Minor in AI

Revision

Reinforcement Learning

May 06, 2025

1 Branches of Machine Learning (ML)

Machine Learning (ML) can be broadly categorized into three branches:

- **Supervised Learning (SL)**: The model learns from labeled data. The goal is to learn a mapping from input to output, where the output (label) is known during training.
- **Unsupervised Learning (UL)**: The model learns from data that is unlabeled. The goal is to identify patterns or groupings in the data.
- **Reinforcement Learning (RL)**: In RL, the agent learns from interacting with an environment. It tries to maximize a cumulative reward signal, often over time, by taking actions based on the current state of the environment.

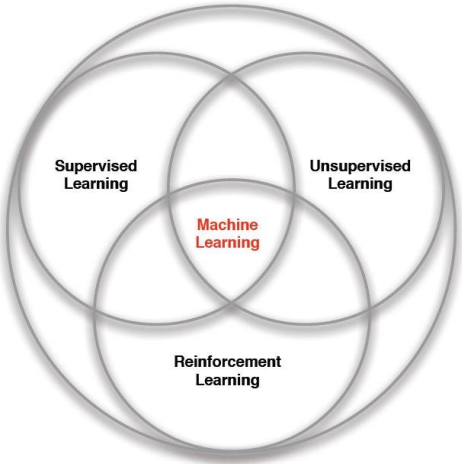

Figure 1: Branches of ML

2 Comparison of ML Techniques

Aspect	Supervised Learning	Unsupervised Learning	Reinforcement Learning
Data and Feedback	Labeled data, feedback during training	Unlabeled data, no feedback	Interaction, rewards as feedback
Learning Process	Learn from labeled data to predict	Find patterns in data	Learn from environment interactions
Goal	Minimize prediction error	Discover patterns	Maximize cumulative reward

3 Supervised, Unsupervised, and Reinforcement Learning

3.1 Supervised Learning (SL)

In supervised learning, the algorithm learns from a training dataset that contains labeled data, i.e., input-output pairs. The objective is to learn a mapping from inputs to outputs.

3.2 Unsupervised Learning (UL)

In unsupervised learning, the model is given unlabeled data and must find patterns or structures within the data. Examples include clustering and dimensionality reduction.

3.3 Reinforcement Learning (RL)

In reinforcement learning, an agent learns to make decisions by interacting with an environment. The agent receives feedback in the form of rewards or penalties, and its goal is to learn an optimal policy for decision-making to maximize cumulative rewards.

4 Sequential Decision Making in RL

In RL, the problem can be viewed as a process of sequential decision-making. The agent interacts with the environment, takes actions, and receives feedback (rewards). The agent aims to make decisions that maximize the cumulative reward over time.

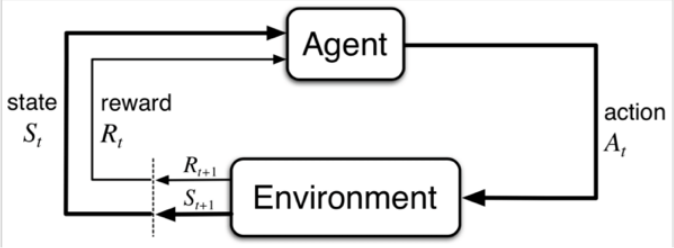

Figure 2: Sequential Decision Making

4.1 Reinforcement Learning Framework

Reinforcement learning can be formalized as a Markov Decision Process (MDP).

4.1.1 Markov Decision Process (MDP)

An MDP is a mathematical framework used to model decision-making problems. It is defined by the following components:

- **States** (S): A set of states that describe the environment.
- **Actions** (A): A set of actions the agent can take.
- **Transition Function** (T): Probability distribution over next states given the current state and action, i.e., $T(s, a, s')$.
- **Reward Function** (R): A function that gives the immediate reward when transitioning from state s to state s' after taking action a.

- **Discount Factor** (γ): A factor that discounts future rewards, typically between 0 and 1.

The objective in RL is to learn a policy $\pi(a|s)$, which specifies the probability of taking action a in state s, that maximizes the expected cumulative reward over time.

5 One-Armed and Multi-Armed Bandit Problem

5.1 One-Armed Bandit Problem

The one-armed bandit is a simplified version of an RL problem where the agent must decide which action to take in a single slot machine (bandit). Each action gives a reward from a fixed, but unknown, distribution. The agent must balance exploration (trying different actions) and exploitation (choosing the best-known action).

