Lecture 4: Defining the RL Problem (2 of 2)

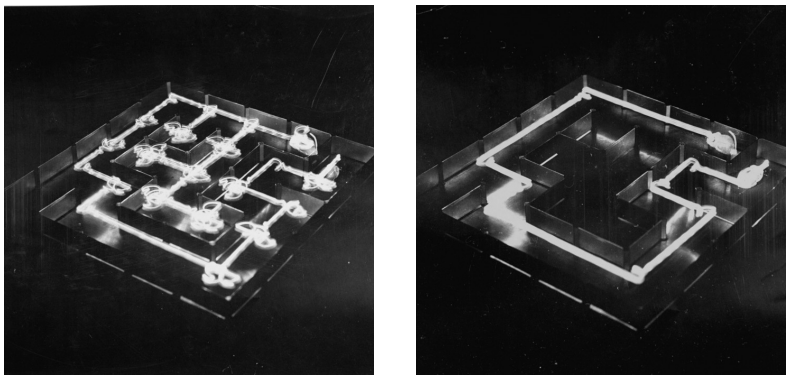

Figure 1: Mechanical RL agent, circa 1950. *(Left)* Path taken when learning to solve the maze. *(Right)* Path taken after learning to solve the maze. Video.

1 Reviewing the RL Problem

The RL problem is defined in terms of observations (also known as "states") and actions. Think of the observations as the inputs and the actions as the outputs. We will use s_t to denote the observation at time t, and a_t as the action at time t. Recall that the RL problem is *sequential*, so we need to consider not just one but many states and actions, one after the next.

After taking an action, we observe a reward r_t as well as the next state s_{t+1}. The reward is computed with an *unknown* function $r_t = r(s_t, a_t)$. The next state is sampled from an *unknown* dynamics function $p(s_{t+1} \mid s_t, a_t)$. This point bears repeating: both the reward function and the dynamics function are unknown. This is a key difference between other planning algorithms (RRT, A*, Dijkstra's).

The aim of the agent is to maximize the sum of all the returns. In other words, the aim of the agent isn't just to choose the action that results in the best outcome right now, but also puts you in the best spot to succeed in the future.

1.1 The Discount Factor

There are two final steps away from formally defining the RL problem. First, when looking at the sum of rewards, we have to decide how much weight to put on the rewards at different time steps. Do you care more about getting high rewards now or in the future? Would you rather get a high grade in freshman or senior year? In RL, the standard assumption is that you care about the rewards at every time step, but assign a slightly higher weight to the rewards in the near-term future. Formally, we encode this with a **discount** parameter $\gamma \in [0, 1)$. The weight associated with the reward t time steps into the future is γ^t.

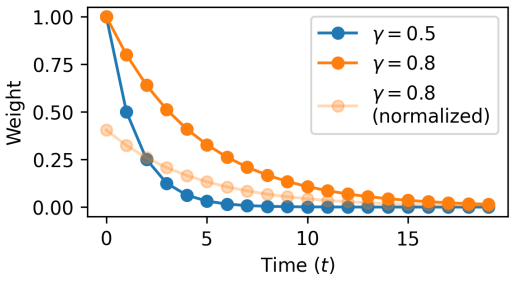

Figure 2

We visualize these weights in Fig. 2. Intuitively, these weights assign higher weights to near-term rewards and lower weights to long-term rewards. We can then express the RL

objective as the *discounted* cumulative sum of rewards:

$$\mathcal{F}(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right]. \tag{1}$$

You might have noticed that these weights resemble a geometric sequence, or a geometric distribution. In fact, we can make this connection precise by thinking about *randomly* sampling the time step at which you care about the reward. We can chat more about this after class if folks are interested.

Now, let's consider the effect of γ on the RL problem. For example, compare the orange and blue lines in Fig. 2.

One thing you might have noticed is that increasing γ increases the weights on *every* time step, in addition to changing the relative weight on different time steps. One way of accounting for this is to normalize the weights by the sum of these weights. However, it turns out that we can ignore such normalization. Why? We'll discuss this later in class. This explains why we defined $0 <= \gamma < 1$.

1.2 Expectation

To finish our definition of the RL problem (the MDP), we focus on the expectation over which we are computing the discounted cumulative sum. How are we sampling these states and actions?

To explain this, we need to explain how experience is collected in the RL problem. We assume that experience is organized into **episodes** or trials. Without each episode, we start at some initial state and action ($t = 0$) and go from there.[1] You can think of an episode as one game of Monopoly, or one attempt that at Roomba makes at cleaning a house.

