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 $\pi$ - map of states to actions
  - Utility $U$ - sum of discounted rewards
  - Values $V$ - expected future utility from a state (max node)
  - Q-Values $Q$ - expected future utility from a q-state (chance node)

Policy Extraction

Computing Actions from Values

- Let's imagine we have the optimal values $V^*(s)$
- How should we act?
  - It's not obvious!
- We need to do a mini-expectimax (one step)
- This is called policy extraction, since it gets the policy implied by the values

Computing Actions from Q-Values

- Let's imagine we have the optimal q-values:
- How should we act?
  - Completely trivial to decide!
- Important lesson: actions are easier to select from q-values than values!

Policy Methods

Problems with Value Iteration

- Value iteration repeats the Bellman updates:
  $$V_{k+1}(s) := \max_a \sum_{s'} T(s,a,s') \left[ R(s,a,s') + \gamma V_k(s') \right]$$
- Problem 1: It's slow - $O(S^2A)$ per iteration
- Problem 2: The “max” at each state rarely changes
- Problem 3: The policy often converges long before the values

[Demos: value iteration (L9D2)]
Policy Iteration

Alternative approach for optimal values:
- 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 (policy extraction) 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 (much) faster under some conditions

Policy Evaluation

Do the optimal action
Do what π says to do

Fixed Policies

Expectimax trees max over all actions to compute the optimal values
- If we fixed some policy π(s), then the tree would be simpler – only one action per state
- ... though the tree’s value would depend on which policy we fixed

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 π:

\[ V^\pi(s) = \text{expected total discounted rewards starting in } s \text{ and following } \pi \]

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 \( \pi \)?**
  - Idea 1: Turn recursive Bellman equations into updates (like value iteration)
    
    \[
    V_{\pi}^0(s) = 0 \\
    V_{\pi+1}^0(s) = \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V_{\pi}^0(s')] 
    \]
  - Efficiency: \( O(S^2) \) per iteration
  - Idea 2: Without the maxes, the Bellman equations are just a linear system
    
    Solve with Matlab (or your favorite linear system solver)

### Policy Iteration

- **Evaluation:** For fixed current policy \( \pi \), find values with policy evaluation:
  - Iterate until values converge:
    
    \[
    V_{\pi+1}^0(s) = \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V_{\pi}^0(s')] 
    \]
  - Improvement: For fixed values, get a better policy using policy extraction
    
    One-step look-ahead:
    
    \[
    \pi_{t+1}(s) = \arg \max_a T(s, a, s') [R(s, a, s') + \gamma V_{\pi}^0(s')] 
    \]

### Comparison

- Both value iteration and policy iteration compute the same thing (all optimal values)
- In value iteration:
  - Every iteration updates both the values and (implicitly) the policy
  - We don’t track the policy, but taking the max over actions implicitly recomputes it
- In policy iteration:
  - We do several passes that update utilities with fixed policy (each pass is fast because we consider only one action, not all of them)
  - After the policy is evaluated, a new policy is chosen (slow like a value iteration pass)
  - The new policy will be better (or we’re done)
- Both are dynamic programs for solving MDPs

### Summary: MDP Algorithms

- **So you want to...**
  - Compute optimal values: use value iteration or policy iteration
  - Compute values for a particular policy: use policy evaluation
  - Turn your values into a policy: use policy extraction (one-step lookahead)
- These all look the same!
  - They basically are · they are all variations of Bellman updates
  - They all use one-step lookahead expectimax fragments
  - They differ only in whether we plug in a fixed policy or max over actions