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).

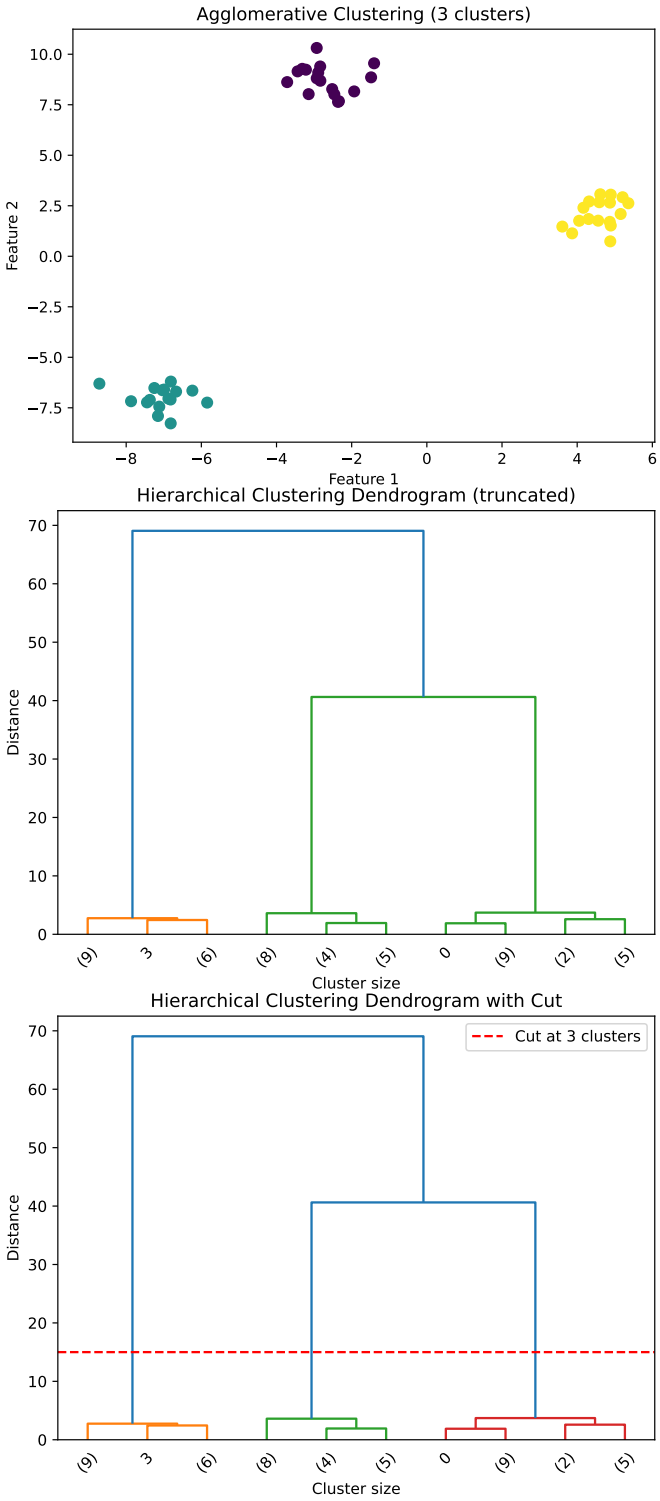


Figure 1: Agglomerative Clustering (top plot), Dendogram (mid plot) and Dendogram with cut (bottom plot).

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.

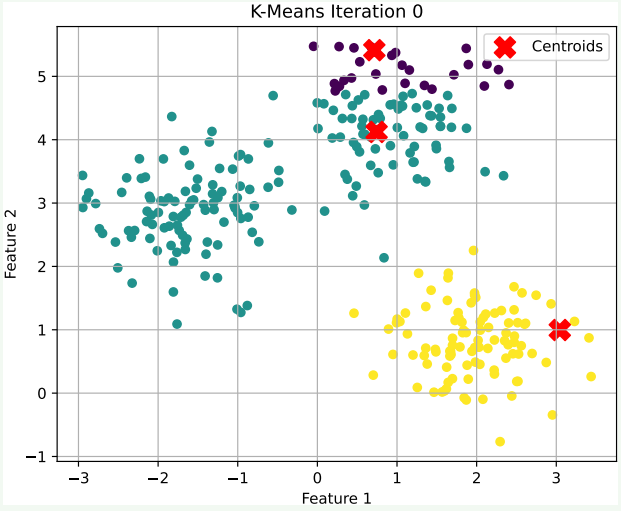
In the last figure, we "cut" the dendogram horizontally at a certain height (distance threshold), and we obtain a chosen number of clusters (here, 3). Everything **below the line** remains as separate clusters. Everything **above the line** (higher merges) is ignored. In the Dendogram, cutting at ≈ 15 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**

