Recap: MDPs

- Markov decision processes:
  - States $S$
  - Actions $A$
  - Transitions $P(s'|s,a)$ (or $T(s,a,s')$)
  - Rewards $R(s,a,s')$
  - 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)

MDP Search Trees

- Each MDP state gives an expectimax-like search tree

Optimal Utilities

- Fundamental operation: compute the optimal utilities of states $s$ (all at once)
- Why? Optimal values define optimal policies!
- Define the utility of a state $s$: $V(s) = \text{expected return starting in } s$ and acting optimally
- Define the utility of a q-state $(s,a)$: $Q(s) = \text{expected return starting in } s$, taking action $a$ and thereafter acting optimally
- Define the optimal policy: $\pi(s) = \text{optimal action from state } s$

The Bellman Equations

- Definition of utility leads to a simple relationship amongst optimal utility values:
  - Optimal rewards = maximize over first action and then follow optimal policy
- Formally:
  \[
  V^*(s) = \max_a Q^*(s,a) \\
  Q^*(s,a) = \sum_{s'} T(s,a,s') [R(s,a,s') + \gamma V^*(s')] \\
  V^*(s) = \max_a \sum_{s'} T(s,a,s') [R(s,a,s') + \gamma V^*(s')] 
  \]

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's!
Why Not Search Trees?

- Why not solve with expectimax?
- Problems:
  - This tree is usually infinite (why?)
  - The same states appear over and over (why?)
  - There's actually one tree per state (why?)
- Idea: Value iteration
  - Compute optimal values 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 \( V_n(s) \)
  - 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 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: Value Iteration

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

Example: Bellman Updates

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

\[
V_{i+1}(3, 3) = \sum_{s'} T(3, 3, \text{right}, s') \left[ R(3, 3) + 0.9 V_i(s') \right] = 0.9 \times [0.8 \cdot 1 + 0.1 \cdot 0 + 0.1 \cdot 0]
\]

Convergence*

- Define the max-norm: \( |U| = \max_s |U(s)| \)
- Theorem: For any two approximations \( U \) and \( V \)
  \[
  |U_{i+1} - V_{i+1}| \leq \gamma |U_i - V_i|
  \]
  - I.e. any distinct approximations must get closer to the true \( U \) and value iteration converges to a unique, stable, optimal solution
- Theorem:
  \[
  |U_{i+1} - U_i| < \epsilon \Rightarrow |U_{i+1} - U_i| < 2\epsilon/(1 - \gamma)
  \]
  - I.e. once the change in our approximation is small, it must also be close to correct
Policy Iteration

- Alternative approach:
  - Step 1: Policy evaluation: calculate utilities for a fixed policy (not optimal utilities!) until convergence
  - Step 2: Policy improvement: update policy based on resulting converged (but not optimal!) utilities
  - Repeat steps until policy converges

- This is policy iteration
  - 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
  \[
  V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s, \pi_k(s), s') \left[ R(s, \pi_k(s), s') + \gamma V_k^\pi(s') \right]
  \]

- Policy improvement: with fixed utilities, find the best action according to one-step look-ahead
  \[
  \pi_{k+1}(s) = \arg \max_a \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V_k^\pi(s') \right]
  \]

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

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!
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)

- In this case:
  - No choice about what actions to take
  - Just execute the policy and learn from experience
  - We’ll get to the general case soon

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
  - \((3,3)\) right -1
  - \((2,3)\) right -1
  - \((2,3)\) right -1
  - \((1,2)\) up -1
  - \((1,2)\) up -1
  - \((1,1)\) up -1

  \( \gamma = 1, R = -1 \)
  \( U(1,1) \approx (92 + -106) / 2 = -7 \)
  \( U(3,3) \approx (99 + 97 + -102) / 3 = 31.3 \)

Model-Based Learning

- Idea:
  - Learn the model empirically (rather than values)
  - Solve the MDP as if the learned model were correct

- 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:
  - \((1,1)\) 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
  - \((3,3)\) right -1
  - \((2,3)\) right -1
  - \((2,3)\) right -1
  - \((1,2)\) up -1
  - \((1,2)\) up -1
  - \((1,1)\) up -1

  \( \gamma = 1 \)
  \( T(<3,3>, \text{right}, <4,3>) = 1 / 3 \)
  \( T(<2,3>, \text{right}, <3,3>) = 2 / 2 \)
  \( U(1,1) = (92 + -106) / 2 = -7 \)
  \( U(3,3) = (99 + 97 + -102) / 3 = 31.3 \)

Model-Based Learning

- In general, want to learn the optimal policy, not evaluate a fixed policy

- Idea: adaptive dynamic programming
  - Learn an initial model of the environment:
  - Solve for the optimal policy for this model (value or policy iteration)
  - Refine model through experience and repeat
  - Crucial: we have to make sure we actually learn about all of the model

Example: Greedy ADP

- Imagine we find the lower path to the good exit first
- Some states will never be visited following this policy from \((1,1)\)
- We’ll keep re-using this policy because following it never collects the regions of the model we need to learn the optimal policy
What Went Wrong?

- Problem with following optimal policy for current model:
  - Never learn about better regions of the space if current policy neglects them

- Fundamental tradeoff: exploration vs. exploitation
  - Exploration: must take actions with suboptimal estimates to discover new rewards and increase eventual utility
  - Exploitation: once the true optimal policy is learned, exploration reduces utility
  - Systems must explore in the beginning and exploit in the limit

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

\[ V^\pi(s) \leftarrow V^\pi(s) + \alpha \sum \text{sample} \]

Example: Passive TD

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

Problems with TD Value Learning

- TD value learning is model-free for policy evaluation
- However, if we want to turn our value estimates into a policy, we're sunk:
  \[ \pi(s) = \arg \max_a Q^\pi(s, a) \]
  \[ Q^\pi(s, a) = \sum_s T(s, a, s') \left[ R(s, a, s') + \gamma V^\pi(s') \right] \]

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