Double Bandits

Double-Bandit MDP

Actions: Blue, Red
States: Win, Lose

No discount
100 time steps
Both states have the same value

Offshore Planning

• Solving MDPs is offline planning
• You determine all quantities through computation
• You need to know the details of the MDP
• You do not actually play the game!

Play Red 150
Play Blue 100

Let’s Play!

Online Planning

• Rules changed! Red’s win chance is different.

Let’s Play!

What Just Happened?

• That wasn’t planning, it was learning!
  • Specifically, reinforcement learning
  • There was an MDP, but you couldn’t solve it with just computation
  • You needed to actually act to figure it out

• Important ideas in reinforcement learning that came up
  • Exploration: you have to try unknown actions to get information
  • Exploitation: eventually, you have to use what you know
  • Regret: even if you learn intelligently, you make mistakes
  • Sampling: because of chance, you have to try things repeatedly
  • Difficulty: learning can be much harder than solving a known MDP

Reinforcement Learning
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!

- 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

Example: Learning to Walk

Initial

Training

[Video: AIBO WALK - initial]

[Video: AIBO WALK - training]

Finished

[Video: AIBO WALK - finished]

The Crawler!

[Demo: Crawler Bot (L10D1)]

Video of Demo Crawler Bot

Offline (MDPs) vs. Online (RL)

Offline Solution

Online Learning

Model-Based 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 \( \pi(s, a, s') \) 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</th>
<th>Observed Episodes (Training)</th>
<th>Learned Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>[\pi(s, a)]</td>
<td>Episode 1: B, east, C, -1; C, east, D, -1; D, east, x, +10</td>
<td>[\pi(s, a, s')]</td>
</tr>
<tr>
<td></td>
<td>Episode 2: B, east, C, -1; C, east, D, -1; D, east, x, +10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Episode 3: E, north, C, -1; C, east, A, -1; A, exit, x, -10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Episode 4: E, north, C, -1; C, east, D, -1; D, exit, x, +10</td>
<td></td>
</tr>
</tbody>
</table>

---

**Model-Free 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.

---

**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</th>
<th>Observed Episodes (Training)</th>
<th>Output Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>[\pi(s, a)]</td>
<td>Episode 1: B, east, C, -1; C, east, D, -1; D, east, x, +10</td>
<td>[\text{Output Values}]</td>
</tr>
<tr>
<td></td>
<td>Episode 2: B, east, C, -1; C, east, D, -1; D, east, x, +10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Episode 3: E, north, C, -1; C, east, A, -1; A, exit, x, -10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Episode 4: E, north, C, -1; C, east, D, -1; D, exit, x, +10</td>
<td></td>
</tr>
</tbody>
</table>

---

**Example: Expected Age**

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

- **Without \( P(A) \), instead collect samples \([a_1, a_2, \ldots, a_N]\)**

- **Known \( P(A) \)**

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

---

**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

---

**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 $V$
  \[
  V_{t+1}^\pi(s) = 0
  \]
  \[
  V_{t+1}^\pi(s) \leftarrow \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V_t^\pi(s')]
  \]
  - 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:
  - Idea: Take samples of outcomes $s'$ (by doing the action!) and average
  \[
  V_{t+1}^\pi(s) \leftarrow \sum_{s'} \text{samples}_t / n
  \]

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!
  - More values toward value of whatever successor occurs: running average

  Sample of $V(s)$: \[\text{sample} = R(s, \pi(s), s') + \gamma V^\pi(s')\]
  Update to $V(s)$: \[V^\pi(s) \leftarrow (1 - \alpha) V^\pi(s) + \alpha \text{sample}\]
  Same update: \[V^\pi(s) \leftarrow V^\pi(s) + \alpha (\text{sample} - V^\pi(s))\]

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:

  \[
  \pi(s) = \arg \max_a Q(s, a)
  \]
  \[
  Q(s, a) = \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^\pi(s')]
  \]

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