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

...
The "Credit Assignment" Problem

I'm in state 43, reward = 0, action = 2

* * * 39, * = 0, * = 4
* * * 22, * = 0, * = 1
* * * 21, * = 0, * = 1
* * * 13, * = 0, * = 1

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**

- Initial
- A Learning Trial
- After Learning [1K Trials]

---

**Example: Learning to Walk**

- Initial
- Finished

---

**The Crawler!**

---

**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 \( a \) 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

Offline (MDPs) vs. Online (RL)

<table>
<thead>
<tr>
<th>Offline Solution</th>
<th>Online Learning</th>
</tr>
</thead>
</table>

Offline Solution

Online Learning

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.

Passive Reinforcement 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 \( T(s, a, s') \)
    - Discover each \( R(s, a, s') \) when we experience \( (s, a, s') \)
  - Step 2: Solve the learned MDP
    - For example, use value iteration, as before
Example: Model-Based Learning

### Input Policy $\pi$

<table>
<thead>
<tr>
<th>Observed Episodes (Training)</th>
<th>Learned Model</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Episode 1</strong></td>
<td>$T(s, a, s')$</td>
</tr>
<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>
<tr>
<td><strong>Episode 2</strong></td>
<td>$T(s, a, s')$</td>
</tr>
<tr>
<td>B, east, C, -1</td>
<td>B, east, D, -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>

### 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, D, -1; D, exit, x, +10

### Model-Free Learning

- **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
- **Direct Evaluation**

Example: Expected Age

**Goal**: Compute expected age of cs473 students

**Model-Free Learning**

- Unknown $P(A)$: "Model-Free"
- Why does this work? Because samples appear with the right frequencies.

**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

### Input Policy $\pi$

<table>
<thead>
<tr>
<th>Observed Episodes (Training)</th>
<th>Output Values</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Episode 1</strong></td>
<td>$T(s, a, s')$</td>
</tr>
<tr>
<td>B, east, C, -1</td>
<td>+8</td>
</tr>
<tr>
<td>C, east, D, -1</td>
<td>+4</td>
</tr>
<tr>
<td>D, exit, x, +10</td>
<td>+10</td>
</tr>
<tr>
<td><strong>Episode 2</strong></td>
<td>$T(s, a, s')$</td>
</tr>
<tr>
<td>B, east, C, -1</td>
<td>-2</td>
</tr>
<tr>
<td>C, east, D, -1</td>
<td>-4</td>
</tr>
<tr>
<td>D, exit, x, +10</td>
<td>-6</td>
</tr>
<tr>
<td><strong>Episode 3</strong></td>
<td>$T(s, a, s')$</td>
</tr>
<tr>
<td>E, north, C, -1</td>
<td>+8</td>
</tr>
<tr>
<td>C, east, D, -1</td>
<td>+4</td>
</tr>
<tr>
<td>D, exit, x, +10</td>
<td>+10</td>
</tr>
<tr>
<td><strong>Episode 4</strong></td>
<td>$T(s, a, s')$</td>
</tr>
<tr>
<td>E, north, C, -1</td>
<td>-2</td>
</tr>
<tr>
<td>C, east, D, -1</td>
<td>-4</td>
</tr>
<tr>
<td>D, exit, x, +10</td>
<td>-6</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’s bad about it?**
  - It wastes information about state connections
  - Each state must be learned separately
  - So, it takes a long time to learn

**Output Values**

- If B and E both go to C under this policy, how can their values be different?
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
    \[ V_0(s) = 0 \]
    \[ V_{n+1}(s) = \sum_{s'} T(s, \pi(s), s') R(s, \pi(s), s') + \gamma V(s') \]
  - 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 do 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:
  \[ V_{n+1}^{(s)}(s) = \frac{1}{n} \sum_{t=1}^{n} \left( R(s, \pi(s), s') + \gamma V(s') \right) \]
- Idea: Take samples of outcomes s' [by doing the action!] and average
  \[ \text{sample}_1 = R(s, \pi(s), s') + \gamma V(s') \]
  \[ \text{sample}_2 = R(s, \pi(s), s') + \gamma V(s') \]
  \[ \text{sample}_n = R(s, \pi(s), s') + \gamma V(s') \]
  \[ V_{n+1}^{(s)}(s) = \frac{1}{n} \sum_{t=1}^{n} \text{sample}_t \]

Temporal Difference Learning

- Big idea: learn from every experience!
  - Update V(s) each time we experience a transition (s, s', r)
  - Likely outcomes s' will contribute updates more often
- Temporal difference learning of values:
  - Policy still fixed, still doing evaluation!
  - Move value toward value of whatever successor occurs: running average
    \[ V(s) = (1 - \alpha) V(s) + (\alpha) \text{sample} \]
    \[ V(s) = V(s) + \alpha (\text{sample} - V(s)) \]
  - Same update:

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, 0, 0, 0</td>
</tr>
<tr>
<td>C</td>
<td>0, 0, 0, 0</td>
</tr>
<tr>
<td>D</td>
<td>0, 0, 0, 0</td>
</tr>
</tbody>
</table>

Assume: \( q = 1, r = 0.5, \alpha = 0.2 \)

\[ V(s) = (1 - \alpha) V(s) + \alpha [R(s, \pi(s), s') + \gamma V(s')] \]

Problems with TD Value Learning

- TD value leaning is a model-free way to do policy evaluation, mimicking Bellman updates with running sample averages
  \[ V(s) = \arg \max Q(s, a) \]
- However, if we want to turn values into a (new) policy, we’re sunk:
  \[ \pi(s) = \arg \max Q(s, a) \]
- Idea: learn Q-values, not values
  - Makes action-values model-free too!
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 $Q$-term 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:

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

Q-Learning

Q-Learning: sample-based Q-value iteration

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

Learn $Q(s,a)$ values as you go
- Receive a sample $(s,a,r,s')$
- Consider your old estimate: $Q_k(s,a)$
- Consider your new sample estimate:

$$\text{sample} = R(s,a,s') + \gamma \max_{a'} Q_k(s',a')$$

Incorporate the new estimate into a running average:

$$Q(s,a) = (1-\alpha)Q_k(s,a) + \alpha \text{sample}$$

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
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=\text{P}(s'|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>Value / policy iteration</td>
<td>Q-learning</td>
</tr>
<tr>
<td>Technique</td>
<td>Policy evaluation</td>
<td></td>
</tr>
<tr>
<td>Compute ( V^<em>, Q^</em>, \pi^* )</td>
<td>Evaluate a fixed policy ( \pi )</td>
<td>Value / policy iteration</td>
</tr>
<tr>
<td>VI/PI on approx. MDP</td>
<td>Evaluate a fixed policy ( \pi )</td>
<td></td>
</tr>
</tbody>
</table>

<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 / policy iteration</td>
</tr>
</tbody>
</table>

Video of Demo Q-Learning Auto Cliff Grid