AI Research Labs

Section 3 of 11 · 4 lessons

Multi-layer Perceptrons

How layers of neurons are written as matrices, how a hidden layer solves XOR, and why depth and non-linearity give networks their power.

3.1 Layers, Matrices and Shapes

3.2 Solving XOR with a Hidden Layer

3.3 Why Depth Helps: Composing Regions

3.4 Why Non-linearity Is Essential



Free course with quizzes, progress tracking and Scoopy, an AI tutor:
airesearchlabs.si/learn/dl

Deep Learning Foundations · Edition November 2026 · © 2026 AI Research Labs. Free for personal study.

Where this section fits

Every lesson is a step in building a model. These are the steps this section teaches; the Big picture PDF and the online map show how they connect to the rest.

M2

BUILDING AND TRAINING A NETWORK · DENSE LANE · STEP M2

Stack layers

How do hidden layers create new representations?

Lessons here: 3.1, 3.2, 3.3, 3.4 · feeds into Choose the activations

Layers, Matrices and Shapes

A multi-layer perceptron stacks layers of neurons. How one layer is written as a matrix product, how to read the shapes, how to count parameters, and the two conventions (lecture vs code) you must not mix.

One layer as a matrix product

A layer applies many neurons to the same input at once. Stack each neuron's weights as a **row** of a matrix W and its bias into a vector b:

$$y^{(0)} = x, \qquad y^{(k)} = f_k\left(W_k\, y^{(k-1)} + b_k\right), \quad k = 1, \ldots, L$$

The output of one layer is the input of the next. Computing all layers in order is the **forward pass**.

Reading the shapes

For a layer with n inputs and m neurons:

- W is $m \times n$ (one row per neuron, one column per input);
- b has m entries;
- the layer has $mn + m$ trainable parameters.

WORKED EXAMPLES: COUNTING PARAMETERS

NETWORK	LAYER 1	LAYER 2	TOTAL
2 → 3 → 1	$3 \times 2 + 3 = 9$	$1 \times 3 + 1 = 4$	13
100 → 50 → 10	$50 \times 100 + 50 = 5{,}050$	$10 \times 50 + 10 = 510$	5,560
784 → 128 → 10	$128 \times 784 + 128 = 100{,}480$	$10 \times 128 + 10 = 1{,}290$	101,770

Almost all the parameters sit in the widest connections. A 28×28 image flattened to 784 inputs already needs 100,352 weights for one modest hidden layer, which is part of why images use convolutions (Section 10).

Worked example: a 2–3–1 network, neuron by neuron

Two inputs, three hidden neurons with ReLU, and one output neuron with the identity activation (so it can output any number, as in regression). Each neuron has its **own** weights and bias.

THE FORWARD PASS FOR $X = (1, 2)$

NEURON	WEIGHTS	BIAS	WEIGHTED SUM z	OUTPUT
h_1	$(0.5,\ -0.2)$	0.1	$0.5(1) - 0.2(2) + 0.1 = 0.2$	$\mathrm{ReLU}(0.2) = 0.2$
h_2	$(-0.3,\ 0.4)$	0	$-0.3(1) + 0.4(2) + 0 = 0.5$	$\mathrm{ReLU}(0.5) = 0.5$
h_3	$(0.2,\ -0.5)$	0.2	$0.2(1) - 0.5(2) + 0.2 = -0.6$	$\mathrm{ReLU}(-0.6) = 0$
$\hat{y}$	$(1.0,\ -0.5,\ 0.8)$	0.3	$1.0(0.2) - 0.5(0.5) + 0.8(0) + 0.3 = 0.25$	identity: 0.25

- Information flows one way: the inputs feed every hidden neuron, and the three hidden outputs feed the output neuron.
- h_3's weighted sum is negative, so ReLU outputs 0: for this input, h_3 contributes nothing.
- Parameters: $3 \times 2 + 3 = 9$ in the hidden layer, $1 \times 3 + 1 = 4$ at the output: **13** in all.

Two conventions: don't mix them

	LECTURE (MATHS)	CODE (NUMPY, PYTORCH)
Input	One example, a column vector x	A batch, matrix X with one example per **row**
Weight shape	W: outputs × inputs	W: inputs × outputs
Layer	$Wx + b$	$XW + b$

Both are correct; they are transposes of each other. State which one you use and check that every product has matching inner dimensions.

Common doubts

FIRST PASS **Why is the weight matrix outputs × inputs and not the other way round?**

Because each **row** holds one neuron's weights, and a matrix–vector product takes the dot product of each row with x. With m neurons and n inputs, that's m rows of length n. Code that stores one example per row uses the transpose, inputs × outputs, so that XW works.

