Announcements

- PS2 online now
  - Due on Wed. Autograder runs tonight and tomorrow.
  - Lydia / Luke office hour:
    - Tue 5-6 006 Lab
- Reading
  - two treatments of MDPs/RL
- Planning Research Opportunity
  - Contact Mausam if interested!
Outline (roughly next two weeks)

- Markov Decision Processes (MDPs)
  - MDP formalism
  - Value Iteration
  - Policy Iteration

- Reinforcement Learning (RL)
  - Relationship to MDPs
  - Several learning algorithms
Review: Expectimax

- What if we don’t know what the result of an action will be? E.g.,
  - In solitaire, next card is unknown
  - In minesweeper, mine locations
  - In pacman, the ghosts act randomly

- Can do **expectimax search**
  - Chance nodes, like min nodes, except the outcome is uncertain
  - Calculate expected utilities
  - Max nodes as in minimax search
  - Chance nodes take average (expectation) of value of children

- Today, we’ll learn how to formalize the underlying problem as a **Markov Decision Process**
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
Reinforcement Learning
Grid World

- The agent lives in a grid
- Walls block the agent’s path
- The agent’s actions do not always go as planned:
  - 80% of the time, the action North takes the agent North (if there is no wall there)
  - 10% of the time, North takes the agent West; 10% East
  - If there is a wall in the direction the agent would have been taken, the agent stays put
- Small “living” reward each step
- Big rewards come at the end
- Goal: maximize sum of rewards
Markov Decision Processes

- An MDP is defined by:
  - A set of states $s \in S$
  - A set of actions $a \in A$
  - A transition function $T(s,a,s')$
    - Prob that $a$ from $s$ leads to $s'$
    - i.e., $P(s' \mid s,a)$
    - Also called the model
  - A reward function $R(s, a, s')$
    - Sometimes just $R(s)$ or $R(s')$
  - A start state (or distribution)
  - Maybe a terminal state

- MDPs: non-deterministic search problems
  - Reinforcement learning: MDPs where we don’t know the transition or reward functions
What is Markov about MDPs?

- Andrey Markov (1856-1922)
- “Markov” generally means that given the present state, the future and the past are independent
- For Markov decision processes, “Markov” means:

\[ P(S_{t+1} = s'|S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1}, \ldots S_0 = s_0) = P(S_{t+1} = s'|S_t = s_t, A_t = a_t) \]
Solving MDPs

- In deterministic single-agent search problems, want an optimal plan, or sequence of actions, from start to a goal.

- In an MDP, we want an optimal policy $\pi^*$: $S \rightarrow A$
  - A policy $\pi$ gives an action for each state.
  - An optimal policy maximizes expected utility if followed.
  - Defines a reflex agent.

Optimal policy when $R(s, a, s') = -0.03$ for all non-terminals $s$. 
Example Optimal Policies

- $R(s) = -0.01$
- $R(s) = -0.03$
- $R(s) = -0.4$
- $R(s) = -2.0$
Example: High-Low

- Three card types: 2, 3, 4
- Infinite deck, twice as many 2’s
- Start with 3 showing
- After each card, you say “high” or “low”
- New card is flipped
- If you’re right, you win the points shown on the new card
- Ties are no-ops
- If you’re wrong, game ends

Differences from expectimax problems:
- #1: get rewards as you go
- #2: you might play forever!
High-Low as an MDP

- States: 2, 3, 4, done
- Actions: High, Low
- Model: \( T(s, a, s') \):
  - \( P(s'=4 \mid 4, \text{Low}) = 1/4 \)
  - \( P(s'=3 \mid 4, \text{Low}) = 1/4 \)
  - \( P(s'=2 \mid 4, \text{Low}) = 1/2 \)
  - \( P(s'=\text{done} \mid 4, \text{Low}) = 0 \)
  - \( P(s'=4 \mid 4, \text{High}) = 1/4 \)
  - \( P(s'=3 \mid 4, \text{High}) = 0 \)
  - \( P(s'=2 \mid 4, \text{High}) = 0 \)
  - \( P(s'=\text{done} \mid 4, \text{High}) = 3/4 \)
  - ...

- Rewards: \( R(s, a, s') \):
  - Number shown on \( s' \) if \( s \neq s' \)
  - 0 otherwise
Search Tree: High-Low

T = 0.5, R = 2
T = 0.25, R = 3
T = 0, R = 4
T = 0.25, R = 0
Each MDP state gives an expectimax-like search tree.

(s, a) is a q-state

(s, a, s') is a transition

T(s, a, s') = P(s'|s, a)
R(s, a, s')
Utilities of Sequences

- In order to formalize optimality of a policy, need to understand utilities of sequences of rewards.
- Typically consider stationary preferences:

\[ [r, r_0, r_1, r_2, \ldots] \succeq [r, r'_0, r'_1, r'_2, \ldots] \]
\[ \iff \]
\[ [r_0, r_1, r_2, \ldots] \succeq [r'_0, r'_1, r'_2, \ldots] \]

- Theorem: only two ways to define stationary utilities:
  - Additive utility:
    \[ U([r_0, r_1, r_2, \ldots]) = r_0 + r_1 + r_2 + \cdots \]
  - Discounted utility:
    \[ U([r_0, r_1, r_2, \ldots]) = r_0 + \gamma r_1 + \gamma^2 r_2 \cdots \]
Problem: infinite state sequences have infinite rewards

Solutions:

