13
Introduction to Reinforcement Learning

This part of the course concerns *Reinforcement Learning (RL)*, the conceptual underpinning of several modern technologies such as self-driving technologies in new cars. It is the third major category of machine learning, in addition to the two previously seen categories of supervised and unsupervised learning. In class we saw a video of robots (made by Boston Dynamics) doing parkour, dancing, and overall doing a pretty good job of imitating the peak human physique. That is also achieved via RL.

The basic idea of RL involves the concept of an *agent* learning to make a *sequence of actions* in a *dynamic* environment. At each discrete time step, the agent is able to take one of a menu of actions. Each choice of action leads to changes in the state of the world (*i.e.*, the agent and its surroundings). The agent has an internal representation of the current and potential states of the world (*e.g.*, using vision or other sensing modules). Under this setting, the agent takes a sequence of actions towards a certain goal.

The world contains uncertainty due to a variety of factors. For instance, there may be other agents in the environment that also take actions to their own benefit, or the sensing modules may be imperfect. Thus taking the same action from the same state of the world may lead to different evolution of state in the future — that is, RL is *non-deterministic*.

In this chapter, we introduce the basic elements of RL using real-world examples, and what it means for the agent to act optimally. Chapter 14 focuses on the setting where the underlying environment (*e.g.*, the number of states, the current state, the probability distribution) is completely known to the agent. [1] In Chapter 15, we will present the case where the environment is not fully available to the agent, and the agent learns about the environment while also learning to act in it. [2]

[1] Think of playing a game where you know the complete set of rules.

[2] Think of playing a Role-playing Game (RPG) where you need to unlock parts of the map by advancing the story.

13.1 Basic Elements of Reinforcement Learning

Now we formalize several of the basic elements of reinforcement learning that were sketched above.

13.1.1 States and Actions

There is a finite set S of states, and the entire system AGENT + ENVIRONMENT exists in one of these states at any time. At each state $s \in S$, the agent makes an action $a \in A_s$, where A_s is the set of allowed actions at state s. We denote $A = \bigcup_{s \in S} A_s$ to be the set of all possible actions in the whole RL environment.

Example 13.1.1. *Consider a game of chess. Each state s can be represented as a pair (C, p) where C denotes the current configuration of pieces and p denotes the player to play next. For example, "white king at $e1$, black king at $e8$, and it is white turn to move" would be a possible state s of the game. An action a is a valid movement of a piece, given a state of the game. For example, "white king to $e2$" (i.e., $Ke2$) would be a possible action of the agent playing white in state s.*

Example 13.1.2. *Self-driving cars, like those built by Tesla, are becoming increasingly popular. Let's imagine how we could construct a state diagram for the task of driving autonomously. Each state can be represented by the current configuration of a number of factors (e.g., the car speed, distance from lane boundaries, distance to nearest vehicle, etc.) Possible actions include increasing/decreasing speed, changing gear, changing direction, changing lane, etc.*

13.1.2 Modeling Uncertainty via Transition Probabilities

As mentioned, the agent has many sources of uncertainty in its knowledge about the environment, and we can use concepts from probability to model uncertainty.

Suppose $S = \{s_1, s_2, \ldots, s_n\}$ contains n states. When the agent takes action a while in state s, it will *transition* into another (potentially the same) state s'. The catch is: the agent does not know exactly which state it will end up in. Instead, there is a probability p_i of ending up in state s_i for each $s_i \in S$. Here $\sum_i p_i = 1$, meaning each (state, action) pair is associated with a *probability distribution* over the next state that the agent will enter. Formally, we define it as follows:

Definition 13.1.3 (Transition Probabilities)**.** *Given a state $s \in S$ and an action $a \in A_s$, there is an associated **transition probability** $p(* \mid s, a)$ distributed over S such that state $s' \in S$ happens with probability $p(s' \mid s, a)$*

when action a is taken at state s and $\sum_{s' \in S} p(s' \mid s, a) = 1$. *If* $p(s' \mid s, a) > 0$, *we say that the state s' is reachable from s when action a is taken.*

In general, not all states are reachable, given a state s and an action a. That is, some transition probability $p(s' \mid s, a)$ is zero. For these states, it is conventional to leave out the corresponding transitions when representing the RL environment as a state diagram as in Figure 13.1 or Figure 13.2.

Example 13.1.4. *Consider the state diagram shown in Figure 13.1. This is a special case where there is only one action a in the set A. In other words, the agent is not making any choices; instead, it is just following probabilistic transition over time steps. To calculate the probability of reaching state s_3 from s_0, we note there are two different paths. The first path is $s_0 - s_1 - s_3$ and the second path is $s_0 - s_2 - s_3$. We thus calculate the probabilities of each of these paths and note that the overall probability of reaching s_3 will be the sum of both:* $0.2 \cdot 0.7 + 0.8 \cdot 0.4 = 0.46$.

