1. Reinforcement Learning: Sequential Decision Making

Notes based on Sutton & Barto

March 29, 2025

Contents

1 Index of Topics

2 Introduction to Sequential Decision Making

Reinforcement learning is fundamentally about sequential decision making - the study of how agents learn to make a sequence of decisions that maximize cumulative reward over time. Unlike supervised learning where correct input-output pairs are presented, in reinforcement learning the agent must learn by interacting with its environment and discovering which actions yield the most reward.

The key challenge in sequential decision making is that decisions not only affect immediate rewards but may also influence future situations and, consequently, future rewards. This creates a temporal credit assignment problem: how do we attribute credit or blame to decisions that have delayed consequences?

2.1 Multi-armed Bandit Problem

The multi-armed bandit problem represents the simplest form of sequential decision making. The name comes from imagining a gambler at a row of slot machines (known as "one-armed bandits"), who must decide which machines to play, how many times to play each machine, and in which order to play them, to maximize reward.

This problem formulation is fundamental to reinforcement learning for several reasons:

- It isolates the exploration-exploitation dilemma that is central to all reinforcement learning problems
- It provides a simplified setting with no state transitions, allowing us to focus solely on action selection
- It has important real-world applications such as clinical trials, website optimization, and adaptive routing

While simplified, the multi-armed bandit captures the essential challenge of learning from interaction: we must make decisions based on limited information, and our choices affect what information we receive in the future. This creates a feedback loop between learning and decision making.

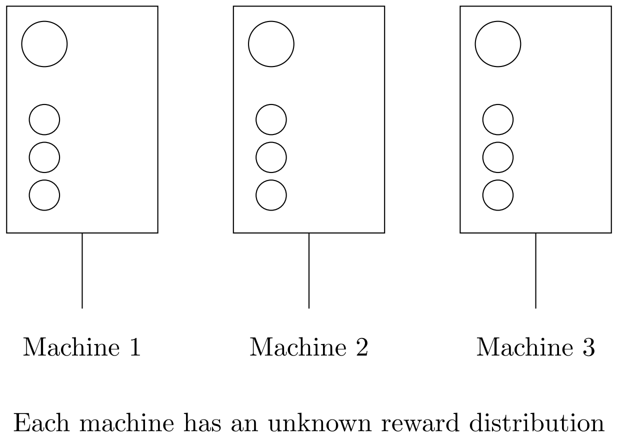

Figure 1: Illustration of a 3-armed bandit problem

2.1.1 Problem Formulation

In the multi-armed bandit problem:

- You face k different options or "arms"
- Each time step t, you select one arm $A_t \in \{1, 2, ..., k\}$
- After choosing arm a, you receive a reward R_t drawn from a probability distribution associated with that arm
- The goal is to maximize the total reward $\sum_{t=1}^{T} R_t$ over some time horizon T

In mathematical terms, choosing an arm a at time t is formalized as an action from a set $\mathcal{A}$. When we choose action A_t, we receive a reward R_t drawn from an unknown probability distribution $p(r|a)$.

$$\text{Action selection:} \quad A_t \in \mathcal{A} \tag{1}$$

Notation Overview

- A_t: The action chosen at time t
- $\mathcal{A}$: The set of all possible actions
- $p(r|a)$: The probability distribution of reward r given action a
- R_t: The reward received at time t after taking action A_{t-1} (following Sutton & Barto's indexing)

2.2 Actions, Rewards, and Policies

In the bandit problem, the agent's objective is to maximize the expected reward over time. The three fundamental components are:

2.2.1 Actions

Actions represent the choices available to the agent. In the multi-armed bandit setting, an action corresponds to selecting one of the k arms. We denote the action taken at time t as $A_t \in \mathcal{A}$, where $\mathcal{A}$ is the set of all possible actions.

2.2.2 Rewards

After taking action A_t at time t, the agent receives a numerical reward R_{t+1}. This reward is sampled from a probability distribution associated with the selected action: $R_{t+1} \sim p(r|A_t)$. The distribution $p(r|a)$ is initially unknown to the agent and must be learned through experience.

2.2.3 Policies

A policy defines the agent's strategy for selecting actions. In the context of bandits, a policy π is a mapping from actions to selection probabilities:

$$\pi(a) = P(A_t = a) \tag{2}$$

Notation Overview

- $\pi(a)$: The probability of selecting action a under policy π
- $P(A_t = a)$: The probability that the action at time t equals a

We can define more sophisticated policies that depend on past actions and rewards, but in the most basic form, a policy simply assigns probabilities to each action.

