

From Linear Models to Neural Networks

Linear & logistic regression, neurons, MLPs, universal approximation

Prof. Nipun Batra

ES 667 · Deep Learning · IIT Gandhinagar

Lecture 1 chose the loss; Lecture 2 expands the prediction rule

Keep from Lecture 1

Regression output $\rightarrow$ Gaussian model
 $\rightarrow$ MSE

Class probabilities $\rightarrow$ Bernoulli /
categorical model $\rightarrow$ cross-entropy

Change in Lecture 2

Start with one weighted sum, expose what it cannot represent, then let hidden layers learn nonlinear features.

$x \rightarrow f_{\theta}(x) \rightarrow$ output parameters $\rightarrow$ same Lecture 1 loss

We will not re-derive the losses; today is about making f_{θ} more expressive.

Outline

1. **Linear prediction** — weighted sums, batches, and output shapes
2. **One neuron** — add an activation to create a nonlinear feature
3. **The limitation** — see exactly why a linear model cannot solve XOR
4. **An MLP** — compose ReLU features and solve the XOR counterexample
5. **Capacity and depth** — approximate functions, then compose representations

One notation covers every prediction rule

Write $f_{\theta}(x)$ for the model's prediction at input x .

x : the input features

θ : all trainable weights and biases

f_{θ} : the rule that maps input to a score or prediction

Starting point — one weighted sum plus a bias:

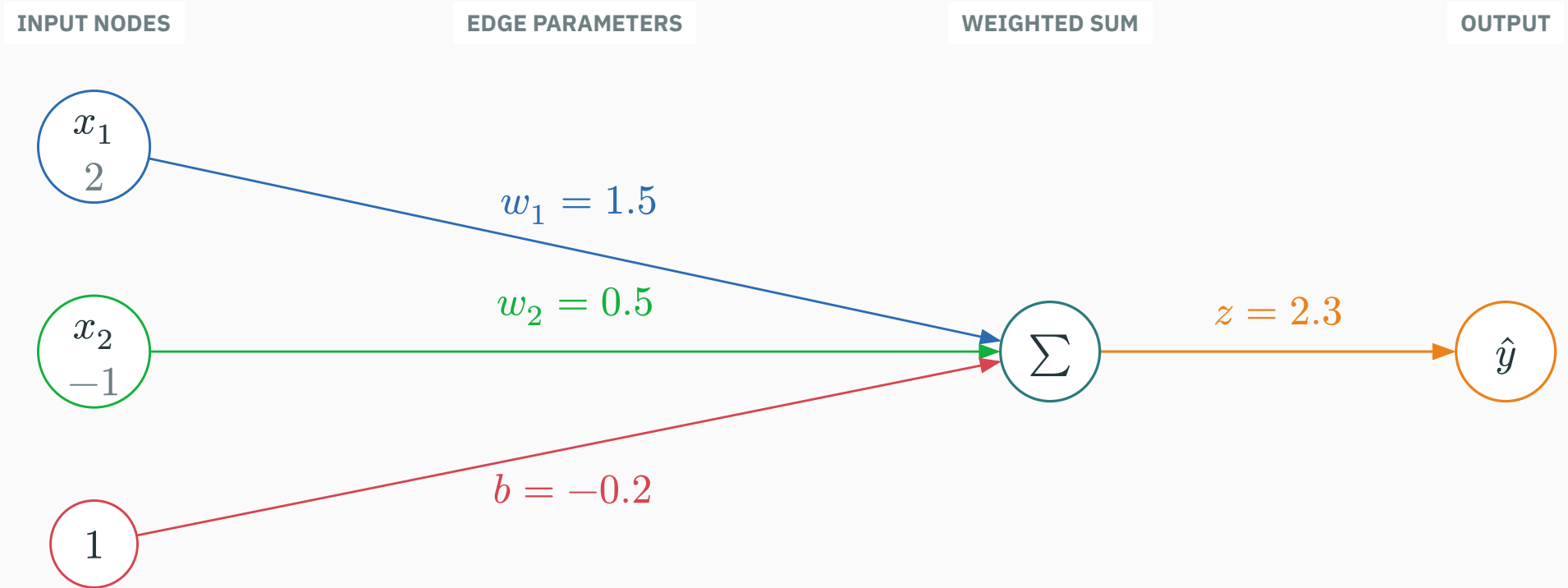
$$f_{\theta}(x) = w^{\top} x + b$$

We will build the network gradually; the full nested formula comes only after each part is concrete.

The linear starting point

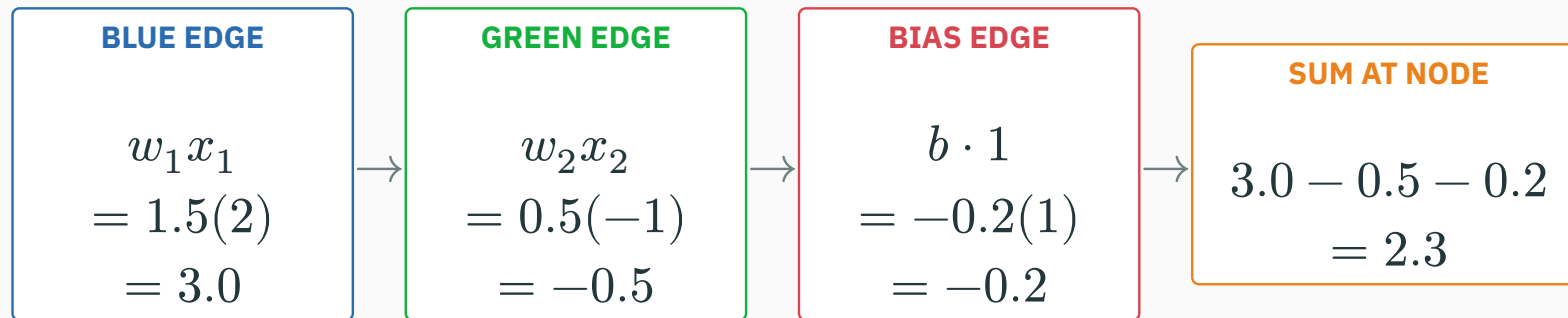
Start with one weighted sum

For one output, every input node sends a weighted value into one summation node.



The constant node 1 turns the bias into one more incoming edge: $b \cdot 1$.

Calculate every contribution entering the node



$$\hat{y} = \underbrace{1.5(2)}_{3.0} + \underbrace{0.5(-1)}_{-0.5} + \underbrace{(-0.2)(1)}_{-0.2} = 2.3.$$

Each edge contributes “weight × incoming value”; the node adds all contributions.

A batch is a labelled data matrix

example	feature x_1	feature x_2	target y
1	2	-1	2.4
2	0	3	1.0
3	-1	1	-1.1

DATA MATRIX X

$$X = \begin{pmatrix} 2 & -1 \\ 0 & 3 \\ -1 & 1 \end{pmatrix}$$

3 rows = examples
2 columns = features

TARGET VECTOR y

$$y = \begin{pmatrix} 2.4 \\ 1.0 \\ -1.1 \end{pmatrix}$$

one observed target
per example

Row i contains the input $x^{(i)}$ and its observed answer $y^{(i)}$.

One matrix multiplication predicts all three rows

Reuse $w = \begin{pmatrix} 1.5 \\ 0.5 \end{pmatrix}$ and $b = -0.2$ for every example.

$$\begin{aligned} Xw + b\mathbf{1} &= \begin{pmatrix} 2 & -1 \\ 0 & 3 \\ -1 & 1 \end{pmatrix} \begin{pmatrix} 1.5 \\ 0.5 \end{pmatrix} + (-0.2) \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \\ &= \begin{pmatrix} 2.5 \\ 1.5 \\ -1.0 \end{pmatrix} + \begin{pmatrix} -0.2 \\ -0.2 \\ -0.2 \end{pmatrix} = \begin{pmatrix} 2.3 \\ 1.3 \\ -1.2 \end{pmatrix} = \hat{y}. \end{aligned}$$

ROW 1

$$1.5(2) + 0.5(-1) - 0.2 = 2.3$$

ROW 2

$$1.5(0) + 0.5(3) - 0.2 = 1.3$$

ROW 3

$$1.5(-1) + 0.5(1) - 0.2 = -1.2$$

$\hat{y}$ contains predictions; y contains observed targets. They are not the same object.

AUGMENT THE DATA MATRIX

$$\tilde{X} = \begin{pmatrix} 2 & -1 & 1 \\ 0 & 3 & 1 \\ -1 & 1 & 1 \end{pmatrix}$$

 $\tilde{X}$

original feature columns
+ one constant column

STACK THE PARAMETERS

$$\theta = \begin{pmatrix} w_1 \\ w_2 \\ b \end{pmatrix} = \begin{pmatrix} 1.5 \\ 0.5 \\ -0.2 \end{pmatrix}$$

 θ

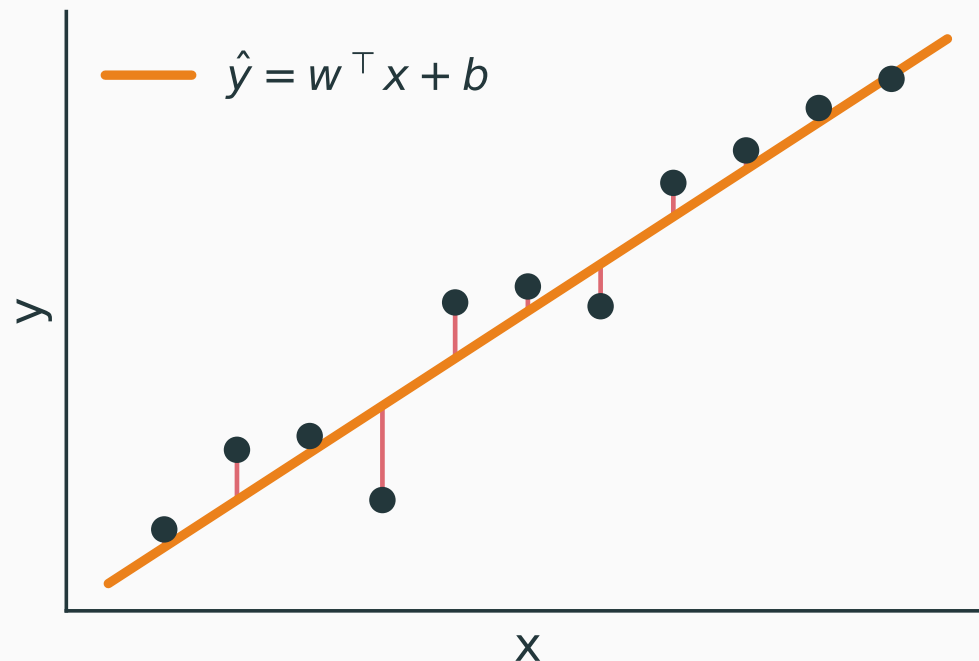
all weights
+ the bias

$$\hat{y} = \tilde{X}\theta, \quad (n \times (d + 1))((d + 1) \times 1) \rightarrow (n \times 1).$$

The column of ones is the batch version of the constant bias node on the previous network diagram.

