ARTIFICIAL NEURAL NETWORKS (ANN) LAB

Lab Demonstrator: Shakeela Shaheen

LAB 01: UNDERSTANDING ANN FROM SCRATCH

LEARNING OBJECTIVES:

- What a biological neuron does and the rough analogy to an artificial neuron.
- The Perceptron: how it works, linear separability, perceptron learning rule.
- Activation functions (sigmoid, tanh, ReLU, softmax): intuition, math, pros/cons.
- High-level biological links: what maps to brain/synapses and what does not clear limitations of the analogy.

0. WHY THIS LAB EXISTS (READ SLOWLY):

Most students think:

"Neural networks work because computers are powerful."

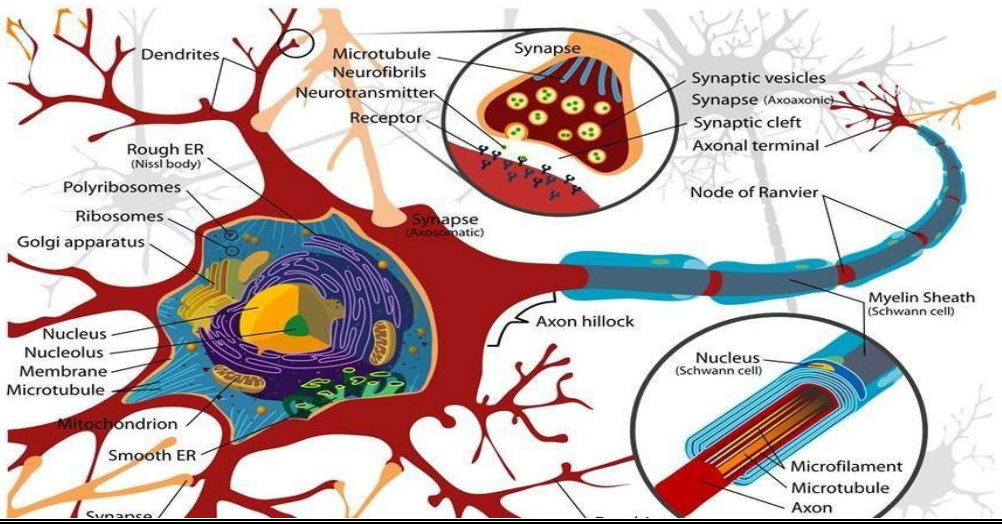
That is **wrong**.

Neural networks work because they are **structured decision systems**:

- They **decide what to keep**
- They **decide what to suppress**
- They **decide when to fire**
- They **decide when to stay silent**

This lab teaches you **how decisions emerge from mathematics**.

1: Biological neuron and Artificial neuron



Electrical portential inside the cell which changes over time but how? 2 mechanisms change this potential (dendrites and axons). If a poetential goes up a threhsold (then cell get rids of the charge as an impulse through axon). The axons are insualted to amke sure charge doesnt come back (short circuited). This in no way is intelligent, where does intelligence come from????& It's just eandom exchange of charge between cells (but how it makes it intelligent?) Note: it's not random but let's just call it (random charges --> eplipisey).& The point where dendrites and axons meet are known as synapses (just like sockets). But these are not actual synapses (not all charges are passed at once like a traditional switch). But in this case synapse control the flow of charge. Synapses have neurotransmitters. Our brain can change the neurotransmitters in the synpses (to make them transmitting and insulating). The can y can be amplified, set to like 60% etc. The intelligence isnt in the cell its in how to cells change torealize the flow of info.

Biological neuron:

- Inputs: signals from many synapses carry chemical/electrical inputs.
- Synapse strength: some inputs are stronger.
- Integration: inputs are summed in the cell body (membrane potential).
- Threshold: if membrane potential crosses threshold → neuron fires (spike).
- Learning: synapse strengths change with activity (plasticity).

Artificial neuron (Perceptron / basic neuron):

- Inputs $x_1, x_2, \ldots, x_n$correspond to sensed features.
- Weights $w_1, \ldots, w_n$correspond to synaptic strengths.
- Weighted sum $z = \sum_i \quad w_i x_i + b$corresponds to integrated membrane potential.
- Activation $a = \phi(z)$corresponds to "firing" — a continuous or threshold response.
- Learning updates weights to reduce prediction error (gradient descent is a training rule, not a biological rule — but conceptually similar to strengthening/weakening synapses).

Important differences:

- Biological neurons spike; artificial neurons usually output real numbers or probabilities.
- Learning rules in deep learning (backprop + SGD) are not the same mechanisms used by brains (biology is more complex: spikes, local plasticity rules, neuromodulators).
- Brains are massively recurrent and structured; standard feedforward MLPs are simple approximations.

WHAT A NEURON REALLY IS??

1. A Neuron Is NOT a Calculator

A neuron is **not**:

- a formula
- a fancy equation
- a random block

A neuron is a **gatekeeper**. It answers ONE question:

"Is the signal strong enough to matter?"

Think of these situations:

- You hear noise → you ignore it
- Someone raises their voice → you pay attention
- Fire alarm rings → you react immediately

Same input type, different **thresholds**.

Threshold = when to care. This is the **soul of a neuron**.

What Is a Neuron in ANN? (Conceptual)

Each neuron:

1. Collects evidence
2. Weighs importance
3. Compares against tolerance
4. Decides output strength

So each neuron is a **micro-decision maker**.

Why Not Treat All Inputs Equally?

Imagine diagnosing a disease:

- Fever matters a lot
- Height matters almost nothing

Weights encode **importance**.

