Artificial Neural Networks and Deep Learning - Notes - v0.1.0

260236

October 2025

Preface

Every theory section in these notes has been taken from two sources:

- Ian Goodfellow and Yoshua Bengio and Aaron Courville, Deep Learning, MIT Press. [1]
- Course slides. [2]

About:

GitHub repository

These notes are an unofficial resource and shouldn't replace the course material or any other book on artificial neural networks and deep learning. It is not made for commercial purposes. I've made the following notes to help me improve my knowledge and maybe it can be helpful for everyone.

As I have highlighted, a student should choose the teacher's material or a book on the topic. These notes can only be a helpful material.

Contents

1 Introduction to Deep Learning

1.1 Machine Learning Foundations

Humans and animals learn from experience. Computers, too, can improve performance when exposed to more data or feedback. But how do we formally define "learning" in a way that's precise enough for a engineering course? Tom Mitchell[1], in 1997, proposed a now-classic definition:

Definition 1: Task, Experience, Performance

A computer program is said to learn from experience **E** with respect to some class of tasks **T** and a performance measure **P**, if its performance at tasks in **T**, as measured by **P**, improves with experience **E**.

- **Task (T)**: what the program is supposed to do. For example, classification (spam vs not spam), regression (predict house prices) or game playing (chess).
- **Experience (E)**: the data the algorithm is exposed to. For example, training set of labeled emails (spam vs ham), past games played by an agent, sensor data from a robot.
- **Performance measure (P)**: the metric used to evaluate progress. For example, classification accuracy (F1 score), mean square error for regression, total reward in reinforcement learning.

A system "learns" if, after seeing more data or interacting more with the environment, its **measured performance improves**.

Example 1: Definition in Action

Some scenarios:

1. **Email Spam Classifier**
   - **T (task)**: Classify emails as spam.
   - **E (experience)**: Training dataset of emails labeled as spam.
   - **P (performance measure)**: Accuracy on unseen emails.

   If accuracy improves as the classifier sees more labeled data, then computer program learning.

2. **Self-Driving Car**
   - **T**: Driving from A to B safely.
   - **E**: Millions of hours of driving footage + sensor readings.
   - **P**: Fewer accidents per mile, shorter trip times.

   If the car improves after more data, it has learned.

[1] Tom Mitchell is a *pioneer of machine learning*, both as a scientist and as an educator. His 1997 textbook, and especially that concise definition, shaped how an entire generation of students and researches understand Machine Learning (ML).

3. **Chess Playing Agent**
   - **T**: Win games.
   - **E**: Past games played against itself or others.
   - **P**: Win rate.

   More games, better play, computer program learning.

This definition matters because is **broad and general** (covers supervised, unsupervised, and reinforcement learning), it stresses **measurable improvement** (no improvement, no learning), and highlights the **central role of data** (E) and evaluation (P).

Why Mitchell's definition doesn't mentions "Machine Learning" explicitly

1. **It's meant to be general**. Mitchell wasn't defining *what ML is as a field*, but rather *what it means for a program to learn*. He avoided vague terms like "machine learning" or "artificial intelligence" and instead described the *process*:
   - A program improves at a **Task (T)**;
   - Thanks to **Experience (E)**;
   - As measured by **Performance (P)**.
2. **Machine Learning = building such programs**. So instead of saying "*Machine Learning is when...*", he framed it as: "*a computer program is said to learn if...*". That's why his definition became the **canonical operational definition of Machine Learning**.
3. **It links directly to practice**. The definition is testable: we can check if a system improves with experience. This is much stronger than a philosophical definition like "*machine learning is making computers intelligent*".

Example 2: Analogy

Think of physics. Newton didn't define "physics". He defined *laws of motion* and *gravity*. From those definitions, physics as a discipline could build itself consistently.
Similarly, Mitchell didn't define "Machine Learning" as a whole discipline. He defined **what it means for a program to learn**. The field then said: "Machine Learning is the study of programs that satisfy this definition".