For regression, one weighted sum gives one flat trend

$$\hat{y} = w^{\top} x + b$$

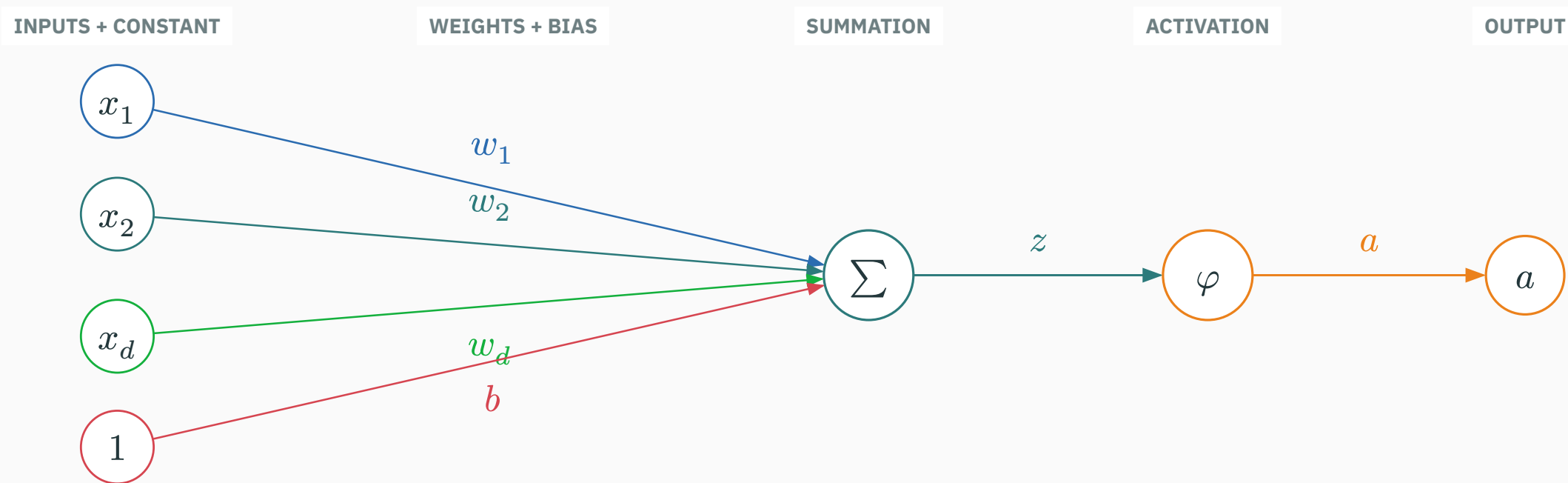


The model can tilt and shift a line or plane, but it cannot bend. Fit it in the Linear GD Colab ↗

From a linear score to a neuron

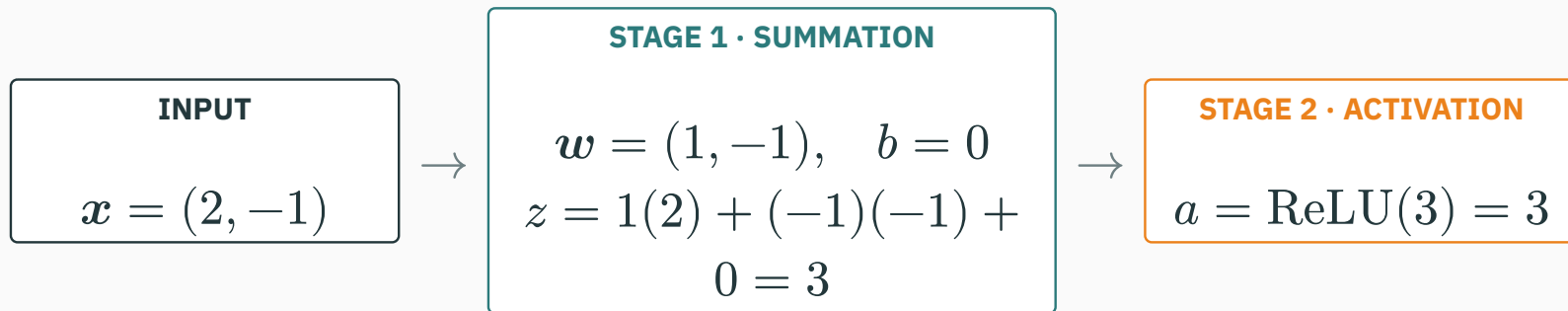
A neuron has two stages

First compute a weighted sum plus bias; then pass that value through an activation.



Weights and bias create z at the Σ node; the separate φ node reshapes z into a .

Trace one concrete neuron through the two stages



Changing w or b changes which input patterns make z large.

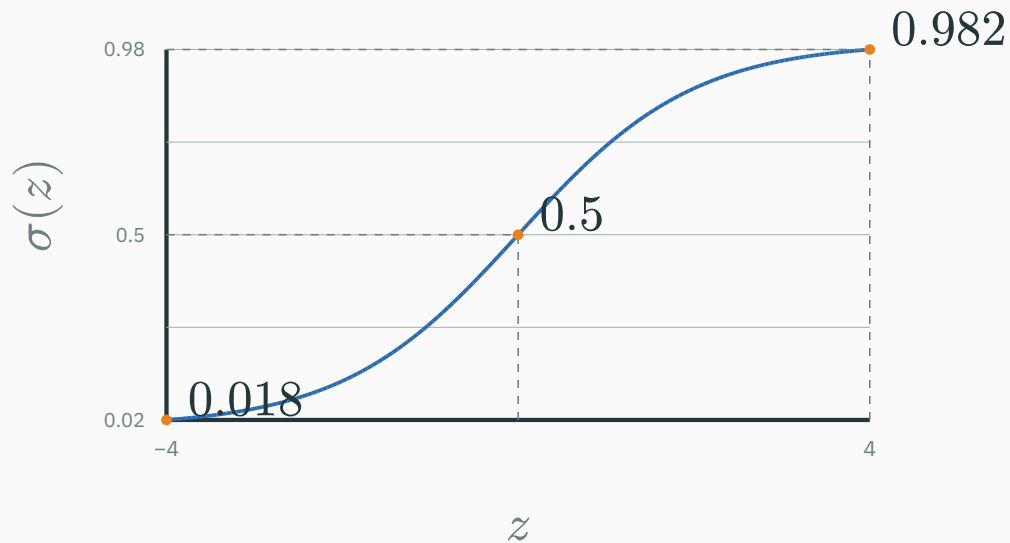
Changing φ changes how the same pre-activation z is gated or reshaped.

Topology tells us where values flow; this calculation tells us the number carried along each stage.

Bounded activations behave like smooth switches

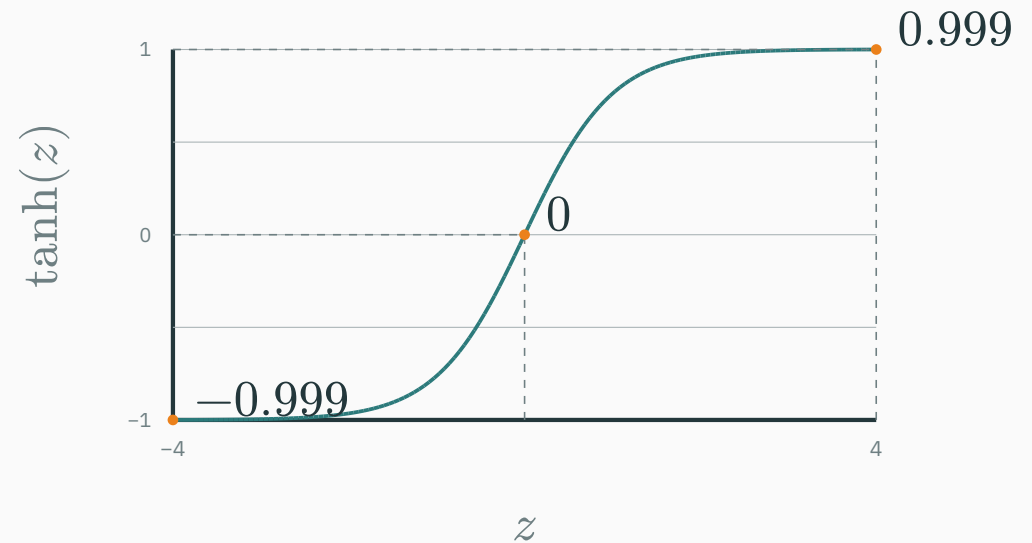
$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$

sigmoid: off $\rightarrow$ on



$$\tanh(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)}$$

tanh: negative $\rightarrow$ positive

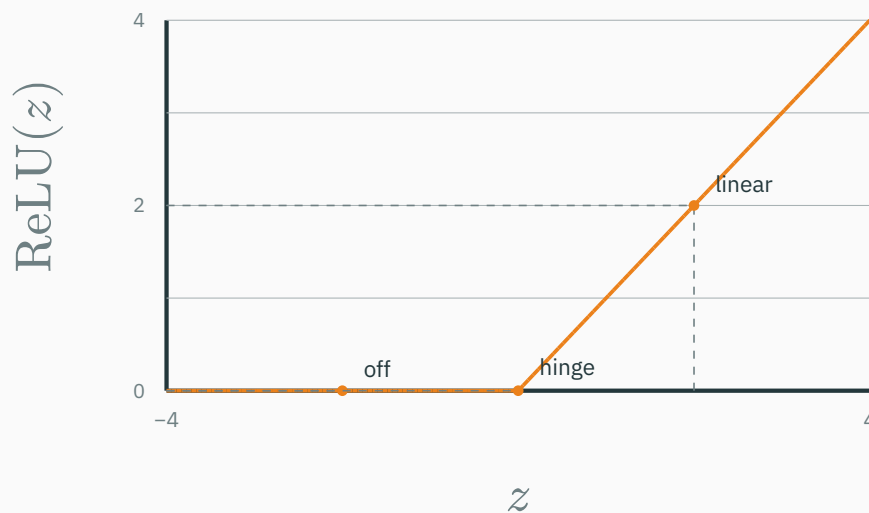


Sigmoid answers “how on?”; tanh answers “negative or positive, and how strongly?”