5.2 Multi-Armed Bandit (MAB) Problem

In the MAB problem, the agent faces multiple slot machines, each with its own reward distribution. The goal is to maximize the total reward by selecting the best arm (slot machine) over time. The agent must again balance exploration and exploitation.

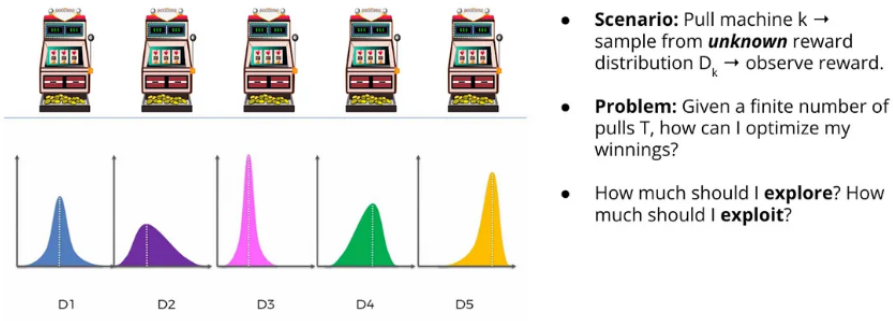

Figure 3: Multi Armed Bandit Problem

5.3 MAB Solution - Numerical Example

Let's consider a scenario with two slot machines (arms), each with different expected rewards:

$$\text{Arm 1: Expected reward} = 0.5$$

$$\text{Arm 2: Expected reward} = 0.7$$

The agent initially has no knowledge of the rewards and must decide which arm to pull. By using exploration (trying both arms), the agent gathers data to update its understanding and gradually shifts towards exploiting the arm with the higher expected reward (Arm 2).

5.4 Why Use Recursive Update?

In the MAB problem, recursive updates help the agent continuously improve its estimates of the expected rewards for each arm. This enables the agent to make better decisions over time by adjusting its estimates based on new experiences.

5.5 Deriving Recursive Update

Let $Q(a)$ denote the estimated value of action a (the expected reward). The recursive update for $Q(a)$ after taking action a and receiving reward r is:

$$Q(a) \leftarrow Q(a) + \alpha(r - Q(a))$$

where α is the learning rate, which controls how much the new reward should influence the estimate. This formula updates the estimate of $Q(a)$ based on the observed reward r and the current estimate.

The value of Q_n is the average of the rewards over n steps:

$$Q_n = \frac{1}{n} \sum_{i=1}^{n} R_i$$

$$Q_n = \frac{1}{n} \left(R_n + \sum_{i=1}^{n-1} R_i \right)$$

$$\sum_{i=1}^{n-1} R_i = (n-1) Q_{n-1}$$

$$Q_n = \frac{1}{n} \left(R_n + (n-1) Q_{n-1} \right)$$

$$Q_n = Q_{n-1} + \frac{1}{n} \left(R_n - Q_{n-1} \right)$$

Thus, the recursive update formula for $Q(a)$ is:

$$Q(a) \leftarrow Q(a) + \alpha(r - Q(a))$$

where $\alpha = \frac{1}{n}$.

6 Epsilon-Greedy Algorithm

6.1 Concept

The epsilon-greedy algorithm is a simple strategy for balancing exploration and exploitation in MAB problems. In each step, the agent either:

- **Exploits**: Chooses the arm with the highest estimated reward with probability $1 - \epsilon$.
- **Explores**: Chooses a random arm with probability ϵ.

Here, ϵ is a small value (e.g., 0.1), which controls the amount of exploration. If ϵ is set to 0, the agent will always exploit. If ϵ is set to 1, the agent will explore randomly.

6.2 Interpretation of Formula

The epsilon-greedy algorithm can be interpreted as a trade-off between exploitation and exploration. The update formula for $Q(a)$ in the epsilon-greedy setting is similar to the standard recursive update:

$$Q(a) \leftarrow Q(a) + \alpha\,(r - Q(a))$$

but the action selection process depends on ϵ. The agent chooses the best action with high probability but occasionally selects a random action to explore other possibilities.

Algorithm 2: Epsilon-Greedy Action Selection

Data: Q: Q-table generated so far, : a small number, S: current state
Result: Selected action
Function *SELECT-ACTION(Q, S, ϵ)* **is**
    n ← uniform random number between 0 and 1;
    if $n < \epsilon$ **then**
        A ← random action from the action space;
    else
        A ← maxQ(S,.);
    end
    return selected action A;
end

Figure 4: Algorithm