Harvard CS 182: Artificial Intelligence November 3, 2022

Section 9: Markov decision processes and Reinforcement Learning

Lecturer: Ariel Procaccia *Authors: Mujin Kwun and Catherine Cui*

Markov Decision Processes

So far, we have considered search problems where state transitions were deterministic. However, this assumption is often unrealistic in practice when we might have to model randomness where taking an action from a given state might lead to a set of different possible successor states.
Markov Decision Processes (MDP) construction:

- S: a set of states
- s_0: an initial state
- A(s): A set of actions for each state $s \in S$
- $P(s'|s, a)$: A transition model that gives us the probability of reaching state s' if action a is taken from state s
- R(s): A reward function that gives us the reward for state s

We assume that the agent's utility is the sum of rewards for now.

A policy π specifies action $\pi(s) \in A(s)$ for each state s. We are interested in finding the optimal policy $\pi*$ that maximizes the expected utility.

Discounted Utilities

Most often, we assume that there is an **infinite horizon** (the process goes on forever,) and that we have a **discount factor** $\gamma \in [0, 1]$ such that for an infinite sequence of states $s_0, s_1, s_2...$, the utility is $R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2)...$

This is ideal for because the "math looks nicer" and because there are many real-world cases where it makes intuitive sense to weight immediate rewards more heavily than future rewards.

Let $r^* \geq R(s)$, $\forall s \in S$. Then if $\gamma < 1$, the utility is bounded by:

$$\sum_{t=0}^{\infty} \gamma^t r^* = \frac{r^*}{1-\gamma}$$

Bellman Equations

For any state s, we can find the utility given by the optimal policy, U(s) using the Bellman equations:

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a) * U(s')$$

Once we have found U(s) for all $s \in S$, we can find the optimal policy by taking:

$$\pi^*(s) \in argmax_{a \in A(s)} \sum_{s'} P(s'|s,a) * U(s')$$

Value Iteration

In the Bellman equations above, we see that our values for U(s) depend on one another. How can we compute the utility of the optimal policy and use this to find our optimal policy? To do so, we start with an initial function U and iteratively update our utility estimates as follows:

$$U_{i+1}(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a) * U_i(s')$$

Given some acceptable $\epsilon > 0$, our stopping condition is:

$$\max_{s \in S} |U_{i+1}(s) + U_i(s)| < \frac{\epsilon(1-\gamma)}{\gamma}$$

Thm: If we choose our stopping condition above, with $\gamma < 1$, value iteration terminates with utility estimates U_t such that:

$$|U(s) - U_t(s)| < \epsilon$$

Policy Iteration

Alternatively, we can use a different algorithm, policy iteration, to iteratively compute policies directly. Beginning with initial policy π_0:

Step 1: Policy Evaluation Given policy π_i we can calculate its utility U_i

$$U_i(s) = R(s) + \gamma \sum_{s'} P(s'|s,\pi_i(s)) * U_i(s')$$

Step 2: Policy Improvement Using this U_i, calculate an updated policy π_{i+1}

$$\pi_{i+1}(s) \in argmax_{a \in A(s)} \sum_{s'} P(s'|s,a) * U_i(s')$$

We repeat these steps until there is no change in utilities.

