Note on The Reinforcement Learning Problem

Santhosh Kumar G

August 5, 2025

The **Reinforcement Learning (RL) problem** is about learning how to map situations to actions to maximize a numerical reward signal. The learner, called an **agent**, is not told which actions to take but instead must discover which actions yield the most reward by trying them. In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards. Formally, RL is a class of methods for solving various kinds of **sequential decision making** tasks, we want to design an **agent** that interacts with an external **environment**. The agent maintains an internal state S_t, which it passes to its **policy** π to choose an action a_i. Sequential decision problems occur when an agent must make a series of decisions in an environment, with each decision affecting not only the immediate outcome but also the future states of the environment. These decisions are interdependent, meaning that the optimal decision at any point depends on the decisions made previously and the potential decisions that will be made in the future.

1 Key Components of the RL Problem

The RL problem is formally described by a few key elements:

1. **Agent**: The entity that perceives its environment and makes decisions. It's the learner. For example, a chess-playing program is an agent (see Fig.2.), In Super Mario, the program which controls Mario is an agent. $a_i \in (left, right, up, down, jump, speed)$

2. **Environment**: The external world with which the agent interacts. For the chess program, the environment is the opponent and the chessboard. In Super Mario, Other than Mario, the world consists of different monsters (Goomba,Koopa, Spiky, Bullet Bill, and Piranha Plant), platforms (hill,gap, cannon, and pipe) and items (coin, mushroom, bricks,flower).

3. **State (S_t)**: A representation of the environment at a specific point in time, t. It's the information the agent uses to make a decision. In chess, the state is the configuration of all pieces on the board. One frame representing current scene as observed by the agent.

4. **Action (A_t)**: A choice made by the agent when it is in a particular state. For instance, moving a pawn forward is an action., Mario takes coins by making a move to UP.

5. **Reward (R_{t+1})**: A scalar feedback signal the agent receives from the environment after performing an action. The reward indicates how good or bad the action was in that state. In chess, a reward might be +1 for winning, -1 for losing, and 0 for all other moves until the end of the game. Mario will get a reward of +1 when a coin is collected, a reward (penalty) of -10000 when he hits Goomba.

6. **Policy (π)**: The agent's strategy or rule for choosing an action given a state. It maps states to actions. $\pi(s, a) \mapsto [0, 1]$. It is a probabilistic function $\pi(a|s) = \mathbb{P}(A = a|S = s)$ ie. it is the probability of taking an action $A = a$ given state $S = s$. The probabilities output by the policy function guides the agent to make decisions. The objective of RL is to find the best policy possible.

---

2 The Goal: Maximizing Cumulative Reward

The central goal of the agent is not to maximize the immediate reward, but the **total cumulative reward** it receives over the long run. This cumulative reward is called the **return**.

To handle potentially infinite sequences of rewards, we often use a **discounted return**. A discount factor, denoted by γ (gamma), where $0 \leq \gamma \leq 1$, is introduced. It determines the present value of future rewards. A reward received k steps in the future is only worth γ^{k-1} times what it would be worth if it were received immediately.

The discounted return (U_t) from time step t is defined as:

$$U_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

The agent's objective is to find an optimal policy (π^*) that maximizes this expected discounted return for every state.

---

3 The Agent-Environment Interaction Cycle

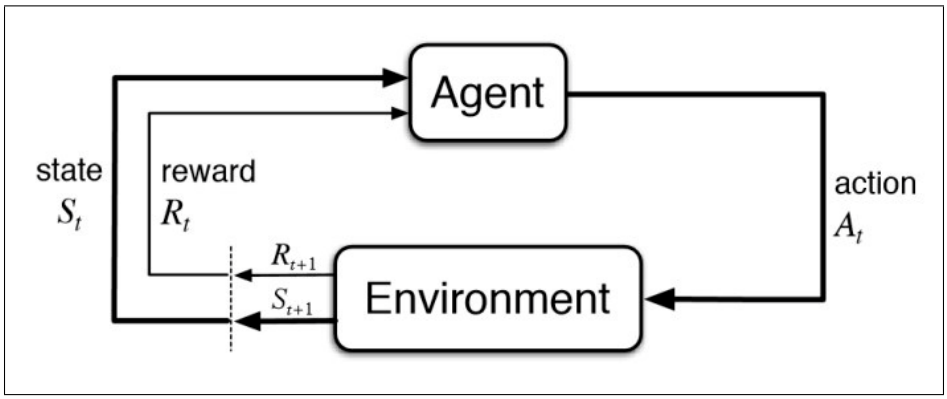

Figure 1: The reinforcement learning agent-environment interaction loop.