2.3 Evaluative Feedback

In reinforcement learning, the agent receives evaluative feedback rather than instructive feedback. The distinction is crucial:

- **Instructive feedback** (as in supervised learning) tells the agent what the correct action would have been
- **Evaluative feedback** only indicates how good the selected action was, but not whether it was the best possible action or what the best action would have been

To understand the difference more clearly:

- In supervised learning for image classification, if the model predicts "cat" when the true label is "dog", the model receives the correct label "dog" as feedback.
- In reinforcement learning, if an agent takes action A_t and receives reward R_{t+1}, it only knows the value of that specific actionnot what reward it would have received had it taken a different action or which action would have been optimal.

This approach mirrors many real-world learning scenarios. When a child touches a hot stove, they learn that this action led to pain, but they don't automatically know what alternative action would have been better. They must explore different approaches to build knowledge through experience.

Evaluative feedback forces the agent to actively explore to gather information about the environment, rather than passively receiving correct answers. This exploration process is what makes reinforcement learning fundamentally different from other machine learning paradigms.

The key challenge is to find the action a that produces the maximum expected reward:

$$\max_a \mathbb{E}[p(r|a)] \tag{3}$$

Notation Overview

- $\mathbb{E}[p(r|a)]$: The expected value of the reward distribution for action a
- $\max_a$: The operation of finding the action a that maximizes the subsequent expression

The true expected reward of each action is unknown, so the agent must estimate it from experience. This leads to the concept of action value.

2.3.1 Action Value

The value of an action a, denoted $q(a)$, is the expected reward when that action is selected:

$$q(a) = \mathbb{E}[R_t | A_{t-1} = a] \tag{4}$$

Notation Overview

- $q(a)$: The true value of action a (expected reward)
- $\mathbb{E}[R_t | A_{t-1} = a]$: The expected reward at time t given that action a was chosen at time $t - 1$

In practice, we estimate $q(a)$ based on observed rewards. Let $Q_t(a)$ denote our estimate of $q(a)$ at time t. As we gather more experience, we refine this estimate to approach the true value.

$$Q_t(a) = \frac{1}{N_t(a)} \sum_{i=1}^{t} R_i \cdot \mathbf{1}_{A_{i-1}=a} \tag{5}$$

Notation Overview

- $Q_t(a)$: The estimated value of action a at time t
- $N_t(a)$: The number of times action a has been selected up to time t
- R_i: The reward received at time i
- $\mathbf{1}_{A_{i-1}=a}$: An indicator function that equals 1 if $A_{i-1} = a$ and 0 otherwise

2.4 Incremental Update of Action-Values

Calculating $Q_t(a)$ using the formula above would require storing all past rewards, which becomes inefficient as t grows. Instead, we can update our estimate incrementally after each new observation.

Incremental updating is a fundamental concept in reinforcement learning for several important reasons:

- **Memory efficiency**: We don't need to store the entire history of rewards for each action
- **Computational efficiency**: We can update estimates with constant time complexity, regardless of how many steps we've taken so far
- **Online learning**: We can immediately incorporate new information as it becomes available

- **Non-stationarity**: With appropriate step sizes, we can track changing reward distributions over time

The incremental update approach is also aligned with how humans and animals appear to learn from experience. We don't reprocess our entire history of experiences when we learn something new; instead, we adjust our existing understanding based on new observations.

Let's derive the incremental update rule:

$$Q_{t+1}(a) = \frac{1}{N_{t+1}(a)} \sum_{i=1}^{t+1} R_i \cdot \mathbf{1}_{A_{i-1}=a} \tag{6}$$

$$= \frac{1}{N_{t+1}(a)} \left(R_{t+1} \cdot \mathbf{1}_{A_t=a} + \sum_{i=1}^{t} R_i \cdot \mathbf{1}_{A_{i-1}=a} \right) \tag{7}$$

$$= \frac{1}{N_{t+1}(a)} \left(R_{t+1} \cdot \mathbf{1}_{A_t=a} + N_t(a) \cdot Q_t(a) \right) \tag{8}$$

If $A_t = a$, then $N_{t+1}(a) = N_t(a) + 1$ and the update becomes:

$$Q_{t+1}(a) = Q_t(a) + \frac{1}{N_{t+1}(a)} [R_{t+1} - Q_t(a)] \tag{9}$$

Notation Overview

- $Q_{t+1}(a)$: The updated estimate of the value of action a after observing the $(t+1)$-th reward
- $Q_t(a)$: The previous estimate of the value of action a
- R_{t+1}: The reward received at time $t+1$
- $\frac{1}{N_{t+1}(a)}$: The learning rate, which decreases as we observe more samples of action a

This is known as the incremental update rule. It adjusts the current estimate $Q_t(a)$ by a fraction of the prediction error $[R_{t+1} - Q_t(a)]$. The prediction error represents the difference between the observed reward and the expected reward.