ReLU and GELU behave like ramps

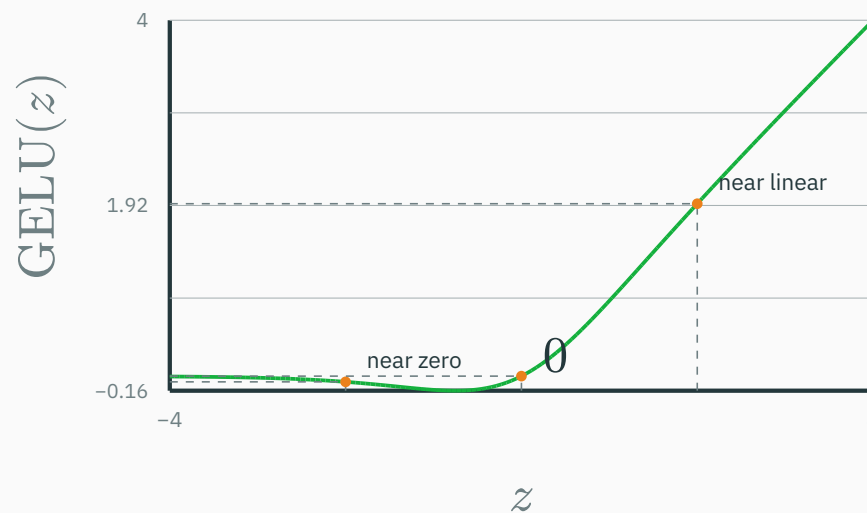
$$\text{ReLU}(z) = \max(0, z)$$

ReLU: hard gate + ramp



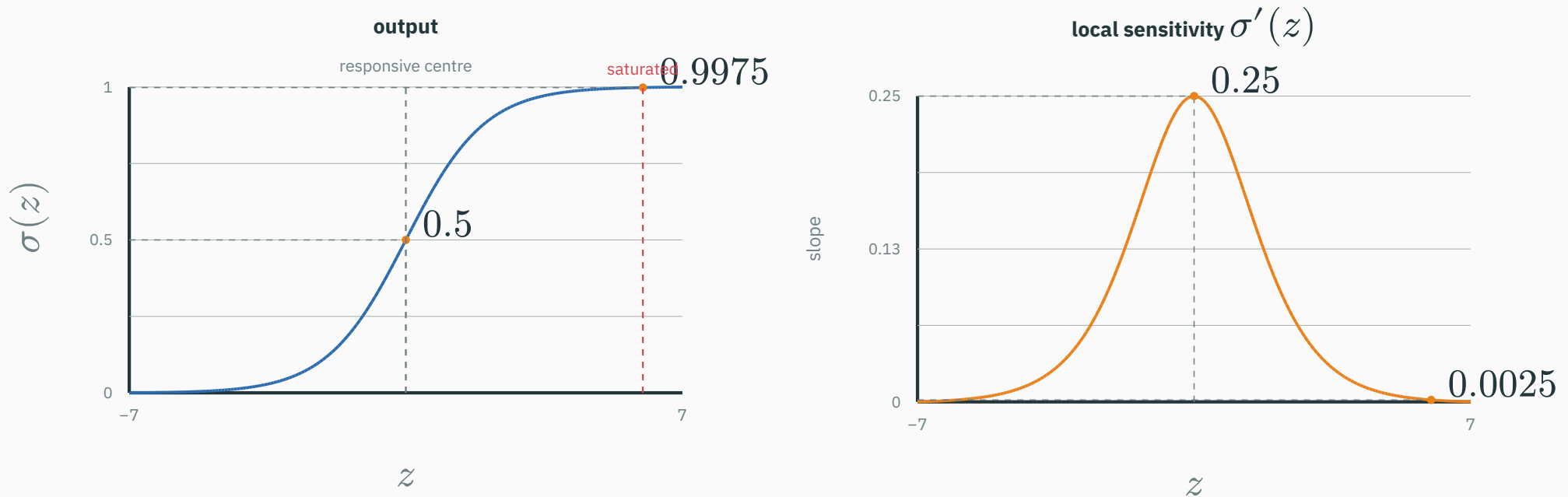
$$\text{GELU}(z) \approx z\sigma(1.702z)$$

GELU: smooth gate + ramp



Both build ramp-like hidden features; ReLU has a sharp hinge, while GELU eases through the gate.

Saturation means the curve has become locally flat



Near $z = 0$, the curve is steep; near $z = 6$, it is almost horizontal.

A saturated unit still outputs a number, but changing its input barely changes that output.

Calculate the effect of saturation

CENTRE: $z = 0$

$$\begin{aligned}\sigma'(0) &= 0.25 \\ \text{for } \Delta z &= 0.1: \\ \Delta h &\approx 0.25(0.1) = 0.025\end{aligned}$$

same
 Δz
 $\rightarrow$

SATURATED: $z = 6$

$$\begin{aligned}\sigma'(6) &\approx 0.0025 \\ \text{for } \Delta z &= 0.1: \\ \Delta h &\approx 0.0025(0.1) = \\ &0.00025\end{aligned}$$

$$\frac{0.025}{0.00025} = 100 \rightarrow \text{the saturated unit responds } \mathbf{100\times \text{ less}}.$$

For a positive ReLU input, $\text{ReLU}'(z) = 1$, so the same $\Delta z = 0.1$ gives $\Delta h \approx 0.1$.

The derivative is a local responsiveness meter; Lecture 3 will connect it to the learning signal.

Remove every activation. What can a stack of “weighted sum + bias” layers represent?

A. One equivalent weighted sum + bias

B. Any curved decision boundary

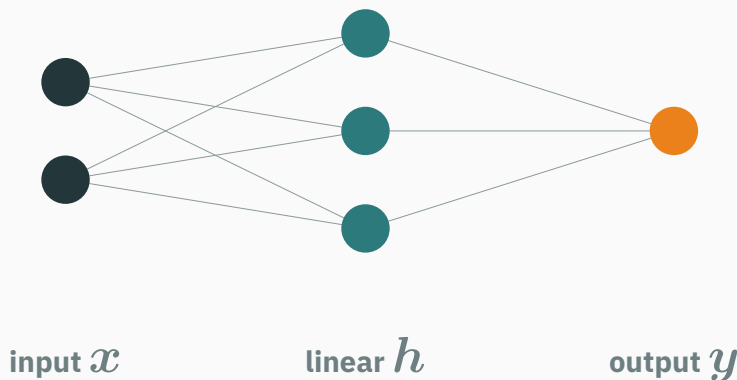
C. Only a constant function

D. A probability distribution automatically

Answer in nodes: two linear layers collapse into one

A ANSWER

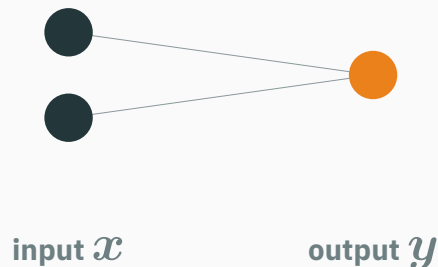
TWO LINEAR LAYERS



$$h = W_1 x + b_1$$

$$y = W_2 h + b_2$$

ONE EQUIVALENT LAYER



$$y = W_{\text{eq}} x + b_{\text{eq}}$$

no hidden representation is needed

$$W_{\text{eq}} = W_2 W_1, \quad b_{\text{eq}} = W_2 b_1 + b_2$$

Without a φ node, the middle column adds parameters but no new bend or feature shape.

INTERACTIVE

↗ nipunbatra.github.io/dl-teaching/interactives/activation-explorer.html

Compare activations and their **derivatives**, and see where gradients vanish.

Controls: activation type · input z · show derivative · compare saturation.

Explore large positive and negative logits, where sigmoid's derivative becomes tiny.

**Where a single weighted sum
fails**

A binary classifier draws one straight boundary

LINEAR SCORE

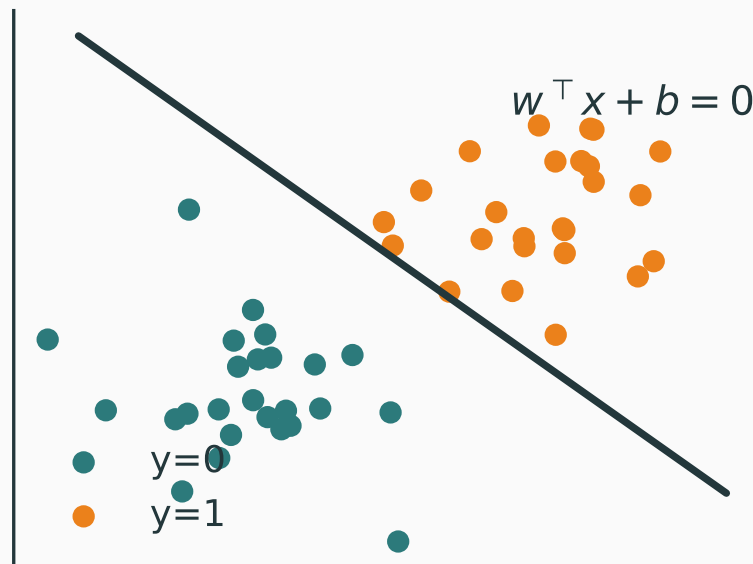
$$z = w^T x + b$$

THRESHOLD

$$\hat{y} = 1 \text{ if } z \geq 0$$

$$\hat{y} = 0 \text{ if } z < 0$$

The boundary is where $z = 0$.