Mitchell's definition tells us ML is **not about hardcoding solutions**, but about **improving performance with data-driven experience**, measurable by a task-specific metric.

Why we start with Tom Mitchell's definition

1. **Machine Learning is broad and fuzzy**. People use "learning", "AI", "intelligence" loosely. By giving a **formal, authoritative definition** at the beginning, the course sets a *clear baseline*: what do we mean by *learning*? How do we recognize it in a program?

2. **It frames the whole course**. Everything we explain later, supervised learning, neural networks, deep learning, must fit inside this triplet (Task, Experience, Performance). For example:

   - Neural Network training? It's about improving P on T given more E.
   - Reinforcement learning? Same template, different E and P.

3. **It's rigorous but simple**. Unlike philosophical definitions of intelligente, Mitchell's version is **operational**: it tells us *how to test if learning is happening*. It works as a **scientific foundation**, "*if we can't measure performance improvement, we can't claim the program learned*".

4. **It avoids confusion later**. If we started with supervised learning or deep learning right away, we'd lack the general umbrella. With this definition first, we can always check: "*what is our T? what is our E? what is our P?*".

Mathematical View

Formally, suppose we have:

- Dataset $D = \{(x_i, t_i)\}_{i=1}^{N}$ (inputs + targets).
- A model $f_\theta(x)$ with parameters θ.
- A loss function $L(f_\theta(x), t)$ that measures errors (P).

Learning means finding θ^* that minimizes the expected loss:

$$\theta^* = \arg\min_{\theta} \mathbb{E}_{(x,t)\sim E} \left[L\left(f_\theta(x), t \right) \right]$$

This equation will be explained more thoroughly in the following sections.

1.1.1 Machine Learning Paradigms

When Tom Mitchell gave us the **triplet (T, E, P)**, he provided a general definition of learning. But in practice, machine learning problems usually fall into a few **big paradigms**; categories defined by *what kind of data (experience) we provide* and *what kind of task we want solved*. These paradigms are like **different ways of framing the learning problem**:

1. **Supervised Learning**: we give the algorithm examples of input and desired output. The goal is learn to map new inputs to outputs.
2. **Unsupervised Learning**: we only give input data, no labels. The goal is discover hidden structures or representations.
3. **Reinforcement Learning**: we don't provide explicit labels. The system interacts with an environment, receives **rewards or penalties**, and learns a strategy (policy) to maximize reward over time.

These paradigms are important because are the **building blocks of the field**. Almost any ML problem can be described belonging to (or combining) these three. They differ mainly in the **nature of the data (E)** and the **type of feedback (P)** available. Understanding them helps in choosing the right algorithms and models for a problem.

Example 3: Analogy

Imagine teaching three kinds of students:

- **Supervised Learning student**: we show them math problems *with answers*, and they learn how to solve similar ones.
- **Unsupervised Learning student**: we give them a pile of problems *without answers*, and they try to find patterns (like grouping similar problems together).
- **Reinforcement Learning student**: we give them a puzzle game. They don't know the rules, but they learn through *trial and error* because we give them rewards when they succeed.

1.1.1.1 Supervised Learning

Supervised Learning is like learning *with a teacher*:

- The algorithm is given **examples of inputs and their correct outputs (labels)**.
- The goal is to learn a **mapping function** that predicts the correct output for new, unseen inputs.

Formally:

- Training dataset:

$$D = \{(x_1, t_1), (x_2, t_2), \ldots, (x_N, t_N)\}$$

  Where x_i are inputs and t_i are targets.

- Model: $f_\theta(x) \approx t$.
- Learning: choose parameters θ that minimize a loss function measuring error.

In other words, **Supervised Learning** is a type of machine learning where the algorithm is trained on a labeled dataset, meaning each training example includes both the input data and the correct output. And the goal is to learn a function that maps inputs to outputs, in order to make predictions on new, unseen data.

