AI: Representation and Problem Solving

Reinforcement Learning II

Instructors: Tuomas Sandholm and Nihar Shah

Slide credits: CMU AI and http://ai.berkeley.edu
Logistics

• Midterm 2 on Wednesday
  • Covers from propositional logic up to (and including) basic Q learning
  • Doesn’t include the rest today’s lecture (after basic Q learning), which will be covered on the Final Exam

• Extra-cool, optional:
  Prof. Bart Selman will give the Inaugural Hans Berliner Lecture on “Mathematical and Scientific Discovery: A New Frontier for AI”
  • On 4/4/2024 at 4 PM in Rashid Auditorium in GHC
Reinforcement Learning (RL) Review So Far

- 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 $\pi(s)$
- The twist: don’t know $T$ or $R$, so must try out actions
- Big idea: Compute all averages over transition probabilities using sample outcomes
Summary so far

- **Passive RL:** agent has to learn from experience

- **Model-based:** Estimate the transition and rewards; run value iteration or policy iteration

- **Model-free:**
  - Direct policy evaluation – empirical average utility
  - Temporal difference learning – sample based policy iteration update via running averages
Temporal Difference learning

- **Main idea:** learn from each experience visiting state \( s \), doing \( \pi(s) \)
- Update \( V(s) \) each time we experience a transition \( (s, a, s', R) \)
  - Not waiting for the whole episode to get utility
- Likely outcomes \( s' \) will contribute updates more often

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

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

- Decreasing learning rate \( (\alpha) \) towards zero leads to convergence
Example: Model-Based Learning

<table>
<thead>
<tr>
<th>Input Policy $\pi$</th>
<th>Observed Episodes (Training)</th>
<th>Learned Model</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$\hat{T}(s, a, s')$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$T(B, east, C) =$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$T(C, east, D) =$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$T(C, east, A) =$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\ldots$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\hat{R}(s, a, s')$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$R(B, east, C) =$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$R(C, east, D) =$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$R(D, exit, x) =$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\ldots$</td>
</tr>
<tr>
<td>Assume: $\gamma = 1$</td>
<td>B, east, C, -1</td>
<td>$T(B, east, C) =$</td>
</tr>
<tr>
<td></td>
<td>C, east, D, -1</td>
<td>$T(C, east, D) =$</td>
</tr>
<tr>
<td></td>
<td>D, exit, $x$, +10</td>
<td>$T(C, east, A) =$</td>
</tr>
<tr>
<td></td>
<td>B, east, C, -1</td>
<td>$\ldots$</td>
</tr>
<tr>
<td></td>
<td>C, east, D, -1</td>
<td>$\hat{T}(s, a, s')$</td>
</tr>
<tr>
<td></td>
<td>D, exit, $x$, +10</td>
<td>$\hat{R}(s, a, s')$</td>
</tr>
<tr>
<td></td>
<td>E, north, C, -1</td>
<td>$R(B, east, C) =$</td>
</tr>
<tr>
<td></td>
<td>C, east, D, -1</td>
<td>$R(C, east, D) =$</td>
</tr>
<tr>
<td></td>
<td>D, exit, $x$, +10</td>
<td>$R(D, exit, x) =$</td>
</tr>
<tr>
<td></td>
<td>A, exit, $x$, -10</td>
<td>$\ldots$</td>
</tr>
</tbody>
</table>
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

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

- $T(B, east, C) = 1$
- $T(C, east, D) = 0.75$
- $T(C, east, A) = 0.25$

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

- $R(B, east, C) = -1$
- $R(C, east, D) = -1$
- $R(D, exit, x) = 10$

...
Example: Model-Free Direct Evaluation

<table>
<thead>
<tr>
<th>Input Policy $\pi$</th>
<th>Observed Episodes (Training)</th>
<th>Output Values</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assume: $\gamma = 1$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Episode 1</th>
<th>Episode 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>B, east, C, -1</td>
<td>B, east, C, -1</td>
</tr>
<tr>
<td>C, east, D, -1</td>
<td>C, east, D, -1</td>
</tr>
<tr>
<td>D, exit, x, +10</td>
<td>D, exit, x, +10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Episode 3</th>
<th>Episode 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>E, north, C, -1</td>
<td>E, north, C, -1</td>
</tr>
<tr>
<td>C, east, D, -1</td>
<td>C, east, A, -1</td>
</tr>
<tr>
<td>D, exit, x, +10</td>
<td>A, exit, x, -10</td>
</tr>
</tbody>
</table>

