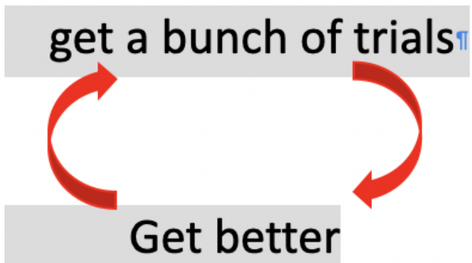
CAS CS 440
Lec 30

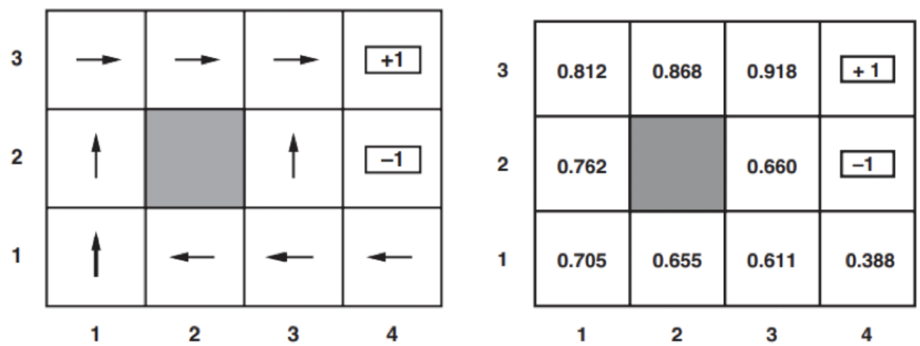
Policy Learning IV: Reinforcement Learning I

1. Last Time
    a. Value Iteration
        i. Calculate U(s) for each state s
            1. Stores the map itself and the utility values
        ii. Use utilities to select optimal action in each state
            1. Once we have the true utilities, we can make the policy from it
        iii. Bellman is a contraction function → converges to optimal solution!
            1. Reason this works
            2. The first iteration may calculate utilities but it may not be good
            3. However, second iteration is better → third iteration is even better → so on
    b. Policy Iteration
        i. Start with some initial policy (can be chosen at random)
            1. If I know the policy, I can measure the goodness of that action
        ii. Policy tells you want to do at particular state (want to do the action that leads to the most favorable state)
            1. Action to take is that leads to the best average value since I may not go to places where I intended
        iii. Alternate between two steps:
            1. Policy evaluation:
                a. Given π_i , calculate $u_i = U^{\pi_i}$
                    i. Calculate utilities from the policies
            2. Policy improvement:
                a. Calculate new MEU policy π_{i+1} using one-step lookahead (bellman equation) from u_i
        iv. Terminate when policy improvement → no changes in utility values
            1. Utility values converge between iterations
2. Today: Learning (from Examples)
    a. The problem:
        i. Value Iteration & Policy Iteration require:
            1. (advanced) knowledge of transition model
            2. (advanced) knowledge of reward function
            3. We need to know the transition probabilities and reward functions for every state beforehand

b. What if we don't know these? (i.e. world is "black box")?
c. The plan:



   i. Let our policy explore the world and record it → repeat this process

3. How?
   a. 

   b. For now, let us fix our policy (blindly follow the policy)
      i. Whatever the policy π is, agent always follows policy
      ii. If it is optimal, good
      iii. If not, still follow blindly
      iv. Skill issue: get a good policy!
   c. Get a bunch of trials
      i. Record state & reward each time
      ii. Keep going until terminal state is reached (do that for many trials)
         1. The path may not be the same due to the stochastic nature
      iii. Goal: Learn the expected utility $U^{\pi}(s)$

$$U^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(S_t)\right]$$

4. How to Learn $U^{\pi}(s)$
   a. Method 1: Direct Utility Estimation (do not need to know the transition model)
      i. Idea: $U^{\pi}(s)$ is expected (total) reward for rest of the trajectory
      ii. Called expected "reward to go"

iii. Each trial provides a sample (for each state visited)

1. Keep running average for $\hat{U}^{\pi}(s)$
2. $\lim_{n \to \infty} \hat{U}^{\pi}(s) = U^{\pi}(s)$

b. Consider this trajectory:

i. $(1,1)_{-.04} \rightsquigarrow (1,2)_{-.04} \rightsquigarrow (1,3)_{-.04} \rightsquigarrow (1,2)_{-.04} \rightsquigarrow (1,3)_{-.04} \rightsquigarrow (2,3)_{-.04} \rightsquigarrow (3,3)_{-.04} \rightsquigarrow (4,3)_{+1}$

ii. 1 sample for a trajectory starting at (1,1) (total reward = 0.72)

iii. 2 samples for a trajectory starting at (1,2) (total rewards = 0.76, 0.84)

1. Starting at s(1,2) has average of 0.8

iv. 2 samples for (1,3) (total rewards = 0.8, 0.88)

1. Starting at s(1,3) has average of 0.84

v. Continue updating these averages as you play more games

vi. …

c.