Mathematically:

- High weight → strong influence
- Low weight → weak influence
- Negative weight → suppressive influence

5. Why Bias Exists (EXTREMELY IMPORTANT)

Bias answers:

"Should this neuron ever activate even if inputs are small?"

Without bias:

- Neuron is forced to pass through origin
- No flexibility

Bias allows:

- Shifting sensitivity
- Adjusting strictness
- Moving threshold left/right

2. Perceptron: the simplest neuron classifier

Intuition

- Takes inputs, computes a weighted sum, applies a threshold:

$$y = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

- Learns a linear decision boundary (a line in 2D, a plane in 3D, a hyperplane in general).

When it works & limitations

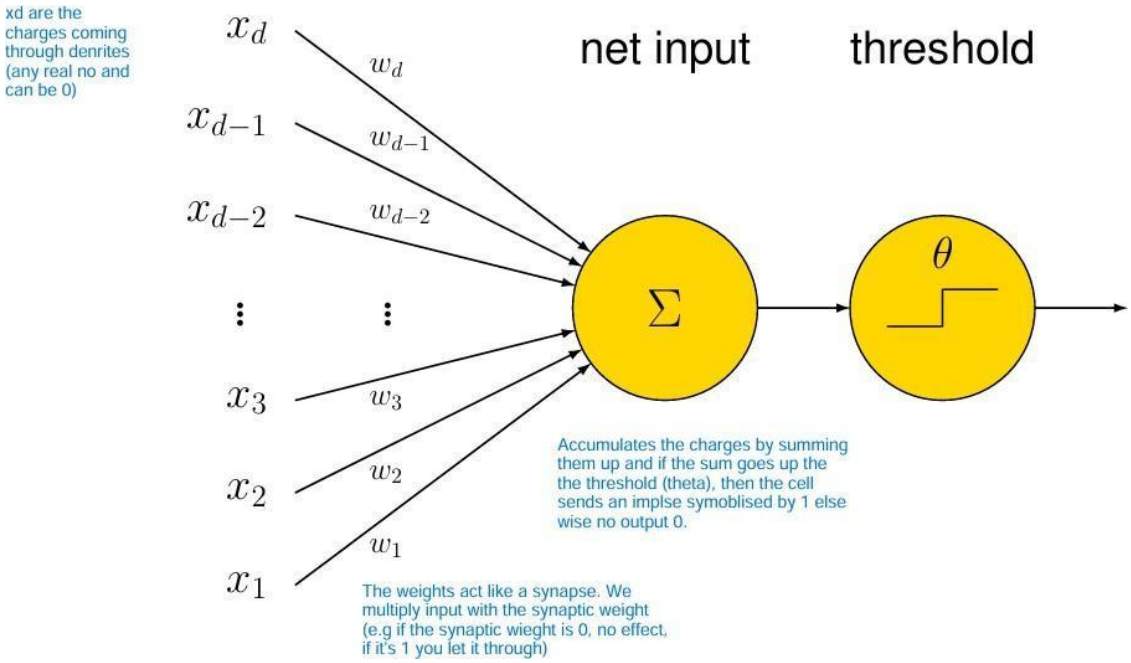
- Works only when classes are **linearly separable** (e.g., OR, AND).
- Cannot learn XOR (not linearly separable). That limitation motivates *multiple layers*.

Perceptron learning rule (intuitive)

- For each misclassified example, move the weights slightly toward correct classification:

$$\mathbf{w} \leftarrow \mathbf{w} + \eta(y_{true} - y_{pred})\mathbf{x}$$

- Repeats until no misclassifications (if data is linearly separable, perceptron converges).



3. Activation functions and "activation energy"

Activation Energy analogy: In biology, a neuron integrates input until membrane potential reaches threshold and fires. In artificial nets, the **activation function** is how the model maps that summed input (the "activation energy") into an output value.

Originally, neurons were written as:

$$\text{output} = \begin{cases} 1 & z > \theta \\ 0 & z \leq \theta \end{cases}$$

This is a **hard threshold**.

Problems:

- Not differentiable
- Small input change → huge output jump
- No gradient information

This makes learning **impossible** using gradient descent.

So we invented **soft thresholds** → activation functions.

A Detector Example:

Example: ‘A’ Detector

Weights plotted on input Grid.

