SCALER School of Technology × Priyansh Saxena

The Perceptron to the MLP

Single Neuron • Activation Functions • Multi-Layer Perceptrons • Forward Pass

Priyansh Saxena

Scaler School of Technology | Deep Learning I: Neural Networks | Week 2 — Session 1/3

6 May 2026

Recap + Today's Goal

Session 1 Recap

- Hand-crafted features don't scale
- Neural nets learn their own representations
- Features go from simple (edges) to complex (objects)

Today's Question

How does a neural network actually work — from the smallest unit to a full network?

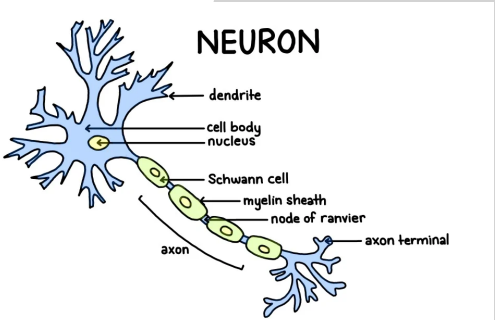
Today we build our first neural network.

Session 1: features → representations (the *why*).
Session 2: neurons → layers → networks (the *how*).

The Biological Inspiration

Real neurons (very simplified):

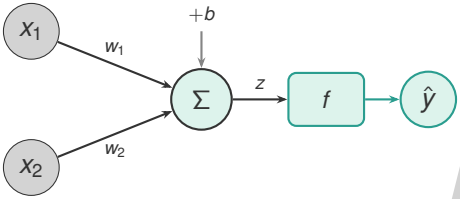
- **Receives** electrical signals through dendrites
- **Fires** if the combined signal exceeds a threshold
- **Connects** to thousands of other neurons via synapses



Important

We're not modelling biology. We're **borrowing the idea**: weighted inputs $\rightarrow$ threshold $\rightarrow$ output.

A Neuron = Weighted Sum + Activation



$$\hat{y} = f(w_1 x_1 + w_2 x_2 + b)$$

1. **Weighted sum:** $z = w_1 x_1 + w_2 x_2 + b$
2. **Activation:** $\hat{y} = f(z)$ — the non-linear function

“You Already Know This”

Logistic Regression

$$P(y{=}1 \mid \mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x} + b)$$

- Weighted sum of features
- Sigmoid squashes to $(0, 1)$
- Binary classification

Single Neuron

$$\hat{y} = \sigma(\mathbf{w}^T\mathbf{x} + b)$$

- Weighted sum of inputs
- Sigmoid activation
- One output

Key Insight

They’re the **same thing**. A single neuron with sigmoid activation *is* logistic regression. You already know neural networks.

A Neuron as a Decision Boundary

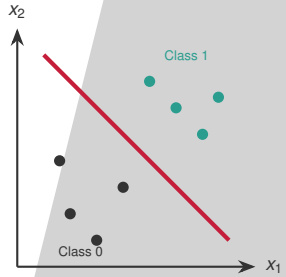
A single neuron with 2 inputs draws a **straight line** in 2D:

$$w_1 x_1 + w_2 x_2 + b = 0$$

- Change **weights** $\rightarrow$ line **rotates**
- Change **bias** $\rightarrow$ line **shifts**

Limitation

What if your data isn't linearly separable? A single neuron **cannot** help.



A single neuron draws **ONE** straight line.

Quiz 1: The Single Neuron

Q1

What does the **bias term** do in a neuron?

Q2

A single neuron can only draw what kind of decision boundary?

Take 1 minute to think, then we'll discuss.

Quiz 1: Answers

A1: The bias shifts the decision boundary

Without a bias, the decision boundary must pass through the origin. The bias allows the neuron to shift the boundary to the right position — like the intercept in a linear equation.

A2: A linear (straight-line) boundary

In 2D: a straight line. In *n*D: a hyperplane. A single neuron can only divide the space with one flat boundary — it cannot curve or bend.

Why Do We Need Activation Functions?