Types of Supervised Learning

In supervised learning we always have:

- **Inputs** x (features).
- **Outputs** t (labels/targets).
- A **model** $f_\theta(x)$ that learns a mapping from inputs to outputs.

The distinction between **classification** and **regression** depends on the **nature of the output**.

- **Classification**: Predict a **discrete class label**. The output space is a finite set of categories. For example:
  - Binary: $\{0, 1\}$, e.g. spam vs not spam.
  - Multi-class: $\{1, \ldots, K\}$, e.g. digits 0-9.

  From a mathematical point of view:

$$f_\theta(x) : \mathcal{X} \to \{1, 2, \ldots, K\}$$

Example 4: Cars vs Motorcycles

Use the classic triplet:

- **Task (T)**: distinguish between two categories (binary classification).
- **Experience (E)**: dataset of images labeled "car" or "motorcycle".
- **Performance (P)**: accuracy (percentage of correct predictions).

Pipeline (how supervised learning was traditionally done before deep learning):

- **Feature Extraction (Hand-Crafted Features)**. Raw data (like an image, sound, or text) is often too complex to give directly to a simple model. Traditionally, humans designed *rules* or *functions* to extract **features** from raw data.
  - Example (images): count edges, corners, textures, or wheel shapes.
  - Example (text): word frequencies, presence of certain keywords.
  - Example (audio): pitch, energy, Mel-frequency coefficients (MFCCs).

  These features are **manually engineered** to capture the most important aspects of the problem. The output is a vector of numbers (feature vector) that represents each example. This step is about "*what information to feed into the model*".

  In this example, hand-crafted features are:

  - Extract "number of circular shapes" (wheels);
  - Extract "dominant color";
  - Extract "edge orientation histograms".

  The photo is now a vector like $[2, 0.6, 0.8]$
- **Learning a Model (Classifier)**. Once we have feature vectors, we train a **machine learning model** that learns to map those features to outputs (labels or numbers). The model **learns decision boundaries** (for classification) or **functions** (for regression) that separate categories or fit numeric values. This is the **actual learning step**: the algorithm adjusts its parameters from the data.

  In this example, the classifier could be a Support Vector Machine (SVM) model, which learns as follows: if "number of wheels ≈ 2" then is a motorcycle; if "number of wheels ≈ 4" then is a car.

- **Regression**: Predict a **continuous value**. The output space is the set of real numbers ($\mathbb{R}$). From a mathematical point of view:

$$f_\theta(x) : \mathcal{X} \to \mathbb{R}$$

Example 5: Price Prediction

Use the classic triplet:

- **Task (T)**: predict a **continuous value** instead of a discrete label.
- **Experience (E)**: dataset of houses (features: size, location, rooms) with their selling prices.
- **Performance (P)**: Mean Squared Error (MSE), Mean Absolute Error (MAE), or R^2 score.

Pipeline:

- **Hand-crafted features**: e.g., number of rooms, square meters, distance to city center.
- **Learned regressor**: a model that predicts a continuous output.

In simple terms, if our labels are:

- Categories, it's **classification**.
- Numbers, it's **regression**.

Why Deep Learning Changed This

In **deep learning**, feature extraction and learning are **not separated anymore**. Neural networks **learn features automatically from raw data** (pixels, sound waves, text). So the pipeline becomes **one end-to-end step**: input raw data → neural network → prediction.

More resources about Supervised Learning can be found in the notes for the Applied Statistics course:

1.1.1.2 Unsupervised Learning

Unsupervised Learning is like learning *without a teacher*:

- We only provide the algorithm with **inputs** $x_1, x_2, \ldots, x_N$.
- There are **no labels/targets** telling the algorithm the "correct answer".
- The goal is to **discover hidden structures** or **representations** in the data.

Formally:

- Dataset:
$$D = \{x_1, x_2, \ldots, x_N\}, \quad x_i \in \mathbb{R}^d$$
- Task: find structure in D, e.g., groups, manifolds, lower-dimension embeddings.
- Performance measure: less obvious (since no labels). It can be internal measures (compact clusters, variance explained) or extrinsic measures (utility in downstream tasks).