Algorithm: Average all total/future rewards that start at each state
Example: Model-Free Direct Evaluation

Input Policy $\pi$

Assume: $\gamma = 1$

Observed Episodes (Training)

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

Output Values

A: -10 [-10]
B: 8, 8 [8]
C: 9, 9, 9, -11 [4]
D: 10, 10, 10 [10]
E: 8, -12 [-2]

Algorithm: Average all total/future rewards that start at each state
Example: Temporal Difference Learning

Assume: $\gamma = 1$, $\alpha = 0.5$

$$sample = R(s, \pi(s), s') + \gamma V^\pi(s')$$

$$V^\pi(s) \leftarrow (1 - \alpha)V^\pi(s) + (\alpha)sample$$
Poll

Which of the following allows you to estimate the **optimal policy?**

- (A) Model-based RL
- (B) Model-free RL: direct policy evaluation
- (C) Model-free RL: temporal difference value learning
From TD Value Learning to Q-learning

- TD value learning is a model-free way to do policy evaluation, mimicking Bellman updates with running sample averages.
- If we want to turn values into a (new) policy, we can learn Q-values, not state values.

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

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

Conceptual; actual details in next slides.
Bootstrapped prediction of Q-values

Estimating $V^\pi(s)$ from $(s, a, s', r)$

Sample of $V^\pi(s)$: $sample = r + \gamma V^\pi(s')$

Update to $V^\pi(s)$: $V^\pi(s) \leftarrow (1 - \alpha)V^\pi(s) + (\alpha)sample$

Estimating $Q^\pi(s)$ from $(s, a, s', r, a')$

Sample of $Q^\pi(s)$: $sample = r + \gamma Q^\pi(s', a')$

Update to $Q^\pi(s)$: $Q^\pi(s) \leftarrow (1 - \alpha)Q^\pi(s) + \alpha sample$
Q-learning

- Expectimax update for optimal Q-values

\[ Q_{k+1}(s, a) \leftarrow \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma \max_{a'} Q_k(s', a') \right] \]

- Q-learning: sample-based Q-value iteration

- Given data \((s, a, s', R)\):
  - \text{sample} = R + \gamma \max_{a'} Q(s, a') \text{ (consider new sample estimate)}
  - \[ Q(s, a) = (1 - \alpha)Q(s, a) + \alpha \text{sample} \text{ (incorporate into running avg)} \]
Q-learning properties

- Important property: Q-learning converges to values of the optimal policy even if you are acting suboptimally.
- This is called off-policy learning.
  - Learning about the optimal policy while the experience is obtained via a different (suboptimal) policy.
- Caveats:
  - Data-collecting policy has to explore enough.
  - Have to lower the learning rate $\alpha$ eventually.
    - But not too quickly.
- Basically, in the limit, doesn’t matter how you select actions!
In-class activity

Assume: $\gamma = 1$
$\alpha = 0.5$

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + (\alpha) \left[ r + \gamma \max_{a'} Q(s', a') \right]$$
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>

Model-Based RL

<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>

Model-Free RL

<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>TD Value Learning</td>
</tr>
</tbody>
</table>
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…
Exploration vs. Exploitation
How to Explore?

