action selection priority: get ordering of Q-values right (prediction)
- We’ll see this distinction between modeling and prediction again later in the course

Solution: learn policies that maximize rewards, not the values that predict them

Policy search: start with an ok solution (e.g. Q-learning) then fine-tune by hill climbing on feature weights
Policy Search

0 Simplest policy search:
   0 Start with an initial linear value function or Q-function
   0 Nudge each feature weight up and down and see if your policy is better than before

0 Problems:
   0 How do we tell the policy got better?
   0 Need to run many sample episodes!
   0 If there are a lot of features, this can be impractical

0 Better methods exploit lookahead structure, sample wisely, change multiple parameters…
Policy Search
# The Story So Far: MDPs and RL

## Known MDP: Offline Solution

<table>
<thead>
<tr>
<th>Goal</th>
<th>Technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compute $V^<em>$, $Q^</em>$, $\pi^*$</td>
<td>Value / policy iteration</td>
</tr>
<tr>
<td>Evaluate a fixed policy $\pi$</td>
<td>Policy evaluation</td>
</tr>
</tbody>
</table>

## Unknown MDP: Model-Based

- **Goal**: Compute $V^*$, $Q^*$, $\pi^*$
- **Technique**: VI/PI on approx. MDP
- **Use features to generalize**

- **Goal**: Evaluate a fixed policy $\pi$
- **Technique**: PE on approx. MDP

## Unknown MDP: Model-Free

- **Goal**: Compute $V^*$, $Q^*$, $\pi^*$
- **Technique**: Q-learning
- **Use features to generalize**

- **Goal**: Evaluate a fixed policy $\pi$
- **Technique**: Value Learning
Discussion: Model-Based vs Model-Free RL
New in Model-Based RL

- [http://deepmpc.cs.cornell.edu/](http://deepmpc.cs.cornell.edu/)
  - Learn a model with a deep neural network and use it for MPC

- [https://sites.google.com/site/visuomotorpolicy/](https://sites.google.com/site/visuomotorpolicy/)
  - G-policy search (GPS) - trains local models around trajectories, planning with local models, then train a policy based on the local plans
Conclusion

- We’re done with Part I: Search and Planning!

- We’ve seen how AI methods can solve problems in:
  - Search
  - Constraint Satisfaction Problems
  - Games
  - Markov Decision Problems
  - Reinforcement Learning

- Next up: Part II: Uncertainty and Learning!