w

+1	+1	+1	+1
+1	-1	-1	+1
+1	-1	-1	+1
+1	+1	+1	+1
+1	-1	-1	+1
+1	-1	-1	+1

$\theta = 15.5$

Example: ‘A’ Detector

x

1	1	1	1
1	0	0	1
1	0	0	1
1	1	1	1
1	0	0	1
1	0	0	1

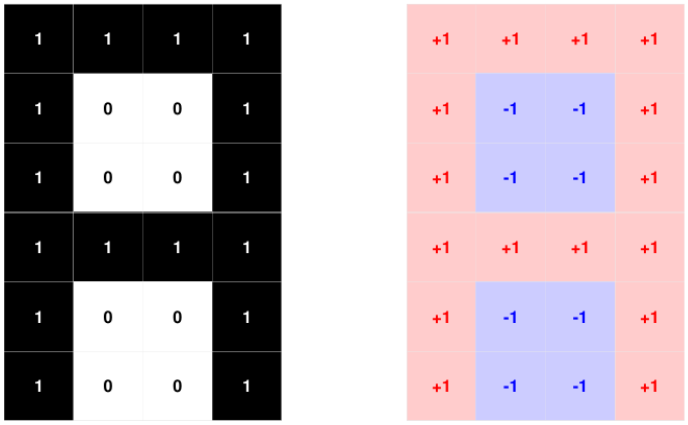
w

+1	+1	+1	+1
+1	-1	-1	+1
+1	-1	-1	+1
+1	+1	+1	+1
+1	-1	-1	+1
+1	-1	-1	+1

$\theta = 15.5$

Example: ‘A’ Detector

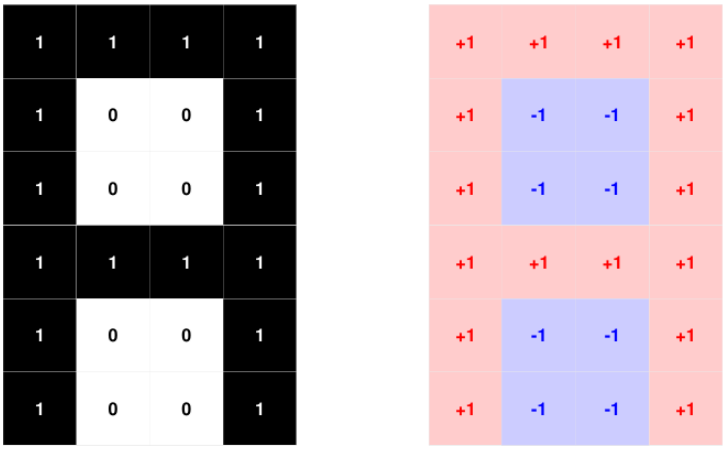
$\mathbf{x} \quad * \quad \mathbf{w} \quad \rightarrow \quad \sum$