Without activation functions, stacking layers does nothing

Layer 1: $\mathbf{h} = W_1\mathbf{x} + b_1$ Layer 2: $\mathbf{y} = W_2\mathbf{h} + b_2$

Substitute: $\mathbf{y} = \underbrace{(W_2 W_1)}_{\text{one matrix}}\mathbf{x} + \underbrace{(W_2 b_1 + b_2)}_{\text{one bias}}$

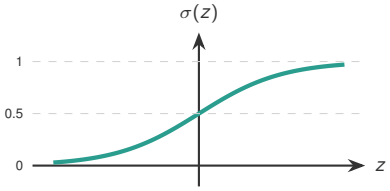
Linear $\times$ linear = still linear. No matter how many layers you stack.

Analogy

Stacking sheets of transparent glass doesn't make a lens. You need a **curved** surface — that's what activation functions provide.

Activation functions add the non-linear "bend" that makes depth meaningful.

Sigmoid: $\sigma(z) = \frac{1}{1+e^{-z}}$



Pros:

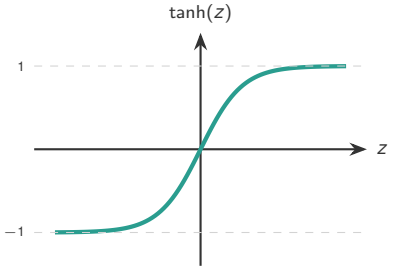
- Outputs in $(0, 1)$ — interpretable as probability
- Smooth and differentiable everywhere

Cons:

- **Vanishing gradient**: for large $|z|$, gradient ≈ 0
- **Not zero-centered**: outputs always positive
- Slow convergence in hidden layers

Today: mostly used at the **output layer** for binary classification.

Tanh: $\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$



Pros:

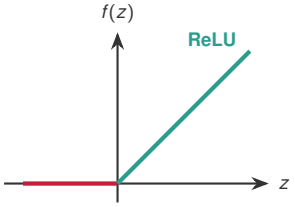
- **Zero-centered**: outputs in $(-1, 1)$
- Stronger gradients than sigmoid near $z = 0$

Cons:

- **Still vanishes** at the extremes ($|z| \gg 0$)
- Computationally heavier than ReLU

Relationship: $\tanh(z) = 2\sigma(2z) - 1$

ReLU and Variants: ReLU$(z) = \max(0, z)$



ReLU: dead simple, computationally cheap.

- No vanishing gradient for $z > 0$
- Sparse activation (many zeros)
- **Dead neuron problem**: if $z < 0$ always, gradient = 0 forever

Variants that fix the dead neuron problem:

- **LeakyReLU**: $\max(\alpha z, z)$ with small α (e.g., 0.01)
- **ELU**: smooth curve for $z < 0$, avoids dead neurons

Default Choice

ReLU is the default activation for hidden layers. Start with ReLU; switch only if needed.

Activation Functions — Side by Side

	Sigmoid	Tanh	ReLU	LeakyReLU
Range	$(0, 1)$	$(-1, 1)$	$[0, \infty)$	$(-\infty, \infty)$
Zero-centered?	No	Yes	No	No
Gradient issue	Vanishing	Vanishing	Dead neurons	None
When to use	Binary output	RNNs	Hidden layers	Hidden layers

Rule of Thumb

ReLU for hidden layers. **Sigmoid** for binary output. **Softmax** for multi-class output.

Quiz 2: Activation Functions

Q1

What happens if you stack 10 linear layers with **no** activation function between them?

Q2

Why is ReLU preferred over sigmoid in hidden layers?

Take 1 minute to think, then we'll discuss.

Quiz 2: Answers

A1: It collapses to a single linear layer

$W_{10} \cdots W_2 W_1 = W'$ — one matrix multiplication. Ten layers give you **no more power** than one. Depth without non-linearity is meaningless.

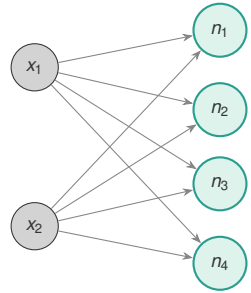
A2: No vanishing gradient + faster computation

Sigmoid gradients shrink toward zero for large $|z|$, slowing learning in deep networks. ReLU has gradient = 1 for $z > 0$, is computationally cheap (just a comparison), and converges much faster.