Linear Programming

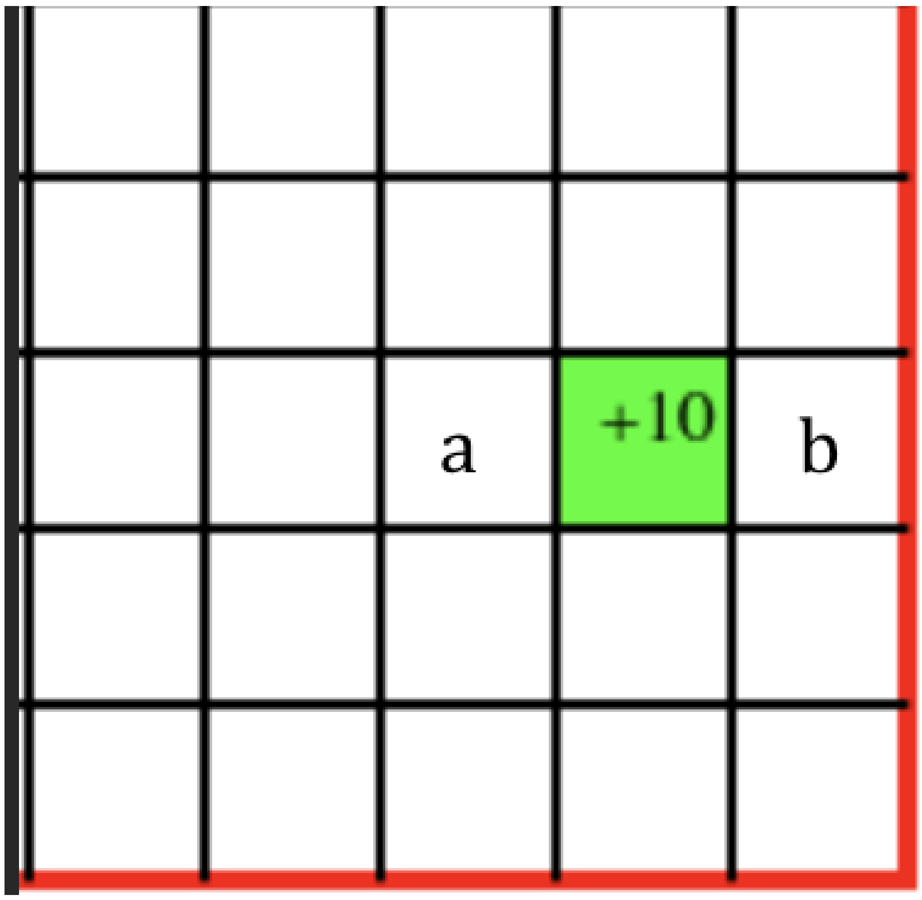
Finally, we can also solve for the optimal utility program by framing it as a linear programming problem.

$$\min_{s \in S} U(s)$$

$$s.t. \forall s \in S, a \in A(s)$$

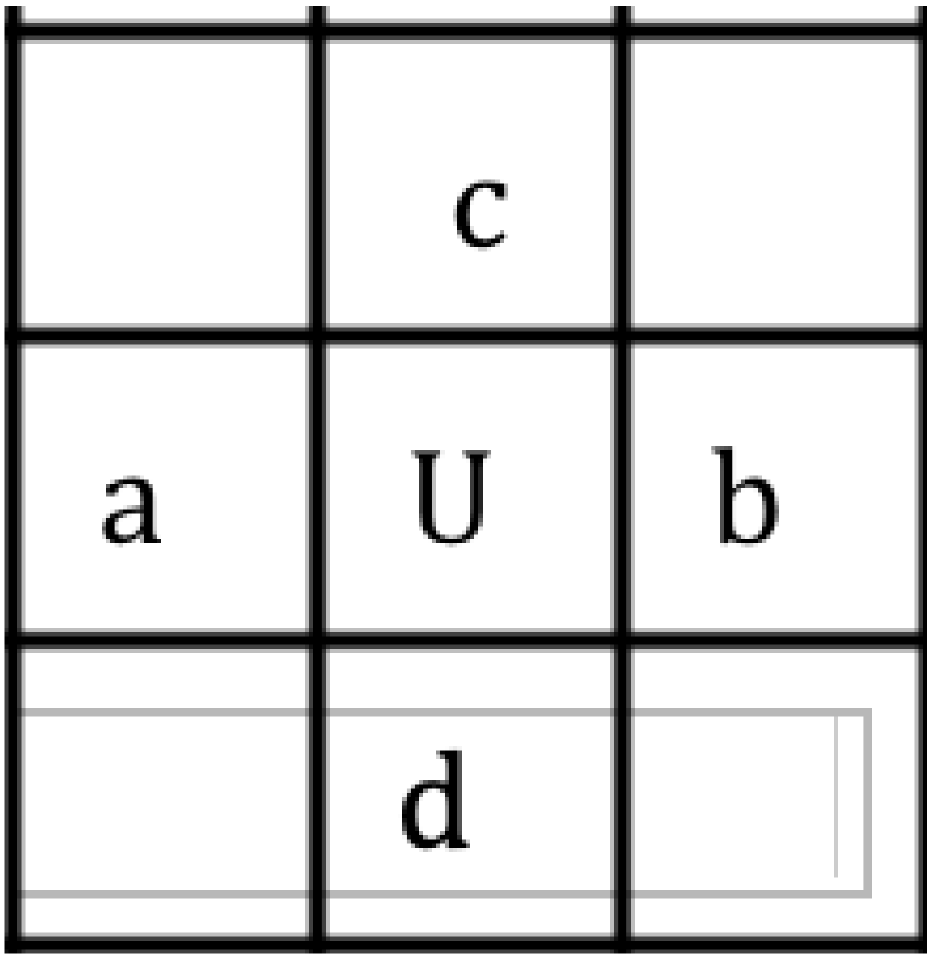
$$U(s) \geq R(s) + \gamma \sum_{s'} P(s'|s,a) * U(s')$$