$\theta = 15.5$

Example: ‘A’ Detector

$\mathbf{x} \quad * \quad \mathbf{w} \quad \rightarrow \quad \sum$



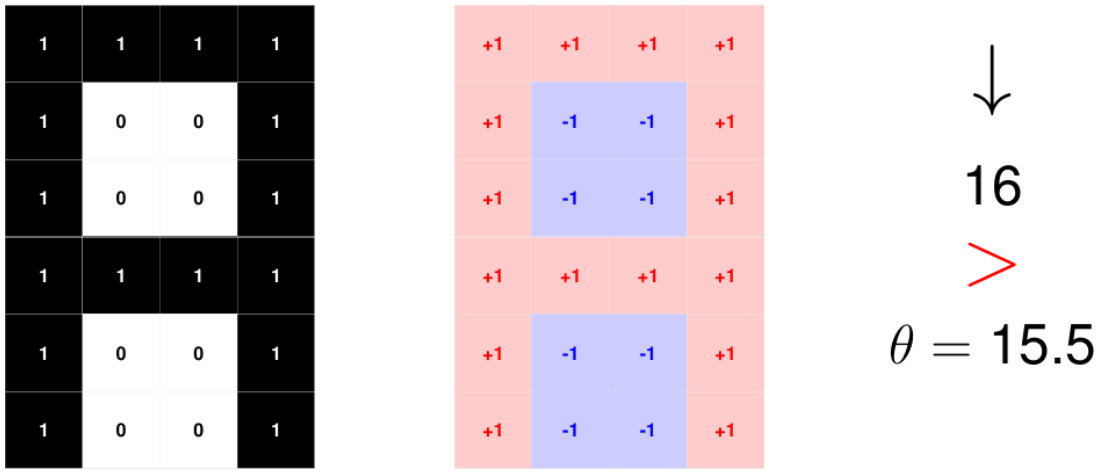
$\downarrow$

16

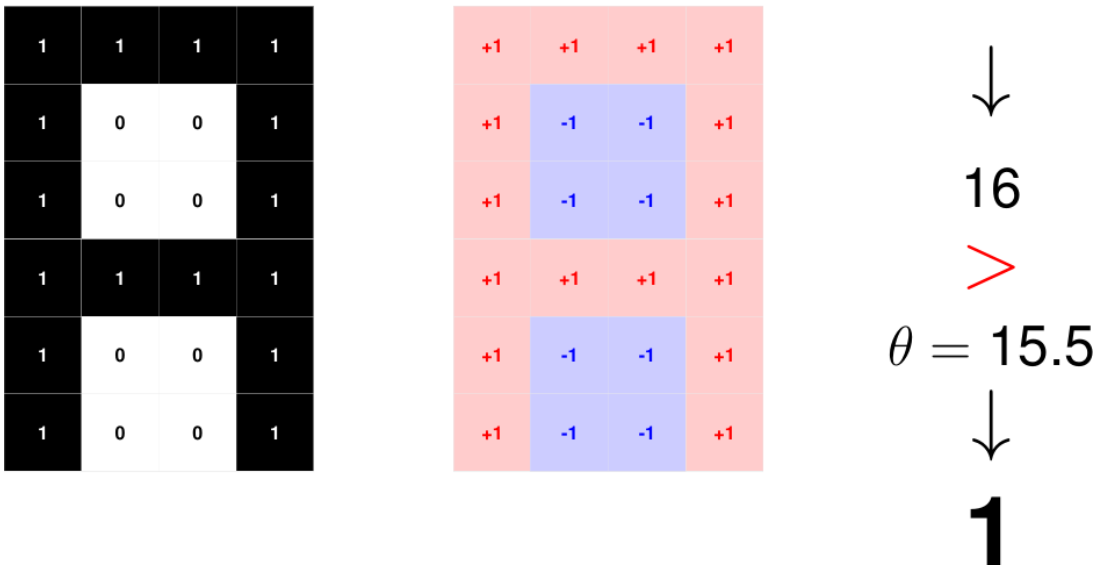
$\theta = 15.5$

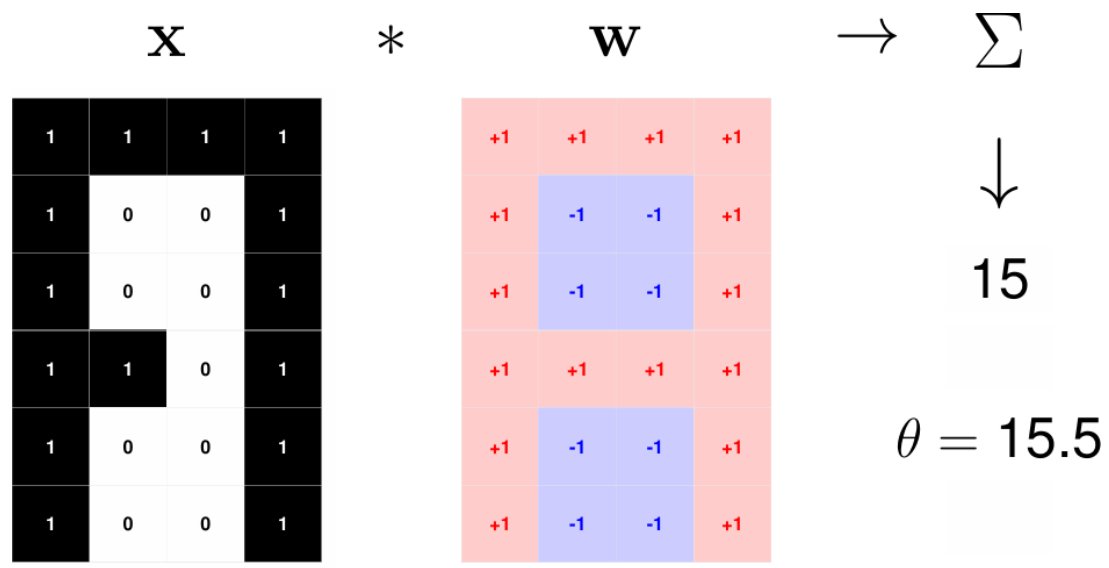
Example: ‘A’ Detector

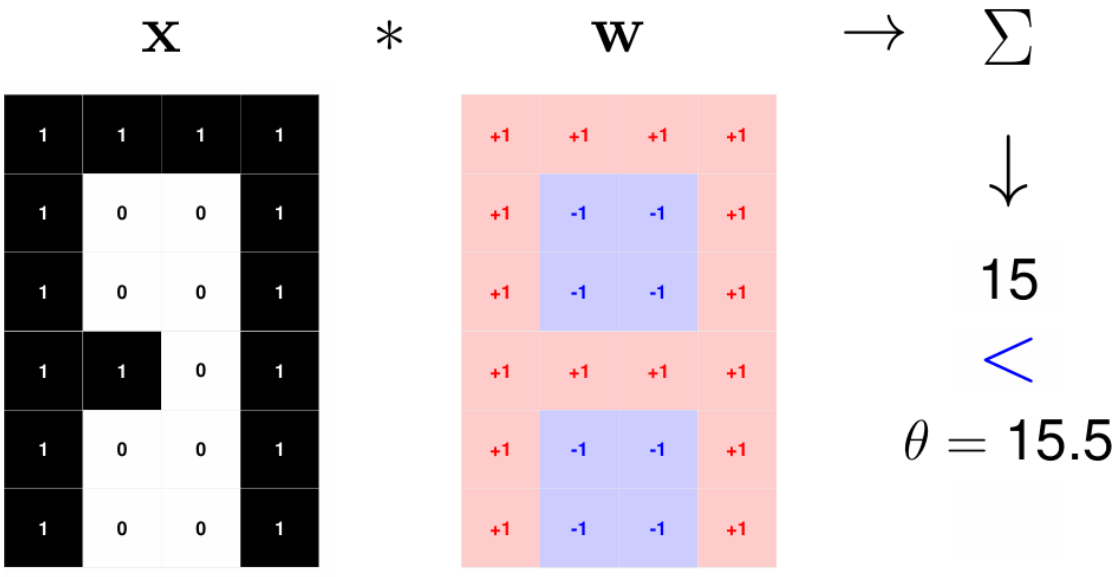
$\mathbf{x}$ $*$ $\mathbf{w}$ $\rightarrow$ $\sum$