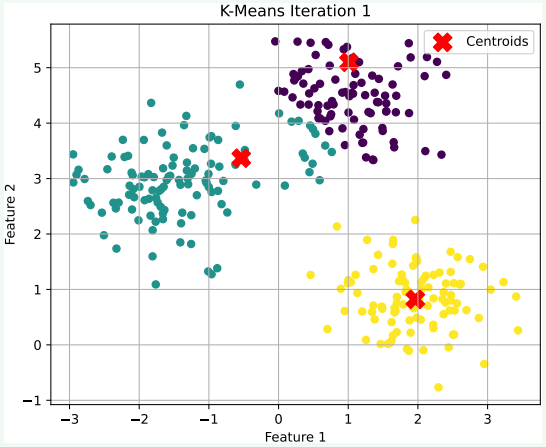


This is the starting point of the K-Means algorithm. **Three centroids are randomly placed in the feature space**.

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.

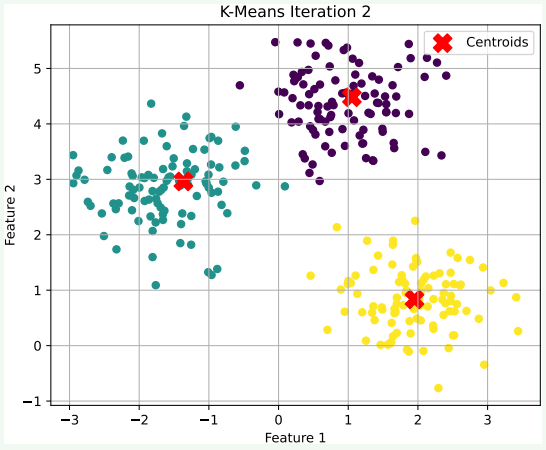
– Iteration 1 - **First Assignment and Update**



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.

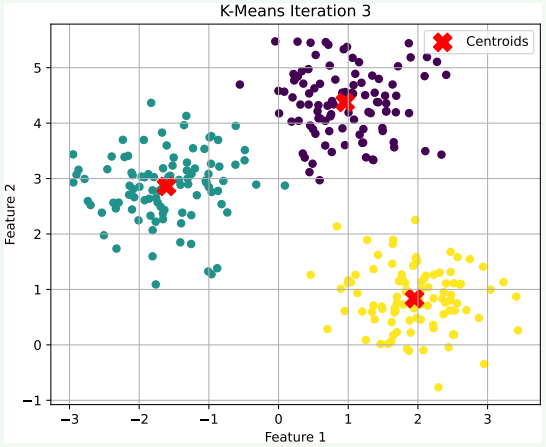
– Iteration 2 - **Re-Assignment and Refinement**



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.

1.2 Towards Deep Learning

This course, and this notes, focuses **mostly on Supervised Learning**, with some unsupervised learning concepts and techniques. ***Why?***

- Supervised Learning is the most widely used paradigm in practice (e.g., image classification, speech recognition, etc.);
- Many deep learning application (image recognition, NLP, etc.) are supervised tasks;
- Unsupervised learning will be touched when needed (e.g., representation learning, generative models, etc.);

Deep Learning is not a new paradigm, it's a **new approach** with supervised/unsupervised learning.

What about Deep Learning? Iris Flower Example

The Iris flower dataset is a classic dataset in machine learning, often used for classification tasks. It consists of 150 samples of iris flowers, each with four features: sepal length, sepal width, petal length, and petal width. The goal is to classify the flowers into three species: Iris setosa, Iris versicolor, and Iris virginica.

