Rational Reinforcement Learning

December 13, 2024

The landscape of reinforcement learning

Reinforcement learning is one of the most successful traditions in the field of artificial intelligence. Reinforcement learning has enabled milestone achievements in several areas of artificial intelligence—most notably game-playing. In addition, the reinforcement learning framework has enjoyed widespread explanatory success in the cognitive and neuro-sciences.

Reinforcement learning (henceforth *RL*) models an agent learning how to act in an environment. The agent can interact with the environment by performing actions. When an agent performs an action, it receives a **reward**.[1] In all but the simplest cases, the agent's actions also influence the environment.[2] The framework assumes that environmental feedback is entirely captured by the reward signal. The agent is assumed to have the goal of maximizing its long-term cumulative reward, also called its **return**.[3] Although these assumptions may appear restrictive, the range of problems that can be framed in the RL framework is surprisingly broad—from playing Space Invaders (Mnih et al. 2015), to navigating rugged terrain (Tang et al. 2024), to controling plasma flow in fusion reactors (Degrave et al. 2022), to optimizing SAT-solving algorithms (Fournier et al. 2022).

[1] The reward signal is a single scalar. It may be negative.

[2] The simplest cases are so-called **bandit problems**, in which the task is to figure out which of several slot machines is best to play. In such cases, the agent's actions correspond to playing a given machine. Playing a machine yields a reward, but does not change the state of the environment.

[3] In the most general setting, the signal need not be a *reward* signal. It may instead be some other variable that the system is trying to predict or control, such as the value of a sensor. Many of the techniques developed to maximize reward apply in this more general context.

In what follows, we present some of the key concepts used in stating and solving the reinforcement learning problem. These concepts are important because they are *explanatorily powerful* and *rationally relevant*. The explanatory power of the RL framework is witnessed by its explanatory successes in the cognitive and neural sciences. We believe that it can also be used to shed light on questions of traditional philosophical interest. Its relevance to rationality will be explained later, in the context of bounded approaches to rational decision-making.

The setup

In a RL model, the environment is represented by a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, and a set of rewards $\mathcal{R}$. A **trajectory** is a (possibly infinite) sequence of random variables of the form

$$S_0, A_0, R_1, S_1, A_1, R_2, \ldots$$

with S_i, A_i, and R_i being states, actions, and rewards.

The environment is equipped with a probability distribution over trajectories. In practice, we assume that the environment is a **Markov Decision Process**, meaning that this distribution is fully determined by the conditional probability distribution $P(S_{t+1}, R_{t+1}|S_t, A_t)$. Informally, this means that the effects of an action (as far as rewards and environmental states are concerned) depend only on the state of the environment when the action is taken. In particular, they do not depend on the actions taken to reach that state, or on the history of states up to this point.

In practice, the environement is often finite, meaning each of $\mathcal{S}, \mathcal{A}$, and $\mathcal{R}$ is finite. In this case, we can assign a probability to each possible state s' and reward r, conditional on taking a given action a in a given state s:

$$p(s', r|s, a)$$

These probabilities fully determine the dynamics of the environment with which the agent is interacting.

The **return** G_t of a trajectory (starting at step t) is the (possibly discounted) sum of rewards in the trajectory:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=1}^{\infty} \gamma^k R_{t+k+1}$$

Here, $\gamma \in [0, 1]$ is a discounting factor. Smaller values of γ mean that later rewards contribute less to the return. We assume that the agent's goal is to maximize its return.

In practice, the best the agent can do is to maximize *expexted* return, beacuse actual returns need not be known ahead of time: as noted above, the results of the agent's action are stochastic. In addition the agent may choose its actions stochastically.

The agent acts according to a **policy** $\pi : \mathcal{A} \times \mathcal{S} \to [0, 1]$.[4] $\pi(a|s)$ is the probability that the agent chooses action a in state s. Different policies specify different ways for the agent to interact with its environment. Policies are *complete*: they specify the agent's choice of action in every possible state of the environment.

[4] While this notation is common, some authors prefer to construe policies as stochastic functions from states to actions. The difference is insubstantial.

In order to guide the design and selection of policies, the agent can use a value-function. The **state-value function** $v_\pi(s)$ associates with each state its expected return under π:

$$\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] \\
&= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| s_t = s\right] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\
&= \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a)\left[r + \gamma v_\pi(s')\right] \\
&= \sum_{a} \pi(a|s) \sum_{s', r} p(s', r|s, a)\left[r + \gamma v_\pi(s')\right]
\end{aligned}$$

This last equation is called the Bellman recurrence equation, so-called because v_π appears on both sides of the identity (and because Bellman discovered it). The Bellman equation implicitly defines v_π: v_π is the only function from states to returns that satisfies the recurrence. Similarly, the **action-value function** $q_\pi(s, a)$ associates with each state-action pair its expected return. A Bellman recurrence also holds for action-value functions. $v_\pi(s)$ and $q_\pi(s, a)$ are measures of how beneficial it is, in expectation, to be in or to take action a in state s.