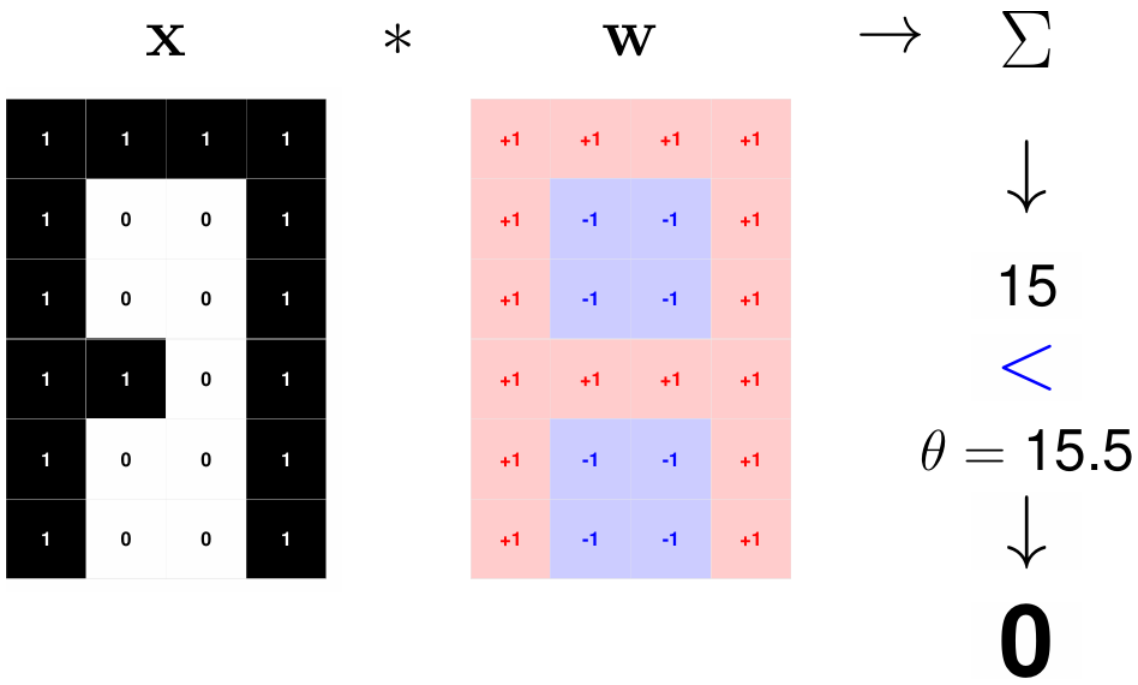


$\mathbf{x}$ $*$ $\mathbf{w}$ $\rightarrow$ $\sum$



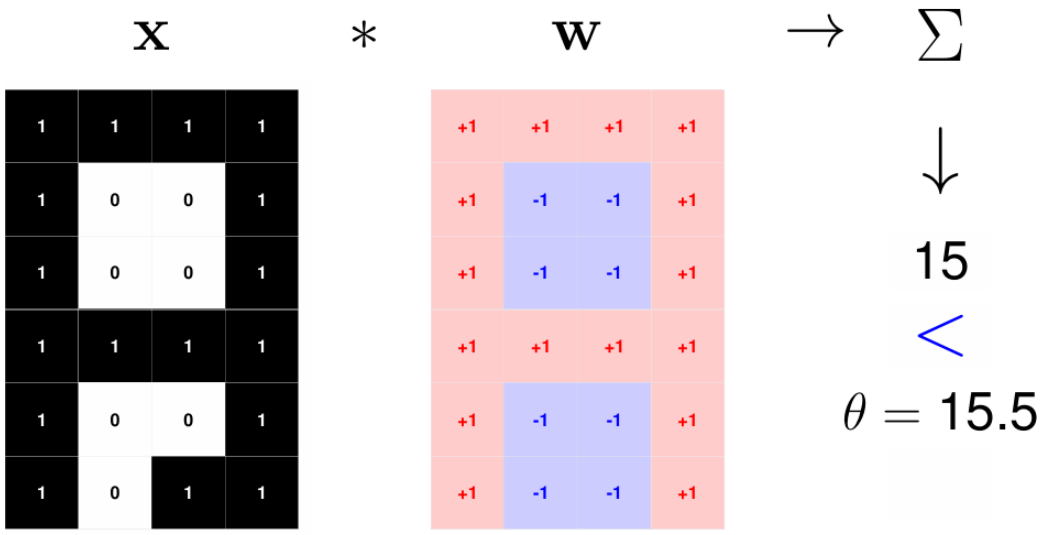

x
*
w
→
∑
1 1 1 1
1 0 0 1
1 0 0 1
1 1 0 1
1 0 0 1
1 0 0 1
+1 +1 +1 +1
+1 -1 -1 +1
+1 -1 -1 +1
+1 +1 +1 +1
+1 -1 -1 +1
+1 -1 -1 +1
↓
15
θ = 15.5

