Reinforcement Learning Notes

Faraaz Akhtar

October 2024

1 Introduction

Reinforcement learning (RL) is a type of machine learning in which an agent learns to make decisions by interacting with an environment. The agent takes actions in the environment to maximize some notion of cumulative reward over time. Unlike supervised learning, where the model is trained on a fixed dataset, in RL the agent learns from the consequences of its actions through a process of trial and error.

1.1 How does reinforcement learning work?

At each step t the agent:

- Receives observation O_t (and reward R_t)
- Executes action A_t

and the environment:

- Receives action A_t
- Emits observation O_{t+1} (and reward R_{t+1})

The agent's job is to maximize cumulative reward

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \dots$$

Reinforcement learning is based on the hypothesis that any goal can be formalized as the outcome of maximizing a cumulative reward.

We call the expected cumulative reward, from a state s, the value

$$v(s) = E\left[G_t \mid S_t = s\right]$$
$$= E\left[R_{t+1} + R_{t+2} + R_{t+3} + \cdots \mid S_t = s\right]$$

Returns and values can be defined recursively

$$G_t = R_{t+1} + G_{t+1}$$
$$v(s) = E\left[R_{t+1} + v(S_{t+1}) \mid S_t = s\right]$$

The environment state is the environment's internal state and is usually invisible to the agent. Even if it is visible, it may contain lots of irrelevant information.

The history is the full sequence of observations, actions, rewards

$$\mathcal{H}_t = O_0, A_0, R_1, O_1, \ldots, O_{t-1}, A_{t-1}, R_t, O_t$$

(This could, for example, include the sensorimeter stream of a robot.)

This history is used to construct the agent state S_t. The agent state could be the environment state if the agent has full observability. i.e., $S_t = O_t =$ environment state.

1.2 Markov Decision Processes (MDPs)

A decision process is Markov if

$$p(r, s \mid S_t, A_t) = p(r, s \mid \mathcal{H}_t, A_t)$$

This means that the future is independent of the past given the present state. In other words, the probability of transitioning to a new state s and receiving a reward r depends only on the current state S_t and the action A_t, and not on the full history $\mathcal{H}_t$.

This simplifies the decision-making process because instead of needing to keep track of the entire history of states and actions, the agent only needs to know the current state and current action to make predictions about the future. This assumption is what makes Markov Decision Processes (MDPs) computationally feasible and widely used in reinforcement learning.

The fact that a state is Markov doesn't mean it contains everything, only that adding more history doesn't help in predicting the future. Once the state is known, the history can effectively be discarded. The full environment and agent state together form a Markov process, though it may be large. Similarly, the full history is also Markov, but it keeps growing over time. Typically, the agent state is a compressed version of the full history.

Often, it's not possible to construct a fully Markovian agent state. In such cases, it may be more useful to focus on the policies and value prediction.

1.3 Policy

A policy defines the agent's behavior by mapping the agent's state to an action. It can either be deterministic, where the action A is a specific function of the state S (i.e., $A = \pi(S)$), or stochastic, where the policy defines a probability distribution over actions given the state, $\pi(A|S) = p(A|S)$.

1.4 Value Functions

The actual value function is the expected return:

$$v_\pi(s) = E[G_t \mid S_t = s, \pi] = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s, \pi]$$

In the above equation, the discount factor, $\gamma \in [0, 1]$, trades off importance of immediate vs long-term rewards.This can also be used to evaluate the desirability of states.

The return has a recursive form:

$$G_t = R_{t+1} + \gamma G_{t+1}$$

And therefore, the value function has a recursive form as well:

$$v_\pi(s) = E\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t \sim \pi(s)\right]$$

$$= E\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t \sim \pi(s)\right]$$

Where, $A_t \sim \pi(s)$ means action a is chosen by policy π in state s, even if π is deterministic. This recursive equation is known as the Bellman equation.

A similar equation holds for the optimal value function, $v_*(s)$, which gives the highest possible value from any state s:

$$v_*(s) = \max_a E[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a]$$

This equation does not depend on a specific policy because it always chooses the action a that maximizes the expected return. This means the agent selects the optimal action for each state, regardless of a fixed policy. The equation is fundamental in algorithms, where agents aim to learn the optimal value function and select actions that maximize reward over time.

While learning the optimal value function is powerful, learning a policy directly can be more practical because it provides an explicit mapping from states to actions. Policies handle stochastic environments, balance exploration vs. exploitation, and can be more efficient in high-dimensional or continuous action spaces.

1.5 Model

A model predicts what the environment will do next.

- $\mathscr{P}$ predicts the next state:

$$\mathscr{P}(s, a, s') \approx p(S_{t+1} = s' \mid S_t = s, A_t = a)$$

- $\mathscr{R}$ predicts the next (immediate) reward:

$$\mathscr{R}(s, a) \approx E[R_{t+1} \mid S_t = s, A_t = a]$$

A model does not directly give us a good policy — we still need to plan based on the model. Additionally, we can also consider stochastic (generative) models, which introduce randomness in the model predictions.

2 Exploration vs Exploitation

In RL, exploitation and exploration are two opposing strategies an agent uses to make decisions:

1. Exploitation: This is when the agent uses the knowledge it has already gained to maximize immediate rewards by choosing actions it knows to yield high rewards. For example, if an agent has found a particular path that provides a high reward, exploitation would involve following that path consistently to keep gaining those rewards. However, excessive exploitation can lead to getting stuck in a local optimum, where the agent may miss discovering potentially better paths or strategies.

2. Exploration: This involves trying out new actions that the agent hasn't experienced much (or at all) to gather more information about the environment. By exploring, the agent can potentially discover even better rewards or strategies that it wouldn't have found by exploiting alone. The downside is that exploration may lead to lower immediate rewards, as the agent might try suboptimal actions while gathering information.