A policy π_* is optimal if $v_{\pi_*}(s) \geq v_\pi(s)$ for all s and all π.[5,6] While the agent's stated goal is to maximize expected returns, finding an optimal, or nearly optimal, policy is a suitable proxy for this goal. Indeed, following an optimal policy suffices to maximize expected returns. In practice, "solving" a reinforcement learning problem consists in finding an optimal (or nearly optimal) policy for that environment.

[5] Note that optimal policies need not be unique.

[6] Optimal objects are usually denoted with a subscripted star.

An optimal value function v_* or q_* maps states (or state-action pairs) to their value under an optimal policy. Optimal value functions are the unique functions satisfying their corresponding Bellman equations:

$$\begin{aligned}
v_{\pi_*}(s) &= \max_a \mathbb{E}_{\pi_*}[G_t|S_t = s, A_t = a] \\
&= \max a \sum_{s', r} p(s', r|s, a)\left[r + \gamma v_{\pi_*}(s')\right]
\end{aligned}$$

With an optimal value function in hand, the agent can follow an optimal policy.[7]

[7] As we explain below, this point is somewhat subtle. Given a state-value function, an agent can only derive the corresponding greedy policy if it has a model of its environment. Without a model, the agent must instead rely on an action-value fuction.

Exploration and Exploitation

In general, a reinforcement learning agent begins with little to no information about its environment. It must act in order to discover how to act. This observation sets up the **explore-exploit dilemma**. At any given point, the agent may choose either an action it currently deems best, or some other action. Choosing the action it deems best is called **exploitation**: the agent exploits the knowledge it has acquired in order to maximize expected returns. An action that maximizes expected return is called **greedy**, and a policy that always selects a greedy action is called a **greedy policy**. Choosing a non-greedy action is called **exploration**. Exploration exposes the agent to a wider range of experiences than pure exploitation. As a result, balancing exploration and exploitation is crucial to discovering optimal policies.

As an example, consider a simple two-armed bandit scenario: at each point in time, the agent can choose between pulling lever a and pulling lever b. The reward from pulling level a follows a normal distribution $\mathcal{N}(0.5, 1)$, while b always offers 0 reward. The optimal strategy for maximizing long term return is clearly to always pull a, since it has higher expected reward than b. However, it is possible that over the course of its first few actions, the agent experiences negative rewards from a. In such cases, a will appear worse than b (it will have a lower expected reward). If the agent were to pursue a greedy policy from that point on, it would always choose b, since that arm appears better. If the agent sometimes explores, however, it will occasionally choose arm a. As long as the agent maintains its exploration for long enough, we will expect its estimate of a's value to converge to the true value 0.5.

How to balance exploration and exploitation is a subtle question. For example, if the agent is playing the last round of a game, then it is probably better off exploiting what it has learned in the previous k rounds rather than exploring: the chances of discovering some better moves are low, and there will be no opportunity to recover from a costly mistake. Thus, the rationality of the decision whether to explore or exploit is influenced by the number of trials left in the task. [Mention Gittins index as optimal explore-exploit strategy here]

Algorithms

In this section, we give a brief overview of some classic reinforcement learning algorithms and indicate some of the results of the recent injection of deep learning into the field.

The goal of a reinforcement learning agent is to find an optimal policy. Since it is often computationally prohibitive to compute an optimal policy, many problems accept *approximately optimal* policies. [[Need to say more here about the kind of standard we impose on RL algorithms—convergence? fast convergence? convergence to approximately optimal policy?]]

Virtually all reinforcement learning algorithms employ some version of the **policy iteration** algorithm. This algorithm iteratively approaches an optimal policy in a sequence of stages, each of which comprises two steps. The first step is **policy evaluation**, whereby the agent computes the value function v_π for the current policy π. The second step is **policy improvement**, whereby the policy π is updated so as to be greedy with respect to v_π. These two steps are repeated indefinitely. It can be shown that this algorithm converges to an optimal policy, in the sense that it yields an optimal policy after a finite number of iterations. We can picture the policy iteration algorithm as follows:

$$\pi_0 \overset{\text{evaluation}}{\rightarrow} v_{\pi_0} \overset{\text{improvement}}{\rightarrow} \pi_1 \overset{\text{evaluation}}{\rightarrow} v_{\pi_1} \overset{\text{improvement}}{\rightarrow} \dots \overset{\text{improvement}}{\rightarrow} \pi_*$$

The algorithm may also be run with an *action*-value function at the evaluation step. Policy improvement is usually conceptually and computationally easy: it suffices to extract the best action for each state, which is trivial if with an action-value function and easy to do with a state-value function (if the agent has a model).

