



# Neurons, the Perceptron and the MLP

Deep learning | The artificial neuron, the perceptron and its learning rule, XOR and why depth matters, activation functions, the multilayer perceptron forward pass, parameter counts, universal approximation

#DeepLearning #Neuron #Perceptron #MLP #Activation

## ● From one neuron to a network

| Artificial neuron       |                                                                                                                                                                                                                                                                                                                                              |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| In one sentence<br>●○○○ | A tiny function: multiply each input by a weight, add them up with a bias, and squash the total through a non-linear curve.                                                                                                                                                                                                                  |
| Practitioner<br>●●○○    | A dense layer is many neurons sharing the same inputs, written as one matrix product $h = \varphi(Wx + b)$ ; a network stacks layers. <ul style="list-style-type: none"><li>weights and biases are learned by gradient descent (sheets 02-03)</li><li>the activation <math>\varphi</math> is what lets stacked layers model curves</li></ul> |
| Expert<br>●●●○          | Each neuron draws a hyperplane $w \cdot x + b = 0$ in input space; the activation decides how the two sides are scored. Hidden layers build new features in which the classes become separable. <ul style="list-style-type: none"><li>depth = composition of features</li><li>width = how many features per layer</li></ul>                  |
| Under the hood<br>●●●●  | Biological neurons inspired the model (McCulloch and Pitts, 1943) but the analogy stops at 'weighted inputs and a threshold': real neurons spike in time, have dendritic computation and do not learn by backpropagation. <ul style="list-style-type: none"><li>modern networks are just differentiable functions</li></ul>                  |
| Analogy                 | <i>A neuron is a committee member who weighs each piece of evidence, adds a personal bias, and votes only when the total crosses a threshold.</i>                                                                                                                                                                                            |

The artificial neuron at four depths: the first band is enough to follow the rest of the folder

Before Neurons and MLPs, can you explain...

|                                                                                                                              | Not yet                  | Partly                   | Yes                      |
|------------------------------------------------------------------------------------------------------------------------------|--------------------------|--------------------------|--------------------------|
| <b>MUST KNOW</b> Vectors and dot products<br>$w \cdot x = \sum w_i x_i$ ; Learn: maths: vectors and dot products             | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| <b>MUST KNOW</b> Matrix times vector<br>row $i$ of $W$ dotted with $x$ ; Learn: maths: matrices and linear maps              | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| <b>MUST KNOW</b> exp and log<br>Learn: maths: algebra refresher                                                              | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| <b>HELPFUL</b> Derivatives<br>slope of a function; used for activation derivatives; Learn: maths: derivatives and chain rule | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| <b>HELPFUL</b> Logistic regression<br>is exactly one sigmoid neuron; Learn: machine-learning: logistic regression            | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |

What this sheet assumes (recalled in one line each, taught in the maths folder)

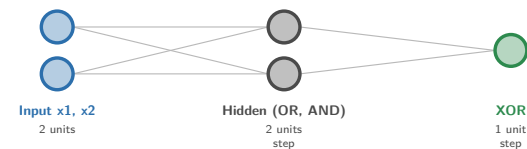
| Object         | Formula                                | Shapes                                        |
|----------------|----------------------------------------|-----------------------------------------------|
| One neuron     | $y = \varphi(w^T x + b)$               | $x$ : $n$ , $w$ : $n$ , $b$ and $y$ : scalars |
| Dense layer    | $h = \varphi(Wx + b)$                  | $W$ : $m \times n$ , $b$ and $h$ : $m$        |
| Softmax output | $p_i = e^{z_i} / \sum_{j=1}^K e^{z_j}$ | $z$ , $p$ : $K$ classes                       |

where  $x$  = input vector (features) ( $n$  numbers);  $w$ ,  $W$  = weight vector of one neuron; weight matrix of a layer (one row per neuron) (learned);  $b$  = bias, one per neuron (learned);  $\phi$  = activation function, applied element-wise (-);  $m$ ,  $n$  = number of outputs and inputs of the layer (count);  $z_i$ ,  $p_i$  = logit and probability of class  $i$  (-, 0-1);  $K$  = number of classes (count)