The reinforcement learning process unfolds in a sequence of discrete time steps:

1. **Observe**: At time t, the agent observes the current state of the environment, S_t.

2. **Act**: Based on its policy π, the agent selects an action, A_t. More precisely, the action is randomly sampled from a probability distribution. $A \sim \pi(.|S)$

3. **Transition & Reward**: The agent performs the action A_t. The environment then transitions to a new state, S_{t+1}, and provides the agent with a reward, R_{t+1}. The state transition can be random or deterministic. Typically we assume that the state transitions are random. The randomness in the state transition comes from the environment (for eg: the position of enemies or coins in the next frame of Mario game is random). The function $p(S_{t+1}|S_t, a_t) = \mathbb{P}(S_{t+1} = s_{t+1}|S_t, A_t = a_t)$ outputs the probability of the new state given old state and the action. More precisely, the next state is randomly sampled from the state transition function $S_{t+1} \sim p(.|S_t, A_t)$

4. **Learn**: The agent uses the information from this interaction (the new state and the reward) to update its policy, hopefully improving it.

5. **Repeat**: This cycle continues, allowing the agent to learn from experience. $s_1 a_1 r_1, s_2 a_2 r_2, \ldots s_n a_n r_n$

Figure 2 shows agent environment interaction cycle in the case of a chess game.

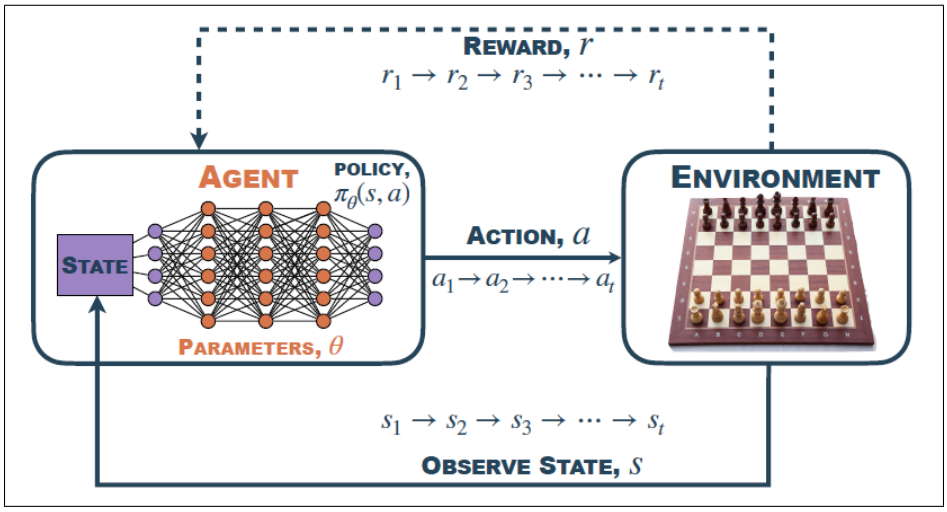

Figure 2: An agent senses (observes) its environmental state s and takes actions a according to a policy π. that is optimized through learning to maximize the future rewards r. In this case, a deep neural network is used to represent the policy pi. This is known as *deep policy network*

4 Randomness in Returns

Consider the game till its end, at the end of the game, we observe all rewards $r_t, r_{t+1}, r_{t+2} \ldots . r_n$. We then know the discounted return $U_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+1} \ldots + \gamma^{n-t} r_n$. At time t, (the game has not ended yet) the rewards $R_t, R_{t+1}, \ldots, R_n$ are random, so the return U_t is random. We have not yet observed the rewards, the rewards are random variables. Let's analyze the source of randomness:

- Reward R_i depends on S_i and A_i
- States are random $S_i \sim p(.|S_{i-1}, A_{i-1})$
- Actions are random $A_i \sim \pi(.|S_i)$

If either S_i or A_i is random, then R_i is random. Reward R_i depends on S_i and A_i. U_t depends on $R_t, R_{t+1}, \ldots, R_n$. Therefore U_t depends on all the states and actions from time t. ie, $S_t A_t, S_{t+1} A_{t+1} \ldots$. Suppose the game has ended, we observe $r_t, r_{t+1}, \ldots, r_n$. U_t is then just a number. Recall that any observation of a random variable is a scalar.

5 Action-Value and State-Value Function in Reinforcement Learning

5.1 Action-Value Function

The **Action-Value function** in Reinforcement Learning, also known as the Q-function and commonly denoted as $Q_\pi(s, a)$, represents the **expected cumulative future rewards** an agent can obtain by taking a specific action a in a specific state s and then following a particular policy thereafter.

The Action-Value function is central to many reinforcement learning algorithms, such as Q-learning. It quantifies not just the benefit of being in a state, but the benefit of choosing a particular action in that state. The **main goal** in typical RL applications is to estimate this function as accurately as possible in order to select actions that maximize the agent's long-term reward.

- $Q_\pi(s, a)$ is the expected sum of rewards, starting from state s, taking action a, and then following a given policy π.
- Formally:
$$Q_\pi(s, a) = \mathbb{E}[U_t \mid S_t = s, A_t = a]$$
where U_t represents the total future rewards received, possibly discounted.

The action-value function $Q_\pi(s, a)$ answers: If Mario does action "a" (say, jumps) in situation "s" (standing on a pipe with Goomba near), how much total reward (like coins, survival, score, progress) can he expect to get from that point onward if he keeps playing wisely? Suppose Mario is at the edge of a pit (state "s"):

- If he jumps forward ("a1"), the Q-function estimates: "If Mario jumps, what's the expected total reward? Maybe he safely lands, grabs a coin, and avoids an enemy, so the expected reward is high."
- If he stands still ("a2"), the Q-function estimates: "If Mario stands still, maybe he gets hit by the approaching Goomba and loses a life, so the expected reward is low."

Optimal Action-Value Function

The optimal action-value function is defined as:

$$Q^*(s, a) = \max_\pi Q_\pi(s, a)$$

Bellman Equation for $Q_\pi(s, a)$

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

Bellman Optimality Equation for $Q^*(s, a)$

$$Q^*(s, a) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} Q^*(S_{t+1}, a') \mid S_t = s, A_t = a\right]$$

Update Rule (For example, in Q-learning)

The Q-values are updated iteratively using experiences of the form (s, a, r, s'):

$$Q(s, a) \leftarrow Q(s, a) + \alpha\left[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right]$$

Here:

- α is the learning rate,
- γ is the discount factor,
- r is the reward received after executing a in s,
- s' is the new state,
- a' are all possible actions in s'.

5.2 State-Value Function

In Reinforcement Learning, the **State-Value Function**, denoted by $V_\pi(s)$, represents the expected return (i.e., cumulative future reward) starting from a given state s, and following a policy π thereafter.

Definition

$$V_\pi(s) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \mid S_0 = s\right]$$

or

$$V_\pi(s) = \mathbb{E}[U_t \mid S_t = s]$$

Where:

- $V_\pi(s)$: Value of state s under policy π.

- $\mathbb{E}_\pi$: Expectation over trajectories generated by policy π.
- $\gamma \in [0, 1]$: Discount factor for future rewards.
- R_{t+1}: Reward received at time step $t + 1$.
- $S_0 = s$: Starting from state s at time $t = 0$.

Interpretation

The state-value function tells us how good it is to be in a particular state s if the agent acts according to policy π from that point onward. It is used to evaluate and improve policies in reinforcement learning algorithms.

