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!

The “Credit Assignment” Problem
I’m in state 43, reward = 0, action = 2
- " " = 22, " " = 0, " " = 1
- " " = 0, " " = 4
- " " = 0, " " = 4
The "Credit Assignment" Problem

Yippee! I got to a state with a big reward!

But which of my actions along the way actually helped me get there??

This is the Credit Assignment problem.

Exploration-Exploitation tradeoff

- You have visited part of the state space and found a reward of 100
  - is this the best you can hope for??
- Exploitation: should I stick with what I know and find a good policy w.r.t.
  this knowledge?
  - at risk of missing out on a better reward somewhere
- Exploration: should I look for states w/ more reward?
  - at risk of wasting time & getting some negative reward
Example: Learning to Walk

[Images of a robot in different stages of learning to walk]

[Text]

Example: Learning to Walk

[Image of robot]

The Crawler!

[Image of robot]

Video of Demo Crawler Bot

[Image of robot]

Reinforcement Learning

- Still assume a Markov decision process (MDP):
  - A set of states $s$ in $S$
  - A set of actions $a$ in $A$
  - A transition function $T(s, a, s')$
  - 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

[Kohl and Stone, ICRA 2004]
**Offline (MDPs) vs. Online (RL)**

**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.

**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 \( T(s,a,s’) \)
  - Discover each \( R(s,a,s’) \) when we experience \( (s,a,t) \)

- **Step 2:** Solve the learned MDP
  - For example, use value iteration, as before

**Example: Model-Based Learning**

<table>
<thead>
<tr>
<th>Input Policy</th>
<th>Observed Episodes (Training)</th>
<th>Learned Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \pi )</td>
<td>Episode 1</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>B, east, C, -1</td>
<td>( T(s,a,s’) )</td>
</tr>
<tr>
<td>B, east, C, -1</td>
<td>C, east, D, -1</td>
<td>T(B, east, C) = 1.00</td>
</tr>
<tr>
<td>C, east, D, -1</td>
<td>D, exit, x, +10</td>
<td>T(C, east, D) = 0.75</td>
</tr>
<tr>
<td>D, exit, x, +10</td>
<td>A, exit, x, -10</td>
<td>T(D, exit, A) = 0.25</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Observed Episodes (Training)</th>
<th>Learned Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Episode 2</td>
<td></td>
</tr>
<tr>
<td>B, east, C, -1</td>
<td>( T(s,a,s’) )</td>
</tr>
<tr>
<td>C, east, D, -1</td>
<td>T(B, east, C) = 1.00</td>
</tr>
<tr>
<td>D, exit, x, +10</td>
<td>T(C, east, D) = 0.75</td>
</tr>
<tr>
<td>A, exit, x, -10</td>
<td>T(D, exit, A) = 0.25</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Observed Episodes (Training)</th>
<th>Learned Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Episode 3</td>
<td></td>
</tr>
<tr>
<td>E, north, C, -1</td>
<td>( T(s,a,s’) )</td>
</tr>
<tr>
<td>C, east, D, -1</td>
<td>T(E, north, C) = 1.00</td>
</tr>
<tr>
<td>D, exit, x, +10</td>
<td>T(C, east, D) = 0.75</td>
</tr>
<tr>
<td>A, exit, x, -10</td>
<td>T(D, exit, A) = 0.25</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Observed Episodes (Training)</th>
<th>Learned Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Episode 4</td>
<td></td>
</tr>
<tr>
<td>E, north, C, -1</td>
<td>( T(s,a,s’) )</td>
</tr>
<tr>
<td>C, east, A, -1</td>
<td>T(E, north, C) = 1.00</td>
</tr>
<tr>
<td>D, exit, x, +10</td>
<td>T(C, east, A) = 0.75</td>
</tr>
<tr>
<td>A, exit, x, -10</td>
<td>T(D, exit, A) = 0.25</td>
</tr>
</tbody>
</table>
Example: Expected Age

Goal: Compute expected age of cs473 students

\[ E(A) = \sum P(A) e^a \]

Known \( P(A) \): 

\[ E(A) = \frac{0.25 \times 20 + \ldots}{N} \]

Without \( P(A) \), instead collect samples \( \{a_1, a_2, \ldots, a_N\} \)

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

Unknown \( P(A) \): “Model Based”

\[ E(A) = \frac{1}{N} \sum a_i \]

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

Unknown \( P(A) \): “Model Free”

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

Why 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’s bad about it?
  - It wastes information about state connections
  - Each state must be learned separately
  - So, it takes a long time to learn

Example: Direct Evaluation

Input Policy \( \pi \)

Observed Episodes (Training)

Output Values

Problem with Direct Evaluation

- Simplified Bellman updates calculate \( V \) for a fixed policy:
  - Each round, replace \( V \) with a one-step-look-ahead layer over \( V \)
  \[ V_{t+1}(s) = 0 \]
  \[ V_{t+1}(s) = \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V_t(s')] \]
- Key question: how can we do this update to \( V \) without knowing \( T \) and \( R \)?
- In other words, how do we take a weighted average without knowing the weights?