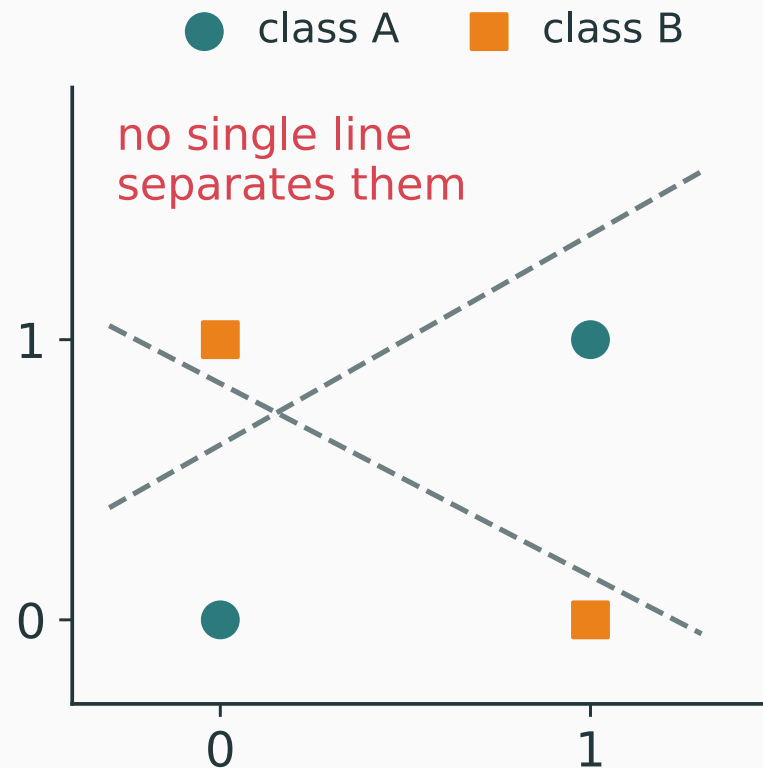
$w^T x + b = 0$ is one line in 2-D, so binary linear classification works only when one line can separate the classes.

XOR is the smallest pattern that defeats one line

CLASS A
(0, 0) and (1, 1)

CLASS B
(1, 0) and (0, 1)

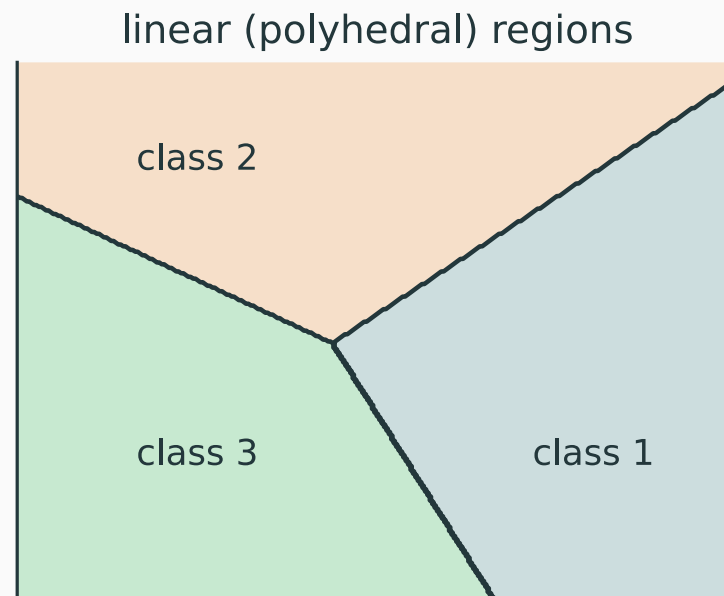
Every proposed line leaves one same-class corner on the wrong side.



XOR requires a nonlinear feature transformation before a final straight separator can work.

For each class k , compute one linear score $z_{k(\mathbf{x})} = \mathbf{w}_k^\top \mathbf{x} + b_k$. Predict the largest score: $\hat{y}(\mathbf{x}) = \arg \max_k z_{k(\mathbf{x})}$.

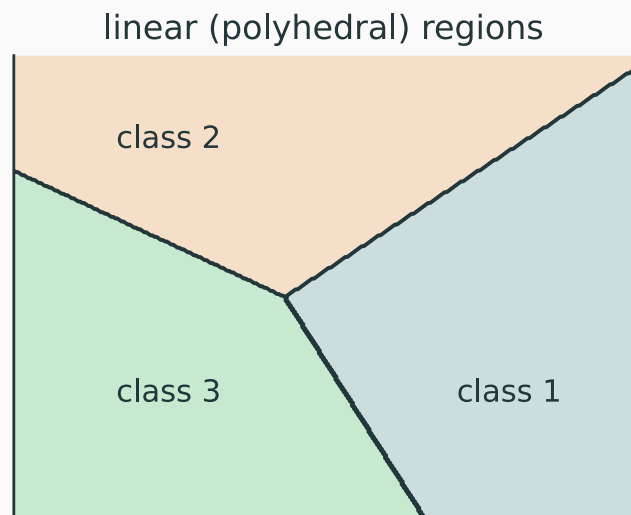
Each pair of classes can exchange the lead only where their two scores tie.



Multi-class regions can form polygons, but every edge still comes from equality between two linear scores.

Question: why are the multi-class boundaries straight?

QUESTION



three linear scores divide the input plane

The classifier predicts

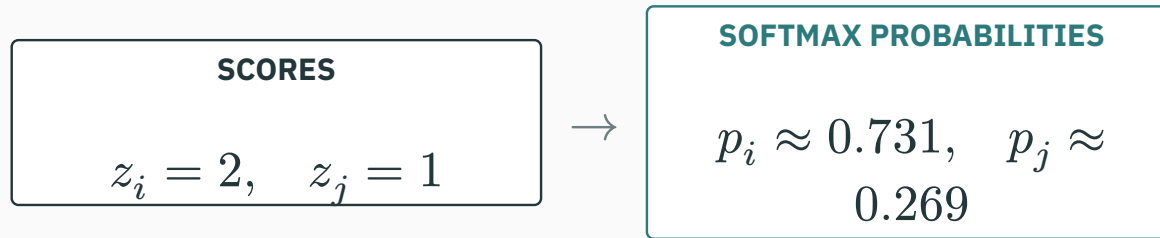
$$\hat{y}(\mathbf{x}) = \arg \max_k p_k(\mathbf{x}).$$

When is a point on the boundary between predicted classes i and j ?

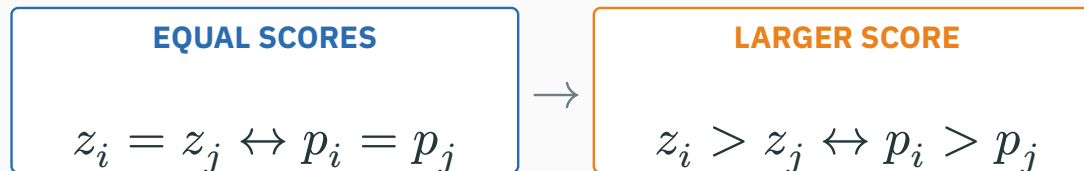
- A.** Whenever $p_i = p_j$
- B.** When $p_i = p_j$ and both are maximal
- C.** Whenever $p_i + p_j = 1$
- D.** Only when all K probabilities are equal

Softmax preserves the winner

A ANSWER



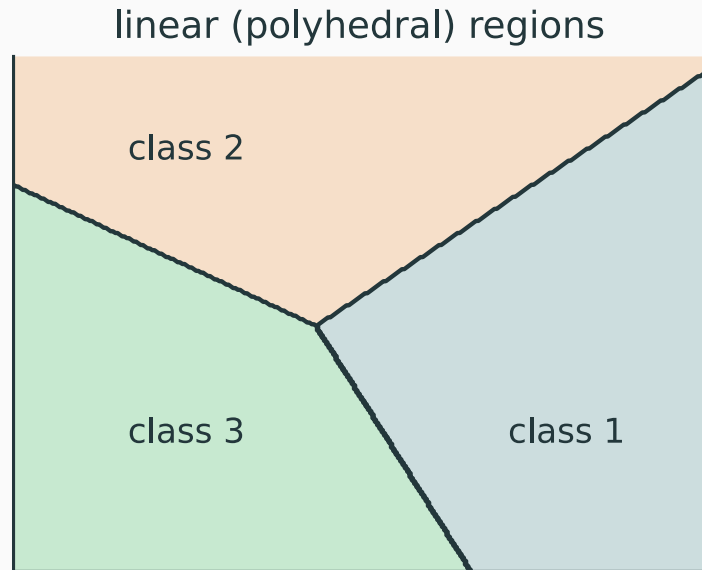
$$\frac{p_i}{p_j} = \frac{\exp(z_i)}{\exp(z_j)} = \exp(z_i - z_j)$$



Softmax changes score scale, not score order: $\arg \max_k p_k = \arg \max_k z_k$.

Linear scores can tie only on a flat set

A ANSWER



For class k , $z_{k(\mathbf{x})} = \mathbf{w}_k^\top \mathbf{x} + b_k$. Two classes tie when $z_{i(\mathbf{x})} = z_{j(\mathbf{x})}$, so $(\mathbf{w}_i - \mathbf{w}_j)^\top \mathbf{x} + (b_i - b_j) = 0$.

This is a line in 2-D, a plane in 3-D, and a hyperplane in d dimensions.

Each coloured region is where one linear score is largest.

Equal linear scores create a candidate straight boundary; softmax cannot bend it.

