Announcements

- P2: Due Wednesday
- P3: MDPs and Reinforcement Learning is up!
- W2: Out late this week
Recap: MDPs

- **Markov decision processes:**
  - States $S$
  - Actions $A$
  - Transitions $P(s'|s,a)$ (or $T(s,a,s')$)
  - Rewards $R(s,a,s')$ (and discount $\gamma$)
  - Start state $s_0$

- **Quantities:**
  - Policy = map of states to actions
  - Episode = one run of an MDP
  - Utility = sum of discounted rewards
  - Values = expected future utility from a state
  - Q-Values = expected future utility from a q-state

Recap: Optimal Utilities

- The utility of a state $s$:
  - $V^*(s) =$ expected utility starting in $s$ and acting optimally

- The utility of a q-state $(s,a)$:
  - $Q^*(s,a) =$ expected utility starting in $s$, taking action $a$ and thereafter acting optimally

- The optimal policy:
  - $\pi^*(s) =$ optimal action from state $s$
Recap: Bellman Equations

- Definition of utility leads to a simple one-step lookahead relationship amongst optimal utility values:

  Total optimal rewards = maximize over choice of (first action plus optimal future)

- Formally:

  \[ V^*(s) = \max_a Q^*(s, a) \]

  \[ Q^*(s, a) = \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V^*(s') \right] \]

  \[ V^*(s) = \max_a \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V^*(s') \right] \]

Practice: Computing Actions

- Which action should we chose from state s:
  - Given optimal values V?
    \[ \arg \max_a \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V^*(s') \right] \]
  - Given optimal q-values Q?
    \[ \arg \max_a Q^*(s, a) \]

  Lesson: actions are easier to select from Q’s!

[DEMO – MDP action selection]
Value Estimates

- Calculate estimates $V_k^*(s)$
  - Not the optimal value of $s$!
  - The optimal value considering only next $k$ time steps (k rewards)
  - As $k \to \infty$, it approaches the optimal value

- Almost solution: recursion (i.e. expectimax)
- Correct solution: dynamic programming

Memoized Recursion?

- Recurrences (basically truncated expectimax):
  \[ V_i^*(s) = \max_a Q_i^*(s, a) \]
  \[ Q_i^*(s, a) = \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V_{i-1}^*(s') \right] \]
  \[ V_0^*(s) = 0 \]
  \[ \pi_i(s) = \arg \max_a Q_i^*(s, a) \]

- Cache all function call results so you never repeat work
- What happened to the evaluation function?
Value Iteration

- Problems with the recursive computation:
  - Have to keep all the $V_k^*(s)$ around all the time
  - Don’t know which depth $\pi_k(s)$ to ask for when planning

- Solution: value iteration
  - Calculate values for all states, bottom-up
  - Keep increasing $k$ until convergence

Value Iteration

- Idea:
  - Start with $V_0^*(s) = 0$, which we know is right (why?)
  - Given $V_i^*$, calculate the values for all states for depth $i+1$:

$$V_{i+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V_i(s') \right]$$

  - Throw out old vector $V_i^*$
  - Repeat until convergence
  - This is called a value update or Bellman update

- Theorem: will converge to unique optimal values
  - Basic idea: approximations get refined towards optimal values
  - Policy may converge long before values do
Convergence*

- Define the max-norm: \( ||U|| = \max_s |U(s)| \)

- Theorem: For any two approximations \( U \) and \( V \)
  \[
  ||U^{t+1} - V^{t+1}|| \leq \gamma ||U^t - V^t||
  \]
  - i.e. any distinct approximations must get closer to each other, so, in particular, any approximation must get closer to the true \( U \) and value iteration converges to a unique, stable, optimal solution

- Theorem:
  \[
  ||U^{t+1} - U^t|| < \epsilon, \Rightarrow ||U^{t+1} - U|| < 2\epsilon \gamma / (1 - \gamma)
  \]
  - i.e. once the change in our approximation is small, it must also be close to correct

