Announcements

- Midterm: 02/12
- Written assignment 2
  - Deadline: 02/16
- Programming project 2
  - Deadline: 02/18
Non-Deterministic Search
Example: Grid World

- A maze-like problem
  - The agent lives in a grid
  - Walls block the agent’s path

- Noisy movement: 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

- The agent receives rewards each time step
  - Small “living” reward each step (can be negative)
  - Big rewards come at the end (good or bad)

- Goal: maximize sum of rewards
Grid World Actions

Deterministic Grid World

Stochastic Grid World
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 \( T(s, a, s') \)
    - Probability that \( a \) from \( s \) leads to \( s' \), i.e., \( P(s' | s, a) \)
    - Also called the model or the dynamics
  - A reward function \( R(s, a, s') \)
    - Sometimes just \( R(s) \) or \( R(s') \)
  - A start state
  - Maybe a terminal state

- MDPs are non-deterministic search problems
  - One way to solve them is with expectimax search
  - We’ll have a new tool soon
What is Markov about MDPs?

- “Markov” generally means that given the present state, the future and the past are independent.

- For Markov decision processes, “Markov” means action outcomes depend only on the current state.

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

- This is just like search, where the successor function could only depend on the current state (not the history).

Andrey Markov (1856-1922)
In deterministic single-agent search problems, we wanted an optimal plan, or sequence of actions, from start to a goal.

For MDPs, we want an optimal policy $\pi^*$: $S \rightarrow A$
- A policy $\pi$ gives an action for each state
- An optimal policy is one that maximizes expected utility if followed
- An explicit policy defines a reflex agent

Expectimax didn’t compute entire policies
- It computed the action for a single state only

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

R(s) = -0.01

R(s) = -0.03

R(s) = -0.4

R(s) = -2.0
Example: Racing
Example: Racing

- A robot car wants to travel far, quickly
- Three states: Cool, Warm, Overheated
- Two actions: Slow, Fast
- Going faster gets double reward
MDP Search Trees

- Each MDP state projects an expectimax-like search tree

(s, a) is a q-state

(s, a, s') called a transition

T(s, a, s') = P(s' | s, a)

R(s, a, s')
Utilities of Sequences
Utilities of Sequences

- What preferences should an agent have over reward sequences?

- More or less? [1, 2, 2] or [2, 3, 4]

- Now or later? [0, 0, 1] or [1, 0, 0]
Discounting

- It’s reasonable to maximize the sum of rewards
- It’s also reasonable to prefer rewards now to rewards later
- One solution: values of rewards decay exponentially

Worth Now

\[ 1 \]

Worth Next Step

\[ \gamma \]

Worth In Two Steps

\[ \gamma^2 \]
Discounting

- **How to discount?**
  - Each time we descend a level, we multiply in the discount once

- **Why discount?**
  - Sooner rewards probably do have higher utility than later rewards
  - Also helps our algorithms converge

- **Example: discount of 0.5**
  - $U([1,2,3]) = 1*1 + 0.5*2 + 0.25*3$
  - $U([1,2,3]) < U([3,2,1])$
Stationary Preferences

- Theorem: if we assume stationary preferences:

\[
[a_1, a_2, \ldots] \succ [b_1, b_2, \ldots] \\
\iff \\
[r, a_1, a_2, \ldots] \succ [r, b_1, b_2, \ldots]
\]

- Then: there are only two ways to define utilities
  - Additive utility: \( U([r_0, r_1, r_2, \ldots]) = r_0 + r_1 + r_2 + \cdots \)
  - Discounted utility: \( U([r_0, r_1, r_2, \ldots]) = r_0 + \gamma r_1 + \gamma^2 r_2 \cdots \)
Quiz: Discounting

- **Given:**
  
  - Actions: East, West, and Exit (only available in exit states a, e)
  - Transitions: deterministic

- **Quiz 1:** For $\gamma = 1$, what is the optimal policy?

- **Quiz 2:** For $\gamma = 0.1$, what is the optimal policy?

- **Quiz 3:** For which $\gamma$ are West and East equally good when in state d?
Infinite Utilities?! 

- **Problem:** What if the game lasts forever? Do we get infinite rewards?