(1,1) samples! $(1,1)_{-.04} \rightsquigarrow (1,2)_{-.04} \rightsquigarrow (1,3)_{-.04} \rightsquigarrow (1,2)_{-.04} \rightsquigarrow (1,3)_{-.04} \rightsquigarrow (2,3)_{-.04} \rightsquigarrow (3,3)_{-.04} \rightsquigarrow (4,3)_{+1}$

(1,2) samples!
$(1,2)_{-.04} \rightsquigarrow (1,3)_{-.04} \rightsquigarrow (1,2)_{-.04} \rightsquigarrow (1,3)_{-.04} \rightsquigarrow (2,3)_{-.04} \rightsquigarrow (3,3)_{-.04} \rightsquigarrow (4,3)_{+1}$
$(1,2)_{-.04} \rightsquigarrow (1,3)_{-.04} \rightsquigarrow (2,3)_{-.04} \rightsquigarrow (3,3)_{-.04} \rightsquigarrow (4,3)_{+1}$

(1,3) samples!
$(1,3)_{-.04} \rightsquigarrow (1,2)_{-.04} \rightsquigarrow (1,3)_{-.04} \rightsquigarrow (2,3)_{-.04} \rightsquigarrow (3,3)_{-.04} \rightsquigarrow (4,3)_{+1}$
$(1,3)_{-.04} \rightsquigarrow (2,3)_{-.04} \rightsquigarrow (3,3)_{-.04} \rightsquigarrow (4,3)_{+1}$

5. Problem with Direct Utility Estimation

a. Utilities are not independent of each other

i. A more accurate representation of utility dependencies:

$$U^{\pi}(s) = R(s) + \gamma \sum_{s'} \Pr\left[s' \middle| s, a\right] U^{\pi}(s')$$

→ Bellman Equation

1. The utility of the state is reward plus the average utility of the action I am going to pick (weighted average)

b. Consider this trial:

$(1,1)_{-.04} \rightsquigarrow (1,2)_{-.04} \rightsquigarrow (1,3)_{-.04} \rightsquigarrow (1,2)_{-.04} \rightsquigarrow (1,3)_{-.04} \rightsquigarrow (2,3)_{-.04} \rightsquigarrow (3,3)_{-.04} \rightsquigarrow (4,3)_{+1}$
$(1,1)_{-.04} \rightsquigarrow (1,2)_{-.04} \rightsquigarrow (1,3)_{-.04} \rightsquigarrow (2,3)_{-.04} \rightsquigarrow (3,3)_{-.04} \rightsquigarrow (3,2)_{-.04} \rightsquigarrow (3,3)_{-.04} \rightsquigarrow (4,3)_{+1}$

c. When we reach (3,2): brand new state

i. Great! Opportunity for learning!

ii. Reaches (3,3): a good state (learned from prior trial)

1. Since (3,3) is reachable from (3,2)…shouldn't (3,2) be a good state (i.e. high utility)?

iii. Problem: Direct Utility Estimation doesn't learn anything until after the trial is done! → have to wait until I see the entire trajectory

            iv. Another problem: Relies on the sample from being good (policy I choose may not be optimal)
                1. Areas of the world I never explore is sensitive to the random policy
                2. Requires magically to get a good initial random policy
    d. Consider all function our model can learn
    e. Really, <u>really</u>, <u>**really**</u> big space (called a "hypothesis space")
        i. Consider: n boolean variables, 1 output (true or false)
            1. Goal: learn the function from data (m examples)
            2. How many functions could exist?
                a. 2^n combinations of variables (x or not x)
                b. Function output t/f (for every assignment)
                c. Therefore, 2^(2^n) functions
                d. Choosing one from them is very risky since most of them are bad
    f. Direct Utility Estimation:
        i. Considers a hypothesis space which is bigger than it needs to be
        ii. Harder to learn!
        iii. Converges really slowly!
6. How to Learn $U^{\pi}(s)$
    a. Method 2: Adaptive Dynamic Programming (ADP)
        i. Uses constraints (i.e. dependencies) between state utilities
        ii. Learn the transition model!
            1. Then solve the MDP (using modified policy iteration, etc)
    b. How to learn transition model?
        i. Fully observable world:
            1. Record (s, a, s') triple and learn the relationship $(s, a) \rightarrow s'$
                a. For example, keep pmf for each state $\Pr[s' | s, a]$
                    i. count frequencies!

7. Passive ADP

function PASSIVE-ADP-AGENT($percept$) **returns** an action
inputs: $percept$, a percept indicating the current state s' and reward signal r'
Global variables **persistent**: π, a fixed policy
mdp, an MDP with model P, rewards R, discount γ
U, a table of utilities, initially empty
N_{sa}, a table of frequencies for state–action pairs, initially zero
$N_{s'|sa}$, a table of outcome frequencies given state–action pairs, initially zero
s, a, the previous state and action, initially null

if s' is new **then** $U[s'] \leftarrow r'$; $R[s'] \leftarrow r'$
if s is not null **then**
increment $N_{sa}[s,a]$ and $N_{s'|sa}[s',s,a]$
for each t such that $N_{s'|sa}[t,s,a]$ is nonzero **do**
$P(t \mid s,a) \leftarrow N_{s'|sa}[t,s,a] \,/\, N_{sa}[s,a]$ Estimate transition model
$U \leftarrow$ POLICY-EVALUATION(π, U, mdp) Make utilities from updated transition model!
if s'.TERMINAL? **then** $s, a \leftarrow$ null **else** $s, a \leftarrow s', \pi[s']$
return a

a.

8. ADP
   a. Bad for large state spaces (same limitation as policy/value iteration)
      i. Why? Too many equations to solve
   b. Another big problem:
      i. The frequencies are sensitive!
         1. For example, keep pmf for each state $\Pr[s'|s,a]$
            a. Count frequencies
         2. What if we don't "explore" some areas of the map! We won't have good probs!
            a. Transition probabilities will not get there (requires non-zero transition probabilities)
         3. Counting frequencies on their own is too aggressive a strategy