A concrete example: three linear scores compete

CLASS 1 SCORE

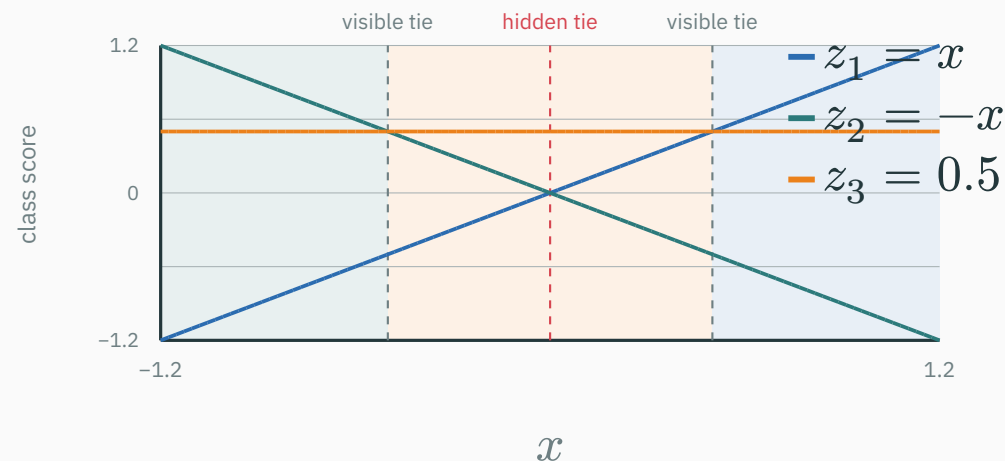
$$z_1(x) = x$$

CLASS 2 SCORE

$$z_2(x) = -x$$

CLASS 3 SCORE

$$z_3(x) = 0.5$$



PREDICTED CLASS = $\arg \max_k z_k(x)$ • boundaries at $x = \pm 0.5$

$$2: x < -.5$$

$$3: |x| < .5$$

$$1: x > .5$$

The learned boundaries are $x = -0.5$ and $x = 0.5$; the class-1/class-2 tie at $x = 0$ remains hidden.

Evaluate five explicit inputs to see which ties are visible

Prediction = $\arg \max_k z_{k(x)}$; a boundary appears only when the tied scores are maximal.

x	$z_1 = x$	$z_2 = -x$	$z_3 = 0.5$	what is visible?
-1	-1	1	0.5	class 2 wins
-0.5	-0.5	0.5	0.5	visible boundary 2/3
0	0	0	0.5	class 3 wins; 1/2 tie hidden
0.5	0.5	-0.5	0.5	visible boundary 1/3
1	1	-1	0.5	class 1 wins

The candidate **1/2 tie** occurs at $x = 0$, but the visible boundaries occur at $x = -0.5$ **and** $x = 0.5$.

Candidate boundary versus visible boundary

A ANSWER

CANDIDATE PAIRWISE TIE

$\mathcal{T}_{ij} = \{x : z_i = z_j\}$
an entire hyperplane

keep
only
winners
→

VISIBLE CLASS BOUNDARY

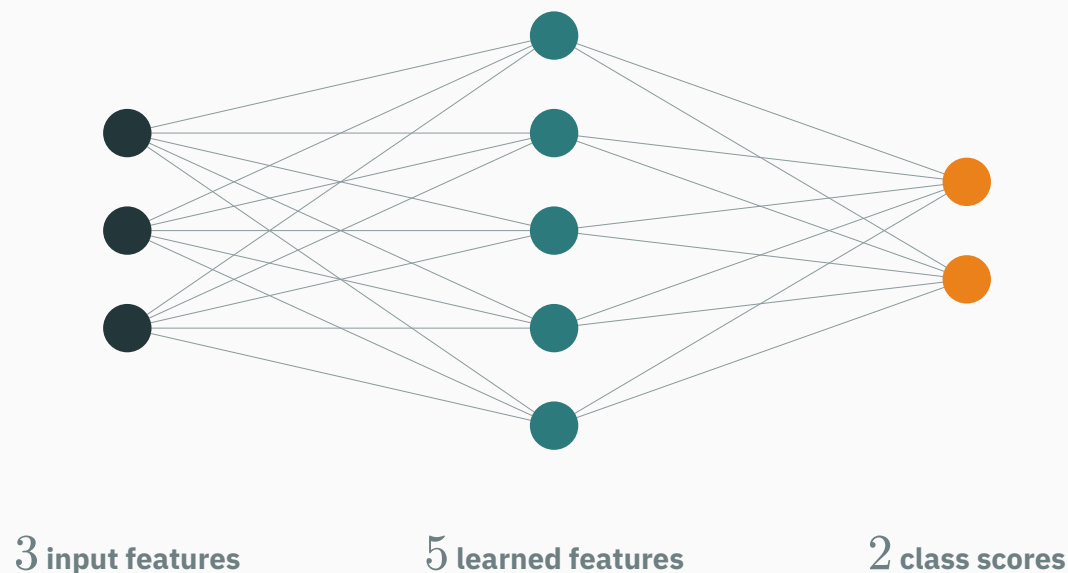
$\mathcal{B}_{ij} = \{x : z_i = z_j \geq z_l \text{ for every } l\}$
the top-scoring portion

In probability language: $p_i = p_j$ **and both are maximal.**

Correct: B. Multi-class linear classifiers have piecewise-straight visible boundaries.

The multilayer perceptron

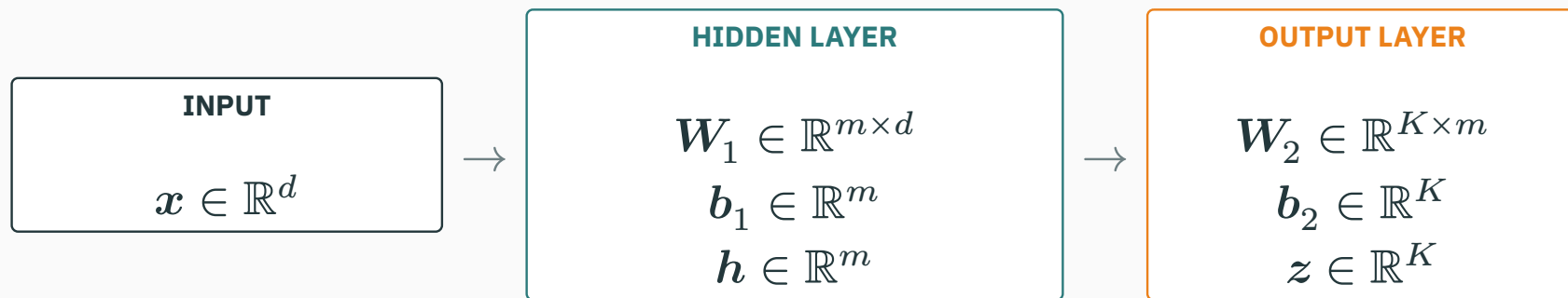
A hidden layer is a new column of learned features



$$h = \varphi(W_1 x + b_1), \quad z = W_2 h + b_2.$$

W_1 stores the $3 \times 5 = 15$ blue-to-teal edges; W_2 stores the $5 \times 2 = 10$ teal-to-orange edges.

General layer dimensions must agree



$$(m \times d)(d \times 1) \rightarrow (m \times 1), \quad (K \times m)(m \times 1) \rightarrow (K \times 1)$$

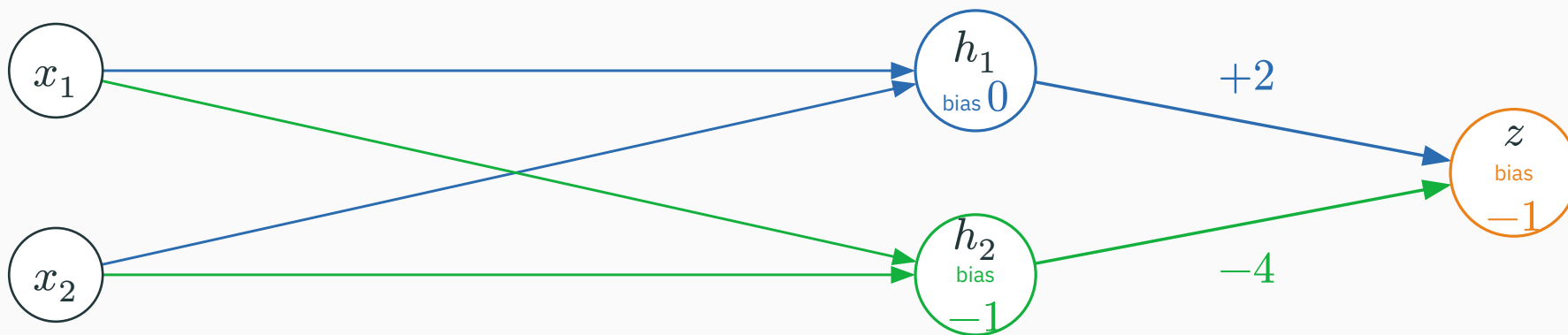
The inner dimensions cancel; trainable scalars $= m(d + 1) + K(m + 1)$.

One XOR network: keep these two features throughout

Both hidden units receive the same pair $x = (x_1, x_2)$:

$$h_1 = \text{ReLU}(x_1 + x_2), \quad h_2 = \text{ReLU}(x_1 + x_2 - 1)$$

all four input weights = $+1$



The output node computes $z = 2h_1 - 4h_2 - 1$; we keep this exact network throughout the XOR example.

Verify the XOR network on all four inputs

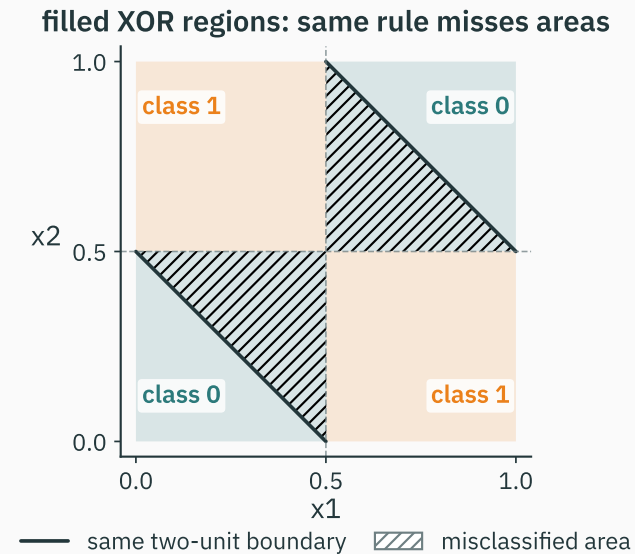
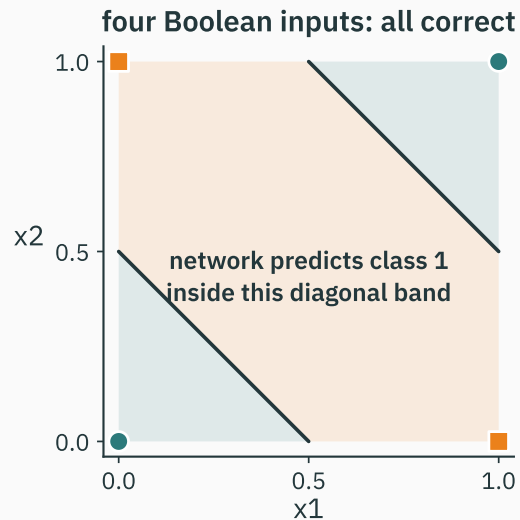
Input	$u_1 =$ $x_1 + x_2$	h_1	$u_2 =$ $x_1 +$ $x_2 - 1$	h_2	$z =$ $2h_1 -$ $4h_2 - 1$	$\hat{y}$
(0, 0)	0	0	-1	0	-1	0
(1, 0)	1	1	0	0	1	1
(0, 1)	1	1	0	0	1	1
(1, 1)	2	2	1	1	-1	0

$$\mathbf{W}_1 = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}, \quad \mathbf{b}_1 = \begin{pmatrix} 0 \\ -1 \end{pmatrix}, \quad \mathbf{w}_{\text{out}} = \begin{pmatrix} 2 \\ -4 \end{pmatrix}, \quad b_{\text{out}} = -1.$$

Predict 1 when $z \geq 0$. One ordinary output node correctly classifies all four XOR inputs.