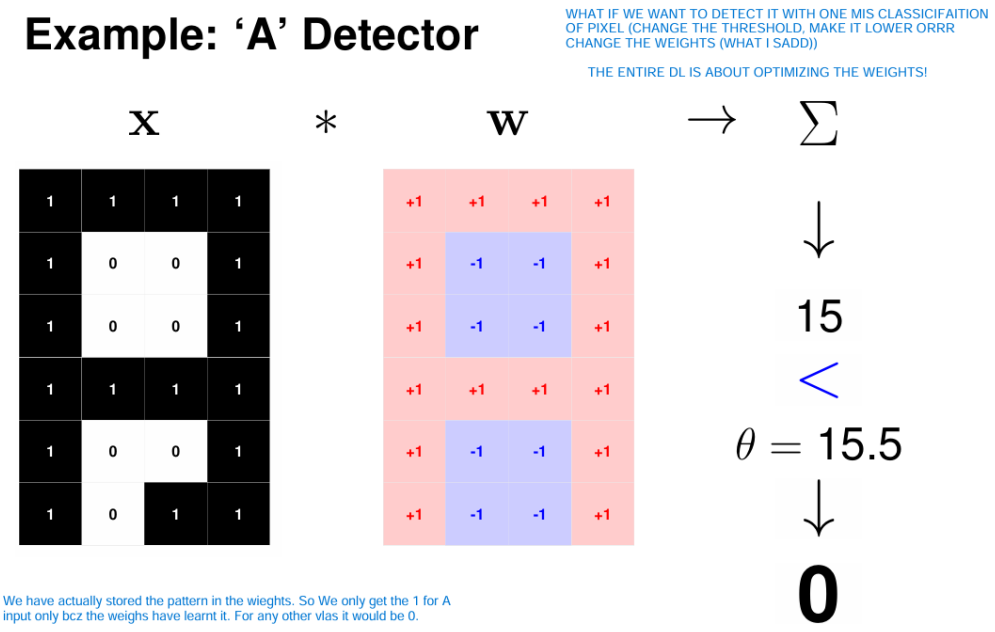

x
*
w
→
∑
1 1 1 1
1 0 0 1
1 0 0 1
1 1 0 1
1 0 0 1
1 0 0 1
+1 +1 +1 +1
+1 -1 -1 +1
+1 -1 -1 +1
+1 +1 +1 +1
+1 -1 -1 +1
+1 -1 -1 +1
↓
15
<
θ = 15.5



Example: 'A' Detector

IMPORTANT





Activation Functions Are NOT Arbitrary
Each activation answers:

"How strongly should I respond once evidence crosses threshold?"

Instead of saying: "Either ON or OFF"

Neural networks say: "How strongly ON?"

That's a **soft margin**.

EXAMPLE:

Inputs as Signals

Let inputs be:

$$x = [x_1, x_2, \dots, x_n]$$

Examples:

- Audio loudness, pitch
- Pixel intensities
- Medical biomarkers

```
x = np.array([x1, x2, x3])
```

Each input has a weight:

$$\boldsymbol{w} = [\boldsymbol{w_1}, \boldsymbol{w_2}, \ldots, \boldsymbol{w_n}]$$

Meaning:

- $w_i > 0 \rightarrow$ encourages activation
- $w_i < 0 \rightarrow$ suppresses activation
- $|\, w_i \,| \rightarrow$ strength of influence

```
w = np.array([w1, w2, w3])
```

Weighted Sum (Evidence Accumulation)

The neuron collects evidence:

$$z = \sum_{i=1}^{n} x_i w_i$$

This is a **dot product**.

```
z = np.dot(x, w)
```

This value answers: **"How strong is the total signal?"**

We now add bias:

$$z = \sum_{i=1}^{n} x_i w_i + b$$

Bias shifts the neuron's sensitivity.

- Large positive bias → neuron fires easily
- Large negative bias → neuron is strict

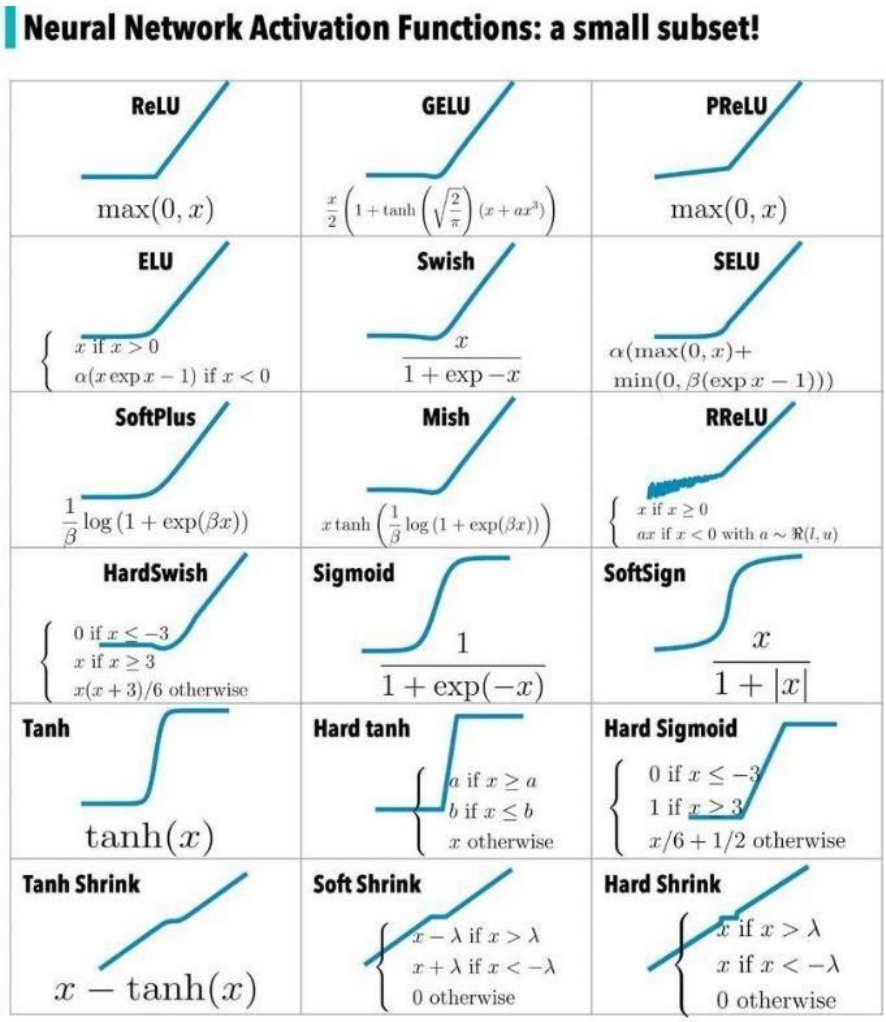
```
z = np.dot(x, w) + b
```

When designing a model, selecting the *right activation function* can dramatically influence its *performance and accuracy*. Each activation function brings a unique approach to handling **data transformation** and **gradient flow** within the network, making them suited for different scenarios. Some of the most prominent activation functions include:

- Sigmoid**:** Common in binary classification tasks, Sigmoid squashes inputs to a range between 0 and 1, providing a smooth gradient that's effective but can be susceptible to the **vanishing gradient problem**.

