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

Training

[Video: AIBO WALK – initial]

[Video: AIBO WALK – training]
Example: Learning to Walk

Kohl and Stone, ICRA 2004

Example: Toddler Robot

Tedrake, Zhang and Seung, 2005

The Crawler!

Demo: Crawler Bot (L10D1)

You, in Project 3

Video of Demo Crawler Bot

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

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 $P(s, a, s')$
  - Discover each $P(s, a, s')$ when we experience (s, a, s')
- **Step 2: Solve the learned MDP**
  - For example, use value iteration, as before

---

Example: Model-Based Learning

<table>
<thead>
<tr>
<th>Input Policy $\pi$</th>
<th>Observed Episodes (Training)</th>
<th>Learned Model $P(s, a, s')$</th>
<th>$R(s, a, s')$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assumed: $\gamma = 1$</td>
<td>Episode 1: B, east, C, -1; C, east, D, -1; D, exit, x, +10</td>
<td>$P(s, a, s')$:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Episode 2: B, east, C, -1; C, east, D, -1; D, exit, x, +10</td>
<td>T(B, east, C) = 1.00; T(C, east, D) = 0.75; T(C, east, A) = 0.25; ...</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Episode 3: E, north, C, -1; C, east, A, -1; A, exit, x, -10</td>
<td>$R(s, a, s')$:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Episode 4: E, north, C, -1; C, east, D, -1; D, exit, x, +10</td>
<td>R(E, north, C) = -1; R(C, east, D) = -1; R(D, exit, x) = +10; ...</td>
<td></td>
</tr>
</tbody>
</table>

---

Example: Expected Age

- **Goal:** Compute expected age of CSE 473 students
- **Unknown P(A):** "Model Based"
  - Known P(A): $E[A] = \sum_{a} P(a) \cdot a = 0.35 \times 20 + ...$
- **Unknown P(A):** "Model Free"
  - Without P(A), instead collect samples $[a_1, a_2, ..., a_N]$
  - Why does this work? Because eventually you learn the right model.

---

Model-Free Learning

- **Passive Reinforcement Learning**
  - $P(a) = \frac{\text{sum}(a)}{N}$
  - $E[A] = \sum_{a} P(a) \cdot a$

---

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

<table>
<thead>
<tr>
<th>Input Policy ( \pi )</th>
<th>Observed Episodes (Training)</th>
<th>Output Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A ) ( B ) ( C ) ( D ) ( E )</td>
<td>Episode 1: B, east, C, -1 C, east, D, 1 D, exit, x, +10</td>
<td>( V_0(s) = 0 )</td>
</tr>
<tr>
<td>( A ) ( B ) ( C )</td>
<td>Episode 2: B, east, C, -1 C, east, D, -1 D, exit, x, +10</td>
<td>( V_{k+1}(s) = \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V_k(s')] )</td>
</tr>
<tr>
<td>( A ) ( B ) ( C ) ( D )</td>
<td>Episode 3: E, north, C, -1 C, east, D, -1 A, exit, x, -10</td>
<td>( V_{k+1}(s) = \sum_{s'} T(s, \pi(s), s')R(s, \pi(s), s') + \gamma V_k(s') )</td>
</tr>
<tr>
<td>( A ) ( B ) ( C )</td>
<td>Episode 4: E, north, C, -1 C, east, D, -1 A, exit, x, -10</td>
<td>( V_{k+1}(s) = \sum_{s'} T(s, \pi(s), s')R(s, \pi(s), s') + \gamma V_k(s') )</td>
</tr>
</tbody>
</table>

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

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 \( V \)
  - 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?

Sample-Based Policy Evaluation?

- We want to improve our estimate of \( V \) by computing these averages:
  - For each \( s \), take \( k+1 \) samples of \( s' \) (by doing the action!) and average
  - \( V_{k+1}(s) = \sum_i T(s, \pi(s), s'_i)[R(s, \pi(s), s'_i) + \gamma V_k(s'_i)] \)

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

\[ V_{k+1}(s) = \frac{1}{n}\sum_{i=1}^{n} R(s, \pi(s), s'_i) + \gamma \sum_{i=1}^{n} V_k(s'_i) \]
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 $V(s)$:

$$V^\pi(s) \leftarrow (1 - \alpha)V^\pi(s) + \alpha \text{sample}$$

Exponential Moving Average

- Exponential moving average
- The running interpolation update:

$$\tilde{x}_n = \frac{x_n + (1 - \alpha) \cdot \tilde{x}_{n-1} + \alpha \cdot x_n}{1 + (1 - \alpha) + (1 - \alpha)^2 + \ldots}$$
- Forgets about the past (distant past values were wrong anyway)
- Decreasing learning rate (alpha) can give converging averages

Example: Temporal Difference Learning

<table>
<thead>
<tr>
<th>States</th>
<th>Observed Transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>8</td>
</tr>
<tr>
<td>D</td>
<td>-1</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
</tr>
</tbody>
</table>

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

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

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:

$$\pi(s) = \arg \max_a Q(s, a)$$

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

- Idea: learn Q-values, not values
- Makes action selection model-free too!

Active Reinforcement Learning

- Full reinforcement learning: optimal policies (like value iteration)
- You don’t know the actions $T(s, a, s')$
- 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:
    \[ V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_k(s')] \]
- 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:
      - Sample $= R(s, a, s') + \gamma \max_{a'} Q_k(s', a')$
    - Incorporate the new estimate into a running average:
      \[ Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha [\text{sample}] \]

**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 (1)