The three formulas of this sheet

## ● The perceptron and the XOR wall

|               |                                                                                                                                                                                                                                                         |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Model         | $\hat{y} = \text{step}(w \cdot x + b)$ : output 1 if the sum is $> 0$ , else 0 (Rosenblatt, 1958)                                                                                                                                                       |
| Learning rule | $w \leftarrow w + \text{lr} \cdot (y - \hat{y}) \cdot x$ and $b \leftarrow b + \text{lr} \cdot (y - \hat{y})$ : change weights only on mistakes; $\text{lr}$ = learning rate (written $\eta$ in papers), $y$ = true label (0/1), $\hat{y}$ = prediction |
| Guarantee     | if the data are linearly separable, the rule stops after a finite number of mistakes (perceptron convergence theorem); otherwise it never settles                                                                                                       |
| Limit         | one neuron = one straight boundary: AND and OR are learnable, XOR is not (Minsky and Papert, 1969, a reason for the first 'AI winter')                                                                                                                  |



XOR with one hidden layer of two step neurons:  $h1 = \text{step}(x1 + x2 - 0.5)$  is OR,  $h2 = \text{step}(x1 + x2 - 1.5)$  is AND,  $y = \text{step}(h1 - h2 - 0.5)$  is 'OR but not AND'. Check (1,1):  $h1 = 1$ ,  $h2 = 1$ ,  $1 - 1 - 0.5 < 0 \rightarrow 0$

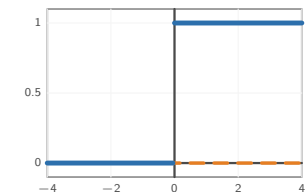
|                                                                                                                              |    |                                                                   |
|------------------------------------------------------------------------------------------------------------------------------|----|-------------------------------------------------------------------|
| <b>Perceptron</b><br>one neuron, step activation, perceptron rule<br>Tell: Does it have a hidden layer trained by gradients? | vs | <b>MLP</b><br>several layers, smooth activations, backpropagation |
| <b>Weight</b><br>scales one input<br>Tell: Is it multiplied by an input?                                                     | vs | <b>Bias</b><br>constant added to the sum                          |
| <b>Logit</b><br>raw score, any real number<br>Tell: Has softmax or sigmoid been applied?                                     | vs | <b>Probability</b><br>0-1, sums to 1 over classes                 |
| <b>Width</b><br>units per layer<br>Tell: Are you counting along a layer or across layers?                                    | vs | <b>Depth</b><br>number of layers                                  |

Notions this sheet's readers mix up

## Activation functions

### Step (Heaviside)

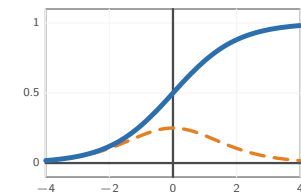
$\text{step}(x) = 1$  if  $x \geq 0$ , else 0  
 $\text{step}'(x) = 0$  (undefined at 0): no gradient to learn from



$f(0) = 1 \cdot \text{range } [0, 1]$   
slope 0 everywhere it is defined

### Sigmoid

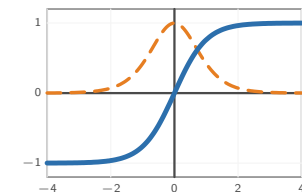
$\sigma(x) = 1 / (1 + e^{-x})$   
 $\sigma'(x) = \sigma(x) \cdot (1 - \sigma(x))$



$f(0) = 0.5 \cdot \text{range } (0, 1)$   
 $\max f' = 0.25$  at  $x \approx 0$

### Tanh

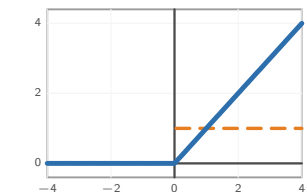
$\tanh(x) = (e^x - e^{-x}) / (e^x + e^{-x})$   
 $\tanh'(x) = 1 - \tanh^2(x)$



$f(0) = 0 \cdot \text{range } (-1, 1)$   
 $\max f' = 1$  at  $x \approx 0$

### ReLU

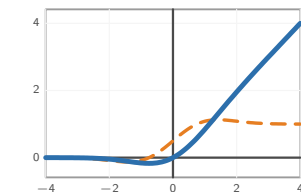
$\text{ReLU}(x) = \max(0, x)$   
 $\text{ReLU}'(x) = 1$  if  $x > 0$ , else 0