- Softmax**:** Used primarily in **multi-class classification** tasks, Softmax produces a probability distribution across classes, helping the model interpret **relative class probabilities**.

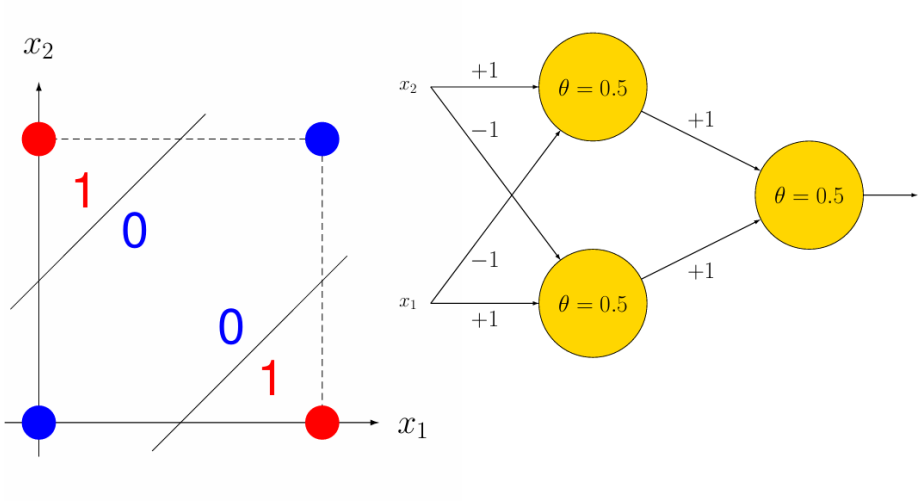
- ReLU **(Rectified Linear Unit):** ReLU has become a popular choice due to its **computational efficiency** and ability to mitigate the vanishing gradient problem. By activating only positive inputs, ReLU accelerates learning while maintaining simplicity.
- **Tanh (Hyperbolic Tangent):** Combining elements of Sigmoid and ReLU, the Tanh activation function outputs values between **-1 and 1**, offering a balanced approach by centering data around zero. This makes Tanh particularly effective for models that benefit from symmetric data and for *improving gradient flow* in deeper networks.

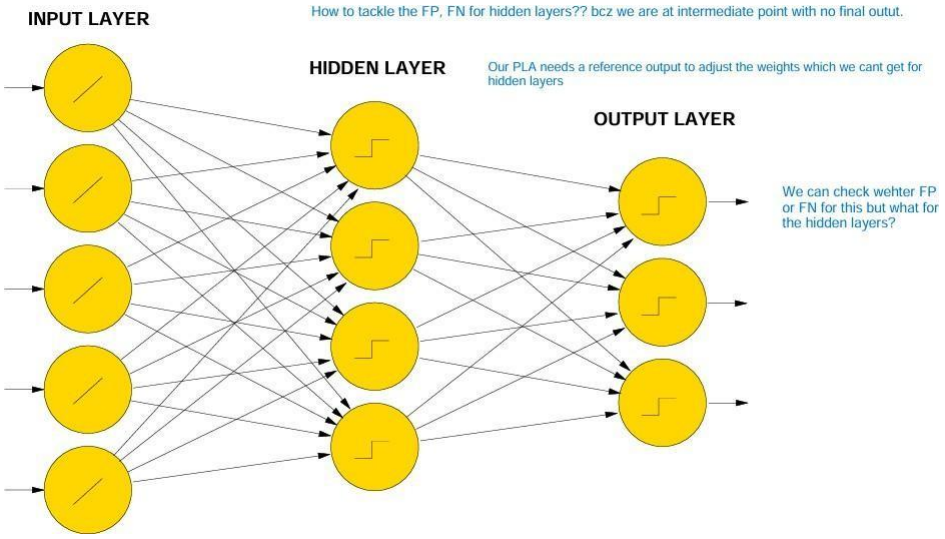


3. Multi-Layer Perceptron (MLP) & Why Depth?

Why add hidden layers?

- A single linear layer with any activation that is linear (or step/perceptron) can only represent linear boundaries.
- Stacking layers with nonlinear activations allows the network to represent **complex nonlinear functions**. Depth lets intermediate layers learn hierarchical features: low-level → mid-level → high-level abstractions.





MLP structure (example: 1 hidden layer)

Input $x\rightarrow$ Linear transform $z^{(1)} = W^{(1)}x + b^{(1)}\rightarrow$ Activation $a^{(1)} = \phi(z^{(1)})\rightarrow$ Output layer $z^{(2)} = W^{(2)}a^{(1)} + b^{(2)}\rightarrow$ Activation/softmax $\rightarrow$ prediction.