5.3 Relationship between $V_\pi(s)$ & $Q_\pi(s, a)$

- State-Value function $V_\pi(s)$ answers the question: **How good is it to be in this state?**
- Action-Value function $Q_\pi(s, a)$ answers the question: **How good is it to take this action from this state?**

State-Value Function

The state-value function can be expressed as the expected value of the action-value function $Q_\pi(s, a)$ over all possible actions from that state, according to the policy π.

$$V_\pi(s) = \sum_{a \in A} \pi(a|s) * Q_\pi(s, a)]$$

Derivation

The derivation is a straightforward application of the law of total expectation. Starting with

$$V_\pi(s) = \mathbb{E}_\pi[U_t \mid S_t = s]$$

The overall expected return from a state s can be broken down by considering all possible actions it could take from s. We can express the expectation as a sum over all actions a from the action set A, weighted by the probability of taking each action according to the policy $\pi(a|s)$.

$$\mathbb{E}_\pi \left[U_t \mid S_t = s\right] = \sum_{a \in A} \pi(a|s) \, \mathbb{E}_\pi \left[U_t \mid S_t = s, A_t = a\right]$$

We know that $\mathbb{E}_\pi\left[U_t \mid S_t = s, A_t = a\right]$ is the action-value function $Q_\pi(s,a)$ hence

$$V_\pi(s) = \sum_{a \in A} \pi(a|s) * Q_\pi(s,a)]$$

Mario is at the edge of a pit (state "s"), the overall *goodness* being at this state is $V_\pi(s)$. The goodness of choosing a specific path (e.g., up action leads to possible collection of coins and avoiding goomba) is its action-value, $Q_\pi(s, up)$; meanwhile which path to choose is based on the policy $\pi(a|s)$. To calculate the overall goodness of Mario's current location (V), Mario would average the goodness of each path (Q), but Mario would give more weight to the paths he is more likely to take. This is exactly what the formula does.

Action-Value Function

We can derive the action-value function $Q_\pi(s,a)$ from the state-value function $V_\pi(s)$ by looking one step into the future. The value of taking a specific action is the immediate reward you get, plus the discounted value of the state you land in. We know;

$$Q_\pi(s,a) = \mathbb{E}[U_t \mid S_t = s, A_t = a]$$

Also, the total return U_t is the immediate reward R_{t+1} plus the discounted future return $\gamma\, U_{t+1}$. ie;

$$U_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \ldots + \gamma^{n-t+1} R_n = R_{t+1} + \gamma\, U_{t+1}$$

substituting in $Q_\pi(s,a)$

$$Q_\pi(s,a) = \mathbb{E}[R_{t+1} + \gamma\, U_{t+1} \mid S_t = s, A_t = a]$$

Now, using the law of linearity of expectations, we can split into two parts

$$Q_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} \mid S_t = s, A_t = a] + \gamma\, \mathbb{E}_\pi[\, U_{t+1} \mid S_t = s, A_t = a]$$

recognizing the term $\mathbb{E}_\pi[\, U_{t+1} \mid S_t = s, A_t = a]$ represents the expected return from the next state, S_{t+1}. This is, by definition, the value of the next state $V_\pi(S_{t+1})$. Now the equation becomes

$$Q_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma\, V_\pi(S_{t+1}) \mid S_t = s, A_t = a]$$

To make this concrete, we expand the expectation. We sum over all possible next states s' and rewards r, weighted by the probability $p(s', r \mid s, a)$ of that outcome occurring. This gives us the final, explicit formula.

$$Q_\pi(s,a) = \sum_{s',r} p(s', r \mid s, a)[r + \gamma V_\pi(s')]$$

6 The Core Challenge: Exploration vs. Exploitation

A fundamental challenge in reinforcement learning is the trade-off between **exploration** and **exploitation**.

- **Exploitation**: The agent makes the best decision it can, given its current knowledge. It chooses actions that it knows will result in a high reward.
- **Exploration**: The agent tries new, seemingly suboptimal actions to gain more information about the environment. This might lead to discovering even better actions in the long run.

To obtain a lot of reward, an agent must prefer actions that it has tried in the past and found to be effective. But to discover such actions, it has to try actions that it has not selected before. The agent must **explore** a variety of actions to find the best ones but also **exploit** what it already knows to get a reward. Balancing these two is crucial for a successful RL agent.