- **Traditional Machine Learning Approach:**
  - Extract "good features" from the raw data (e.g., petal length and width);
  - Train a classifier (e.g., decision tree) on these features;
- **Deep Learning Approach:**
  - Learn both **features** and **classifier** simultaneously from the raw data;

For example:

1. If **features are simple** (e.g., petal length and sepal width), then the classification task is **easy**, and a simple model (e.g., decision tree) can achieve high accuracy;
2. If **features are complex** (e.g., raw pixel values of flower images), then the classification task is **hard**, and the traditional approach **struggles to extract meaningful features**;
3. If **impossible to know** which features matter, then handcrafted features are **not enough**, and we need a model that can **learn features** from the data itself (e.g., a deep neural network).
4. Deep Learning learns features **directly from raw data**, making it suitable for complex tasks where feature engineering is challenging or infeasible (hierarchical representations).

⚖ Feature Engineering vs. Learned Features

- **Feature Engineering (Traditional ML)**:
  - Feature Engineering is the process of **using domain knowledge to extract features from raw data** that make machine learning algorithms work. It needs human experts to design and select features that are relevant to the task.
  - **Problem:** requires domain expertise, time-consuming, and may not capture all relevant information. It is often brittle and not transferable to new tasks or domains.
- **Learned Features (Deep Learning)**:
  - Learned Features are features that are **automatically learned by the model** from the raw data during training.
  - Layers learn progressively:
    - Lower layers learn simple patterns (e.g., edges, corners);
    - Middle layers learn more complex patterns (e.g., eyes, wheels);
    - Higher layers learn high-level concepts (e.g., faces, cars).
  - **Advantage:** optimized for the task at hand, can capture complex patterns, and are transferable to new tasks or domains. It requires less manual effort and often generalizes better to unseen data.

1.3 Modern Pattern Recognition (Pre-DL)

Before the rise of deep learning, modern pattern recognition techniques were primarily based on traditional machine learning algorithms and statistical methods. These techniques focused on feature extraction, dimensionality reduction, and classification using various algorithms.

Speech Recognition (early 1990s-2011)

Speech recognition systems used a **multi-stage pipeline** approach, which included:

- **Low-level features**: extracted from the raw audio waveform, such as MFCCs (Mel-Frequency Cepstral Coefficients), a compact representation of the spectral properties of the audio signal.
- **Mid-level features**: built by grouping/encoding low-level features over short time windows, capturing temporal dynamics. For example Mixture of Gaussians (MoG) used to model acoustic units (phonemes).
- **Classifier (high-level features)**: used to map mid-level features to words or phrases. Common classifiers included Hidden Markov Models (HMMs) combined with Gaussian Mixture Models (GMMs) to decode sequences of acoustic units into words.

This pipeline worked decently but was very **hand-crafted** and success depended heavily on the quality of feature engineering.

Object Recognition (2006-2012)

Computer vision systems followed a similar multi-stage pipeline approach:

- **Low-level features**: detect edges, corners, gradients using methods like SIFT (Scale-Invariant Feature Transform) or HOG (Histogram of Oriented Gradients).
- **Mid-level features**: combine low-level descriptors into higher-level "visual words". For example, clustering with k-means to create a codebook of visual words, and Sparse Coding to represent images as sparse combinations of these words.
- **Classifier (high-level features)**: train SVMs (Support Vector Machines) or Random Forests to classify images based on mid-level features.

Again, this approach was heavily reliant on hand-crafted features and required significant domain expertise to design effective features. However, before 2012, these methods were the state-of-the-art in many computer vision tasks.

General Pipeline (Pre-DL Pattern Recognition)

The general pattern recognition pipeline before deep learning can be summarized as follows:

1. **Low-level features**: raw signal transformation (e.g., edges, frequencies).

2. **Mid-level features**: encode or cluster low-level descriptors (e.g., visual words, acoustic units).
3. **Classifier (high-level features)**: learns categories from hand-designed representations.

⚠ Limitations

- **Domain expertise required**: Designing MFCCs, SIFT, HOG, etc. required significant knowledge of the specific domain (speech, vision).
- **task specific**: features built for one task often did not generalize well to others (e.g., MFCCs don't work well for images).
- **Brittleness**: sensitive to noise, illumination, scaling, speaker accents, etc.
- **Limited expressiveness**: as dataset grew, hand-crafted pipelines saturated in accuracy.

Before deep learning, pattern recognition was a **multi-stage pipeline** heavily **reliant on hand-crafted features and domain expertise**. While effective for its time, it had significant limitations in scalability, generalization, and robustness that deep learning would later address.

1.4 What is Deep Learning after all?

After showing the historical context, what Machine Learning is, the three paradigms and how pre-DL pattern recognition worked, we can finally answer the question:

Now that we know what ML does, what makes Deep Learning *different* from classic ML?