$f(0) = 0 \cdot \text{range } [0, \infty)$   
 $\max f' = 1$  for  $x > 0$

### GELU

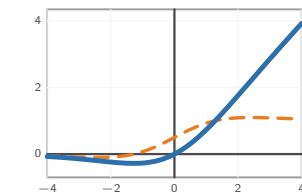
$\text{GELU}(x) = x \cdot \Phi(x)$   
 $\text{GELU}'(x) = \Phi(x) + x \cdot \varphi(x)$



$f(0) = 0 \cdot \text{range } [-0.17, \infty)$   
 $\max f' \approx 1.13$  at  $x \approx 1.4$

### SiLU / Swish

$\text{SiLU}(x) = x \cdot \sigma(x)$   
 $\text{SiLU}'(x) = \sigma(x) \cdot (1 + x \cdot (1 - \sigma(x)))$



$f(0) = 0 \cdot \text{range } [-0.28, \infty)$   
 $\max f' \approx 1.1$  at  $x \approx 2.4$

—  $f(x)$ , the activation      - - -  $f'(x)$ , its derivative

Where:  $x$ : pre-activation of a neuron,  $z = w \cdot x + b$  (unitless);  $f(x)$ : the neuron's output;  $f'(x)$ : derivative, the factor backpropagation multiplies the gradient by (near 0 = saturated, the gradient vanishes);  $e \approx 2.718$ ;  $\sigma$ : sigmoid;  $\Phi$ ,  $\varphi$ : standard normal cumulative distribution and density

Six activations and their derivatives (dashed). The derivative is what backpropagation multiplies through each layer: a flat region means no learning signal

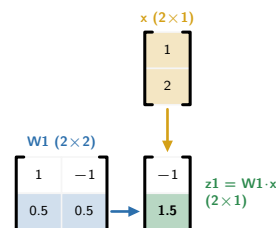
| Function                | Formula                                            | Derivative max                    | Use today                                                                        |
|-------------------------|----------------------------------------------------|-----------------------------------|----------------------------------------------------------------------------------|
| <b>Sigmoid</b> $\sigma$ | $1 / (1 + e^{-z})$ , output 0-1                    | 0.25 at $z = 0$                   | binary output, gates of LSTMs (sheet 07); avoid in deep hidden layers            |
| <b>tanh</b>             | $(e^z - e^{-z}) / (e^z + e^{-z})$ , output -1 to 1 | 1 at $z = 0$                      | RNN hidden states; zero-centred, still saturates                                 |
| <b>ReLU</b>             | $\max(0, z)$                                       | 1 for $z > 0$ , 0 below           | default for CNNs and MLPs since AlexNet (2012); cheap, no saturation for $z > 0$ |
| <b>Leaky ReLU</b>       | $\max(\alpha z, z)$ , $\alpha \approx 0.01$        | 1 ( $\alpha$ for $z < 0$ )        | avoids dying ReLU                                                                |
| <b>GELU</b>             | $z \times (\text{standard normal CDF at } z)$      | $\approx 1.13$ at $z \approx 1.4$ | BERT, GPT-2/3, ViT; smooth ReLU                                                  |
| <b>SiLU / Swish</b>     | $z \cdot \sigma(z)$                                | $\approx 1.1$                     | inside SwiGLU feed-forward blocks of most 2024-2026 LLMs                         |
| <b>Softmax</b>          | $e^{z_i} / \sum e^{z_j}$                           | -                                 | multi-class output layer, attention weights (sheet 08)                           |

Which activation where (as of 2026)

## ✗ Forgetting the activation

Two dense layers without an activation between them are one dense layer:  $W_2(W_1x + b_1) + b_2 = (W_2W_1)x + (W_2b_1 + b_2)$ . Any depth of purely linear layers can only draw a straight boundary. The non-linearity is not decoration; it is what depth buys.