Utilities for a Fixed Policy

- Another basic operation: compute the utility of a state \( s \) under a fixed (generally non-optimal) policy

- Define the utility of a state \( s \), under a fixed policy \( \pi \):
  \[V^\pi(s) = \text{expected total discounted rewards (return) starting in } s \text{ and following } \pi\]

- Recursive relation (one-step look-ahead / Bellman equation):
  \[
  V^\pi(s) = \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V^\pi(s')]
  \]
Policy Evaluation

- How do we calculate the $V$'s for a fixed policy?
- Idea one: turn recursive equations into updates
  \[ V_0^\pi(s) = 0 \]
  \[ V_{i+1}^\pi(s) \leftarrow \sum_{s'} T(s, \pi(s), s') \left[ R(s, \pi(s), s') + \gamma V_i^\pi(s') \right] \]
- Idea two: it’s just a linear system, solve with Matlab (or whatever)

Policy Iteration

- Alternative approach:
  - **Step 1: Policy evaluation**: calculate utilities for some fixed policy (not optimal utilities!) until convergence
  - **Step 2: Policy improvement**: update policy using one-step look-ahead with resulting converged (but not optimal!) utilities as future values
  - Repeat steps until policy converges

- This is policy iteration
  - It’s still optimal!
  - Can converge faster under some conditions
Policy Iteration

- Policy evaluation: with fixed current policy $\pi$, find values with simplified Bellman updates:
  - Iterate until values converge

$$V_{i+1}^{\pi_i}(s) \leftarrow \sum_{s'} T(s, \pi_k(s), s') \left[ R(s, \pi_k(s), s') + \gamma V_i^{\pi_k}(s') \right]$$

- Policy improvement: with fixed utilities, find the best action according to one-step look-ahead

$$\pi_{k+1}(s) = \arg\max_a \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V_k(s') \right]$$

Comparison

- Both compute same thing (optimal values for all states)
- In value iteration:
  - Every pass (or “backup”) updates both utilities (explicitly, based on current utilities) and policy (implicitly, based on current utilities)
  - Tracking the policy isn’t necessary; we take the max

$$V_{i+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V_i(s') \right]$$

- In policy iteration:
  - Several passes to update utilities with fixed policy
  - After policy is evaluated, a new policy is chosen
  - Together, these are dynamic programming for MDPs
Asynchronous Value Iteration

- In value iteration, we update every state in each iteration.
- Actually, *any* sequences of Bellman updates will converge if every state is visited infinitely often.
- In fact, we can update the policy as seldom or often as we like, and we will still converge.
- Idea: Update states whose value we expect to change:
  If \(|V_{init}(s) - V_t(s)|\) is large then update predecessors of s.

Reinforcement Learning

- Reinforcement learning:
  - Still have an 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. don’t know which states are good or what the actions do
  - Must actually try actions and states out to learn
Example: Animal Learning

- RL studied experimentally for more than 60 years in psychology
  - Rewards: food, pain, hunger, drugs, etc.
  - Mechanisms and sophistication debated

- Example: foraging
  - Bees learn near-optimal foraging plan in field of artificial flowers with controlled nectar supplies
  - Bees have a direct neural connection from nectar intake measurement to motor planning area

Example: Backgammon

- Reward only for win / loss in terminal states, zero otherwise
- TD-Gammon learns a function approximation to \( V(s) \) using a neural network
- Combined with depth 3 search, one of the top 3 players in the world

- You could imagine training Pacman this way...

- … but it’s tricky! (It’s also P3)
Passive Learning

- **Simplified task**
  - You don’t know the transitions $T(s,a,s')$
  - You don’t know the rewards $R(s,a,s')$
  - You are given a policy $\pi(s)$
  - **Goal: learn the state values** (and maybe the model)
  - I.e., policy evaluation

- **In this case:**
  - Learner “along for the ride”
  - No choice about what actions to take
  - Just execute the policy and learn from experience
  - We’ll get to the active case soon
  - This is NOT offline planning!