However, this often yields a running time larger than that of Value and Policy iteration, so in most cases, it is not as practical



Problem 1 *Consider the grid world above where the agent can choose to move up, down, left, or right. Assume that if an agent chooses to carry out an action, there is a 0.7 chance it performs this desired action and 0.1 chance it performs each of the other three. There is one square with a reward of +10. If the agent moves into the red wall, it receives a reward of -1.*

1. *What is the value for a and b after one round of value iteration, assuming a discount factor of 0.9 and initial rewards of 0 for all non-labelled states?*
2. *If we choose an ϵ of 0.1, what is our stopping condition?*
3. *Considering the grid below with labelled states a,b,c,d and a policy that specifies moving up from the middle square, s', write out the equation for policy evaluation at the middle state in terms of U_a, U_b, U_c, U_d. Assume that $R(s') = 0.5$*



4. *Explain why Policy Iteration is guaranteed to converge.*

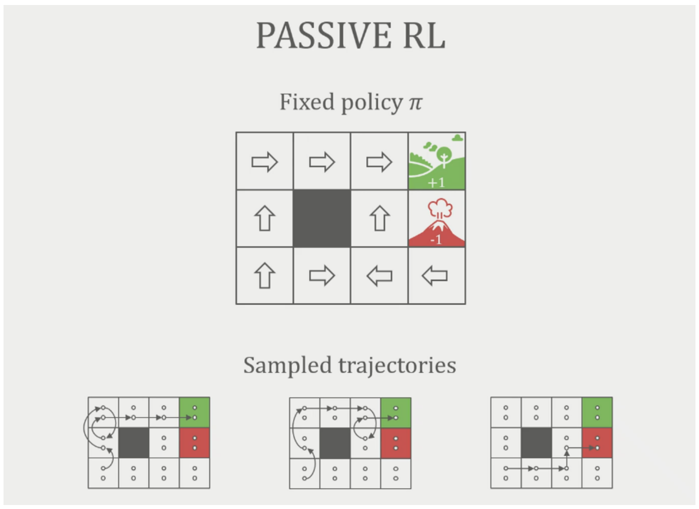
5. *Prove that policy iteration converges at the optimal policy and value function*

Reinforcement Learning

Reinforcement learning refers to getting feedback through rewards. Note that this differs from MDP: MDP also have rewards, but the environment is known. In contrast, reinforcement learning focuses on learning to act in an unknown environment, which is represented as an MDP with unknown transitions and rewards. Reinforcement learning can either be passive or active, model-based or model-free.

Passive Reinforcement Learning

In passive reinforcement learning, we're given a fixed policy π, and we want to find the utility of π. To do this, we sample many possible trajectories by following π many times and use this information to estimate the utility of π.



PASSIVE RL

Fixed policy π

Sampled trajectories

Depicted above is an example of a fixed policy π and examples of possible sampled trajectories.

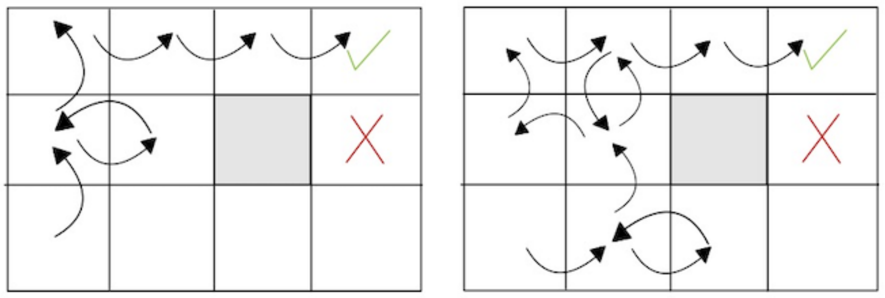
Model-based, Passive RL

The goal is to learn an estimate $\hat{U}(s)$ of the utility $U^{\pi}(s)$ of each state s for the fixed policy π where $U^{\pi}(s)$ is the utility under π if we start from state s and follow π.