## ● The multilayer perceptron forward pass



Shapes:  $W1$  is  $2 \times 2$ ,  $x$  is  $2 \times 1 \rightarrow z1 = W1 \cdot x$  is  $2 \times 1$  (the inner sizes 2 = 2 must match)

$z1[2,1] = \sum W1[2,k] \cdot x[k,1]$  for  $k = 1 \dots 2 = 0.5 \cdot 1 + 0.5 \cdot 2 = 1.5$

Cost:  $m \cdot n \cdot p = 2 \cdot 2 \cdot 1 = 4$  multiplications for the whole of  $z1$

Where:  $W1$ :  $m \times n$  matrix ( $m = 2$  rows,  $n = 2$  columns);  $x$ :  $n \times p$  matrix ( $p = 1$ );  $z1[i,j]$ : entry in row  $i$ , column  $j$  of  $z1$  = dot product of row  $i$  of  $W1$  with column  $j$  of  $x$ ;  $k$ : index over the  $n$  shared columns of  $W1$  / rows of  $x$ ;  $\Sigma$ : sum

The first layer of the worked example as a matrix product: each pre-activation is one row of  $W1$  dotted with  $x$

**PROBLEM** Compute the output probability of a 2-2-1 MLP (ReLU hidden layer, sigmoid output) for  $x = (1, 2)$

**GIVEN**

- $W1 = \begin{bmatrix} 1 & -1 \\ 0.5 & 0.5 \end{bmatrix}$ ,  $b1 = (0, -1)$ : hidden layer weights and biases
- $w2 = (2, 3)$ ,  $b2 = 0.5$ : output neuron weights and bias
- $\text{ReLU}(z) = \max(0, z)$ ;  $\sigma(z) = 1 / (1 + e^{-z})$

#### 1 Hidden pre-activations

$$z1 = W1 \cdot x + b1 = (1 - 2 + 0, 0.5 + 1 - 1) = (-1, 0.5)$$

Why: one dot product per hidden neuron

#### 2 Hidden activations

$$h = \text{ReLU}(z1) = (0, 0.5)$$

Why: the first unit is off for this input

#### 3 Output logit

$$z2 = w2 \cdot h + b2 = 2 \cdot 0 + 3 \cdot 0.5 + 0.5 = 2.0$$

Why: the output neuron reads the learned features, not  $x$

#### 4 Probability

$$p = \sigma(2.0) = 1 / (1 + e^{-2}) = 0.881$$

Why: sigmoid maps a logit to 0-1 for a binary class

#### 5 Count parameters

$$(2 \cdot 2 + 2) + (2 + 1) = 9$$

Why: weights plus biases of each layer

**ANSWER**  $p \approx 0.88$  that  $x$  belongs to the positive class, from 9 parameters

**CHECK** ✓ If  $b1$  were  $(0, 0)$ ,  $z1 = (-1, 1.5)$ ,  $h = (0, 1.5)$ ,  $z2 = 5.0$  and  $p = 0.993$ : the bias moves the threshold

A 2-2-1 network by hand: the same arithmetic a GPU does billions of times per second

#### Parameters of a dense layer

**FORMULA**  $P = n_{in} \times n_{out} + n_{out}$

**WHERE**

- $P$  = trainable parameters of the layer (count)
- $n_{in}$  = inputs to the layer (count)
- $n_{out}$  = units (neurons) of the layer (count)

**WORKED**

$n_{in} = 784$   
 $n_{out} = 256$   
 $P = 784 \times 256 + 256$   
 $P = 200\,960$  parameters

**RESULT**  $P = 200\,960$  parameters

**READ IT** An MNIST MLP 784-256-10 has  $200,960 + 2,570 = 203,530$  parameters; the first layer holds 99 % of them. Dense layers on raw pixels explode in size, which is why images use convolutions (sheet 06).

**MISTAKES**

- forgetting the biases (one per output unit)
- counting activations as parameters: they have none, except learnable variants such as PReLU

The same MLP in PyTorch (sheet programming/PyTorch fundamentals for the API)

```
import torch, torch.nn as nn
mlp = nn.Sequential(
    nn.Linear(784, 256), nn.ReLU(), # hidden layer
    nn.Linear(256, 10), # 10 logits
) # softmax lives in the loss
logits = mlp(torch.randn(32, 784)) # batch of 32 -> (32, 10)
print(sum(p.numel() for p in mlp.parameters())) # 203530
```

## Why depth: universal approximation and its limits

### Building a curve with LEGO bricks

One brick: a step up at some position

Brick height

More, thinner bricks: a smoother staircase

Any silhouette, given enough bricks

### One hidden layer approximating a function

One ReLU or sigmoid unit: a bend or step at  $w \cdot x + b = 0$

Output weight of that unit

More hidden units: a smaller approximation error

Any continuous function on a bounded region (Cybenko 1989, Hornik 1991)

### WHERE THE ANALOGY BREAKS

The theorem guarantees that the bricks exist, not that gradient descent finds them, nor that the number needed is reasonable: some functions need exponentially many units in one layer but few with depth.