A more general form replaces $\frac{1}{N_{t+1}(a)}$ with a step-size parameter $\alpha \in (0, 1]$:

$$Q_{t+1}(a) = Q_t(a) + \alpha [R_{t+1} - Q_t(a)] \tag{10}$$

Notation Overview

- α: The step-size parameter or learning rate, controlling how quickly the estimates are updated
- $[R_{t+1} - Q_t(a)]$: The prediction error or TD error

Using a constant α can be advantageous in non-stationary problems where the true action values change over time. The constant step size gives more weight to recent rewards, allowing the estimates to track changing values.

2.5 Exploitation vs. Exploration

The multi-armed bandit problem presents a fundamental dilemma: exploitation versus exploration.

- **Exploitation** means selecting the action that currently appears best based on accumulated knowledge
- **Exploration** means trying different actions to gather more information about their values

If we always exploit (by selecting the action with the highest estimated value), we risk missing better actions whose value we have underestimated due to limited samples. If we spend too much time exploring, we may waste opportunities to gain higher rewards from the best actions.

The exploration-exploitation dilemma extends far beyond reinforcement learning:

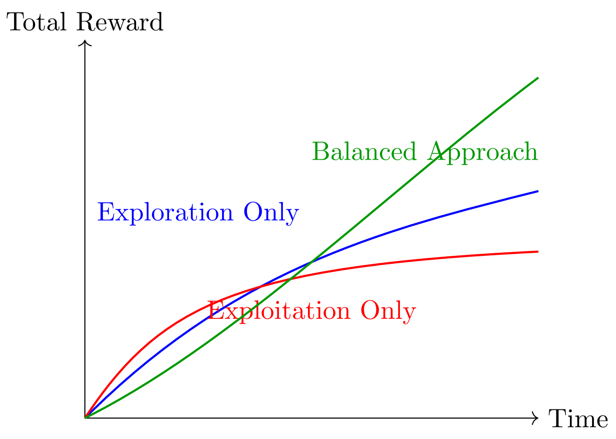

Figure 2: Conceptual illustration of different exploration-exploitation strategies and their long-term reward outcomes

- A restaurant-goer must decide whether to visit a new restaurant (exploration) or return to a favorite one (exploitation)
- A business must choose between investing in new products (exploration) or optimizing existing ones (exploitation)
- A researcher must decide whether to pursue novel research directions (exploration) or build on established findings (exploitation)

Finding the right balance is criticaltoo much exploitation leads to suboptimal solutions, while too much exploration wastes resources on unpromising options. Different contexts require different balances, and the optimal balance often changes over time as knowledge accumulates.

2.5.1 Greedy Action Selection

The simplest policy is to always select the action with the highest estimated value:

$$A_t = \arg\max_a Q_t(a) \tag{11}$$

Notation Overview

- $\arg\max_a Q_t(a)$: The action a that maximizes the estimated value $Q_t(a)$

This policy is known as the greedy policy. It always exploits current knowledge but never explores, which can lead to suboptimal performance if the initial estimates are inaccurate.

2.6 Action Sampling Methods

Several methods have been developed to balance exploration and exploitation. Each has unique characteristics that make it suitable for different scenarios:

2.6.1 ϵ-Greedy Selection

The ϵ-greedy approach introduces random exploration by selecting a random action with probability ϵ and the greedy action with probability $1-\epsilon$:

$$\pi(a) = \begin{cases} 1-\epsilon & \text{if } a = \arg\max_a Q_t(a) \\ \frac{\epsilon}{|\mathcal{A}|-1} & \text{otherwise} \end{cases} \tag{12}$$

Notation Overview

- $\pi(a)$: Probability of selecting action a
- ϵ: The exploration rate ($0 \leq \epsilon \leq 1$)
- $|\mathcal{A}|$: The number of actions (cardinality of the action set)
- $\arg\max_a Q_t(a)$: The action with the highest estimated value

The ϵ-greedy approach ensures that every action has some probability of being selected, which allows the agent to continue learning about all actions.