We will take our time answering this question. First, we need to understand the meanings of "features" and "classifiers".

What are "features"?

Features are **numerical representations of the raw data** that capture something meaningful for the task.

Type of Data	Raw data example	Example of features
Images	Pixels (RGB values)	Edges, corners, textures
Audio	Waveform (amplitude over time)	Pitch, frequency spectrum, MFCCs
Text	Words or sentences	Word counts, syntactic structure

In **classical ML**, these features were **manually designed** by humans; engineers decided *what* was important and *how to compute it*. For example:

Input image → extract edges manually → feed into SVM classifier

So we had:

Handcrafted Features → Learned Classifier

Where "handcrafted" means "coded by humans". So, before Deep Learning, the **feature extraction** and the **classifier** were two separate stages in the pipeline, and humans designed the first stage. This approach worked, but only if the human correctly guessed *what features matter* for the task.

What does "Learned Features" mean?

Deep Learning says: "*Stop handcrafting features; let the machine learn them automatically, layer by layer, together with the final classifier*".

In **Deep Learning**, the model itself learns how to transform raw data into useful internal representations. Each layer of a neural network acts as a **feature extractor** that learns automatically *what patterns matter*:

- First layers: detect edges, colors, or simple shapes.
- Intermediate layers: detect object parts (e.g., eyes, wheels, leaves).
- Deep layers: detect abstract categories (e.g., "face", "car", "flower").

So instead of telling the machine *what to look for*, we let it **discover patterns directly from data**. This is the "**learned features**" part.

What does "Learned Classifier" mean?

After features are extracted (automatically or manually), the model still needs to **make a decision**: classify, predict, or generate.

- In traditional ML, this is the final **classifier** stage (e.g., SVM, logistic regression, random forest).
- In Deep Learning, the **last few layers** of the network act as that classifier, they map high-level learned features to output labels.

So, both parts, the *feature extractor* and the *decision function*, are **learned jointly** through backpropagation.

$$\text{Raw Data} \xrightarrow[\text{learned weights}]{\text{Feature Extractor}} \text{Representations} \xrightarrow[\text{learned weights}]{\text{Classifier}} \text{Predictions}$$

So DL uses a single model to learn both **features** and **classifier** together: Learned Features + Learned Classifier. The model not only learns *how to decide* but also *how to see the world*, both are learned from data.

So, "What is Deep Learning after all?"

Deep Learning is **not just a new algorithm**, it's a new way of *approaching representation learning*. If we had to answer in one line: "Deep Learning is the automatic learning of hierarchical data representations and decision functions directly from raw data". That's why it's so powerful: it *adapts* to the data and the task, without relying on human intuition about features.

Deepening: Why Not Everything Is Deep Learning

Deep Learning is *powerful*, but it's not a silver bullet, it's not *free*. It's the best tool **when** we have: large amounts of diverse data, high compute, a task based on perception or pattern recognition. Otherwise, **traditional ML or statistical models** can be simpler, faster, and just as effective.

- **Deep Learning needs a *lot of data.*** Deep models have **millions (sometimes billions)** of parameters. They only generalize well when trained on **massive labeled datasets** (e.g., ImageNet: 14M images). If we have small data, like 300 samples from an industrial machine, a deep model will likely **overfit** and perform worse than simpler methods. In other words, Deep Learning shines when there is **data abundance**, but struggles in **data scarcity**.
- **Deep Learning needs a *lot of computation.*** Training is computationally heavy, requiring specialized hardware: GPUs, TPUs, clusters, or cloud computing. Classic ML (SVMs, Decision Trees, Random Forests) can run on a laptop. Deep nets require weeks of GPU training, hyperparameter tuning, and energy cost. So, if the

task doesn't justify the cost, simpler ML is more efficient.

