**NEURAL NETWORKS**

**Introduction to Neural Networks**

A **neural network** is a type of machine learning model inspired by the structure and functioning of the human brain. Neural networks are a fundamental component of **deep learning**, which is a subset of **artificial intelligence (AI)**.

**🔍 What is a Neural Network?**

A **neural network** is made up of layers of **neurons** (also called **nodes**), where:

* Each **neuron** receives input,
* Processes it using a **weighted sum** and a **non-linear function** (called an **activation function**),
* And passes the result to the next layer.

**🧠 Structure of a Neural Network**

A typical neural network has the following layers:

1. **Input Layer**
   * Takes in the raw data (e.g., image pixels, text embeddings).
2. **Hidden Layer(s)**
   * One or more layers that process the data. These layers "learn" patterns or features from the input.
3. **Output Layer**
   * Produces the final result (e.g., classification label, regression value).

Example:

Input → [Hidden Layer 1] → [Hidden Layer 2] → Output

**🔗 How It Works (High-Level)**

1. **Forward Propagation**
   * Data flows through the network layer by layer.
   * Each neuron applies weights and an activation function to its inputs.
2. **Loss Function**
   * Compares the network's output to the actual result to measure error.
3. **Backpropagation**
   * The network adjusts the weights using **gradient descent** to minimize the error.
   * This is how the model "learns".

**🧩 Key Concepts**

* **Weights & Biases**: Parameters the model adjusts during training.
* **Activation Function**: Introduces non-linearity. Examples:
  + ReLU (Rectified Linear Unit)
  + Sigmoid
  + Tanh
* **Loss Function**: Measures prediction error.
* **Learning Rate**: Controls how much the weights change during learning.

**🧠 Common Types of Neural Networks**

| **Type** | **Use Case** |
| --- | --- |
| **Feedforward Neural Network (FNN)** | Basic structure, used for general-purpose tasks |
| **Convolutional Neural Network (CNN)** | Image recognition and processing |
| **Recurrent Neural Network (RNN)** | Time-series data and sequences (e.g., text, speech) |
| **Transformer** | Modern NLP models (e.g., GPT, BERT) |

**✅ Applications**

* Image and speech recognition
* Natural Language Processing (NLP)
* Medical diagnosis
* Self-driving cars
* Game playing (e.g., AlphaGo)

**Biological Neurons (Inspiration for Neural Networks)**

* Biological neurons are **cells in the brain** that **process and transmit information** using **electrical and chemical signals**.
* They are connected through **synapses**, forming complex networks.
* A neuron “fires” an **action potential** (electrical signal) when it receives enough input from other neurons.

**🧠 Role of Biological Neurons in Neural Networks (Conceptually)**

Biological neurons served as a **model** for the **artificial neuron**, which is the basic unit in a neural network.

| **Biological Neuron** | **Artificial Neuron** |
| --- | --- |
| Dendrites (input receivers) | Inputs (numbers/features) |
| Soma (cell body) | Summation + Activation function |
| Axon (signal sender) | Output to the next layer |
| Synapse (connection) | Weight (strength of connection) |

**⚙️ How Biological Neurons Inspired Artificial Neurons**

Researchers modeled artificial neurons based on how biological neurons:

* **Receive signals** from other neurons (inputs),
* **Summarize and process** those signals (weighted sum + activation),
* **Decide** whether to "fire" (activation function),
* **Send signals** to other neurons (output to next layer).

### What is an ****Artificial Neuron**** (or ****Perceptron****)?

An **artificial neuron** is a **mathematical model** inspired by a **biological neuron**. It is the **basic unit** of an **artificial neural network (ANN)** — just like a biological neuron is the basic unit of the brain.

The **Perceptron** is the **simplest type** of artificial neuron, introduced by **Frank Rosenblatt** in 1958.

### 🧠 Structure of a Perceptron

A **perceptron** takes multiple inputs, processes them, and produces a single output.

#### 📥 Inputs:

* Inputs are typically numerical values (e.g., features of data).
* Each input is associated with a **weight** — a number that tells how important the input is.