Four-point XOR and filled XOR are different tasks

The same hand-chosen logit is positive only in the band $0.5 \leq x_1 + x_2 \leq 1.5$.



Exact on the **four Boolean inputs** is not exact on filled regions. Audit both constructions in the [XOR Colab ↗](#)

INTERACTIVE

↗ nipunbatra.github.io/interactive-articles/mlp-decision-boundary

Open the **ReLU classification lab**. Keep the model fixed while you switch from the four XOR corners to filled XOR.

1 · COMPARE THE DATA

four XOR corners →
filled XOR

2 · COMPARE THE RULES

two-ReLU corner
rule → four-ReLU
field rule

3 · INSPECT THE FUNCTION

logit map, signed
ReLU terms, and a
rotatable 3-D
surface

**Then reset and train: representation tells us what is possible;
optimization determines what is found.**

First derive the feature functions for a general input

Let $\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$. The weights and bias define a **function** before we substitute any particular input.

BLUE HIDDEN UNIT

$$\mathbf{w}_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad b_1 = 0$$

$$\begin{aligned} u_1(\mathbf{x}) &= \mathbf{w}_1^\top \mathbf{x} + b_1 \\ &= 1x_1 + 1x_2 + 0 = x_1 + x_2 \end{aligned}$$

$$\begin{aligned} h_1(\mathbf{x}) \\ &= \text{ReLU}(x_1 + x_2) \end{aligned}$$

GREEN HIDDEN UNIT

$$\mathbf{w}_2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad b_2 = -1$$

$$\begin{aligned} u_2(\mathbf{x}) &= \mathbf{w}_2^\top \mathbf{x} + b_2 \\ &= 1x_1 + 1x_2 - 1 = x_1 + x_2 - 1 \end{aligned}$$

$$\begin{aligned} h_2(\mathbf{x}) \\ &= \text{ReLU}(x_1 + x_2 - 1) \end{aligned}$$

These are exactly the two XOR features already shown in the node diagram; now we evaluate them at one input.

Check both hidden calculations in matrix form

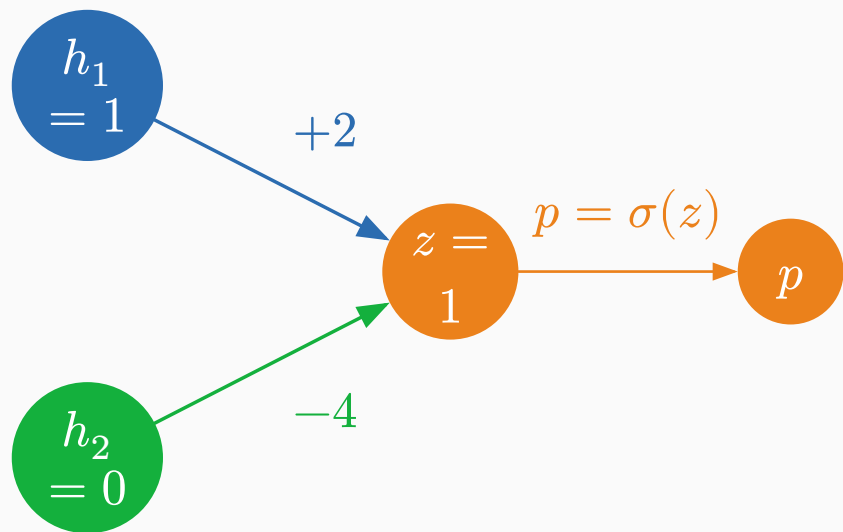
$$\begin{aligned} u &= W_1 x + b_1 \\ &= \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ -1 \end{pmatrix}. \end{aligned}$$

$$u_1 = 1(1) + 1(0) + 0 = 1$$

$$u_2 = 1(1) + 1(0) - 1 = 0$$

$$h = \text{ReLU}(u) = \begin{pmatrix} \max(0, 1) \\ \max(0, 0) \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

Rows of W_1 are neurons: each row computes one pre-activation, then ReLU acts elementwise.



Output logit

$$\begin{aligned} z &= (2 \ -4) \begin{pmatrix} 1 \\ 0 \end{pmatrix} - 1 \\ &= 2(1) - 4(0) - 1 = 1. \end{aligned}$$

Probability

$$p = \sigma(1) = \frac{1}{1 + \exp(-1)} \approx 0.731.$$

Binary cross-entropy for $y = 1$

$$\mathcal{L} = -\log(0.731) \approx 0.313.$$

This is the same output node shown in the XOR network: weights $+2, -4$ and bias -1 . Sigmoid only turns z into p .

Each ReLU is flat on one side of a line and rises on the other

$$h_1(x) = \text{ReLU}(x_1 + x_2)$$

active above $x_1 + x_2 = 0$



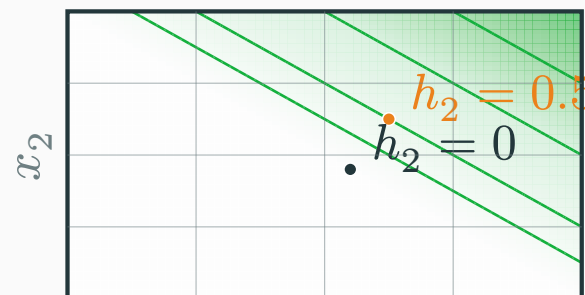
-0.5 0 0.5 1 1.5

x_1

$h_{j(x)} : 0$  3

$$h_2(x) = \text{ReLU}(x_1 + x_2 - 1)$$

active above $x_1 + x_2 = 1$



-0.5 0 0.5 1 1.5

x_1

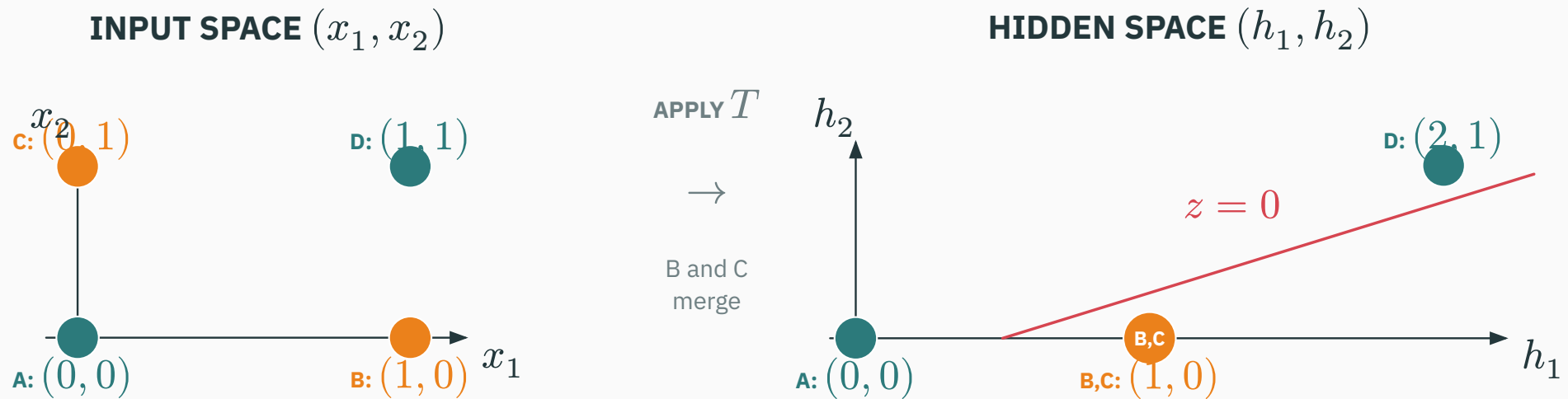
$h_{j(x)} : 0$  2

The grid is input space; each colour bar reports a hidden-unit activation value—not a probability.

Compute each XOR point's new hidden-space coordinates

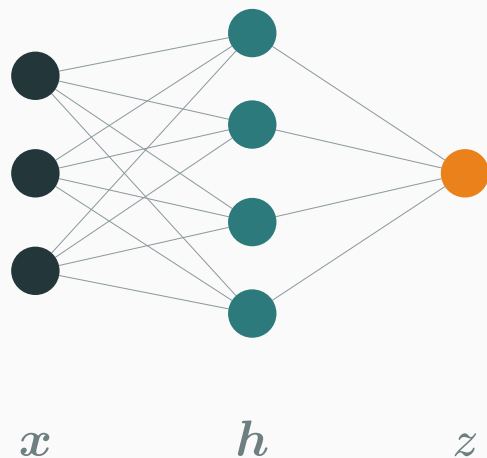
Use the **same map** for every point: $T(x_1, x_2) = (h_1, h_2) = (\text{ReLU}(x_1 + x_2), \text{ReLU}(x_1 + x_2 - 1))$.