- **Deep models are *black boxes.*** We can rarely explain *why* a deep network made a decision. For critical systems (healthcare, law, finance, safety) we need **interpretability** and **traceability**. Simpler models like linear regression or decision trees are **transparent**, easy to justify in front of regulators or domain experts. For example, a hospital won't risk a deep net saying "tumor" without being able to explain which features caused that prediction.
- **Deep models are *hard to train and tune.*** Choosing architecture (layers, neurons, learning rate, dropout, etc.) is an art. Training can **diverge** or **get stuck** (vanishing gradients, overfitting, exploding losses). We often need extensive experimentation and deep knowledge of optimization tricks. So, not every team or project can afford the expertise and trial cycles DL requires.
- **Deep Learning doesn't always fit the problem.** Some tasks simply:
  1. Have **structured or tabular data** (e.g., bank records, tabular logs). Here, traditional ML (XGBoost, Random Forests) often outperforms DL.
  2. Require **symbolic reasoning** or **logic**, not pattern recognition. Here, DL struggles to capture rules and relationships that classical AI or rule-based systems handle better.
  3. Need **causal inference**, not just correlations. DL finds patterns but doesn't understand cause-effect relationships, which are crucial in many scientific and policy domains. Let's think about a real-world example: predicting disease spread based on interventions (lockdowns, vaccinations) requires understanding causality, not just correlations in data (not just "if X happens, Y follows", but "if we do X, Y will change").
- **Deep Learning needs *good data.*** DL is extremely sensitive to: label noise (wrong annotations ruin learning); biases in the dataset (can reproduce or amplify them); distribution shifts (fails badly if test data differ from training). Traditional methods often handle noise and small variations more robustly. So, "Garbage in $\rightarrow$ garbage out" is even more true with DL.
- **Deep Learning doesn't mean *understanding.*** DL recognizes **patterns**, not meaning. It can detect a cat, but it doesn't *know* what a cat is. It can predict outcomes, but not always *why* they happen. That's why current research explores **hybrid systems** combining DL with: symbolic reasoning (neuro-symbolic AI), knowledge graphs, logic and interpretability layers.

Deepening: ChatGPT, LLaMA & Modern AI Models - What Are They?

ChatGPT, LLaMA, Gemini, Claude, etc. are all based on a specific kind of **Deep Neural Network** called a **Transformer**, introduced in 2017 by Vaswani et al. ("Attention is All You Need"). So, fundamentally:

$$\text{ChatGPT, LLaMA, Gemini, etc.} \in \text{Deep Learning}$$

They are not "beyond" DL, they are its **current frontier**.

What kind of Deep Learning model? They belong to the family of **Large Language Models (LLMs)**.

- **Architecture**: Transformer (a type of deep neural network specialized for sequences and attention).
- **Learning paradigm**: mainly *self-supervised learning*, a subform of unsupervised learning.
- **Objective**: predict the next word (token) given the previous ones.

Mathematically:

$$P\left(w_t \mid w_1, w_2, \ldots, w_{t-1}\right)$$

"Given this context, what's the next most probable word?". That's the only thing it learns. Everything else (reasoning, style, facts) *emerges* from learning this next-token distribution on vast text corpora.

Why are they still called "Deep Learning"? They perfectly fit the definition we discussed earlier: "Deep Learning is the learning data representation and decision functions directly from data".

- They learn **representations** of words, sentences, and even concepts automatically.
- They have **layers upon layers** (up to 100+ in GPT-4).
- They **don't rely on hand-crafted linguistic features** (no human tells them grammar rules).
- They learn everything **directly from raw text data** (syntax, semantics, even reasoning patterns).

So they exemplify:

$$\begin{gathered}\text{Learned Features (embeddings)} + \\ \text{Learned Classifier (next word predictor)}\end{gathered}$$

But at **massive scale**, with **billions of parameters** and trained on **terabytes of text**. This scale is what enables their surprising capabilities.

What makes them *different* from earlier Deep Learning. Traditional DL (e.g., CNNs, RNNs) had strong **task specialization**: CNNs for vision, RNNs for sequences, LSTMs for time series. Instead, Transformers with LLMs changed the game because they are **general-purpose learners**:

- They can handle language, code, images, audio, even multimodal data.
- Their **attention mechanism** learns relationships between all parts of the input simultaneously.

They are sometimes called: "Foundation Models", because they can be *fine-tuned* for many downstream tasks (translation, summarization, question answering, etc.).

Why do they feel intelligent? When we train on *massive data* (trillions of words) and *huge models* (hundreds of billions of parameters), the model starts showing **emergent behaviors**:

- Understanding context, humor, and nuance.
- Performing reasoning and arithmetic.
- Generating coherent, creative text.
- Translating languages fluently.
- Writing code snippets.

But still, it's pattern prediction. There is **no explicit symbolic reasoning** or understanding; it's just learned statistical structure at enormous scale. So we say: "They are **Deep Learning models**, trained on **massive dataset**, showing **emergent intelligence**".