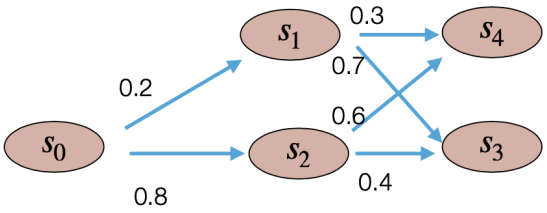

Figure 13.1: An example diagram where $|A| = 1$. The agent simply follows probabilistic transitions.

Now we consider an example where there is more than one action to make. In this case, each action induces a different probability distribution on the set of states, so we need to draw a diagram for each option.

Example 13.1.5. *We can model a baby learning to walk through RL. As shown in Figure 13.2, we can define the state $s_0 =$* ***standing but feeling unsteady****, $s_1 =$* ***standing and feeling secure****, and $s_2 =$* ***on ground****. The baby has two actions to take: $a =$* ***not grab onto nearest support*** *and $a' =$* ***grab onto nearest support****. The state diagram on the top represents the transition probabilities when the baby takes the action a. See that the baby has a high chance of entering state s_2 — falling to the ground. On the other hand, the state diagram on the bottom represents the transition probabilities when the baby takes the action a'. The baby now has a high chance of entering state s_1 — standing securely on the ground. The two actions have* ***different*** *probability distributions associated with the relevant transitions.*

Example 13.1.6. *Mechanical ventilators are used to stabilize breathing for patients. Suppose we wished to construct a state diagram. We could define*

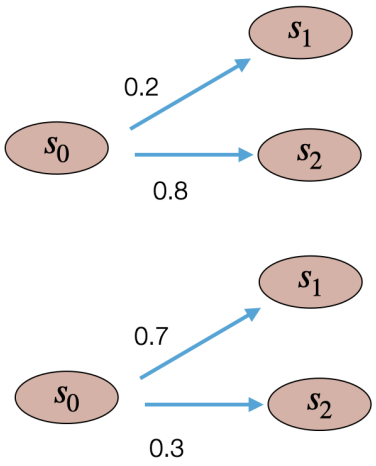

Figure 13.2: An example diagram showing how an action determines the probability of outgoing transitions.

states that consider the pressure and CO_2 level in the patient's level for the past k seconds. Actions might include adjusting the flow rate of oxygen via valve settings as needed. Possible transitions might include the typical mechanical response of the lungs, or unexpected spasms. Finally, we can define the goal to be maintaining steady pressure in the patient without "overshooting" and causing damage.

13.1.3 Agent's Motivation/Goals

In general, an agent is a participant in RL models driven by the need to maximize "rewards." In a probabilistic setting, the agent wishes to maximize their *expected* rewards. In a natural setting, the "rewards" could be innate satisfaction, such as getting to eat food, being entertained, etc. But in the usual artificial settings such as robots and self-driving cars, rewards are sprinkled by the system designer into the framework. Some examples appear later.[3]

At each step, the agent takes an action, and is given a reward (which could be negative, *i.e.*, is a punishment) based on the action, current state, and next state.

[3] While reward/punishment as a way to shape human or animal behavior is a very old idea, mathematical modeling of agents as reward-maximisers appears in several disciplines that flowered around the middle of the 20th century (*e.g., behaviorism* in psychology, profit-maximisation in economics, and of course RL).

Definition 13.1.7 (Reward)**.** *For each valid 3-tuple (a, s, s') where $s' \in S$ is a state reachable from state $s \in S$ by taking action $a \in A_s$, we define a corresponding **reward** $r(a \mid s_1, s_2) \in \mathbb{R}$.*

Example 13.1.8 (Example 13.1.5 revisited)**.** *When the baby stands and feels secure after grabbing onto something, the parents applaud the baby, and the baby receives a positive reward: $r(a' \mid s_0, s_1) = 5$. When the baby feels secure without grabbing onto the nearest support, the parents feel even prouder and the baby gets a more positive reward: $r(a \mid s_0, s_1) = 10$. When the baby falls to the ground, the baby feels pain and receives negative reward:*

$r(a \mid s_0, s_2) = r(a' \mid s_0, s_2) = -5$.

Typically, the designer of a RL model gets to define the rewards throughout the framework based on the designer's judgment. For instance, in Example 13.1.2, we might design an RL model such that if the car drifts into an adjacent lane, we assign a negative reward. If another vehicle is in the lane, we might assign an even larger negative reward. This will induce a RL model to "learn" the proper way to driving — staying in lane.

13.1.4 Comparison with NFA

Recall the Non-deterministic Finite Automata (NFA) you learned in COS 126. In an NFA, there is a finite number of states, and for each state, we know the set of next possible states, based on the next input character.