○ Several schemes for forcing exploration
  ○ Simplest: random actions (ε-greedy)
    ○ Every time step, flip a coin
    ○ With (small) probability ε, act randomly
    ○ With (large) probability 1-ε, act on current policy

○ Problems with random actions?
  ○ You do eventually explore the space, but keep thrashing around once learning is done
  ○ One solution: lower ε over time
  ○ Another solution: exploration function …
Exploration Functions

- **When to explore?**
  - Random actions: explore a fixed amount
  - **Better idea:** explore areas whose badness is not (yet) established, eventually stop exploring

- **Exploration function**
  - Takes a *value estimate* $u$ and a *visit count* $n$, and returns an optimistic utility, e.g.
    \[
    f(u, n) = u + k/(n + 1)
    \]

Regular Q-Update: \[
Q(s, a) = (1 - \alpha)Q(s, a) + \alpha [r + \gamma \max_{a'} Q(s', a')]\]

Modified Q-Update: \[
Q(s, a) = (1 - \alpha)Q(s, a) + \alpha [r + \gamma \max_{a'} f(Q(s', a'), N(s', a'))]\]

- **Note:** this propagates the “bonus” back to states that lead to unknown states as well!
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
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!
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
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
- Can also describe a q-state \((s, a)\) with features (e.g., action moves closer to food)

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

- $V_w(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$
- $Q_w(s, a) = w_1 f_1(s, a) + w_2 f_2(s, a) + \ldots + w_n f_n(s, a)$

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

**Disadvantage:** states may share features but actually be very different in value!
Updating a linear value function

- Original Q learning rule tries to reduce prediction error at $s, a$:
  - $Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha[R(s, a, s') + \gamma \max_{a'} Q(s', a')]$
  - $Q(s, a) \leftarrow Q(s, a) + \alpha[R(s, a, s') + \gamma \max_{a'} Q(s', a') - Q(s, a)]$

- Instead, we update the weights to try to reduce the error at $s, a$:
  - $w_i \leftarrow ?$
Detour: Minimizing Error 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

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

**Goal:** find \( x \) that minimizes \( f(x) \)

1. Start with initial guess, \( x_0 \)
2. Update \( x \) by taking a step in the direction that \( f(x) \) is changing fastest (in the negative direction) with respect to \( x \):
   \[
   x ← x - α \nabla_x f,
   \]
   where \( α \) is the step size or learning rate
3. Repeat until convergence
Gradient Descent and Q learning

- Gradient descent on \( f(x) = \frac{1}{2}(y - x)^2 \)
- We know that \( \frac{df}{dx} = -(y - x) \); so \( x \leftarrow x + \alpha (y - x) \)
- **Q-learning:** find values \( Q(s, a) \) that minimizes difference between samples and \( Q(s, a) \)
  - \( Error(Q(s, a)) = \frac{1}{2} (sample - Q(s, a))^2 \)
  - \( Q(s, a) \leftarrow Q(s, a) - \alpha \nabla_{Q(s,a)} Error \)
  - \( Q(s, a) \leftarrow Q(s, a) + \alpha [(R(s, a, s') + \gamma \max_{a'} Q(s', a')) - Q(s, a)] \)

  "target" (sample)  "prediction"
Approximate Q-learning and gradient descent

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:

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

"target" (sample)     "prediction"
Updating a linear value function

- Original Q learning rule tries to reduce prediction error at $s, a$:
  - $Q(s, a) \leftarrow Q(s, a) + \alpha [(R(s, a, s') + \gamma \max_{a'} Q(s', a')) - Q(s, a)]$

- Instead, we update the weights to try to reduce the error at $s, a$:
  - $w_i \leftarrow w_i + \alpha \cdot f_i(s, a) \cdot [(R(s, a, s') + \gamma \max_{a'} Q(s', a')) - Q(s, a)]$
Approximate Q-Learning summary

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

- 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
Poll: Pacman with approximate Q learning

- Two features: $f_{\text{dot}}(s, a)$ and $f_{\text{gst}}(s, a)$
- Current weights: $w_{\text{dot}} = 4, w_{\text{gst}} = -1$