---

Example: Direct Estimation

- **Episodes:**
  - (1,1) up -1
  - (1,2) up -1
  - (1,2) up -1
  - (1,3) right -1
  - (2,3) right -1
  - (3,3) right -1
  - (3,2) up -1
  - (3,3) right -1
  - (4,3) exit +100
  - (done)

\[
\gamma = 1, \ R = -1
\]

\[
V(1,1) \sim (92 + -106) / 2 = -7
\]

\[
V(3,3) \sim (99 + 97 + -102) / 3 = 31.3
\]
Model-Based Learning

- **Idea:**
  - Learn the model empirically (rather than values)
  - Solve the MDP as if the learned model were correct
  - Better than direct estimation?

- **Empirical model learning**
  - Simplest case:
    - Count outcomes for each s,a
    - Normalize to give estimate of $T(s,a,s')$
    - Discover $R(s,a,s')$ the first time we experience $(s,a,s')$
  - More complex learners are possible (e.g. if we know that all squares have related action outcomes, e.g. “stationary noise”)

---

Example: Model-Based Learning

- **Episodes:**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- $(1,1)$ up -1
- $(1,2)$ up -1
- $(1,2)$ up -1
- $(1,3)$ right -1
- $(2,3)$ right -1
- $(3,3)$ right -1
- $(3,2)$ up -1
- $(3,3)$ right -1
- $(4,3)$ exit +100
- $(4,3)$ exit +100

- $T(<3,3>, \text{ right, } <4,3>) = 1 / 3$
- $T(<2,3>, \text{ right, } <3,3>) = 2 / 2$
Recap: Model-Based Policy Evaluation

- Simplified Bellman updates to calculate $V$ for a fixed policy:
  - New $V$ is expected one-step-look-ahead using current $V$
  - Unfortunately, need $T$ and $R$

$$V^\pi_0(s) = 0$$

$$V^\pi_{i+1}(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^\pi_i(s')]$$

Sample Avg to Replace Expectation?

- Who needs $T$ and $R$? Approximate the expectation with samples (drawn from $T$!)

$$V^\pi_{i+1}(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^\pi_i(s')]$$

$$s_{sample_1} = R(s, \pi(s), s_1') + \gamma V^\pi_i(s_1')$$

$$s_{sample_2} = R(s, \pi(s), s_2') + \gamma V^\pi_i(s_2')$$

$$\ldots$$

$$s_{sample_k} = R(s, \pi(s), s_k') + \gamma V^\pi_i(s_k')$$

$$V^\pi_{i+1}(s) \leftarrow \sum_k s_{sample_k}$$
Model-Free Learning

- Big idea: why bother learning $T$?
  - Update $V$ each time we experience a transition
  - Frequent outcomes will contribute more updates (over time)

- Temporal difference learning (TD)
  - Policy still fixed!
  - Move values toward value of whatever successor occurs: running average!

\[
V^\pi(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^\pi(s')] \\
\text{sample} = R(s, \pi(s), s') + \gamma V^\pi(s') \\
V^\pi(s) \leftarrow (1 - \alpha)V^\pi(s) + (\alpha)\text{sample}
\]

Example: TD Policy Evaluation

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

(1,1) up -1  (1,1) up -1  
(1,2) up -1  (1,2) up -1  
(1,2) up -1  (1,3) right -1  
(1,3) right -1  (2,3) right -1  
(2,3) right -1  (3,3) right -1  
(3,3) right -1  (3,2) up -1  
(3,2) up -1  (4,2) exit -100  
(3,3) right -1  (done)  
(4,3) exit +100  (done)

Take $\gamma = 1$, $\alpha = 0.5$
Problems with TD Value Learning

- TD value leaning is model-free for policy evaluation (passive learning)
- However, if we want to turn our value estimates into a policy, we’re sunk:

\[
\pi(s) = \arg\max_a Q^*(s, a)
\]

\[
Q^*(s, a) = \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V^*(s') \right]
\]

- Idea: learn Q-values directly
- Makes action selection model-free too!