2.1 Multi-Armed Bandit

A multi-armed bandit is defined by a set of reward distributions $\{\mathcal{R}_a \mid a \in \mathcal{A}\}$, where:

- $\mathcal{A}$ is a (known) set of actions (or "arms"),
- $\mathcal{R}_a$ is a distribution on rewards, given action a,
- At each step t, the agent selects an action $A_t \in \mathcal{A}$,
- The environment generates a reward $R_t \sim \mathcal{R}_{A_t}$,
- The goal is to maximize cumulative reward $\sum_{i=1}^{t} R_i$,

- We do this by learning a **policy**: a distribution on $\mathcal{A}$.

Understanding multi-armed bandits is important because it provides insight into the trade-off between exploring new options and exploiting known rewards, a challenge faced in many real-world applications, from online advertising (deciding which ads to show) to clinical trials (testing treatment effectiveness).

2.1.1 Regret

The action value for action a is the expected reward:

$$q(a) = E\left[R_t \mid A_t = a\right]$$

The optimal value is:

$$v_* = \max_{a \in \mathcal{A}} q(a) = \max_a E\left[R_t \mid A_t = a\right]$$

Then, the regret of an action a is:

$$\Delta_a = v_* - q(a)$$

The regret for the optimal action is zero, and so ideally we would like to get as close to this as possible. Minimizing total regret can be written as:

$$L_t = \sum_{n=1}^{t} v_* - q(A_n) = \sum_{n=1}^{t} \Delta_{A_n}$$

3 Algorithms

3.1 Action Values

Many algorithms use action value estimates. We can define an average action-value function $Q_t(a)$ as:

$$Q_t(a) = \frac{\sum_{n=1}^{t} \mathcal{I}(A_n = a)\, R_n}{\sum_{n=1}^{t} \mathcal{I}(A_n = a)}$$

where $\mathcal{I}(\cdot)$ is the indicator function:

$$\mathcal{I}(\text{True}) = 1 \quad \text{and} \quad \mathcal{I}(\text{False}) = 0$$

The count for action a is:

$$N_t(a) = \sum_{n=1}^{t} \mathcal{I}(A_n = a)$$

This finds the average reward for action a over all time. This can also be written iteratively as:

$$Q_t(A_t) = Q_{t-1}(A_t) + \alpha_t \left(R_t - Q_{t-1}(A_t) \right),$$

where $R_t - Q_{t-1}(A_t)$ represents the *error* term.

Here, $\forall\ a \neq A_t$:

$$Q_t(a) = Q_{t-1}(a)$$

with

$$\alpha_t = \frac{1}{N_t(A_t)} \quad \text{and} \quad N_t(A_t) = N_{t-1}(A_t) + 1,$$

In a greedy policy, the agent chooses the action with the highest value: where $N_0(a) = 0$.

3.2 Greedy Policy

In a greedy policy, the agent chooses the action with the highest value:

$$A_t = \arg\max_a Q_t(a)$$

or equivalently:

$$\pi_t(a) = \mathcal{I}(A_t = \arg\max_a Q_t(a))$$

A regular greedy algorithm always chooses the action with the highest known reward, leading to a lack of exploration. This can cause the agent to get stuck in a suboptimal choice, as it doesn't try other actions that might yield better rewards. To solve this, we often use something called an ε-greedy policy.

3.2.1 ε-Greedy Policy

The ε-greedy algorithm is a method in reinforcement learning to balance exploration and exploitation. With probability $1 - \varepsilon$, the agent chooses the action with the highest estimated value (exploitation). With probability ε, it selects a random action from the set, allowing it to explore other options. This ensures that the agent mostly follows the best-known actions while occasionally trying new ones, preventing it from getting stuck in potentially suboptimal choices.

Equivalently:

$$\pi_t(a) = \begin{cases} (1-\epsilon) + \frac{\epsilon}{|\mathcal{A}|} & \text{if } Q_t(a) = \max_b Q_t(b) \\ \frac{\epsilon}{|\mathcal{A}|} & \text{otherwise} \end{cases}$$

3.3 Policy Gradient

Here, we pose a question: Instead of learning values, can we learn the policy $\pi(a)$ directly?
For instance, let us define action preferences $H_t(a)$ and a policy:

$$\pi(a) = \frac{e^{H_t(a)}}{\sum_b e^{H_t(b)}}$$

These preferences are not values, they are only learnable policy parameters. The goal is to learn by optimizing these preferences. To do this, we can use gradient ascent. In the bandit case, we want to update:

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta E\left[R_t \mid \pi_{\theta_t}\right]$$

Where, θ_t are the current policy parameters.
We can write this down differently using the REINFORCE trick (Williams, 1992):

$$\begin{aligned}
\nabla_\theta E[R_t \mid \pi_\theta] &= \nabla_\theta \sum_a \pi_\theta(a)\, E[R_t \mid A_t = a] \\
&= \sum_a q(a)\, \nabla_\theta \pi_\theta(a) \\
&= \sum_a q(a) \frac{\pi_\theta(a)}{\pi_\theta(a)} \nabla_\theta \pi_\theta(a) \\
&= \sum_a \pi_\theta(a) \frac{q(a)}{\pi_\theta(a)} \nabla_\theta \pi_\theta(a) \\
&= E\left[R_t \frac{\nabla_\theta \pi_\theta(A_t)}{\pi_\theta(A_t)}\right] = E\left[R_t \nabla_\theta \log \pi_\theta(A_t)\right]
\end{aligned}$$

Thus, this way we can write

$$\nabla_\theta E[R_t \mid \theta] = E\left[R_t \nabla_\theta \log \pi_\theta(A_t)\right]$$