1.5 What's Behind Deep Learning?

If the concept of neural networks exists since the 1950s, ***why did Deep Learning explode only after 2012?*** This is a natural question that comes *after* we've seen what Deep Learning is. To answer this question, we show two perspectives: the **MIT view** and **The Economist view**.

The MIT view: Computational Power

According to MIT and many early researchers, Deep Learning became possible only when **computational resources** caught up with the theory. It means that the mathematics and algorithms (backpropagation, perceptrons, convolutional nets) existed for decades, but **training deep networks** requires enormous computation:

- Millions of matrix multiplications.
- Thousands of gradient updates per sample.
- Gigantic datasets.

Before 2010, this was impractical. Around 2011-2012, **GPUs** (Graphics Processing Units) changed everything:

- They made large-scale matrix computations thousands of times faster.
- Deep learning frameworks (Theano, TensorFlow, PyTorch) made GPU computing accessible.
- Hardware parallelism allowed training networks with **hundreds of layers** instead of 3-4.

So from the MIT perspective: Deep Learning rose because **we finally had the computational power to train deep models**.

The Economist view: Big Data

In 2012, *The Economist* (yes, the famous magazine) proposed a different, and equally valid, explanation: "Deep Learning exploded because the world finally generated **enough data** to feed it". It means that the Internet, social media, smartphones, sensors, and cloud storage created **massive labeled datasets**:

- ImageNet (over 14 million labeled images).
- YouTube (millions of labeled videos).
- Text from web, Wikipedia, books, perfect for LLM pretraining.

Deep neural networks thrive on data volume: they don't generalize well with few examples. The more data, the better they learn **hierarchical representations**. So from the Economist perspective: "Deep Learning rose because **we finally had Big Data**, the fuel it needs to work".

The Real View: Both Matter

In reality, both perspectives are correct and complementary. Deep Learning's success is due to the **synergy of computational power and big data**:

- Before 2010, algorithms existed but computing was too slow and data too scarce. Then neural networks were limited to shallow architectures and small datasets.
- Around 2012, hardware (GPUs, TPUs, distributed training) made computation feasible. Simultaneously, the explosion of digital data provided the massive labeled datasets needed.

This combination triggered the **Deep Learning revolution**. The turning point was **ImageNet 2012**, where Krizhevsky, Sutskever, and Hinton demonstrated that a deep convolutional network (AlexNet) could drastically outperform traditional methods on image classification. This success was possible only because:

- They used two NVIDIA GPUs to train a deep network with millions of parameters.
- They trained on the large ImageNet dataset with 1.2M labeled images.

The result was an error rate of 15%, compared to 26% for the best traditional method. This landmark event showcased the **power of deep learning when both computational resources and big data are available**.

1.6 Summary

Everything we've seen, supervised, unsupervised, or reinforcement learning, ultimately depends on **how we represent data**. In traditional ML, features are *hand-crafted*. In Deep Learning, features are *learned automatically* through hierarchical representations. The revolution of Deep Learning wasn't new math, it was learning **what matters** in the data instead of coding it by hand.

$$\text{Success of ML} \Rightarrow \text{Success of its feature representation}$$

Deep networks just made the **representation learning** automatic and scalable.

Deep Learning = Learning Data Representation from Data

Deep Learning is not a specific architecture (like CNN, RNN, or Transformers) or algorithm. It's the **paradigm** where:

1. Input → raw data (e.g., pixels, text, audio)
2. Model → multiple non-linear layers learning internal representations.
3. Output → desired prediction/task.
4. Learning → end-to-end optimization of all layers together.

So instead of:

$$\text{Human designs features} \rightarrow \text{Model learns mapping}$$

We now have:

$$\text{Model learns both features and mapping} \rightarrow \text{directly from data}$$

This is the essence of Deep Learning: **learning hierarchical representations directly from raw data**.

"Which data?" - The key question of the course

This is the **transition line** to the rest of the course (notes). Now that we know *what* Deep Learning is, the next question is *what data we use and how*. Different data types define the upcoming sections:

Data Type	Upcoming Section
Tabular / numerical	Perceptrons & Feed-Forward NNs
Images	Convolutional Neural Networks (CNNs)
Sequential (text, time series)	Recurrent Neural Networks (RNNs) & Transformers
Unlabeled data	Autoencoders & Word Embeddings

So this question of "which data?" becomes the **roadmap** for the rest of the course (notes).

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