A: $(0, 0) \rightarrow (0, 0)$ **B:** $(1, 0) \rightarrow (1, 0)$ **C:** $(0, 1) \rightarrow (1, 0)$ **D:** $(1, 1) \rightarrow (2, 1)$

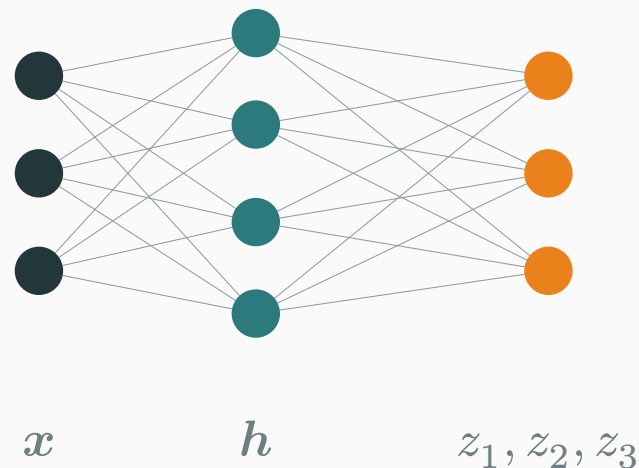


B and C both become $(1, 0)$. Since $z = 2h_1 - 4h_2 - 1$, the boundary $z = 0 \leftrightarrow h_1 - 2h_2 = 0.5$ separates them from A and D.

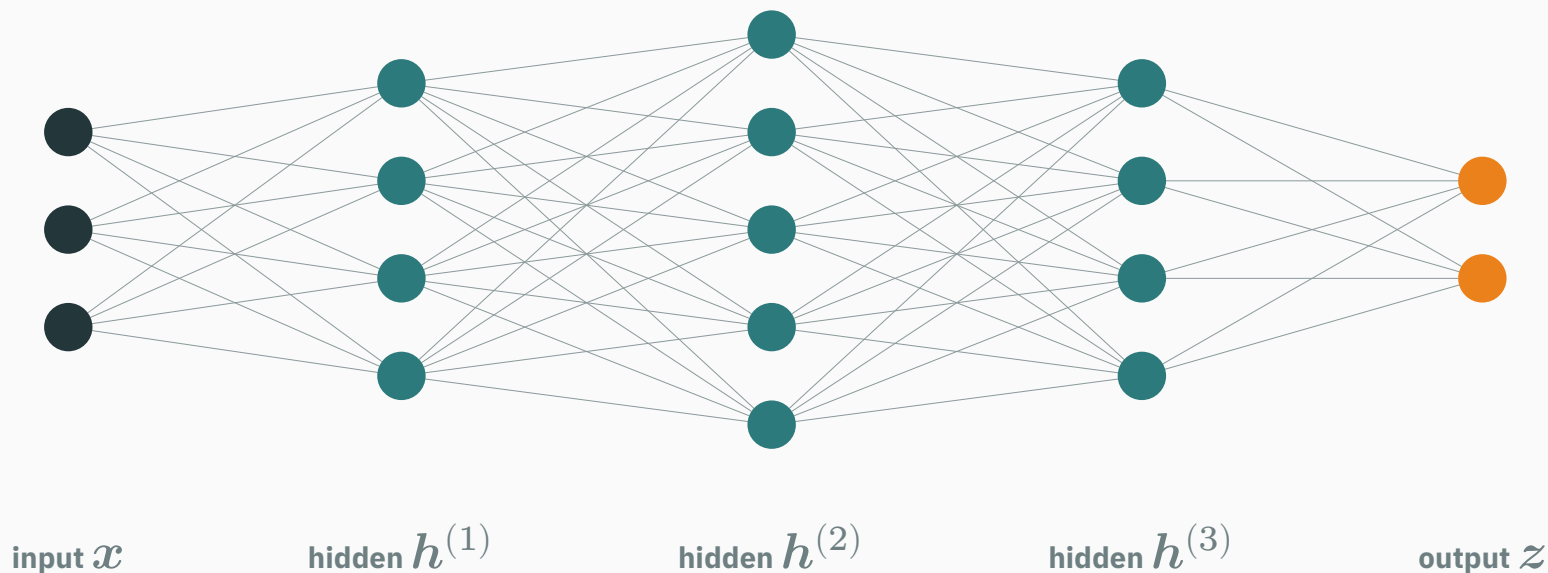
BINARY



THREE CLASSES



The hidden representation can stay the same; only the number and interpretation of output nodes changes.



Every node connects only to the next column; information reaches the output through successive learned transformations.

Depth is visible as the number of hidden columns, not the number of nodes within one column.

Universal approximation

Universal approximation asks one precise question

TARGET REGION TOLERANCE
continuous f $\rightarrow$ closed, bounded D $\rightarrow$ $\varepsilon > 0$

Can some finite network g make $|f(x) - g(x)| < \varepsilon$ for **every** $x \in D$?

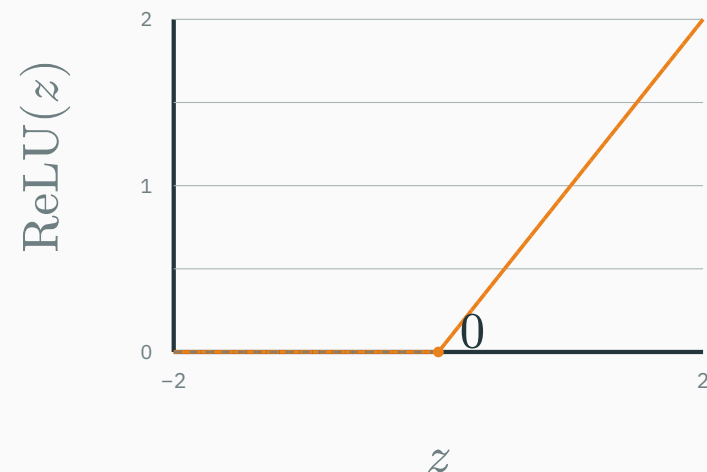
With a suitable nonlinear activation, the theorem says: **yes—such a finite one-hidden-layer network exists.**

This is an existence claim about representational capacity; it does not promise that training will find the network.

A ReLU contributes one hinge

$$\varphi(z) = \max(0, z)$$

- $z < 0$: output is zero;
- $z > 0$: output grows linearly;
- at $z = 0$: the slope changes—a **hinge**.



A hidden layer combines many shifted hinges into a piecewise-linear function.

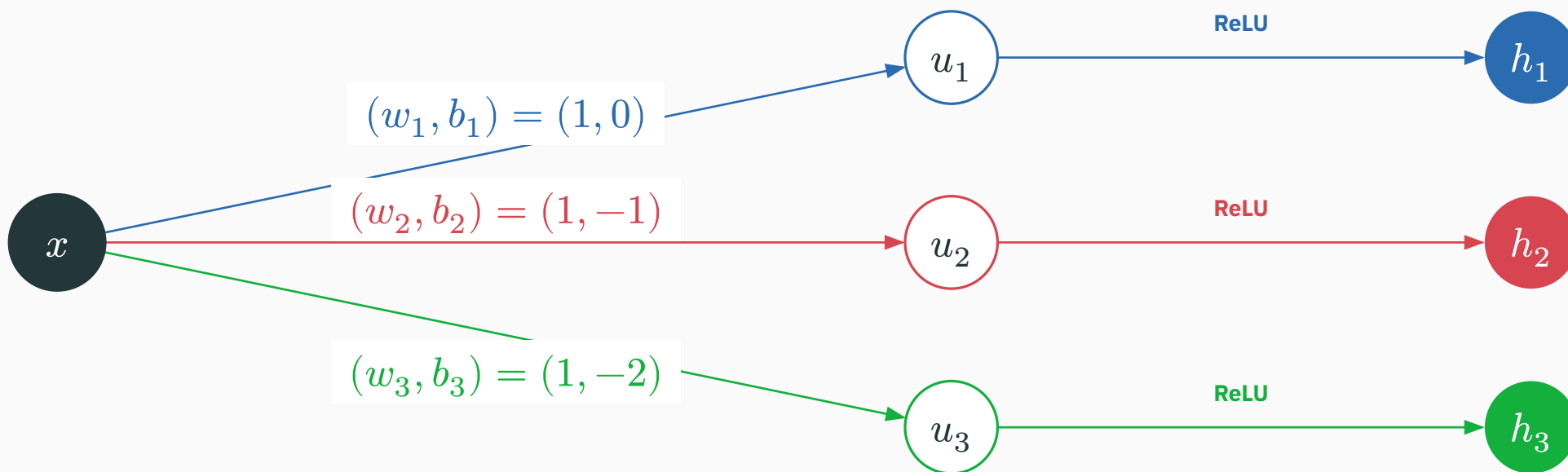
One input becomes three symbolic hinge features

Every unit first computes $u_i = w_i x + b_i$, then applies $h_i = \text{ReLU}(u_i)$.