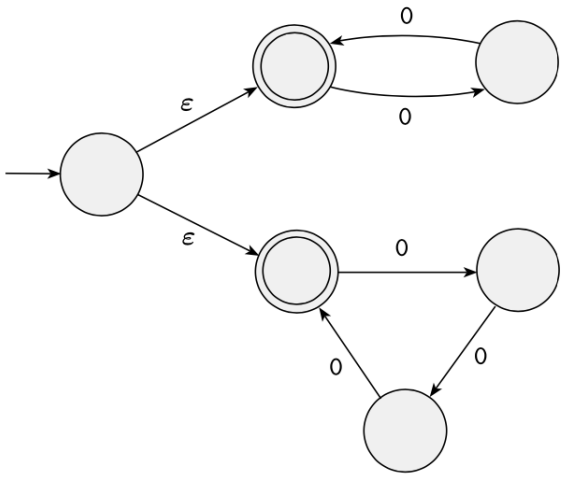

Figure 13.3: A sample Non-deterministic Finite Automata. Source: *Introduction to the Theory of Computation* by Michael Sipser.

We can consider the following analogy between RL and NFA — there is someone behind an NFA, who can observe its current state and type in the next input character. This person will be called an *agent*, and the choice of input character that is typed in will be called an *action*. Each action can lead to a finite set of next possible states, but because of some uncertainty in the world, the agent cannot specify which particular state will be the next one. This is similar to an NFA in the sense that the actions are non-deterministic. Also, just like in an NFA, the change in the current state is also referred to as a *transition*. One major difference between RL and NFA is that while an NFA only cares about the final state of the automata (*i.e.*, whether it is an accept state or a reject state), in RL, the agent is given a *reward* after each transition. The goal of the agent will be to take a sequence

of actions so as to maximize the sum of the reward throughout the sequence of actions.

13.2 *Useful Resource: MuJoCo-based RL Environments*

Real-life robots with precise and reliable hardware can get very expensive to buy, let alone train. An easier playground for students (especially those trying to work with a single GPU on CoLab) is doing RL in a virtual environment.

MuJoCo is a famous physics engine that allows creating virtual objects with somewhat realistic "joints" that can be commanded to move similar to real-life robots. OpenAI and DeepMind have open-source environments that allow experimentation in the MuJoCo environment. The official website gives a pretty good overview of the software: [4]

[4] Source: https://mujoco.org.

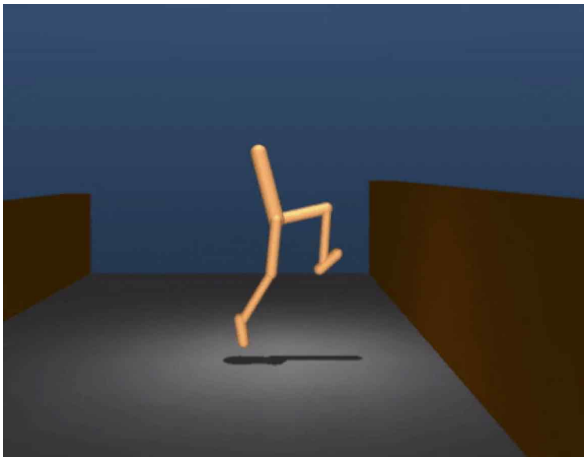

Figure 13.4: An example of a MuJoCo Walker.

> *MuJoCo is a physics engine that aims to facilitate research and development in robotics, biomechanics, graphics and animation, and other areas where fast and accurate simulation is needed. MuJoCo offers a unique combination of speed, accuracy and modeling power, yet it is not merely a better simulator. Instead it is the first full-featured simulator designed from the ground up for the purpose of model-based optimization, and in particular optimization through contacts.*

One aspect of MuJoCo simulation involves a representation of a humanoid figure (*i.e.*, the agent) learning how to navigate an obstacle course (*i.e.*, the environment). Training videos are readily available online and show how the agent learns over time (sometimes, to comedic effect).

Example 13.2.1. *Let's analyze the example of an agent navigating an obstacle course through a RL framework. The states can be considered to be the set of coordinates, velocity, and acceleration for each limb, the velocity*

and acceleration for the motors in each joint, and the environment itself straight ahead. The actions can include the agent increasing or decreasing motor speed in their joints. Finally, the final goal is to stay upright, run forward at a reasonable pace, and avoid obstacles.

13.3 Illustrative Example: Optimum Cake Eating

Let's consider an extended example which ties together the elements of RL discussed previously. Suppose you buy a small cake with three slices. The reward of eating one slice at one sitting is 1, but eating two or three slices at one sitting is 1.5 and 1.8 respectively. [5]

[5] This sense of diminishing rewards is known as the satiation effect.

