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

- 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
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.

**Environment**
- Agent
- Actions: $a$  
- State: $s$  
- Reward: $r$

**Example:**
- Sidewinding (Andrew Ng)
- Toddler Robot (Tedrake, Zhang, Seung, 2005)

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
  - Get 'measurement' of $R$ at each step

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 \( \hat{P}(s, a, s') \)
  - Discover each \( P(s, a, s') \) when no experience \( (s, a, s') \)

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

---

Example: Model-Based Learning

- **Input Policy:**
- **Observed Episodes (Training):**
  - Episode 1: B, east, C, -1
  - Episode 2: B, east, C, -1
  - Episode 3: E, north, C, -1
  - Episode 4: E, north, C, -1

- **Learned Model**

---

Analogy: Expected Age

Goal: Compute expected age of cs188 students

- **Known P(A):**
  \[
  E(A) = \sum_A P(A) \cdot A = (0.5 \times 10) + (0.5 \times -10) = 0
  \]

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

---

Model-Free Learning

- **Passive Reinforcement Learning**

---

Passive Reinforcement 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

- Input Policy \( \pi \)
- Observed Episodes (Training)
- Output Values

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:
  - 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?
Temporal Difference Learning

- Big idea: learn from every experience!
  - Update $V(s)$ each time we experience a transition $(s, a, s')$
  - 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

$$V(s) = \alpha \mu(s) + (1 - \alpha)V(s) + \gamma V(s')$$

Sample of $V(s)$:
$$V(s) = \alpha \mu(s) + (1 - \alpha)V(s) + \gamma V(s')$$

Update to $V(s)$:
$$V(s) = \alpha \mu(s) + (1 - \alpha)V(s) + \gamma V(s')$$

Same update:
$$V(s) = \alpha \mu(s) + (1 - \alpha)V(s) + \gamma V(s')$$

Temporal Difference Learning

- Exponential moving average
  - The running interpolation update: $x_n = (1 - \alpha) x_{n-1} + \alpha x_n$
  - Makes recent samples more important
  - Forgets about the past (distant past values were wrong anyway)
  - Decreasing learning rate (alpha) can give converging averages

Exponential Moving Average

- Exponential moving average
  - Forgets about the past (distant past values were wrong anyway)

Example: Temporal Difference Learning

- Observed Transitions
  - States: A, B, C, D, E
  - Transitions: East, West

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-values are more useful, so compute them instead
  $$Q(s, a) = \sum_{s'} P(s, a, s') [R(s, a, s') + \gamma V(s')]$$

Detour: Q-Value Iteration

- Value iteration: find successive (depth-limited) values
  - Start with $V_0(s) = 0$, which we know is right
  - Given $V_k(s)$, calculate the depth $k+1$ values for all states:
    $$V_{k+1}(s) = \max_a \sum_{s'} P(s, a, s') [R(s, a, s') + \gamma V_k(s')]$$

- But Q-values are more useful, so compute them instead
  - Start with $Q_0(s, a) = 0$, which we know is right
  - Given $Q_k(s, a)$, calculate the depth $k+1$ Q-values for all states:
    $$Q_{k+1}(s, a) = \sum_{s'} P(s, a, s') [R(s, a, s') + \gamma \max_a Q_k(s', a')]$$
Q-Learning:
- Sample-based Q-value iteration
  \[ Q_{t+1}(s,a) = \sum_s T(s,a,s') \left[ R(s,a,s') + \gamma \max_{a'} Q_t(s',a') \right] \]
- Learn Q(s,a) values as you go
  - Receive a sample \((s,a,s',r)\)
  - Consider your old estimate: \(Q_t(s,a)\)
  - Consider your new sample estimate: \(\text{sample} = R(s,a,s') + \gamma \max_{a'} Q_t(s',a')\)
  - Incorporate the new estimate into a running average:
    \[ Q(s,a) = (1 - \alpha)Q(s,a) + \alpha [\text{sample}] \]
- Act according to current optimal (and also explore...)
  - 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...

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 (!)
<table>
<thead>
<tr>
<th>Discussion: Model-Based vs Model-Free RL</th>
</tr>
</thead>
</table>

""