Example: Expected Age

Model-Free Learning

Goal: Compute expected age of cs473 students

\[ E(A) = \sum P(A) e^a \]

Without \( P(A) \), instead collect samples \( \{a_1, a_2, \ldots, a_N\} \)

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

Unknown \( P(A) \): “Model Based”

\[ E(A) = \frac{1}{N} \sum a_i \]

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

Unknown \( P(A) \): “Model Free”

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 \( V \)
  \[ V_{t+1}(s) = 0 \]
  \[ V_{t+1}(s) = \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V_t(s')] \]
- This approach fully exploited the connections between the states
- Unfortunately, we need \( T \) and \( R \) to do it!
Sample-Based Policy Evaluation?

- We want to improve our estimate of $V$ by computing these averages:
  $$v_{T+1}^\pi(s) = \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma v_{T+1}^\pi(s')]$$
  
- Idea: Take samples of outcomes $s'$ [by doing the action!] and average
  
  \[
  \begin{align*}
  \text{sample}_1 &= R(s, \pi(s), s') + \gamma v_{T+1}^\pi(s') \\
  \text{sample}_2 &= R(s, \pi(s), s') + \gamma v_{T+1}^\pi(s') \\
  \text{sample}_3 &= R(s, \pi(s), s') + \gamma v_{T+1}^\pi(s') \\
  v_{T+1}^\pi(s) &= \frac{1}{n} \sum \text{sample}_i
  \end{align*}
  \]

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

$$\text{Sample of } V(s): \quad \text{sample} = R(s, \pi(s), s') + \gamma V(s')$$

$$\text{Update to } V(s): \quad v_{T+1}^\pi(s) \leftarrow (1 - \alpha) v_{T}^\pi(s) + \alpha \text{sample}$$

$$\text{Same update: } \quad v_{T+1}^\pi(s) \leftarrow v_{T}^\pi(s) + \alpha [\text{sample} - v_{T}^\pi(s)]$$

Exponential Moving Average

- Exponential moving average
  - The running interpolation update: $\tilde{x}_n = (1 - \alpha) \tilde{x}_{n-1} + \alpha \cdot x_n$
  - Makes recent samples more important:
    $$\tilde{x}_n = x_n \underbrace{(1 - \alpha) x_{n-1} + (1 - \alpha)^2 x_{n-3} + \ldots}_{\text{forgets about the past (distant past values were wrong anyway)}}$$
  - 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>b, east, C, 2</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0, -1</td>
</tr>
<tr>
<td>D</td>
<td>a, -1</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
</tr>
</tbody>
</table>

Assume $\gamma = 0.9$ or $\lambda = 0.5$

$$v_{T}^\pi(s) \leftarrow (1 - \alpha) v_{T}^\pi(s) + \alpha [R(s, \pi(s), s') + \gamma v_{T}^\pi(s')]$$

Problems with TD Value Learning

- TD value learning is a model-free way to do policy evaluation, mimicking Bellman updates with running sample averages
- However, if we want to turn values into a (new) policy, we’re sunk:
  $$\pi(s) = \arg \max_a Q(s, a)$$
  $$Q(s, a) = \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma v_{T+1}^\pi(s')]$$

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

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...

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:
    $$V_{k+1}(s) = \max_{a} \sum_{s'} T(s,a,s') \left[ R(s,a,s') + \gamma V_k(s') \right]$$
  - 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 states:
      $$Q_{k+1}(s,a) = \sum_{s'} T(s,a,s') \left[ R(s,a,s') + \gamma \max_{a'} Q_k(s', a') \right]$$

Q-Learning

- Q-Learning: sample-based Q-value iteration
  $$Q_{k+1}(s,a) = \sum_{s'} T(s,a,s') \left[ R(s,a,s') + \gamma \max_{a'} Q_k(s', a') \right]$$
  - 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:
      $$Q(s,a) = (1-\alpha)Q(s,a) + \alpha [sample]$$

Q-Learning

- For all $s,a$
  - Initialize $Q(s,a) = 0$
- Repeat Forever
  - Where are you? $s$
  - Choose some action $a$
  - Execute it in real world: $(s,a,r,s')$
  - Do update:
    $$Q(s,a) = (1-\alpha)Q(s,a) + \alpha \left[ r + \gamma \max_{a'} Q(s', a') \right]$$

Video of Demo Q-Learning -- Gridworld

Video of Demo Q-Learning -- Crawler
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 (!)

Two main reinforcement learning approaches

- Model-based approaches:
  - explore environment & learn model, \( T(p(s|a)) \) and \( R(s,a) \), (almost) everywhere
  - use model to plan policy, MDP-style
  - approach leads to strongest theoretical results
  - often works well when state-space is manageable
- Model-free approach:
  - don’t learn a model, learn value function or policy directly
  - weaker theoretical results
  - often works better when state space is large

The Story So Far: MDPs and RL

<table>
<thead>
<tr>
<th>Known MDP: Offline Solution</th>
<th>Unknown MDP: Model-Based</th>
<th>Unknown MDP: Model-Free</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>Technique</td>
<td>Goal</td>
</tr>
<tr>
<td>Compute ( V^<em>, Q^</em>, \pi^* )</td>
<td>Value / policy iteration</td>
<td>Compute ( V^<em>, Q^</em> )</td>
</tr>
<tr>
<td>Evaluate a fixed policy ( \pi )</td>
<td>Policy evaluation</td>
<td>Evaluate a fixed policy ( \pi )</td>
</tr>
</tbody>
</table>

Video of Demo Q-Learning Auto Cliff Grid