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

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
  - 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 an action leads to a state
    - 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
- 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:
  \[
  P(S_{t+1} = s' | S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1} = a_{t-1}, \ldots, S_0 = s_0)
  =
  \]
  \[
  P(S_{t+1} = s' | S_t = s_t, A_t = a_t)
  \]
  - First order Markov

Announcements

- Assignment 1 graded
- Come and see me after class if you have questions
**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**

<table>
<thead>
<tr>
<th>R(s)</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.01</td>
<td>2</td>
</tr>
<tr>
<td>-0.03</td>
<td>0</td>
</tr>
<tr>
<td>-0.4</td>
<td>4</td>
</tr>
<tr>
<td>-2.0</td>
<td>4</td>
</tr>
</tbody>
</table>

**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.

**High-Low as an MDP**

- States: 2, 3, 4, done.
- Actions: High, Low.
- Model: \( T(s, a, s') \):
  - \( P(s'=4 \mid 4, \text{Low}) = 1/4 \)
  - \( P(s'=3 \mid 4, \text{Low}) = 1/4 \)
  - \( P(s'=2 \mid 4, \text{Low}) = 3/4 \)
  - \( P(s'=\text{done} \mid 4, \text{Low}) = 0 \)
  - \( P(s'=4 \mid 4, \text{High}) = 1/4 \)
  - \( P(s'=3 \mid 4, \text{High}) = 0 \)
  - \( P(s'=2 \mid 4, \text{High}) = 0 \)
  - \( P(s'=\text{done} \mid 4, \text{High}) = 3/4 \)
- Rewards: \( R(s, a, s') \):
  - Number shown on \( s' \) if \( s \neq s' \).
  - 0 otherwise.
- Start: 3.

**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:
  \[[r, r_0, r_1, r_2, \ldots] \Rightarrow [r, r_0, r_1, r_2, \ldots]\]
  \[[r_0, r_1, r_2, \ldots] \Rightarrow [r_0, r_1, r_2, \ldots]\]
- 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:
    \[U([r_0, r_1, r_2, \ldots]) = r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots\]

Infinite Utilities?!

- Problem: infinite state sequences have infinite rewards.
- Solutions:
  - Finite horizon:
    Terminating episodes after a fixed \(T\) steps (e.g., life).
  - Absorbing state: guarantee that for every policy, a terminal state will eventually be reached (like "done" for High-Low).
  - Discounting: for \(0 < \gamma < 1\)
    \[U([r_0, \ldots, r_n]) = \sum_{i=0}^{\infty} \gamma^i r_i \leq R_{\text{max}}/(1 - \gamma)\]
    - 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

- Fundamental operation: compute the values (optimal expectations utilities) of states \(s\).
- Why? Optimal values define optimal policies.
- 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:
  - Optimal rewards = maximize over first action and then follow optimal policy
- Formally:
  \[V^*(s) = \max_a Q^*(s,a)\]
  \[Q^*(s,a) = \sum_{s'} T(s,a,s') \left[ R(s,a,s') + \gamma V^*(s') \right]\]
  \[V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[ R(s,a,s') + \gamma V^*(s') \right]\]
Solving MDPs

- We want to find the optimal policy $\pi^*$
- Proposal 1: modified expectimax search, starting from each state $s$:
  \[
  \pi^*(s) = \arg\max_a Q^*(s, a) \\
  Q^*(s, a) = \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')] \\
  V^*(s) = \max_a Q^*(s, a)
  \]

Why Not Search Trees?

- Why not solve with expectimax?
- Problems:
  - This tree is usually infinite
  - Same states appear over and over
  - We would search once per state
- 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 $V_k(s)$
  - 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
- 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

Memoized Recursion?

- Recurrences:
  \[
  V_k^*(s) = 0 \\
  V_k^*(s) = \max_a Q_k^*(s, a) \\
  Q_k^*(s, a) = \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_{k-1}^*(s')] \\
  \pi_k^*(s) = \arg\max_a Q_k^*(s, a)
  \]
  - 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 $i+1$:
  \[
  V_{i+1}^*(s) = \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_i(s')] \\
  \]
  - 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
**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:**
  - Returns = sum of discounted rewards
  - Values = expected future returns from a state (optimal, or for a fixed policy)
  - Q-values = expected future returns from a q-state (optimal, or for a fixed policy)

**Example: Bellman Updates**

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

**Example: Value Iteration**

\[
V_2(3.3) = \sum_{s'} T(3.3, \text{right}, s') \left[ R(3.3) + \gamma V_1(s') \right] = 0.9 \left[ 0.8 \cdot 1 + 0.1 \cdot 0 + 0.1 \cdot 0 \right]
\]

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

**Practice: Computing Actions**

- Which action should we chose from state \( s \):
  - Given optimal values \( V^* \)
    \[ \arg \max_a \sum_{s'} T(s,a,s') [R(s,a,s') + \gamma V^*(s')] \]
  - Given optimal q-values \( Q^* \)
    \[ \arg \max_a Q^*(s,a) \]
- Lesson: actions are easier to select from \( Q^* \)

**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's for a fixed policy?
- Idea one: modify Bellman updates
  \[ V_{t+1}^\pi(s) = 0 \]
  \[ V_{t+1}^\pi(s) = \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V_t^\pi(s')] \]
- 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 look-ahead with resulting converged (but not optimal!) utilities (slow but infrequent)
  - 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 π, find values with simplified Bellman updates:
  - Iterate until values converge

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