Universal approximation with one hidden layer, in LEGO terms

### Depth composes

Each layer reuses the previous layer's features: edges  $\rightarrow$  textures  $\rightarrow$  parts  $\rightarrow$  objects. Deep nets represent some functions with exponentially fewer units than shallow ones.

### Inductive bias

Architecture choices (convolution, recurrence, attention) build assumptions in; the plain MLP has the fewest, so it needs the most data.

### Width memorises

Very wide shallow nets can fit training data but generalise worse per parameter on images, text and audio.

### Where MLPs live in 2026

Tabular heads, the feed-forward block inside every transformer layer (sheet 09), reward and value heads, small control policies.

### EXAM TIP

Interview trap: 'Can one hidden layer represent any function, so why go deep?' Answer: representable is not learnable or efficient. Depth reduces the number of units needed and gives hierarchical features that gradient descent finds and that generalise.

#### MYTH

✗ Neural networks work like the brain

✗ More hidden layers always help

✗ Sigmoid is the standard activation

✗ Universal approximation means any MLP can learn anything

#### FACT

✓ They borrow one idea: weighted inputs and a threshold  
No spikes, no time dynamics, and learning by backpropagation has no confirmed biological counterpart.

✓ Plain deep stacks stop training past  $\sim 10$ -20 layers  
Vanishing gradients (sheet 02) needed ReLU, careful initialisation, normalisation and residual connections (sheet 05) to go deeper.

✓ ReLU-family and GELU/SiLU replaced it in hidden layers  
Sigmoid saturates and its derivative never exceeds 0.25; it survives in binary outputs and gates.

✓ It is an existence result  
It promises neither a size, nor a training method, nor generalisation to unseen data.

## 🚩 Recap and self-test

**Neuron** =  $\varphi(\mathbf{w} \cdot \mathbf{x} + b)$   
weighted sum, bias, non-linearity

**Perceptron** = one hyperplane  
learns AND/OR, fails XOR

**Hidden layers make new features**  
XOR = OR and not AND

**No activation, no depth**  
linear layers collapse into one

**ReLU, GELU, SiLU in hidden layers**  
sigmoid/softmax at outputs

**Dense params** =  $n_{in} \cdot n_{out} + n_{out}$   
first layer on raw inputs dominates

**Q1 Write the output of one neuron.**

**A**  $y = f(\mathbf{w} \cdot \mathbf{x} + b)$ ,  $f$  the activation: weighted sum of inputs plus bias, through an activation.

**Q2 Perceptron learning rule?**

**A**  $\mathbf{w} \leftarrow \mathbf{w} + \text{lr} \cdot (y - \hat{y}) \cdot \mathbf{x}$ ,  $b \leftarrow b + \text{lr} \cdot (y - \hat{y})$ ,  $\text{lr}$  = learning rate; it updates only on mistakes.

**Q3 Why can a single perceptron not learn XOR?**

**A** XOR is not linearly separable; one neuron draws one straight boundary.

**Q4 Why are activation functions needed?**

**A** Without them, stacked linear layers equal one linear layer and can only draw straight boundaries.

**Q5 Maximum derivative of the sigmoid, and why it matters?**

**A** 0.25 at  $z = 0$ ; multiplied over many layers it shrinks gradients (vanishing gradients).

**Q6 Parameters of `nn.Linear(512, 128)`?**

**A**  $512 \times 128 + 128 = 65,664$ .

**Q7 What is a logit?**

**A** An unnormalised score; softmax or sigmoid turns it into a probability.

**Q8 What does the universal approximation theorem guarantee, and not guarantee?**

**A** One wide hidden layer can approximate any continuous function on a bounded region; not how many units, not that training finds them, not generalisation.

**Q9 What is a dying ReLU and one fix?**

**A** A unit stuck at negative pre-activation outputs 0 with zero gradient; use Leaky ReLU, lower learning rate or better initialisation.

**Q10 Which activation sits inside most 2024-2026 LLM feed-forward blocks?**

**A** SiLU (Swish), inside the gated SwiGLU block.