The most intuitive unsupervised task: Clustering

In supervised learning, we had "car vs motorcycle", categories are known. In unsupervised, no labels are given. The simplest question becomes: "*can we group the data into natural categories, even if we don't know their names?*". That's exactly what clustering does. **Clustering** is the process of grouping data points into **clusters** such that:

- Points in the same cluster are **similar** to each other.
- Points in different clusters are **dissimilar**.

Clustering uses a **similarity measure**, such as Euclidean distance. The algorithm groups data into clusters that minimize within-cluster distance and maximize between-cluster distance. Some common algorithms include:

- **Hierarchical Clustering**. Build a tree of clusters by progressively merging or splitting. Exists two approach: Agglomerative Clustering (Bottom-Up) or Divisive Clustering (Top-Down).

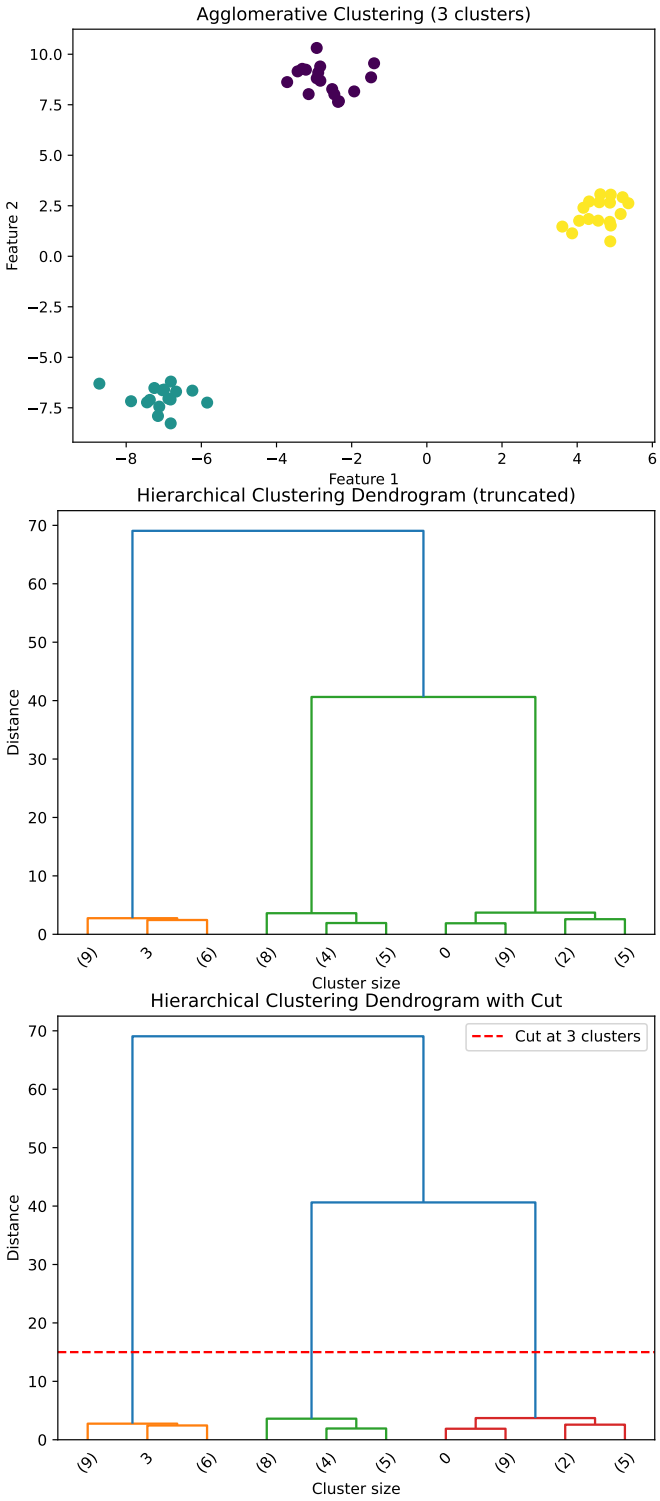


Figure 1: Agglomerative Clustering (top plot), Dendogram (mid plot) and Dendogram with cut (bottom plot).

