## 🧠 1. The Core Idea: From Biology to Math

The central goal of Deep Learning is to mimic the human brain's ability to learn patterns. This starts with its basic component:

- **Biological Neuron:** A cell that receives signals (from dendrites) and "fires" an output signal (down the axon) only if the combined inputs cross a certain **threshold** (e.g., -50mV from -7-mv).
- **MCP Neuron (1943):** The first mathematical model. It was very simple:
    - **Inputs & Weights:** Were **binary** (e.g., {0, 1} for inputs, {+1, -1} for weights).
    - **Rule:** It summed the weighted inputs ($\sum w_i x_i$) and "fired" a 1 if the sum was $\ge$ a **Threshold ($T$)**, otherwise it output 0.
- **Perceptron (1958):** A more advanced model by Frank Rosenblatt.
    - **Inputs & Weights:** Could be **real numbers** (any value, not just 0 or 1).
    - **Rule:** It replaced the Threshold ($T$) with a learnable **bias ($b$)** and computed an **affine transformation** ($z = w^T x + b$).
    - **Activation:** It used a **Sign (or Step) function** to output a 1 or 0 based on whether the result was positive or negative.

---

## 🚧 2. The First Big Problem: Linear Separability

A single neuron (both MCP and Perceptron) can only solve problems that are **linearly separable**—meaning you can separate the two classes (e.g., 0s and 1s) with a single straight line.

- **What it CAN solve:** Simple logic gates like **AND**, **OR**, **NAND**, and **NOR**.
- **What it CANNOT solve:** The **XOR problem**. It's impossible to draw one straight line to separate the outputs of XOR. This limitation showed that single neurons were not enough and that **multiple layers** would be needed to solve complex, non-linear problems.

---

## ⚙️ 3. How a Neuron "Learns": The Training Process

The goal of "training" is to find the perfect values for the weights ($w$) and bias ($b$). Your notes cover two different ways to do this.

**Method 1: The Perceptron Learning Algorithm ("The Trick")**

This was an early method for training a Perceptron.

- **How it works:** It updates its weights **only when it makes a mistake** (misclassifies a point).
    - **If False Negative** (actual=1, pred=0): It "adds" the input to the weights ($w = w + x$) to move the boundary closer to the point.
    - **If False Positive** (actual=0, pred=1): It "subtracts" the input from the weights ($w = w - x$) to push the boundary away.
- **Its Limitation:** This method only considers one point at a time and has no way to track the **overall performance** of the model. Fixing one point might misclassify many others.

**Method 2: Modern Training (Gradient Descent)**

This is the standard approach for all modern deep learning.

Step 1: Forward Propagation

This is the process of getting a prediction.

1. Take inputs ($x$) and calculate the **affine transformation** $z = w^T x + b$.
2. Pass $z$ through an **activation function** $h = f(z)$.
3. This gives the final **prediction $\hat{y}$**.
4. Compare the prediction $\hat{y}$ to the actual $y$ using a **Loss Function** (e.g., $L = \frac{1}{2}(y - \hat{y})^2$) to get a single error score.

Step 2: Backward Propagation

This is the process of finding who to blame for the error.

- **The Problem:** The Perceptron's **Sign function** is not differentiable (it's a hard step), so you can't calculate its gradient.

- **The Solution:** We use smooth, differentiable functions like the **Sigmoid function** ($\sigma(z) = \frac{1}{1 + e^{-z}}$).
- **The Goal:** To find the **gradient** (the slope of the loss) for each weight and bias in the network.
- **The Method:** It uses the **Chain Rule** of calculus to work backward from the Loss ($L$) to each weight, calculating exactly how much that weight contributed to the error (e.g., $\frac{\partial L}{\partial w_1}$).

Step 3: Gradient Descent (The Update)

This is the process of applying the correction.

- The gradients (e.g., $\frac{\partial L}{\partial w_1}$) tell us the direction to update the weights to *minimize* the loss.
- The weights are updated using the rule: $w_{new} = w_{old} - \eta \cdot \text{gradient}$, where $\eta$ is the **learning rate**.

---

## 📋 4. Key Training Terminology

- **Epoch:** One complete pass through the *entire* training dataset.
- **Batch / Mini-Batch:** A small subset of the training data (e.g., 32 samples). The model updates its weights after seeing this small batch.
- **Stochastic Gradient Descent (SGD):** An update strategy where the "batch" is just one single, randomly chosen data point.
- **Iteration:** A single update of the model's weights. (If your batch size is 32, one epoch contains [Total Data / 32] iterations).