Advantages of ϵ-greedy:

- Simple to implement and understand
- Computationally efficient
- Guaranteed to eventually visit all actions
- Can be effective in practice despite its simplicity

Implementation considerations:

- A common practice is to start with a high value of ϵ (e.g., 0.1) and gradually decrease it over time
- This approach, called ϵ-*decay*, allows more exploration early in learning when uncertainty is high
- As more information is gathered, ϵ decreases to favor exploitation

Algorithm 1 ϵ-Greedy Action Selection

```
1: procedure EpsilonGreedy($Q_t, \epsilon$)
2:     $p \leftarrow$ Random(0, 1)
3:     if $p < \epsilon$ then
4:         return Random action from $\mathcal{A}$
5:     else
6:         return $\arg\max_a Q_t(a)$
7:     end if
8: end procedure
```

2.6.2 Softmax Selection

While ϵ-greedy treats all non-greedy actions equally, the softmax approach (also known as Boltzmann exploration) selects actions with probabilities relative to their estimated values:

$$\pi(a) = \frac{e^{Q_t(a)/\tau}}{\sum_{a' \in \mathcal{A}} e^{Q_t(a')/\tau}} \tag{13}$$

Notation Overview

- $\pi(a)$: Probability of selecting action a
- $e^{Q_t(a)/\tau}$: The exponential of the estimated value divided by temperature
- τ: The temperature parameter controlling the randomness of the selection
- $\sum_{a' \in \mathcal{A}} e^{Q_t(a')/\tau}$: Normalization term to ensure probabilities sum to 1

The temperature parameter τ controls the degree of exploration:

- As $\tau \to 0$, softmax approaches greedy selection
- As $\tau \to \infty$, softmax approaches uniform random selection

2.6.3 Upper Confidence Bound (UCB) Selection

The UCB algorithm addresses the exploration-exploitation dilemma by selecting actions based not only on their estimated values but also on the uncertainty in those estimates:

$$A_t = \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right] \tag{14}$$

Notation Overview

- $Q_t(a)$: The estimated value of action a at time t
- c: A parameter controlling the degree of exploration
- $\ln t$: Natural logarithm of the current time step
- $N_t(a)$: The number of times action a has been selected up to time t
- $\sqrt{\frac{\ln t}{N_t(a)}}$: The uncertainty term, which increases with time and decreases with the number of selections

The UCB approach favors actions that either have high estimated values or have been selected infrequently. As an action is selected more often, its uncertainty term decreases, reducing the incentive for further exploration unless its estimated value is truly high.

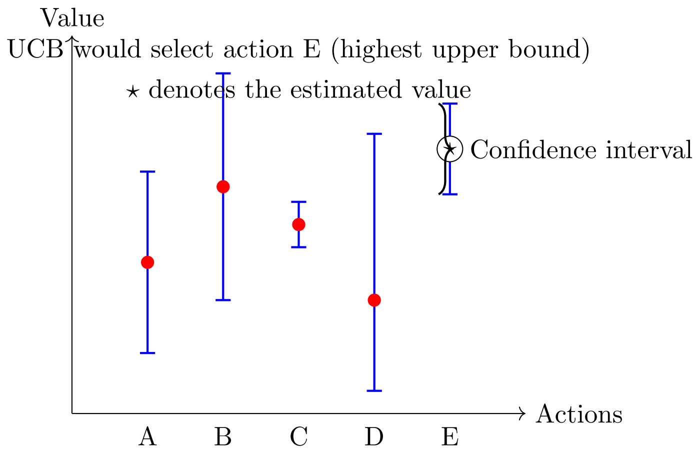

Figure 3: Illustration of Upper Confidence Bound selection

The UCB algorithm has several important theoretical and practical advantages:

- It is **principled**: The exploration term is derived from statistical confidence bounds
- It is **adaptive**: Unlike ϵ-greedy, which explores randomly, UCB directs exploration toward actions with high uncertainty
- It has **regret guarantees**: Under certain conditions, UCB algorithms have provably optimal regret bounds
- It is **parameter-sensitive**: The constant c controls the balance between exploration and exploitation

UCB exemplifies the principle of *optimism in the face of uncertainty*: by optimistically selecting actions based on the upper bound of their confidence intervals, the algorithm efficiently explores the action space and converges to the optimal policy.

3 Summary

Sequential decision making forms the foundation of reinforcement learning. The multi-armed bandit problem provides a simplified setting where:

- An agent repeatedly chooses among k actions
- Each action yields a reward from an unknown distribution
- The agent must balance exploration (trying different actions to learn their values) and exploitation (selecting the best-known action to maximize reward)

Key concepts include:

- Action values and their estimation
- Incremental updating of estimates
- Exploration strategies like ϵ-greedy, softmax, and UCB

These concepts extend to more complex reinforcement learning problems where actions also affect the environment state, leading to Markov Decision Processes and the full reinforcement learning framework.