About Figure 1, page 12. In the Agglomerative Clustering result, each dot is a **data point** (here we generated 50 synthetic points). The algorithm grouped them into **3 clusters**. We can see points within each cluster are **close together** in space. Also, the clusters are **well separated**, this is why hierarchical clustering works well here. The Dendogram shows the **hierarchical merging process**:

- At the **bottom**, each point starts as its own cluster.
- Going **upwards**, clusters that are close together are merged.
- The **height of each merge** (y-axis = distance) indicates how far apart the clusters were when merged.
- At the **top**, all points are eventually merged into a single cluster.

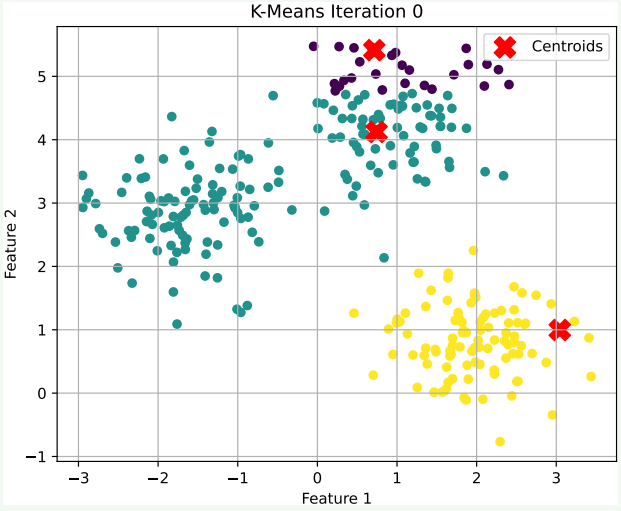
In the last figure, we "cut" the dendogram horizontally at a certain height (distance threshold), and we obtain a chosen number of clusters (here, 3). Everything **below the line** remains as separate clusters. Everything **above the line** (higher merges) is ignored. In the Dendogram, cutting at ≈ 15 gives **3 vertical "branches" crossing the red line**. Each branch corresponds to one cluster. These branches include **all 3 groups of points**.

- **K-Means**. Choose k clusters; assign points to the nearest cluster centroid; and update centroids until convergence.

Example 6: K-Means, taken from the Applied Statistics course

Below is a simple run of the K-means algorithm on a random dataset.

- Iteration 0 - **Initialization**

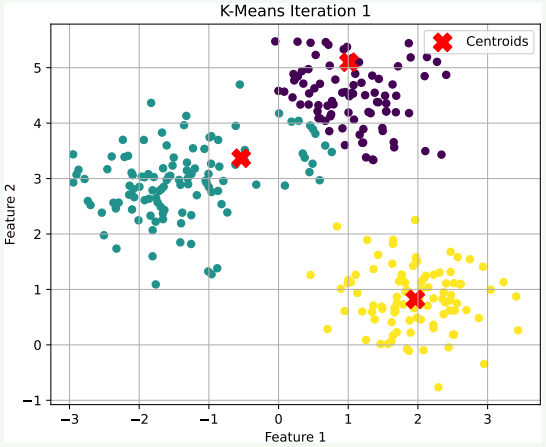


This is the starting point of the K-Means algorithm. **Three centroids are randomly placed in the feature space**.

At this point, no data points are assigned to clusters yet, or all are assumed to be uncolored/unclustered. The positions of the centroids will strongly influence how the algorithm proceeds.

The goal here is to start with some guesses. The next step will use these centroids to form the initial clusters.