Policy evaluation is more difficult. To each state there corresponds an instance of the Bellman equation specifying $v_\pi(s)$ partly in terms of $v_\pi(s')$. If there are n states, these Bellman equations form a system of n equations in n unknowns. Solving these equations yields the true value function v_π. However, it is prohibitively expensive to solve this sytem of equations for all but the most trivial environments.

Instead, policy evaluation algorithms are almost exclusively concerned with **approximating** a value function for π. Many such algorithms are usually organized around the Bellman recurrence equation. The key idea is to treat the Bellman equation as an *update rule*:

$$V_\pi(s) \leftarrow \sum_a \pi_*(a|s) \sum_{s',r} p(s',r|s,a)\big[r + V_\pi(s')\big]$$

where $V_\pi(s)$ is the current estimate of the value of s (under policy π). Intuitively, the iterative Bellman rule updates an estimate $V_\pi(s)$ of the value of a state with a combination of factual information (in the form of the immediate reward r) and further estimates (in the form of the estimated returns of next states $V_\pi(s')$.

Key distinctions

There are a number of ways of refining the basic RL setup introduced in the previous subsection. Here we cover some of the most important ones.

Model-free vs. model-based: a reinforcement learning algorithm may require its user to possess a model of its environment. Models come in two varieties. **Distribution models** allow their users to compute the distribution $p(s', r|s, a)$. **Sampling models** allow their users to sample from the distribution $p(s', r|s, a)$. Thus, predictive models can be used to plan ahead: in principle, an agent equipped with an accurate model need not rely on its experience. It can simply compute the expected consequences of its actions in every possible state to determine an optimal policy. In practice, even if the agent possess an accurate model, the computations involved in determining the optimal policy are intractable for all but the simplest environments, and other methods must be employed. Such methods still make use of the model, but only use it to look, say, one step ahead. Sampling models do not allow the agent to plan in the straightforward way enabled by predictive models. Instead, sampling models allow the agent to *simulate* experiences. If the agent's model is perfectly accurate, then sample simulated experiences follow the same distribution as actual experiences, and so are just as good from the perspective of learning. Sampling models are less demanding than predictive models: it is often easier to sample from a distribution than it is to compute the distribution.

Model-free algorithms do not require either a predictive or a sampling model. Agents employing model-free algorithms learn in a trial-and-error fashion: they can only obtain information about the effects of their actions by taking those actions.

An important special case of model-based reinforcement learning is the use of approximate models. In general, we cannot expect the agent to have a perfectly accurate model of its environment. Instead, the agent maintains an imperfect model of its environment and improves it on the basis of its experience.

The three-way distinction between predictive, sampling, and model-free algorithms is important.

Stationary vs. nonstationary environments: in a stationary environment, the transition probabilities given by $p(s', r|s, a)$ are fixed. In a nonstationary environment, the transitions may evolve over time. A nonstationary environment is more difficult to navigate, since what the agent learns may become obsolete as the environment changes. A number of techniques are available to deal with nonstationary environments. The most common one is to induce the agent to "forget" what it has learned. This forgetting is implemented by weighing recent evidence about the value of a state or action more heavily than past evidence.

In general, model-free agents are preferred in nonstationary environments. This is because models (especially distribution models) are generally costly to acquire, maintain, and use. If the nonstationary environment shifts rapily enough, the agent may not have time to build an accurate model. Even if it succeeds in building such a model, the model is unlikely to remain accurate for long. Instead of investing computational resources in building a model, then, it is often preferable to "go with the flow": make decisions about what to do on the basis of one's recent experiences.

An important exception to this analysis of nonstationary environments is the case when the environment is nonstationary but periodic: that is, transition probabilities change over time, but the changes themselves follow a predictable cycle. In such cases, the environment exhibits a kind of higher-order stability, and models may be worth the investment. The day-night cycle provides an example of (an aspect of) a nonstationary but periodic environment: the state transitions evolve over time, but in a predictably cyclical way.

Finally, it is worth noting that some environments, while technically nonstationary, evolve so slowly that they may be treated as stationary for all intents and purposes (consider changes in the Earth's rotational axis, inducing subtle changes in the direction of the magnetic north).

On-policy vs. off-policy algorithms: some reinforcement learning agents learn **on-policy**. This means that they are improving the policy that guides their behavior. By contrast, **off-policy** agents act according to one policy, but evaluate and improve another. Why would an agent improve a policy which it is not following? [Explain off policy learning]