FIRST PASS **Do all neurons in a layer have the same weights and biases?**

No. Each neuron has its own weights and bias, and that's the point: each one learns to detect something different. If they all started identical, they would get identical gradients and stay identical forever, so the layer would act like one neuron. That's why weights start **random**.

FIRST PASS

How do the weights and the bias change a neuron's output?

Each weight scales how much one input matters, and its sign says whether that input pushes the output up or down. The bias shifts the weighted sum up or down regardless of the inputs, which moves the point where ReLU switches on. In the example, raising h_3's bias from 0.2 to 1.0 would make its sum $+0.2$, so it would switch on.

FIRST PASS

How do I multiply a 1×2 by a 2×1, and a 2×1 by a 1×2?

Write the shapes side by side. The **inner** numbers must match, and the result has the **outer** numbers.

PRODUCT	SHAPES	RESULT	EXAMPLE
Row × column	(1×**2**)(**2**×1)	1×1: one number (a **dot product**)	$\begin{bmatrix}1 & 2\end{bmatrix}\begin{bmatrix}3\\4\end{bmatrix} = 1\cdot 3 + 2\cdot 4 = 11$
Column × row	(2×**1**)(**1**×2)	2×2: a full table (an **outer product**)	$\begin{bmatrix}3\\4\end{bmatrix}\begin{bmatrix}1 & 2\end{bmatrix} = \begin{bmatrix}3 & 6\\4 & 8\end{bmatrix}$

Both appear in a network. The forward pass uses dot products: $z = w^\top x + b$. Backpropagation uses an outer product for a layer's weight gradient, $\partial L/\partial W = \delta\, a^\top$: every entry is (error at that output) × (input on that connection).

Thumb rule: "row times column gives a number; column times row gives a table."

Cheatsheet

Layer and parameter count

$y^{(k)} = f_k(W_k y^{(k-1)} + b_k)$. n inputs, m neurons: W is $m \times n$, $mn + m$ parameters.

Shape conventions

Lecture: $Wx + b$, W is outputs×inputs. Code: $XW + b$, rows are examples, W is inputs×outputs.

Online version with the interactive quiz: airesearchlabs.si/learn/dl/multi-layer-perceptrons/layers-matrices-and-shapes

Solving XOR with a Hidden Layer

An explicit 2–2–1 network that computes XOR, checked for all four inputs. The hidden layer changes the representation so that one output neuron can separate the classes.

The network

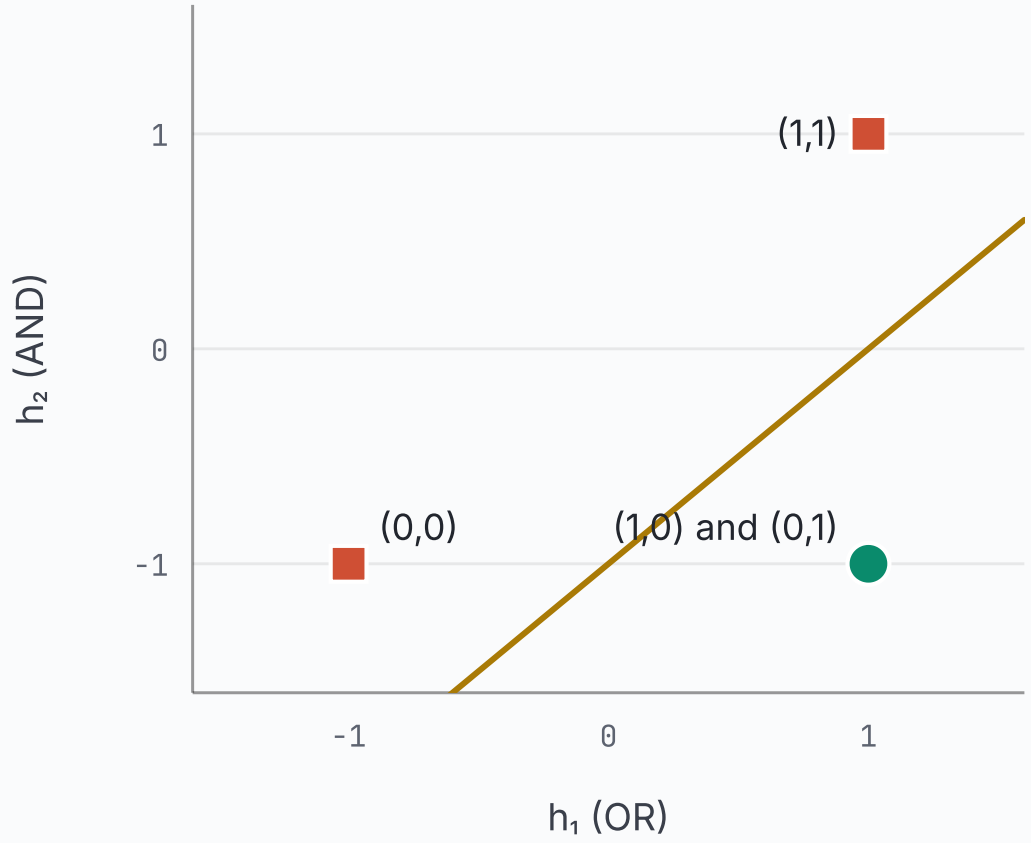
Two inputs, two hidden neurons, one output, all with the sign activation:

$$W_1 = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \quad b_1 = \begin{bmatrix} -0.5 \\ -1.5 \end{bmatrix}, \qquad W_2 = \begin{bmatrix} 1 & -1 \end{bmatrix}, \quad b_2 = -1$$

Hidden unit 1 computes $x_1 + x_2 - 0.5$: that's **OR**. Hidden unit 2 computes $x_1 + x_2 - 1.5$: that's **AND**. The output computes $h_1 - h_2 - 1$: "OR and not AND", which is XOR.

INPUT	HIDDEN PRE-ACTIVATIONS	HIDDEN OUTPUTS (H_1, H_2)	OUTPUT $H_1 - H_2 - 1$	XOR
(0, 0)	(−0.5, −1.5)	(−1, −1)	−1 + 1 − 1 = −1	−1 ✓
(1, 0)	(0.5, −0.5)	(+1, −1)	1 + 1 − 1 = +1	+1 ✓
(0, 1)	(0.5, −0.5)	(+1, −1)	1 + 1 − 1 = +1	+1 ✓
(1, 1)	(1.5, 0.5)	(+1, +1)	1 − 1 − 1 = −1	−1 ✓

What the hidden layer did



In the hidden layer's coordinates, the two positive inputs land on the same point and the classes are linearly separable, so one output neuron finishes the job.

The hidden layer maps the four inputs to new coordinates in which the classes **are** linearly separable. Both positive inputs even land on the same point.

A COMMON MISSTATEMENT

"The hidden layer works by mapping to a higher dimension." Not here: the hidden layer has 2 units, the same as the input. It's the **non-linear** mapping that makes the classes separable, not extra dimensions.

Common doubts

FIRST PASS **Would gradient descent find these exact weights?**

Not with sign activations: their derivative is zero almost everywhere, so there is no gradient to follow (Section 4). With sigmoid units, training finds *some* solution, usually a different one, and not always: with only 2 hidden units, a fair share of random starts get stuck. With 3 or more hidden units training succeeds far more reliably.

FIRST PASS **Where do the thresholds 0.5 and 1.5 come from?**

Look at the sum $x_1 + x_2$ for the four inputs: it's 0, 1, 1 or 2.

- **OR** should fire when the sum is 1 or more, so put its threshold **between 0 and 1**: 0.5.
- **AND** should fire only when the sum is 2, so put its threshold **between 1 and 2**: 1.5.

Any value inside those gaps works; the middle of the gap is the safest choice. The output works the same way: $h_1 - h_2$ is 0 for the negatives and 2 for the positives, so its threshold sits in the middle, at 1 (the bias -1).

FIRST PASS **Is there an easy way to remember this network?**

XOR = OR and not AND. The hidden layer computes OR and AND; the output says "yes" when OR is on and AND is off.

In the input square, OR and AND are two parallel lines. The two positive points, (1,0) and (0,1), lie in the strip between them. A single neuron can draw only one line, which is why it needs the hidden layer to draw the second.

Cheatsheet

XOR with a hidden layer

Hidden units compute OR ($x_1 + x_2 - 0.5$) and AND ($x_1 + x_2 - 1.5$); output $h_1 - h_2 - 1$. The non-linear hidden representation makes XOR separable.

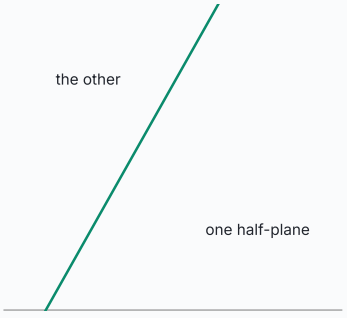
Online version with the interactive quiz: airesearchlabs.si/learn/dl/multi-layer-perceptrons/solving-xor-with-a-hidden-layer

Why Depth Helps: Composing Regions

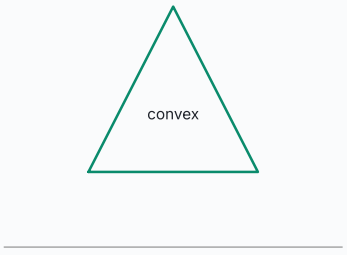
With threshold neurons, the first layer cuts the space into half-planes, the next combines them into convex regions, and the next into unions of regions. That is the geometric picture of what depth adds.