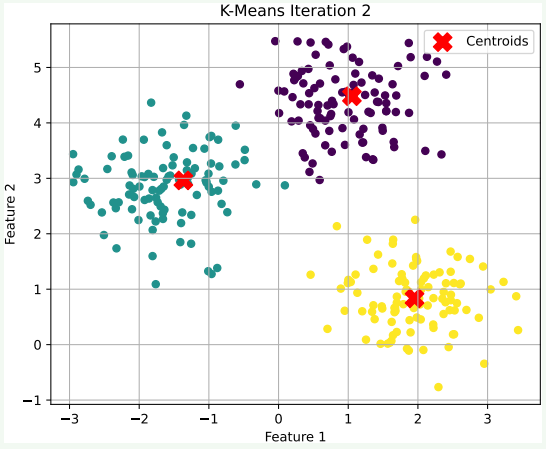
– Iteration 1 - **First Assignment and Update**



Each data point is assigned to the closest centroid, forming the first version of the clusters. New centroids are computed by taking the average of the points in each cluster. We can already see structure forming in the data, as points begin grouping around centroids.

This step is the first real clustering, and centroids begin to move toward dense regions of data.

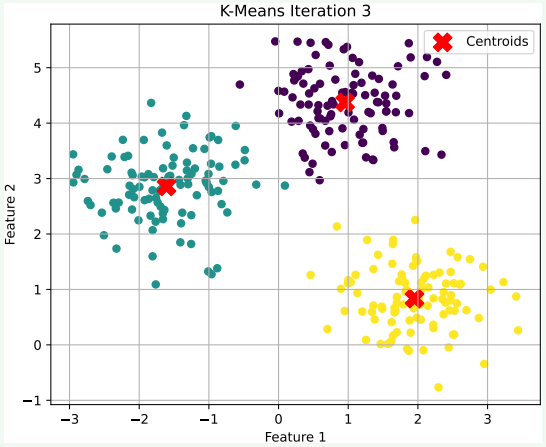
– Iteration 2 - **Re-Assignment and Refinement**



Clusters are recomputed based on updated centroids. Many points remain in the same clusters, but some may shift to a new cluster if a centroid has moved. Centroids continue moving closer to the center of their respective groups.

The algorithm is now refining the clusters and reducing the total distance from points to centroids.

– Iteration 3 - **Further Convergence**



At iteration 3, the K-Means algorithm reached convergence. The centroids no longer moved, and no points changed cluster.This means:

* The algorithm has found a locally optimal solution.
* Further iterations would not improve or change the clustering.
* The final configuration is considered the result of the algorithm.

In practice, this is how K-Means stops: it checks whether the centroids remain unchanged, and if so, it terminates automatically.

More resources about Unsupervised Learning and Clustering can be found in the notes for the Applied Statistics course:

1.1.1.3 Reinforcement Learning

Reinforcement Learning (RL) is like *learning by trial and error*. An **agent** interacts with an **environment** by taking **actions** and receiving **rewards** or **punishments**. The goal of the agent is to learn a policy that maximizes the cumulative reward over time.

At each step, the agent:

1. **Observes a state** s_t from the environment.
2. **Selects an action** a_t based on its current policy $\pi(a_t \mid s_t)$.
3. **Receives a reward** r_t and a **new state** s_{t+1}.

The agent's goal is to learn a **policy** $\pi(a \mid s)$ that maximizes the expected cumulative reward. Unlike supervised learning, no teacher gives the right answer; the agent learns from the **consequences** of its actions.

What is an Agent?

An **agent** is an *entity* that **makes decisions and takes actions in an environment to achieve a specific goal**. In reinforcement learning, the agent learns to optimize its behavior based on feedback from the environment.

With ***entity***, we mean anything that can perceive its environment through sensors and act upon that environment through actuators.

Example 7: Robot Navigation

For example, consider a robot navigating a maze. The robot (agent) perceives its surroundings (state), decides to move left or right (action), and receives feedback (reward) based on whether it gets closer to the exit or hits a wall. The robot's goal is to learn a strategy (policy) that maximizes its chances of reaching the exit while avoiding obstacles.