Coming Up Next

Part 2: Multi-Layer Perceptrons and the XOR Problem

From One Neuron to a Layer

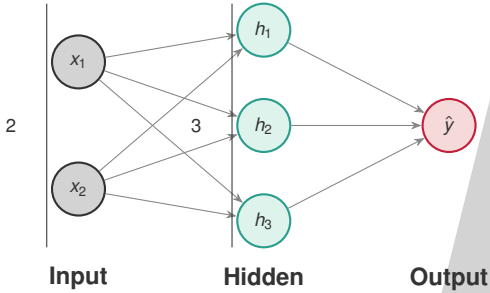


Each neuron has its own weights and draws its own decision line.

Key Point

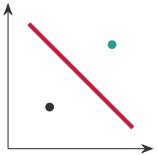
A layer = many boundaries **in parallel**.

Stacking Layers = MLP

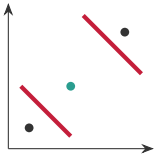


Multi-Layer Perceptron (MLP): Input $\rightarrow$ Hidden layer(s) $\rightarrow$ Output.
Every neuron in one layer connects to every neuron in the next ("fully connected").

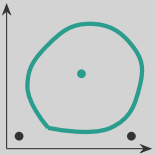
What Each Layer Does



1 layer
Straight lines



2 layers
Convex shapes



3+ layers
Arbitrary shapes

One layer = lines. Two layers = shapes. Three layers = anything.

More layers → more complex decision boundaries → more powerful models.

MLP Architecture Choices

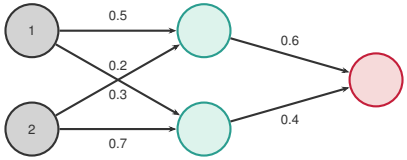
Common questions when designing an MLP:

- **How many hidden layers?**
  Start with 1–2. Add more only if underfitting.
- **How many neurons per layer?**
  Common choices: 32, 64, 128, 256. Powers of 2 for GPU efficiency.
- **Which activation?**
  ReLU for hidden layers. Sigmoid (binary) or softmax (multi-class) for output.
- **Is there a magic formula?**
  No. Architecture design is mostly **empirical** — try, evaluate, adjust.

Note

We'll cover systematic approaches to architecture selection in **Week 3**.

The Forward Pass — Step by Step



Step	Computation	Result
Hidden h_1	$z_1 = 0.5(1) + 0.3(2) + 0 = 1.1$	$\text{ReLU}(1.1) = 1.1$
Hidden h_2	$z_2 = 0.2(1) + 0.7(2) + 0 = 1.6$	$\text{ReLU}(1.6) = 1.6$
Output	$z_o = 0.6(1.1) + 0.4(1.6) + 0 = 1.3$	$\sigma(1.3) \approx 0.79$

Input $[1, 2] \to$ hidden $[1.1, 1.6] \to$ output ≈ 0.79.

Forward Pass in Matrix Form

The same computation, written as matrices:

$$\mathbf{h} = f\Big(W^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\Big) \qquad \hat{y} = g\Big(\mathbf{w}^{(2)} \cdot \mathbf{h} + b^{(2)}\Big)$$

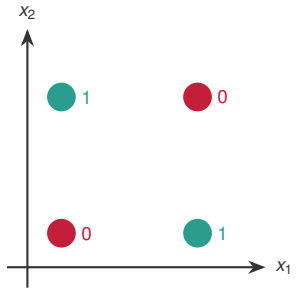
$$\mathbf{h} = \text{ReLU}\left(\begin{bmatrix}0.5 & 0.3\\ 0.2 & 0.7\end{bmatrix}\begin{bmatrix}1\\ 2\end{bmatrix} + \begin{bmatrix}0\\ 0\end{bmatrix}\right) = \text{ReLU}\left(\begin{bmatrix}1.1\\ 1.6\end{bmatrix}\right) = \begin{bmatrix}1.1\\ 1.6\end{bmatrix}$$

$$\hat{y} = \sigma\left(\begin{bmatrix}0.6 & 0.4\end{bmatrix}\begin{bmatrix}1.1\\ 1.6\end{bmatrix}\right) = \sigma(1.3) \approx 0.79$$

Why matrices?

PyTorch does exactly this under the hood — `nn.Linear` is a matrix multiply + bias. Understanding the math helps you debug and design networks.