- **Solutions:**
  - Finite horizon: (similar to depth-limited search)
    - Terminate episodes after a fixed $T$ steps (e.g. life)
    - Gives nonstationary policies ($\pi$ depends on time left)
  
  - Discounting: use $0 < \gamma < 1$
    
    $$U([r_0, \ldots r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_t \leq R_{\text{max}}/(1 - \gamma)$$
    
    - Smaller $\gamma$ means smaller “horizon” – shorter term focus

- **Absorbing state:** guarantee that for every policy, a terminal state will eventually be reached (like “overheated” for racing)
Recap: Defining MDPs

- **Markov decision processes:**
  - Set of states $S$
  - Start state $s_0$
  - Set of 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 = sum of (discounted) rewards
Solving MDPs
Optimal Quantities

- The value (utility) of a state $s$:
  \[ V^*(s) = \text{expected utility starting in } s \text{ and acting optimally} \]

- The value (utility) of a q-state $(s,a)$:
  \[ Q^*(s,a) = \text{expected utility starting out having taken action } a \text{ from state } s \text{ and (thereafter) acting optimally} \]

- The optimal policy:
  \[ \pi^*(s) = \text{optimal action from state } s \]
Snapshot of Demo – Gridworld V Values

VALUES AFTER 100 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0
Snapshot of Demo – Gridworld Q Values

Q-VALUES AFTER 100 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0
Values of States

- Fundamental operation: compute the (expectimax) value of a state
  - Expected utility under optimal action
  - Average sum of (discounted) rewards
  - This is just what expectimax computed!

- Recursive definition of value:

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

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

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

- We’re doing way too much work with expectimax!

- Problem: States are repeated
  - Idea: Only compute needed quantities once

- Problem: Tree goes on forever
  - Idea: Do a depth-limited computation, but with increasing depths until change is small
  - Note: deep parts of the tree eventually don’t matter if $\gamma < 1$
Time-Limited Values

- Key idea: time-limited values

- Define $V_k(s)$ to be the optimal value of $s$ if the game ends in $k$ more time steps
  - Equivalently, it’s what a depth-$k$ expectimax would give from $s$
$k=0$

VALUES AFTER 0 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0
$k=1$

VALUES AFTER 1 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0
k=2

Noise = 0.2
Discount = 0.9
Living reward = 0

VALUES AFTER 2 ITERATIONS
$k=3$

VALUES AFTER 3 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0
$k = 4$

VALUES AFTER 4 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0
\( k = 5 \)

VALUES AFTER 5 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0
$k = 6$

VALUES AFTER 6 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0
$k=7$

Values after 7 iterations:

- Top row: 0.62, 0.74, 0.85, 1.00
- Middle row: 0.50, 0.57, -1.00
- Bottom row: 0.34, 0.36, 0.45, 0.24

Additional parameters:
- Noise = 0.2
- Discount = 0.9
- Living reward = 0
$k = 8$

VALUES AFTER 8 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0
$k=9$

VALUES AFTER 9 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0
$k=10$

VALUES AFTER 10 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0
$k=11$

VALUES AFTER 11 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0
$k = 12$

VALUES AFTER 12 ITERATIONS

0.64 → 0.74 → 0.85 → 1.00

0.57 ▲ 0.57 ▲ -1.00

0.49 ▲ 0.42 ▲ 0.47 ▲ 0.28

Noise = 0.2  
Discount = 0.9  
Living reward = 0
$k = 100$

VALUES AFTER 100 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0
Computing Time-Limited Values

$V_0(\_\_\_) \quad V_1(\_\_\_) \quad V_2(\_\_\_) \quad V_3(\_\_\_) \quad V_4(\_\_\_) \quad V_4(\_\_\_) \quad V_4(\_\_\_)$
Value Iteration
Value Iteration

- Start with $V_0(s) = 0$: no time steps left means an expected reward sum of zero

- Given vector of $V_k(s)$ values, do one ply of expectimax from each state:

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

- Repeat until convergence

- Complexity of each iteration: $O(S^2A)$

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

\[ V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V_k(s') \right] \]
Convergence*

- How do we know the $V_k$ vectors are going to converge?

- Case 1: If the tree has maximum depth $M$, then $V_M$ holds the actual untruncated values

- Case 2: If the discount is less than 1
  - Sketch: For any state $V_k$ and $V_{k+1}$ can be viewed as depth $k+1$ expectimax results in nearly identical search trees
  - The difference is that on the bottom layer, $V_{k+1}$ has actual rewards while $V_k$ has zeros
  - That last layer is at best all $R_{\text{MAX}}$
  - It is at worst $R_{\text{MIN}}$
  - But everything is discounted by $\gamma^k$ that far out
  - So $V_k$ and $V_{k+1}$ are at most $\gamma^k \max |R|$ different
  - So as $k$ increases, the values converge