Each episode starts with an initial state s_0. In some problems (e.g., Monopoly), the initial state is fixed – every episode starts with the same state. In other problems, the initial state is randomized, sampled from an **initial state distribution** $s_0 \sim p_0(s_0)$ (e.g., your Roomba starts at a random place in your house; the card game war).

After this initial state, and at every subsequent step, the action a_t is sampled from the policy $\pi(a_t \mid s_t)$.[2] This policy is the thing that we are trying to learn. Like in other areas of machine learning, we can think about π as a mapping from inputs (observations) to outputs (actions). The states are sampled from the (unknown) transition function $s_{t+1} \sim p(s_{t+1} \mid s_t, a_t)$.[3] The notation in this section is sometimes (though not always) included when writing the RL objective:

$$\mathcal{F}(\pi) = \mathbb{E}_{\substack{s_0 \sim p_0(s_0), a_t \sim \pi(a_t|s_t) \\ s_{t+1} \sim p(s_{t+1}|s_t, a_t)}}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right] \tag{2}$$

Some times in this course we may want to talk about the entire sequence of states and actions. We will use $\tau = (s_0, a_0, s_1, a_1, \cdots)$ to refer to the **trajectory**.

1.3 Estimating the RL objective

How to we estimate and (ultimately) optimize this RL objective? We will not try to enumerate all possible trajectories and look at the probability and return associated with each – that would just take way, way too long. Instead, we will make use of **Monte Carlo** sampling, sampling k trajectories and looking at the average return:

$$\mathcal{F}(\pi) \approx \frac{1}{k}\sum_{i=0}^{k-1} R(\tau) \quad \text{where } \tau \sim \pi \tag{3}$$

[1] We'll try to stick with zero-indexing because it will reduce notational clutter later in the course.

[2] In this course, we use the Greek letter "π" (pronounced "pi") to denote this conditional probability distribution, not to refer to the constant $3.14\cdots$.

[3] The transition function is also referred to as the dynamics function, the transition kernel, the model, and the system dynamics.

The homework this week will look at implement this to estimate the expected return associated with different policies.

1.4 Horizon.

One thing you might have noticed above is that we're summing over an *infinite* number of time steps. This is mostly for mathematical convenience. In practice, of course, we have to stop at some point. But, note that because $\gamma^t \to 0$ for large t, this truncation will have little effect.

How should you set the discount γ? As a rule of thumb, using a discount factor of γ corresponds to reasoning $\frac{1}{1-\gamma}$ steps into the future. For example, a discount of $\gamma = 0.5$ corresponds to planning for the next 2 steps, while a discount of $\gamma = 0.99$ corresponds to planning for the next 100 steps. Of course, this is just a heuristic. How did we get this? Recall from before that γ^t looks like the PDF of a geometric distribution with parameter γ, and such a distribution has an expected value of $\frac{1}{1-\gamma}$.

2 MDP transformations

The discussion above provides one way of defining the RL problem. To better understand this problem statement, we will discuss a few variations of this problem, showing that many variants are actually one and the same. Working through these examples should help build intuition into what the RL problem is, and highlight how it can be applied to a wide range of problems, including those that (on the surface) seem to not exactly accord to the MDP specification.

Fixed initial state. While we defined the MDP above using an initial state sampled $s_0 \sim p_0(s_0)$ from an initial state distribution, some problems have a fixed initial state. Given an MDP with a fixed initial state, can you convert it into another MDP with a full initial state distribution? What about the opposite, taking an MDP with a (stochastic) initial state distribution and converting it into an MDP with a singled fixed initial state?

Terminating states. The discussion above assumed that episodes last forever; that they are an infinite number of steps long. However, many practical problems terminate after a fixed number of steps. How might you represent one of these problems using the MDP formalism described above? Make sure that the transition dynamics remain Markovian.

Reward transformation. Given an MDP with reward function $r(s, a)$ and another with reward function $r(s, a) + b$ where $b \in \mathbb{R}$, what is the relationship between the expected return of a policy π under the first MDP versus under the second MDP? Are the optimal policies for these two MDPs the same? What is the MDP terminates?

Given an MDP with reward function $r(s, a)$ and another with reward function $c \cdot r(s, a) + b$ where $b, c \in \mathbb{R}$, what is the relationship between the expected return of a policy π under the first MDP versus under the second MDP? Are the optimal policies for these two MDPs the same?