- Finite horizon:
  - Terminate episodes after a fixed $T$ steps (e.g. life)
  - Gives nonstationary policies ($\pi$ depends on time left)
- Absorbing state: guarantee that for every policy, a terminal state will eventually be reached (like “done” for High-Low)
- Discounting: for $0 < \gamma < 1$

$$U([r_0, \ldots r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_t \leq R_{\text{max}}/(1 - \gamma)$$

- Smaller $\gamma$ means smaller “horizon” – shorter term focus
Discounting

Typically discount rewards by $\gamma < 1$ each time step

- Sooner rewards have higher utility than later rewards
- Also helps the algorithms converge

\[ U([r_0, \ldots r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_t \leq R_{\text{max}}/(1 - \gamma) \]
Recap: Defining MDPs

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

- **MDP quantities so far:**
  - Policy = Choice of action for each state
  - Utility (or return) = sum of discounted rewards
Define the value of a state $s$: $V^*(s) = \text{expected utility starting in } s \text{ and acting optimally}$

Define the value of a q-state $(s,a)$: $Q^*(s,a) = \text{expected utility starting in } s, \text{ taking action } a \text{ and thereafter acting optimally}$

Define the optimal policy: $\pi^*(s) = \text{optimal action from state } s$
The Bellman Equations

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

- Formally:

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

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

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

Why not solve with expectimax?

Problems:
- This tree is usually infinite (why?)
- Same states appear over and over (why?)
- We would search once per state (why?)

Idea: Value iteration
- Compute optimal values for all states all at once using successive approximations
- Will be a bottom-up dynamic program similar in cost to memoization
- Do all planning offline, no replanning needed!
Value Estimates

- **Calculate estimates** $V^*_k(s)$
  - The optimal value considering only next $k$ time steps ($k$ rewards)
  - As $k \to \infty$, it approaches the optimal value

- **Why:**
  - If discounting, distant rewards become negligible
  - If terminal states reachable from everywhere, fraction of episodes not ending becomes negligible
  - Otherwise, can get infinite expected utility and then this approach actually won’t work
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 $i+1$:
    \[
    V_{i+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V_i(s') \right]
    \]
  - This is called a **value update** or **Bellman update**
  - Repeat until convergence

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

\[ V_{i+1}(s) = \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_i(s')] = \max_a Q_{i+1}(s, a) \]

\[ Q_1(\langle 3, 3 \rangle, \text{right}) = \sum_{s'} T(\langle 3, 3 \rangle, \text{right}, s') [R(\langle 3, 3 \rangle, \text{right}, s') + \gamma V_i(s')] \]

\[ = 0.8 \times [0.0 + 0.9 \times 1.0] + 0.1 \times [0.0 + 0.9 \times 0.0] + 0.1 \times [0.0 + 0.9 \times 0.0] \]
Example: Value Iteration

- Information propagates outward from terminal states and eventually all states have correct value estimates.
Example: Value Iteration

VALUES AFTER 0 ITERATIONS

```
0.00  0.00  0.00
0.00  0.00  0.00
0.00  0.00  0.00
```
Practice: Computing Actions

Which action should we chose from state $s$:

- Given optimal values $Q$?
  \[
  \arg \max_a Q^*(s, a)
  \]

- Given optimal values $V$?
  \[
  \arg \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')] 
  \]

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

\[
\| U^{t+1} - U^t \| < \epsilon, \implies \| U^{t+1} - U \| < 2\epsilon\gamma / (1 - \gamma)
\]

- I.e. once the change in our approximation is small, it must also be close to correct.
Value Iteration Complexity

- **Problem size:**
  - $|A|$ actions and $|S|$ states

- **Each Iteration**
  - Computation: $O(|A| \cdot |S|^2)$
  - Space: $O(|S|)$

- **Num of iterations**
  - Can be exponential in the discount factor $\gamma$
Utilities for Fixed Policies

- Another basic operation: compute the utility of a state $s$ under a fix (general 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: modify Bellman updates

\[ V^\pi_0(s) = 0 \]

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

- Idea two: it’s just a linear system, solve with Matlab (or whatever)
Policy Iteration

- **Problem with value iteration:**
  - Considering all actions each iteration is slow: takes $|A|$ times longer than policy evaluation
  - But policy doesn’t change each iteration, time wasted

- **Alternative to value iteration:**
  - **Step 1: Policy evaluation:** calculate utilities for a fixed policy (not optimal utilities!) until convergence (fast)
  - **Step 2: Policy improvement:** update policy using one-step lookahead with resulting converged (but not optimal!) utilities (slow but infrequent)
  - Repeat steps until policy converges
Policy Iteration

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

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

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

\[
\pi_{k+1}(s) = \arg \max_a \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V^{\pi_k}(s') \right]
\]
Policy Iteration Complexity

- **Problem size:**
  - $|A|$ actions and $|S|$ states

- **Each Iteration**
  - Computation: $O(|S|^3 + |A| \cdot |S|^2)$
  - Space: $O(|S|)$

- **Num of iterations**
  - Unknown, but can be faster in practice
  - Convergence is guaranteed
Comparison

- **In value iteration:**
  - Every pass (or “backup”) updates both utilities (explicitly, based on current utilities) and policy (possibly implicitly, based on current policy)

- **In policy iteration:**
  - Several passes to update utilities with frozen policy
  - Occasional passes to update policies

- **Hybrid approaches (asynchronous policy iteration):**
  - Any sequences of partial updates to either policy entries or utilities will converge if every state is visited infinitely often