Code blocks for MLP:

```
# Perceptron from scratch (2D toy dataset)
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score


# Create a simple linearly separable dataset
X, y = make_blobs(n_samples=300, centers=2, random_state=42, cluster_std=1.2)
y = y.astype(int)  # labels 0/1
# For plotting convenience, pick first two features (already 2D)


# Add bias term by appending 1 to each input vector
Xb = np.hstack([X, np.ones((X.shape[0], 1))])  # shape (n, 3)


# Initialize weights (including bias)
np.random.seed(42)
w = np.random.randn(Xb.shape[1]) * 0.1


def perceptron_predict(Xb, w):
    return (np.dot(Xb, w) > 0).astype(int)


def perceptron_train(Xb, y, w, lr=0.01, epochs=50):
    n = Xb.shape[0]
    for epoch in range(epochs):
        errors = 0
```

```
        # iterate through dataset (stochastic)
        for i in range(n):
            xi = Xb[i]
            yi = y[i]
            pred = perceptron_predict(xi.reshape(1,-1), w)[0]
            if pred != yi:
                # update rule
                w += lr * (yi - pred) * xi
                errors += 1
        if epoch % 10 == 0:
            print(f"Epoch {epoch}, errors = {errors}")
    return w
# Train/test split
Xb_train, Xb_test, y_train, y_test = train_test_split(Xb, y, test_size=0.25, random_state=1)
w = perceptron_train(Xb_train, y_train, w, lr=0.1, epochs=100)

# Evaluate
y_pred = perceptron_predict(Xb_test, w)
print("Perceptron test accuracy:", accuracy_score(y_test, y_pred))

# Plot decision boundary
plt.figure(figsize=(6,5))
# points
plt.scatter(X[:,0], X[:,1], c=y, cmap='bwr', alpha=0.6)
# decision boundary: w0*x + w1*y + b =0 => y = -(w0/w1) x - b/w1
w0, w1, b = w
xs = np.array([X[:,0].min()-1, X[:,0].max()+1])
```

```
ys = -(w0/w1)*xs - (b/w1)
plt.plot(xs, ys, 'k--')
plt.title("Perceptron decision boundary")
plt.show()
```

Neural networks do not think.

They decide, again and again, until intelligence emerges.

LAB TASKS

You are part of a research team building an **AI system to analyze human speech**. The goal is to detect **emotional stress** from speech signals. Speech is **not binary**.
Stress is not "ON or OFF", it exists in degrees.

You are asked to **start small**. Design a **single artificial neuron**, understand how it behaves, then gradually build intelligence into it.

All experiments must be done in **Google Colab** using **NumPy and Matplotlib**.

TASK 1: Build a Neuron That Thinks, Not Just Calculates

Scenario

Your neuron must decide **how stressed a person is** based on speech characteristics.

Step 1: Choose Input Features

Select **any three** meaningful speech-related features, for example:

- Speech rate (slow ↔ fast)
- Pitch variation (flat ↔ shaky)
- Pause duration (short ↔ long)

You are not given data, you simulate values to understand behavior.

Step 2: Assign Weights

Assign weights based on **importance**:

- Which feature contributes **most** to stress?
- Which contributes **least**?

You must **justify** your choices in words.

Step 3: Add Bias (Sensitivity Control)

Bias represents **tolerance**.

Explain:

- What does a **high bias** mean emotionally?
- What does a **low bias** mean?

Step 4: Apply Different Activation Functions

Using the **same inputs, weights, and bias**, apply:

- Sigmoid
- Tanh
- ReLU

Deliverable:

Answer in markdown cells:

Why does the **same neuron** behave differently with different activation functions?

You must explain in terms of:

- Soft margin
- Confidence
- Suppression vs amplification
- Non-linearity

TASK 2: The Threshold Experiment

Your system is **too sensitive**. It flags stress too often. You are asked to **control when the neuron fires**.

Step 1: Fix Everything Except Bias

- Keep inputs constant
- Keep weights constant

- Only change **bias**

Step 2: Sweep Bias Values

Gradually change bias from:

- Very negative → very positive

For each bias value:

- Compute neuron output
- Store result

Step 3: Plot Behavior

Plot:

- X-axis → Bias
- Y-axis → Neuron output

Do this for:

- Sigmoid
- ReLU
- Tanh (optional comparison)

Critical Thinking Question

At what bias does the neuron **"wake up"** for each activation?

Explain:

- Why sigmoid wakes up gradually
- Why ReLU wakes up suddenly
- Why tanh has a neutral zone

TASK 3: Why One Neuron Is Not Enough (The Depth Problem)

Scenario

You test your neuron on two types of speech:

- Calm but fast speakers
- Stressed but slow speakers

The neuron **fails** to separate them.

Part A: Try With ONE Neuron

- Manually compute output for multiple inputs
- Observe overlaps

No training, only reasoning.

Part B: Add a Hidden Layer (Manual Forward Pass)

Design:

- 2 neurons in hidden layer
- 1 neuron in output layer

Manually compute:

1. Hidden layer outputs
2. Final output

Reflection Question (VERY IMPORTANT)

Why does adding a layer help even though no new data was added?

Expected reasoning:

- Feature transformation
- Space bending

- Combination of soft decisions
- Non-linear composition

TASK 4: Softmax

Scenario

Your system now predicts **three emotional states**:

- Calm
- Anxious
- Stressed

The network outputs **raw scores**, not probabilities.

Step 1: Apply Softmax Manually

Given three raw outputs:

$$z_1, z_2, z_3$$

Compute softmax **step by step**.

Step 2: Verify Probability Behavior

Students must verify:

- All outputs are between 0 and 1
- Sum equals 1

Interpretation Question

Why does increasing one score reduce others even if they didn't change?

Expected understanding:

- Competition

- Relative confidence
- Soft margins across classes

FINAL REFLECTION (MANDATORY)

You must answer:

How do activation functions turn rigid mathematical rules into human-like decision-making?

They must reference:

- Soft margins
- Non-linearity
- Sensitivity control
- Confidence vs certainty