Stochastic rewards. In defining the MDP formalism, we assumed that the reward function $r(s, a)$ was a deterministic function of the state and action. However, some practical settings involve stochastic rewards. For example, the amazing Susan Murphy uses RL to decide which health reminders to show on patients' smartphones (see Fig. 3). Here, the reward signal might depend on whether the patient goes on a walk, but the patient's might occasionally forget to log their walk.

Let's say that you have a problem with a stochastic reward function $\mathbf{r} \sim p(r \mid s, a)$. How might you convert this into an MDP (with a deterministic reward function)?

Bonus: Stochastic transitions. Many real-world problems have stochastic transitions, and the MDP formalism allows for such randomness. In contrast, a field known as often (but not always) looks at problems with deterministic transitions. When is possible to convert a problem with stochastic transitions into an equivalent problem with deterministic transitions? (This is a fun problem, but gets pretty nasty. If interested, check out the PEGASUS construction [2].)

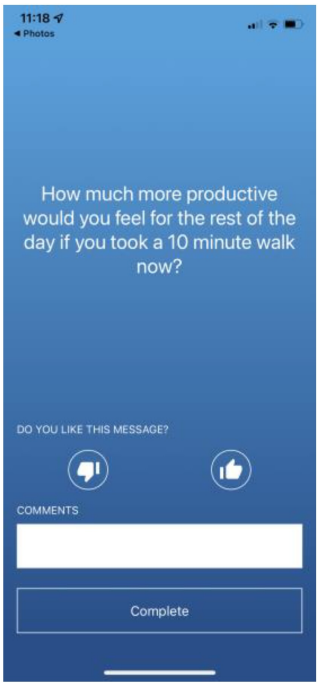

Figure 3: Application of RL to healthcare apps [1]

3 What are the inputs and outputs?

When defining a new problem, it's often very useful to think about what the inputs and outputs of the algorithm will be.

Output. In RL, the output will be the policy, $\pi(a \mid s)$. The policy can be thought of as a particular strategy for solving the task at hand. There will be many ways of *representing* this policy:

1. Lookup table: rows correspond to states and columns show the probabilities associated with each action
2. "Proabilistic" neural network: The input is the state, and the output are the parameters of some distribution over actions (e.g., the neural network outputs the mean and variance of a Gaussian distribution over actions).
3. "Implicit" neural network: The input is a state and some random noise, and the output is some action.

At the end of the day, it's this policy that we are trying to learn.

Input. The **input** of the RL problem is a bit harder to pin down, because it takes a few different forms depending on the precise problem setting:

- **Offline** setting.[4]
  - Static dataset of transitions $\{(s, a, r, s')\}$.
  - Static dataset of trajectories $\{\tau = (s_{1:T}, a_{1:T}, r_{1:T})\}$.
- **Online** or interactive setting.
  - Collect some transitions/trajectories, do some learning, collect some more data, repeat.
  - Known transition probabilities (can solve analytically).
  - Known dynamics (can sample infinite data).

Another factor is whether the reward function is known.

More broadly, when talking about RL there are roughly two schools of thought about how to think about the problem. *One model* is to think of the input as being the MDP. This mental model is very in line with graph search algorithms that you may have seen before. *The second model* is to think of the input as being data, datat that is either collected in the onlien setting of the offline setting. This mental model is very in line with the ML courses you've seen before. This is why this is a reinforcement *learning* course.

One of the beautiful things about RL is that these two models connect to many of the things that you've seen in prior classes. The first model relates to graph search, dynamic

[4]Later, we'll talk about **off-policy** and **on-policy** methods. You can apply both online and offline RL methods in both off-policy and on-policy settings, though you're mileage may vary.

programming, discrete math, algorithms. The second model relates to pattern recognition, machine learning, computer vision, NLP.

Some of the most exciting ongoing research lies at the intersection of both of these fields.

4 Coding: (Lecture 1) Engineered Controller for CartPole [20 min]

Setup: classical control problem. How to you balance a pole by driving a cart Turn to a partner to write some code for doing this (3 min)

References

[1] Klasnja, P., Smith, S., Seewald, N. J., Lee, A., Hall, K., Luers, B., Hekler, E. B., and Murphy, S. A. (2019). Efficacy of contextually tailored suggestions for physical activity: a micro-randomized optimization trial of heartsteps. *Annals of Behavioral Medicine*, 53(6):573–582.

[2] Ng, A. Y. and Jordan, M. I. (2013). Pegasus: A policy search method for large mdps and pomdps. *arXiv preprint arXiv:1301.3878*.