In simple terms, the robot through cameras and sensors perceives the maze (environment), decides its next move (action), and learns from the outcomes (rewards) to improve its navigation strategy (policy).

In summary:

- **Agent**: The robot.
- **Environment**: The maze.
- **State**: The robot's current position in the maze.
- **Action**: Moving left, right, forward, or backward.
- **Reward**: Positive reward for reaching the exit, negative reward for hitting a wall.
- **Policy**: The strategy the robot uses to decide its next move based on its current state.

The agent's **primary objective** is to **learn a policy that maximizes the cumulative reward** it receives over time by interacting with the environment.

Formalization of Reinforcement Learning

Reinforcement learning problems are often modeled using **Markov Decision Processes (MDPs)**. An MDP is defined by:

- **Task (T)**: learn a policy $\pi(a \mid s)$ mapping states to actions. In other words, the task is to find the best action to take in each state to maximize cumulative reward.
- **Experience (E)**: consists of sequences of states, actions, and rewards obtained by interacting with the environment.
- **Performance Measure (P)**: expected return (sum of discounted rewards):

$$P = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$$

  Where $\gamma \in [0, 1]$ is the discount factor that determines the importance of future rewards.

Key Concepts in Reinforcement Learning

The goal of this section is to introduce the Reinforcement Learning paradigm and its key concepts. These concepts will be covered in more detail in later sections. However, here are some of those concepts:

- **Exploration vs. Exploitation**: The dilemma of choosing between exploring new actions to discover their effects (***exploration***) and exploiting known actions that yield high rewards (***exploitation***).

  Why a dilemma? Because if the agent only exploits known actions, it may miss out on potentially better actions. Conversely, if it only explores, it may not accumulate enough reward.

- **Reward Signal**: The feedback received from the environment after taking an action, used to evaluate the action's effectiveness. It could be sparse or dense:
  - **Sparse Reward**: Rewards are infrequent, making it challenging for the agent to learn. For example, in a game, the agent might only receive a reward upon winning or losing.
  - **Dense Reward**: Rewards are given frequently, providing more immediate feedback. For example, in a driving simulation, the agent might receive small rewards for staying on the road and penalties for going off-road.

- **Delayed reward**: The reward for an action may not be immediate, making it challenging to associate actions with their long-term consequences. For example, in a chess game, a move may not yield an immediate reward but could lead to a win several moves later. The agent must learn to evaluate actions based on their long-term impact rather than immediate outcomes. This requires the agent to consider future rewards when making decisions.

⚖ RL vs. Supervised Learning

Reinforcement learning differs from supervised learning in several key ways:

Aspect	Supervised Learning	Reinforcement Learning
Data	Fixed labeled dataset (in-out pairs)	No labels; agent generates data by acting
Feedback	Correct answer for each example	Rewards (possibly delayed, sparse)
Goal	Minimize error (classification/regression)	Maximize cumulative reward
Typical methods	Regression, SVM, Neural Nets	Q-learning, Policy Gradients, Actor-Critic

⚠ Challenges of Reinforcement Learning

Reinforcement learning presents several challenges:

- **Exploration**: need to try enough actions to discover good strategies.
- **Delayed Feedback**: rewards may not be immediate, complicating reward assignment.
- **Sample inefficiency**: often requires millions of trials to learn effective policies.
- **Stability**: training can be unstable with neural nets.

Despite these challenges, RL has achieved remarkable success in various domains, including game playing, robotics, and autonomous systems.

In summary, reinforcement learning is a powerful paradigm for training agents to **make decisions in complex environments** by **learning from the consequences** of their actions. RL is distinct from supervised learning in its approach to data, feedback, and goals, making it suitable for a wide range of applications where direct supervision is not feasible.

References

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[2] Matteucci Matteo. Artificial neural networks and deep learning. Slides from the HPC-E master's degree course on Politecnico di Milano, 2025-2026.

Index