We observe the reward $R(s)$ when we visit a state s and we estimate the transition function $\hat{P}(s'|s,\pi(s))$ from s to s' by observing how many times state s' was reached when taking action $\pi(s)$ in s and normalizing. Using these values, we compute $\hat{U}$ for all $s \in S$:

$$\hat{U}(s) = R(s) + \gamma \sum_{s' \in S} \hat{P}(s'|s,\pi(s)) \cdot \hat{U}(s')$$

Problem 2 *Below, we have two sampled trajectories based on some fixed policy. What are the estimated transition probabilities? Suppose we start in the bottom left square.*



Model-free, Passive RL (Temporal-Difference Learning)

Similarly to the model-based passive RL, we observe the fixed policy π acting in the world. Each time we transition from s to s', update $\hat{U}(s)$ in the following way:

$$\hat{U}(s) \leftarrow (1-\alpha)\hat{U}(s) + \alpha\left(R(s) + \gamma\hat{U}(s')\right)$$

Note that we are essentially updating $\hat{U}(s)$ with a convex combination of the new estimate and old estimate. The learning rate α is a function of the number of times state s is visited, k_s. As k_s increases, α should decrease because we become more confident in $\hat{U}(s)$ and thus give less weight to the new observation. When α decreases appropriately as k_s increases, $\hat{U}$ converges to U^π.

Active Reinforcement Learning

In active reinforcement learning, our goal is to find a food policy in an unknown environment by observing what happens as we act in the world. Similarly to passive reinforcement learning, we have model-based and mode-free.

Model-based, Active RL

For now, we take a random action at each step. We estimate $\hat{P}(s'|s,a)$ by observing how many times s' was reached when taking action a in s and normalizing. Using these values, we solve the Bellman equations for each $s \in S$ to achieve the optimal policy:

$$\hat{U}(s) = R(s) + \gamma \max_{a\in A(s)} \sum_{s'\in S} \hat{P}(s'|s,a)\cdot\hat{U}(s')$$

Problem 3 *Using the equations above, we want to derive an optimal policy. How can we do this?*

Model-free, Active RL (Q-learning)

Instead of estimating the utility for each state, $\hat{U}(s)$, we'll instead estimate $Q(s,a)$ for each state-action pair and for all $s \in S$, we choose the action

$$\pi(s) \in \max_{a\in A_s} \text{argmax} Q(s,a)$$

In equilibrium, optimal Q values satisfy Bellman-like equations for all $s \in S, a \in A_s$:

$$Q(s,a) = R(s) + \gamma \sum_{s'\in S} P(s'|s,a) \max_{a'\in A_{s'}} Q(s',a')$$

Instead of following a fixed policy π, take a random action at each step. Each time we transition from s to s' through action a update $Q(s,a)$ in the following way:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\left(R(s) + \gamma \max_{a'} Q(s',a')\right)$$

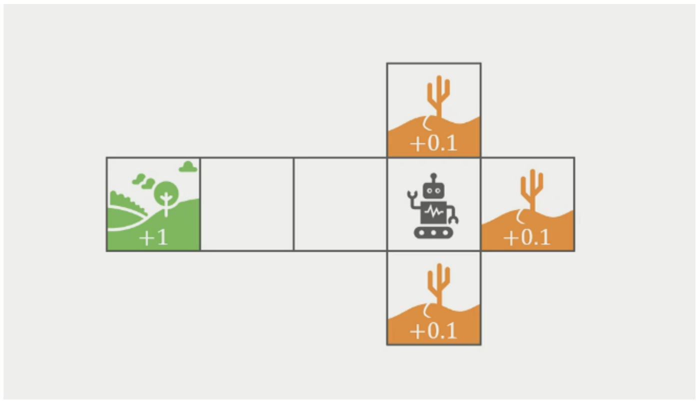
where α once again is the learning rate. Now, α depends on the number of takes action a was taken in state s.

Exploration vs. Exploitation

So far, we assume we take a random action at every stage. This is nice because we will reach every state and give us information about the transition function. In other words, our Q-learning agent currently only explores. What if this Q-learning only exploited and always selected the action $\text{argmax}_a Q(s,a)$?

In that case, we may not find "larger" or riskier rewards: example below.

Example 1 *This is an example where if we only exploited, we may end up going to the desert every time and miss out on the larger rewards to the further left.*



So what should we do? We need to maintain a balance between exploring and exploiting. To satisfy the exploration requirement and make sure that all state action pairs are visited infinitely often, there are two methods we discuss in this course:

1. ϵ-exploration: Use $\text{argmax}_a Q(s,a)$ with probability $1-\epsilon$ and a random action wirh probability ϵ
2. Softmax: Choose each action with probability

$$\frac{e^{Q(s,a)/\theta}}{\sum_{a' \in A_s} e^{Q(s,a')/\theta}}$$

If all state action pairs are visited infinitely often and $\alpha(k_{sa})$ goes to zero appropriately, then Q-learning converges to an optimal policy!