The XOR Problem

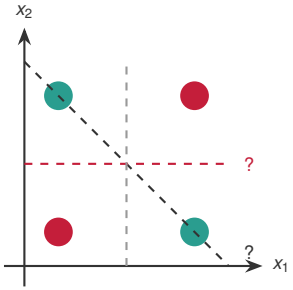


x_1	x_2	XOR
0	0	0
0	1	1
1	0	1
1	1	0

Challenge

Can you draw **ONE** straight line to separate the red from the blue?

Why One Neuron Fails



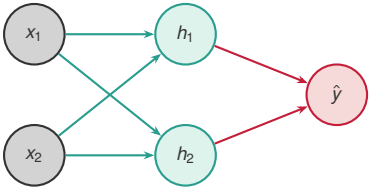
Every possible line fails.

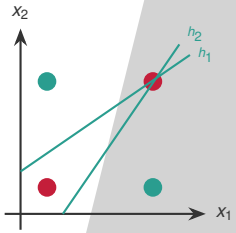
No matter how you rotate or shift a single line, you always misclassify at least one point.

Historical Impact

XOR is **NOT** linearly separable. Minsky & Papert proved this in 1969 — and it nearly killed neural network research.

The Solution: Add a Hidden Layer

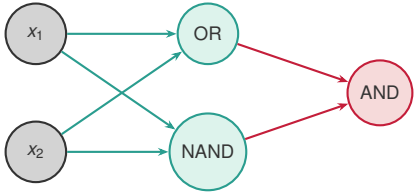




Key Insight

Two neurons = two lines. The output neuron **combines** them to carve out the correct region. One neuron can't do it; two can.

XOR Solved — Step by Step



x_1	x_2	OR (h_1)	NAND (h_2)	AND ($\hat{y}$) = XOR
0	0	0	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	0

XOR = OR **AND** NAND. Two hidden neurons $\rightarrow$ problem solved.

Discussion: The 17-Year Gap

Discussion Prompt

Minsky & Papert killed neural networks in 1969 by showing a single neuron can't solve XOR. The fix — MLPs trained with backpropagation — came in 1986.

What held things back for 17 years?

Consider:

- The proof showed single-layer networks fail. Did people try adding layers?
- What was missing — the architecture or the **training algorithm**?
- How does academic consensus shape (or block) research directions?

The Bigger Picture

XOR is a toy problem. But the lesson is universal:

- **Single neurons are limited** — they can only draw straight lines.
- **Depth gives you power** — hidden layers transform the input space.
- **Non-linearity is essential** — without it, depth is meaningless.

Connecting to Session 1

Remember **representation learning**? The hidden layer learns the right internal features so that XOR becomes linearly separable in the new space. That's exactly what we said last session: neural networks learn their own representations.

Recap — 5 Key Takeaways

1. **Neuron = weighted sum + activation** = logistic regression. The atom of deep learning.
2. **Activation functions add non-linearity.** Without them, stacking layers is useless.
3. **MLPs stack layers** to learn complex, non-linear decision boundaries.
4. **Forward pass: multiply, add, activate, repeat.** Data flows from input to output through matrix operations.
5. **XOR proves you need depth.** One neuron fails; a hidden layer succeeds by transforming the input space.

Final Quiz

Q1

What's the difference between a single neuron with sigmoid activation and logistic regression?

Q2

Why can't a single neuron solve XOR?

Q3

What does adding a hidden layer allow the network to do?

Take 2 minutes, then we'll go through the answers.

Final Quiz: Answers

A1: Nothing — they're the same model

A single neuron with sigmoid computes $\hat{y} = \sigma(\mathbf{w}^T\mathbf{x} + b)$, which is exactly logistic regression. Neural networks generalize this by stacking many such units.

A2: XOR is not linearly separable

A single neuron can only draw one straight line (hyperplane). No single line can separate the XOR pattern — the classes are interleaved diagonally.

A3: Non-linear decision boundaries via space transformation

A hidden layer re-maps inputs into a new representation where previously inseparable classes become separable. Each layer adds expressive power through non-linear transformations.

Thank You!

Questions?

Priyansh Saxena

All lecture materials available at:

www.priyanshsaxena.com/sst-deep-learning-nn