Feedback

- The projects are a bit tricky and/or vague.
- The lectures move too fast.

Last Time

- Adversarial search
  - Consider a kind of search tree where each node represents a state of the game and each edge represents an action taken from a state.
  - The value of a node is the best achievable reward (assuming optimal play)
  - Minimax – assume optimal opponent
  - Expectimax – assume random opponent
- You will program this in Project 2!

Today

- More about utilities
- Markov decision processes (MDPs)
- Solving MDPs: Value iteration

There will be math.

Maximizing Expected Utility

- Principle of maximum expected utility:
  - A rational agent should choose the action that maximizes its expected utility, given its knowledge
- Last time:
  - Utility expresses relative values of different outcomes.
  - Utilities are defined so that expected utilities make sense.
- Today:
  - More details about utilities, where they come from, and why they make sense.

Preferences

- An agent chooses among:
  - Outcomes: $A, B$, etc.
  - Lotteries: situations with uncertain prizes
  - $L = [p, A; (1 − p), B]$
- Notation:
  - $A \succ B$ \quad A preferred over $B$
  - $A \sim B$ \quad indifference between $A$ and $B$
  - $A \succeq B$ \quad $B$ not preferred over $A$
Rational Preferences

- We want some constraints on preferences before we call them rational.

\[(A > B) \land (B > C) \Rightarrow (A > C)\]

- For example: an agent with intransitive preferences can be induced to give away all of its money.
  - If \(B > C\), then an agent with \(C\) would pay (say) 1 cent to get \(B\).
  - If \(A > B\), then an agent with \(B\) would pay (say) 1 cent to get \(A\).
  - If \(C > A\), then an agent with \(A\) would pay (say) 1 cent to get \(C\).

Preferences of a rational agent must obey constraints.

- The axioms of rationality:
  - Orderability: \((A > B) \lor (B > A) \lor (A \sim B)\)
  - Transitivity: \((A > B) \land (B > C) \Rightarrow (A > C)\)
  - Continuity: \(A > B > C \Rightarrow 3p [A, \ 1 - p, C] \sim B\)

Theorem: Rational preferences imply behavior describable as maximization of expected utility.

MEU Principle

- Theorem: [Ramsey, 1931; von Neumann & Morgenstern, 1944]
  - Given any preferences satisfying these axioms, there exists a real-valued function \(U\) such that:
    \[U(A) \geq U(B) \iff A \succeq B\]
    \[U(p_1, S_1; \ldots; p_n, S_n) = \sum_i p_i U(S_i)\]

- Maximum expected utility (MEU) principle:
  - Choose the action that maximizes expected utility
  - Note: an agent can be entirely rational (consistent with MEU) without ever representing or manipulating utilities and probabilities – this is about behavior
  - E.g., a lookup table for perfect tic-tac-toe

Utility Scales

- Normalized utilities: \(u_i = 1.0, u_0 = 0.0\)
- Micromorts: one-millionth chance of death, useful for paying to reduce product risks, etc. Worth about $50.
- QALYs: quality-adjusted life years, useful for medical decisions involving substantial risk
- Note: behavior is invariant under positive linear transformation
  \[U'(x) = k_1 U(x) + k_2 \quad \text{where} \quad k_2 > 0\]
- With deterministic prizes only (no lottery choices), only ordinal utility can be determined, i.e., total order on prizes

Money

- Money does not behave as a utility function, but we can talk about the utility of having money (or being in debt)
- Given a lottery \(L = [p, $X; (1-p), $Y]\)
  - The expected monetary value \(EMV(L) = pX + (1-p)Y\)
  - \(U(L) = pU($X) + (1-p)U($Y)\)
  - Typically, \(U(L) < U(EMV(L))\): people prefer a sure thing
  - In this sense, people are risk-averse
  - When deep in debt (or in Vegas), we are risk-prone
- Utility curve: for what probability \(p\) am I indifferent between:
  - Some sure outcome \(x\)
  - A lottery \([p, $M; (1-p), $0]\)

Are Humans Rational?

- A: [0.8, $4k ; 0.2, $0]
- B: [1.0, $3k ; 0.0, $0]
- C: [0.2, $4k ; 0.8, $0]
- D: [0.25, $3k ; 0.75, $0]
- Famous example of Allais (1953)
- Most people prefer \(B > A, C > D\)
- But if \(U(0) = 0\), then
  - \(B > A \Rightarrow U($3k) > 0.8 U($4k)\)
  - \(C > D \Rightarrow 0.8 U($4k) > U($3k)\)

One explanation: people don’t want to feel regret.
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
- Big rewards come at the end

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., \( P(s' | s, a) \)
  - 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 are a family of non-deterministic search problems

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

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

Example Optimal Policies

- 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

Example: High-Low

- 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
- Start: 3

Search Tree: High-Low

MDP Search Trees

- Each MDP state gives an expectimax-like search tree

Utilities of Sequences

- In order to formalize optimality of a policy, need to understand utilities of sequences of rewards
- Typically consider stationary preferences:
  \[ [r, r_0, r_1, r_2, ...] > [r, r_0', r_1', r_2', ...] \]
  \[ [r_0, r_1, r_2, ...] > [r_0', r_1', r_2', ...] \]
- Theorem: only two ways to define stationary utilities
  - Additive utility:
    \[ U([r_0, r_1, r_2, ...]) = r_0 + r_1 + r_2 + ... \]
  - Discounted utility:
    \[ U([r_0, r_1, r_2, ...]) = r_0 + \gamma r_1 + \gamma^2 r_2 + ... \]

Infinite Utilities?!

- 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 \( 0 < \gamma < 1 \)
    \[ U([r_0, r_1, r_2, ...]) = \sum_{i=0}^{\infty} \gamma^i r_i \leq R_{\text{max}}/(1 - \gamma) \]
    - 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
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

Optimal Utilities

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

\[
V_{t+1}(s) = \max_a \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V_t(s') \right] = \max_a Q_{t+1}(s, a)
\]

\[
Q_{t+1}(3, 3, \text{right}) = \sum_{s'} T(3, 3, \text{right}, s') \left[ R(3, 3, \text{right}, s') + \gamma V_t(s') \right] = 0.8 \times [0.6 + 0.9 \times 1.0] + 0.1 \times [0.3 + 0.9 \times 0.0] + 0.1 \times [0.0 + 0.9 \times 0.0]
\]

**Example: Value Iteration**

- Information propagates outward from terminal states and eventually all states have correct value estimates.

**Example: Value Iteration**

**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 value and value iteration converges to a unique, stable, optimal solution.

- Theorem:
  \[
  \|U^{t+1} - U^t\| < \epsilon \Rightarrow \|U^{t+1} - U^t\| < 2\epsilon/(1 - \gamma)
  \]
  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 fixed (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 \( V^\pi \)'s for a fixed policy?
- Idea one: modify Bellman updates
  \[ V^\pi_0(s) = \mathcal{C} \]
  \[ V^\pi_{i+1}(s) = \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V^\pi_i(s')] \]
- 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 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
**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

**Diagram:**
- Agent
- Environment
- State $s_t$
- Reward $r_t$
- Action $a_t$

Reinforcement Learning

**Image:**
- Outdoor setting with a vehicle in a controlled environment.