$\alpha = 0.004$

$Q(s, N) = 4 \times 0.5 + (-1) \times 1 = 1$

$Q(s', \cdot) = 0$

\begin{align*}
(A) & \quad w_{\text{dot}} \text{ and } w_{\text{gst}} \text{ both increase by same amount} \\
(B) & \quad w_{\text{dot}} \text{ and } w_{\text{gst}} \text{ both decrease by same amount} \\
(C) & \quad w_{\text{dot}} \text{ and } w_{\text{gst}} \text{ both increase, } w_{\text{dot}} \text{ increases by larger amount} \\
(D) & \quad w_{\text{dot}} \text{ and } w_{\text{gst}} \text{ both increase, } w_{\text{gst}} \text{ increase by larger amount} \\
(E) & \quad w_{\text{dot}} \text{ and } w_{\text{gst}} \text{ both decrease, } w_{\text{dot}} \text{ decreases by larger amount} \\
(F) & \quad w_{\text{dot}} \text{ and } w_{\text{gst}} \text{ both decrease, } w_{\text{gst}} \text{ decreases by larger amount}
\end{align*}
Poll: Pacman with approximate Q learning

- Two features: $f_{\text{dot}}(s, a)$ and $f_{\text{gst}}(s, a)$
- Current weights: $w_{\text{dot}} = 4, w_{\text{gst}} = -1\quad \alpha = 0.004, \gamma = 1.0$

$Q(s, N) = 4 \times 0.5 + (-1) \times 1 = 1$

$w_i \leftarrow w_i + \alpha \text{ [difference]} f_i(s, a)$

$s'$

$Q(s', a) = 0 \forall a$

$\alpha = N$

$R = -500$

$sample = R + \gamma \max_{a'} Q(s', a') = -500$

$estimate = Q(s, a) = 1$

$w_{\text{dot}} \leftarrow 4 + \alpha(-501) 0.5 = 3.0$

$w_{\text{gst}} \leftarrow -1 + \alpha(-501) 1.0 = -3.0$
All equations we saw so far

Standard expectimax: \[ V(s) = \max_a \sum_{s'} P(s'|s, a) V(s') \]

Bellman equations: \[ V^*(s) = \max_a \sum_{s'} P(s'|s, a)[R(s, a, s') + \gamma V^*(s')] \]

Value iteration: \[ V_{k+1}(s) = \max_a \sum_{s'} P(s'|s, a)[R(s, a, s') + \gamma V_k(s')], \quad \forall s \]

Q-iteration: \[ Q_{k+1}(s, a) = \sum_{s'} P(s'|s, a)[R(s, a, s') + \gamma \max_{a'} Q_k(s', a')], \quad \forall s, a \]

Policy extraction: \[ \pi_V(s) = \arg\max_a \sum_{s'} P(s'|s, a)[R(s, a, s') + \gamma V(s')], \quad \forall s \]

Policy evaluation: \[ V_{k+1}^\pi(s) = \sum_{s'} P(s'|s, \pi(s))[R(s, \pi(s), s') + \gamma V_k^\pi(s')], \quad \forall s \]

Policy improvement: \[ \pi_{\text{new}}(s) = \arg\max_a \sum_{s'} P(s'|s, a)[R(s, a, s') + \gamma V_{\pi_{\text{old}}}(s')], \quad \forall s \]

Value (TD) learning: \[ V^\pi(s) = V^\pi(s) + \alpha [r + \gamma V^\pi(s') - V^\pi(s)] \]

Q-learning: \[ Q(s, a) = Q(s, a) + \alpha [r + \gamma \max_{a'} Q(s', a') - Q(s, a)] \]
Recent Reinforcement Learning Milestones
TDGammon

- 1992 by Gerald Tesauro
- 4-ply lookahead using $V(s)$ trained from 1,500,000 games of self-play
- 3 hidden layers, ~100 units each
- Input: contents of each location plus several handcrafted features
- Experimental results:
  - Approximately as strong as world champion
  - Led to radical changes in the way humans play backgammon
Deep Q-Networks

- Deep Mind, 2015
- Used a deep learning network to represent Q:
  - Input is last 4 images (84x84 pixel values) plus score
- 49 Atari games, incl. Breakout, Space Invaders, Seaquest, Enduro
OpenAI Gym

- 2016+
- Benchmark problems for learning agents
- https://gym.openai.com/envs

Images: Open AI
AlphaGo, AlphaZero

- Deep Mind, 2016+
Autonomous Vehicles?