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:
  - Policy = map of states to actions
  - Episode = one run of an MDP
  - Utility = sum of discounted rewards
  - Values = expected future utility from a state
  - Q-Values = expected future utility from a q-state

Recap: Optimal Utilities

- The utility of a state $s$: $V^*(s) = \text{expected utility starting in } s \text{ and acting optimally}$
- The utility of a q-state $(s,a)$: $Q^*(s,a) = \text{expected utility starting in } s, \text{ taking action } a \text{ and thereafter acting optimally}$
- The optimal policy: $\pi^*(s) = \text{optimal action from state } s$

Recap: Bellman Equations

- Definition of utility leads to a simple one-step lookahead relationship amongst optimal utility values:
  - Total optimal rewards = maximize over choice of (first action plus optimal future)

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

Practice: Computing Actions

- Which action should we chose from state $s$:
  - Given optimal values $V$?
    $$\arg\max_a \sum_{s'} T(s,a,s') \left[ R(s,a,s') + \gamma V^*(s') \right]$$
  - Given optimal q-values $Q$?
    $$\arg\max_a Q^*(s,a)$$
- Lesson: actions are easier to select from Q’s!
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
- Almost solution: recursion (i.e. expectimax)
- Correct solution: dynamic programming

Memoized Recursion?

- Recurrences (basically truncated expectimax):
  \[
  V_k^*(s) = \max\{Q_k^*(s, a) \}
 \]
  \[
  Q_k^*(s, a) = \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V_{k-1}^*(s') \right]
  \]
  \[
  V_0^*(s) = 0
  \]
  \[
  \pi_i(s) = \arg \max_{a} Q_i^*(s, a)
  \]
- 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

Utilities for a Fixed Policy

- Another basic operation: compute the utility of a state $s$ under a fixed (generally 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):
  \[
  V^\pi(s) = \sum_{s'} T(s, \pi(s), s') \left[ R(s, \pi(s), s') + \gamma V^\pi(s') \right]
  \]
Policy Evaluation

- How do we calculate the V’s for a fixed policy?
- Idea one: turn recursive equations into updates
  \[ V_0(s) = 0 \]
  \[ V_{i+1}(s) = \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V_i(s')] \]
- Idea two: it's just a linear system, solve with Matlab (or whatever)

Policy Iteration

- Alternative approach:
  - Step 1: Policy evaluation: calculate utilities for some fixed policy (not optimal utilities!) until convergence
  - Step 2: Policy improvement: update policy using one-step look-ahead with resulting converged (but not optimal!) utilities as future values
  - 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 \( \pi \), find values with simplified Bellman updates:
  \[ V_{i+1}(s) \leftarrow \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V_i(s')] \]
  - Policy improvement: with fixed utilities, find the best action according to one-step look-ahead
  \[ \pi_{i+1}(s) = \arg \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_i(s')] \]

Comparison

- Both compute same thing (optimal values for all states)
- In value iteration:
  - Every pass (or "backup") updates both utilities (explicitly, based on current utilities) and policy (implicitly, based on current utilities)
  - Tracking the policy isn’t necessary; we take the max
  \[ V_{i+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_i(s')] \]
- In policy iteration:
  - Several passes to update utilities with fixed policy
  - After policy is evaluated, a new policy is chosen
  - Together, these are dynamic programming for MDPs

Asynchronous Value Iteration*

- In value iteration, we update every state in each iteration
- Actually, any sequences of Bellman updates will converge if every state is visited infinitely often
- In fact, we can update the policy as seldom or often as we like, and we will still converge
- Idea: Update states whose value we expect to change: if \( |P_{\pi(s)} - V_i(s)| \) is large then update predecessors of \( s \)

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

Goal: learn the state values

This is NOT offline planning!

Just execute the policy and learn from experience

You are given a policy \( \pi(s) \)

Bees learn near-optimal foraging plan in field of artificial flowers with controlled nectar supplies

Rewards: food, pain, hunger, drugs, etc.

Example: Direct Estimation

\[
\begin{align*}
V(3,3) & \approx (99 + 97 - 102) / 3 = 31.3 \\
V(1,1) & \approx (92 - 106) / 2 = -7
\end{align*}
\]

Count outcomes for each \( (s,a) \) and solve the MDP as if the learned model were correct

Learn the model empirically (rather than values)

Example: 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)
  - I.e., policy evaluation

In this case:

- Learner “along for the ride”
- No choice about what actions to take
- Just execute the policy and learn from experience
- We’ll get to the active case soon
- This is NOT offline planning!

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! (It’s also P3)

Example: Model-Based Learning

- Idea:
  - Learn the model empirically (rather than values)
  - Solve the MDP as if the learned model were correct
  - Better than direct estimation?
- 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")

Episodes:

\[
\begin{align*}
T(<2,3>, \text{right}, <3,3>) & = 1 / 3 \\
T(<2,3>, \text{right}, <3,3>) & = 2 / 2
\end{align*}
\]
Recap: Model-Based Policy Evaluation

- Simplified Bellman updates to calculate \( V \) for a fixed policy:
  - New \( V \) is expected one-step-look-ahead using current \( V \)
  - Unfortunately, need \( T \) and \( R \)

\[
V^0(s) = 0 \\
V^i(s) = \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V^{i-1}(s')] 
\]

Sample Avg to Replace Expectation?

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

- Who needs \( T \) and \( R \)? Approximate the expectation with samples (drawn from \( T! \))

\[
\text{sample}_1 = R(s, \pi(s), s_1') + \gamma V^0(s_1') \\
\text{sample}_2 = R(s, \pi(s), s_2') + \gamma V^0(s_2') \\
\cdots \\
\text{sample}_k = R(s, \pi(s), s_k') + \gamma V^0(s_k') \\
V^i(s) = \left( \sum_k \text{sample}_k \right) / k
\]

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: running average!

\[
V^\pi(s) = \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V^\pi(s')] \\
\text{sample} = R(s, \pi(s), s') + \gamma V^\pi(s') \\
V^\pi(s) = (1 - \alpha) V^\pi(s) + \alpha \text{sample} 
\]

Example: TD Policy Evaluation

\[
V^\pi(s) = (1 - \alpha) V^\pi(s) + \alpha \left[ R(s, \pi(s), s') + \gamma V^\pi(s') \right] 
\]

Problems with TD Value Learning

- TD value learning is model-free for policy evaluation (passive learning)
- However, if we want to turn our value estimates into a policy, we’re sunk:

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

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