Layer by layer

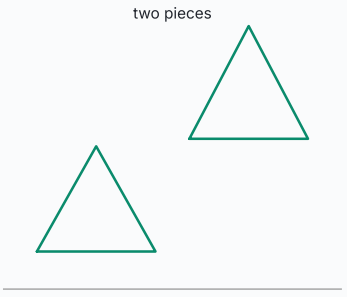
LAYER	EACH UNIT COMPUTES	REGION IT CAN DESCRIBE
1st hidden layer	A line (hyperplane) test	A **half-plane**
2nd layer, AND-like unit	"All of these half-plane tests are true"	An **intersection** of half-planes: a **convex** region (triangle, polygon)
3rd layer, OR-like unit	"Any of these convex regions"	A **union** of convex regions: non-convex or even disconnected shapes



Layer 1: each unit draws one line.



Layer 2: AND of three lines gives a triangle.



Layer 3: OR of two triangles gives a disconnected region.

The general lesson: composition

Later layers build on what earlier layers detect. In images the same idea appears as edges → textures and parts → objects (Section 10). Depth lets a network express structured, complicated functions by combining simple pieces, often with far fewer units than a single wide layer would need.

DON'T OVERSTATE IT

This geometric picture uses idealised threshold units. Real networks use smooth activations, and depth also makes training harder (vanishing gradients, Section 4). Section 11 weighs depth against width.

Common doubts

GOING DEEPER **Can a single hidden layer not also describe a triangle?**

A single hidden layer of threshold units followed by one output unit can combine half-planes with a weighted vote, so with the right weights it can carve out a convex region. What needs another layer is **combining regions**, such as the union of two separate triangles. With smooth activations and enough units, one hidden layer can approximate any continuous function (Section 5), but it may need vastly more units than a deeper network.

Cheatsheet

What depth adds

Layer 1: half-planes. Layer 2 (AND): convex regions. Layer 3 (OR): unions, so non-convex or disconnected regions.

Online version with the interactive quiz: airesearchlabs.si/learn/dl/multi-layer-perceptrons/why-depth-helps

Why Non-linearity Is Essential

Stacking layers without non-linear activations gives nothing new: any stack of linear layers collapses to a single linear layer. The algebra, a numerical check, and the consequence for XOR.

The collapse

Suppose both layers use the identity activation $f(z) = z$:

$$W_2(W_1 x + b_1) + b_2 = \underbrace{(W_2 W_1)}_{W} x + \underbrace{(W_2 b_1 + b_2)}_{b}$$

That is a single affine layer $Wx + b$. Repeating the argument, **any number of linear layers collapses to one**. Its decision boundary is still a hyperplane, so it still can't solve XOR.

NUMERICAL CHECK

$W_1 = \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix}$, $b_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, $W_2 = \begin{bmatrix} 1 & -1 \end{bmatrix}$, $b_2 = 2$.

- Collapsed: $W_2 W_1 = \begin{bmatrix} 1 & 1 \end{bmatrix}$ and $W_2 b_1 + b_2 = 1 + 2 = 3$, so $y = x_1 + x_2 + 3$.
- Check with $x = (1, 1)$: layer 1 gives $(1 + 2 + 1,\ 0 + 1 + 0) = (4, 1)$; layer 2 gives $4 - 1 + 2 = 5$. The collapsed formula gives $1 + 1 + 3 = 5$. ✓

What non-linearity buys

Put a non-linear f between the layers and the composition no longer simplifies: each layer can bend and fold the space, which is how the hidden layer in the XOR network made the classes separable. Non-linear activations are what give depth its power.

EXAM PHRASING

"Without non-linear activations, stacked layers compose to a single affine map $(W_2 W_1)x + (W_2 b_1 + b_2)$, so a deep linear network is no more expressive than one linear layer and cannot solve XOR."

Common doubts

FIRST PASS **So is a linear activation ever useful?**

Yes, at the **output** of a regression network, where you want an unbounded real number (Section 6). What you shouldn't do is use only linear activations in the **hidden** layers, because then the hidden layers add nothing.

FIRST PASS **Does linear regression need a non-linear activation function?**

No. Linear regression is one neuron with the identity activation: $\hat{y} = w^\top x + b$. That is exactly what it should be, because it models a straight-line (linear) relationship. Non-linear activations matter when you stack **hidden** layers: without them, the layers collapse into one linear map, as shown above.

Cheatsheet

Linear layers collapse

$W_2(W_1 x + b_1) + b_2 = (W_2 W_1)x + (W_2 b_1 + b_2)$: one affine layer. Depth needs non-linear activations.

Online version with the interactive quiz: airesearchlabs.si/learn/dl/multi-layer-perceptrons/why-non-linearity-is-essential