#### ⚙️ Computation:

The perceptron calculates a **weighted sum** of the inputs:

z=w1​.x1+w2.​x2​+⋯+wn​.xn​+b

Where:

* x1,x2,...,,...,xn​ = inputs
* w1,w2,...wn​ = weights
* b = bias (like a threshold offset)

#### Activation Function:

The result zzz is passed through an **activation function** to produce the output.

In the **original perceptron**, the activation function was:

* output={1 if z>0 otherwise​ 0}

### Perceptron Diagram

x1 ─┬──►

│ \

x2 ─┼──► [ Weighted Sum → Activation ] → Output

│ /

x3 ─┴──►

(w1, w2, w3, b)

### Example

Imagine a perceptron that decides whether an email is spam:

* Inputs:
  + x1x\_1x1​: contains the word "free"
  + x2x\_2x2​: contains the word "win"
  + x3x\_3x3​: email has an attachment
* Weights and bias are learned from data:
  + w1=0.7,w2=0.8,w3=0.3,b=−1.0
  + x1=0.1,x2=0.2,x3=0.5

If the weighted sum > 0, the perceptron classifies it as **spam (1)**. Otherwise, **not spam (0)**.

**Learning in Perceptrons**

Perceptrons learn by adjusting the weights and bias using a learning algorithm, typically based on error correction. If the output is wrong, the weights are changed slightly to reduce the error next time.

**Limitations of a Single Perceptron**

Can only solve linearly separable problems.

Cannot handle problems like XOR, where the output is not a straight-line separation.

To overcome this, we use multi-layer perceptrons (MLPs) — which stack multiple perceptrons into layers, allowing the network to learn nonlinear functions.

**Working of a Perceptron — Step by Step**

A **Perceptron** is a type of **artificial neuron** that takes input, processes it, and produces an output based on a **decision boundary** (like a line). Here's how it works in detail:

**Components Recap**

* **Inputs**: x1,x2,..​,...,xn​
* **Weights**: w1,w2...,wn​
* **Bias**: b
* **Activation Function**: Usually a **step function** (in basic perceptrons).

### Step-by-Step Working of a Perceptron

#### ****Step 1: Receive Input****

The perceptron takes in multiple input values x1,x2,...,xnx\_1, x\_2, ..., x\_nx1​,x2​,...,xn​.  
These could be:

* Pixel values in an image
* Features of a dataset (like height, weight, etc.)

#### ****Step 2: Multiply Each Input by its Weight****

Each input is multiplied by its corresponding weight:

#### ****Step 3: Add Bias****

The bias b is added to the weighted sum to shift the output left or right:

z=w1.x1+w2.x2+⋯…..wn​xn​+b

#### ****tep 4: Pass the Sum Through Activation Function****

The value z is passed through an **activation function**, often:

output={1if z>0

0 otherwise​ }

This "fires" the neuron if the sum is above a threshold (0 in this case).

#### ****Step 5: Output the Result****

The result is either:

* **1** (Neuron is "activated")
* **0** (Neuron is not activated)

This output can be used for:

* Binary classification (e.g., spam vs not spam)
* Feeding into another layer in a neural network

### ✅ Example

Let’s walk through an example:

| **Feature** | **Value (x)** | **Weight (w)** |
| --- | --- | --- |
| x1​ = free in subject | 1 | 0.8 |
| x2 = has link | 1 | 0.6 |
| x3​ = is long message | 0 | 0.2 |
| Bias b=−1.0. |  |  |

#### Step-by-step:

1. Weighted sum:
2. z=(1)(0.8)+(1)(0.6)+(0)(0.2)+(−1.0)=0.8+0.6+0−1.0=0.4.
3. Activation:

z=0.4>0 Output=1

✅ The perceptron outputs **1**, meaning it classifies the email as **spam**.

**Limitation**

A single perceptron **can only solve linearly separable problems** (i.e., problems where a straight line can divide the classes, like AND, OR).  
It **cannot solve** non-linearly separable problems like **XOR** — for that, we need **multi-layer networks** (MLPs).