Problem 13.3.1. *Suppose you plan to eat the cake over a period of three days. What eating schedule will maximize the internal reward?*

Now let's introduce your roommate, who is oblivious to basic understandings of ownership and adheres to the "finders keepers" faith. We define the probability $\Pr[\text{sneakily eats a slice overnight}] = \frac{1}{2}$. To account for this uncertainty, we can create a *look-ahead tree* for different initial actions. We first consider the action where you decide to eat two out of the three slices on the first night. Successive states and associated probabilities are shown in the Figure 13.5.

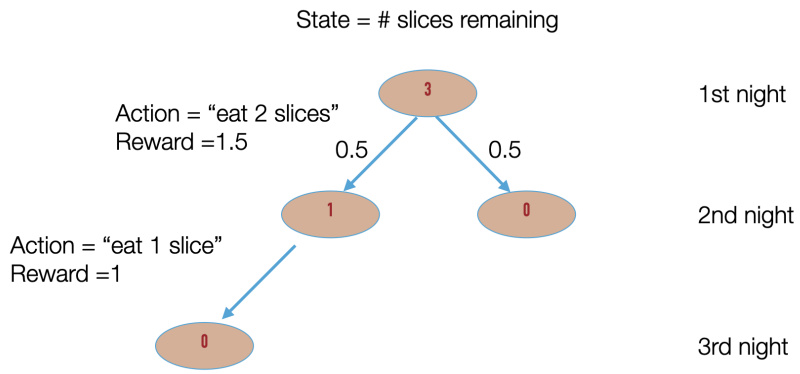

Figure 13.5: The diagram (look-ahead tree) of the cake problem where you decide to eat two slices on the first night. Each state represents the number of slices remaining, and each action represents the number of slices eaten on one night.

Even though you only eat only two out of the three slices during the first night, there is a $\frac{1}{2}$ chance that your roommate eats the remaining slice overnight. Therefore, the action of "eating 2 slices" can lead to two possible states — "1 slice left" or "0 slice left" — each with probability $\frac{1}{2}$.

Example 13.3.2. *We can calculate the expected reward associated with eating two slices on the first night by analyzing the Figure 13.5. You first gain reward of* 1.5 *by eating the two slices on the first night. Then with probability* $\frac{1}{2}$ *(where the roommate does not eat the remaining slice overnight),*

you get to eat the last slice on the second night and gain additional reward of 1. *With probability of* $\frac{1}{2}$ *(where the roommate eats the remaining slice), you cannot gain anymore reward. That is, the expected reward is* $1.5 + 0.5 \cdot 1 + 0.5 \cdot 0 = 2$.

Problem 13.3.3. *Consider the result of Example 13.3.2. Would you prefer to take two slices on the first night or three slices?*

We next consider the action where you decide to eat one out of the three slices on the first night. Successive states and associated probabilities are shown in the Figure 13.6.

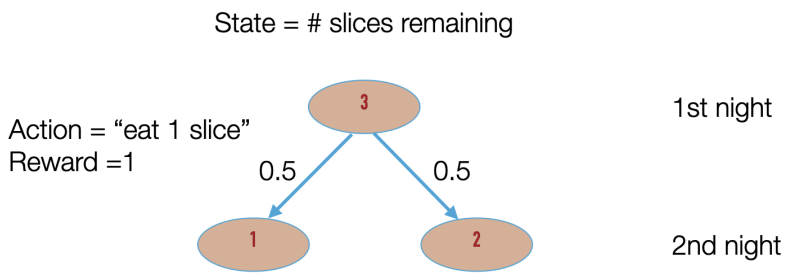

Figure 13.6: The diagram (look-ahead tree) of the cake problem where you decide to eat one slice on the first night.

The difficulty in this example in contrast to the Figure 13.5 is that if the roommate does not eat a slice after the first night, you have two slices at your disposal on the second night. You have two actions you can take in this "2 slices left" state — "eat 1 slice" (and hope the third slice is still there on the third night) or "eat 2 slices" — and it is not immediately obvious which one is more optimal. It turns out that the expected reward you can get from the remaining 2 slices is 1.5 for both options.

Problem 13.3.4. *Verify the previous claim that both options on the second night have the same expected reward.*

Example 13.3.5. *Given the previous analysis and the look-ahead tree in the Figure 13.6, we note that the total expected reward is* $1 + 0.5 \cdot 1 + 0.5 \cdot 1.5 = 2.25$. *You first receive a reward of* 1 *by eating* 1 *slice on the first night. Then with probability* $\frac{1}{2}$, *the roommate eats one slice over night, and you gain reward of* 1 *by eating the last slice on the second night. With the remaining probability* $\frac{1}{2}$, *the roommate does not eat a slice, and you are expected to gain reward of* 1.5 *from the remaining* 2 *slices, regardless of the action you choose to take on the second night.*

Problem 13.3.6. *Consider the result of Example 13.3.5. Would you prefer to take two slices on the first night or one slice?*