ONE INPUT

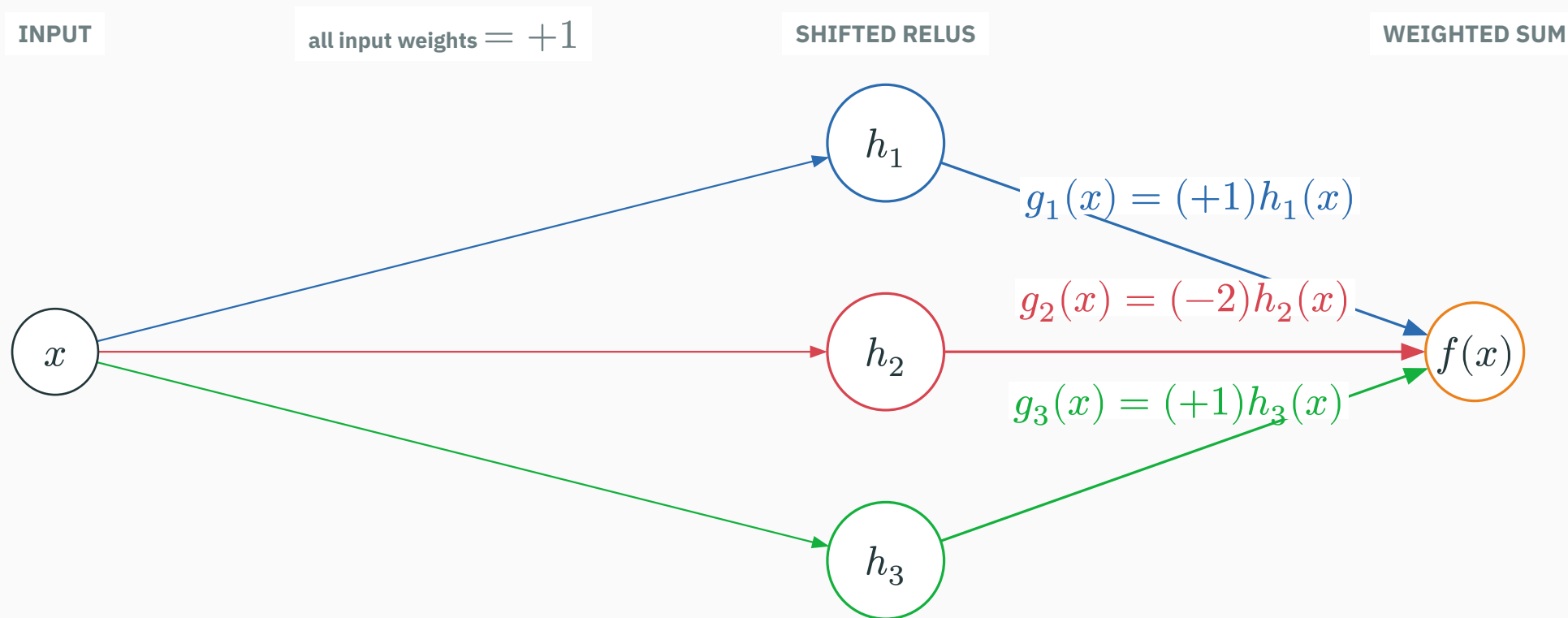
AFFINE SUMS

RELU FEATURES



The same x creates $h(x) = (\text{ReLU}(x), \text{ReLU}(x - 1), \text{ReLU}(x - 2))$; the hinges are at 0, 1, 2.

Output weights combine the three symbolic hinges

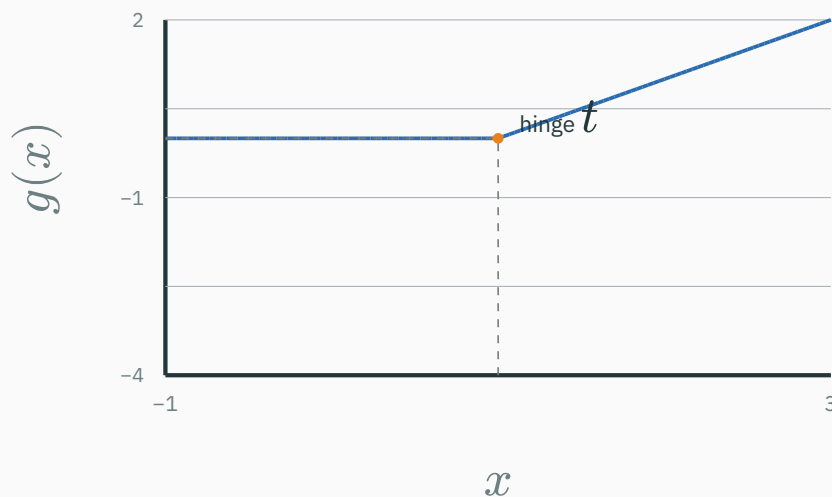


The output adds $f(x) = g_1(x) + g_2(x) + g_3(x)$ —the theorem’s “sum of hidden units” is literal.

Output weights choose each hinge's slope change

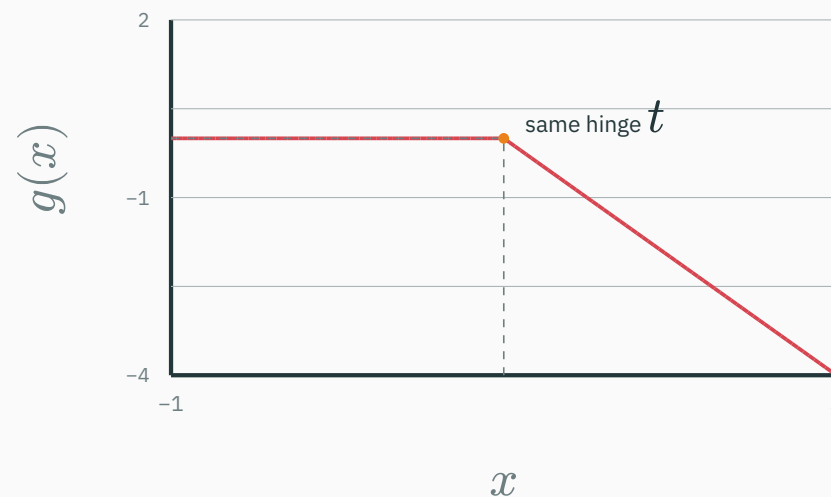
Start with $h(x) = \text{ReLU}(x - t)$; its output contribution is $g(x) = vh(x)$.

$$v = +1 \rightarrow g(x) = +h(x)$$



slope change $\Delta s = +1$

$$v = -2 \rightarrow g(x) = -2h(x)$$



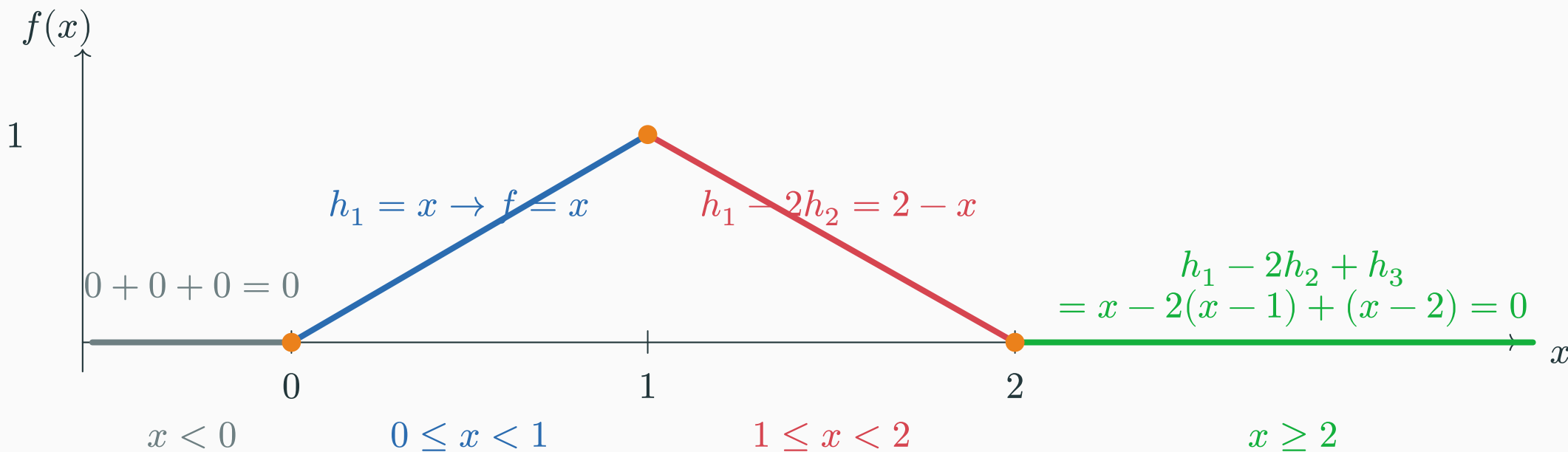
slope change $\Delta s = -2$

The hinge stays at t ; the sign of v chooses the bend, and $|v|$ sets its strength.

Three hinges assemble a tent interval by interval

D DERIVATION

$$f(x) = h_1 - 2h_2 + h_3 = \text{ReLU}(x) - 2\text{ReLU}(x - 1) + \text{ReLU}(x - 2).$$



Region 4: h_3 switches on and cancels the remaining slope, so $f(x) = 0$.

Reading left to right: flat $\rightarrow$ rise $\rightarrow$ fall $\rightarrow$ flat—the weighted sum is a tent.

Learned ReLU units are the theorem's building blocks

For a one-dimensional input, a trained hidden unit has the form

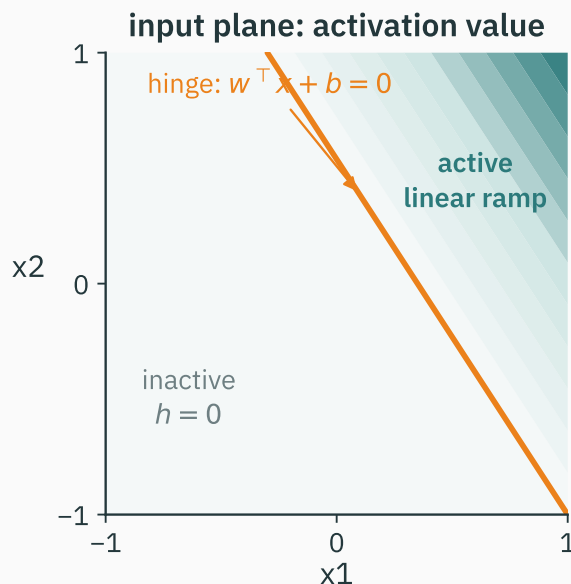
$$h_{j(x)} = \text{ReLU}(w_j x + b_j), \quad g(x) = c + \sum_j v_j h_{j(x)}.$$

j	hinge t_j	w_j	b_j	output v_j	signed contribution $g_{j(x)}$
1	0	1	0	+1	+ReLU(x)
2	1	1	-1	-2	-2ReLU($x - 1$)
3	2	1	-2	+1	+ReLU($x - 2$)

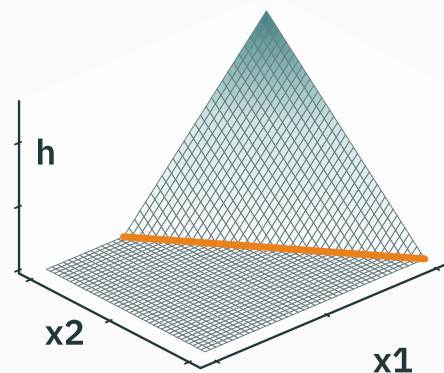
Training learns hinge locations and signed contributions; universal approximation says enough such units can make the remaining gap arbitrarily small.

One ReLU folds a 2-D input plane into a 3-D surface

$$h(x) = \text{ReLU}(w^T x + b) = \text{ReLU}(x_1 + 0.65x_2 - 0.35)$$



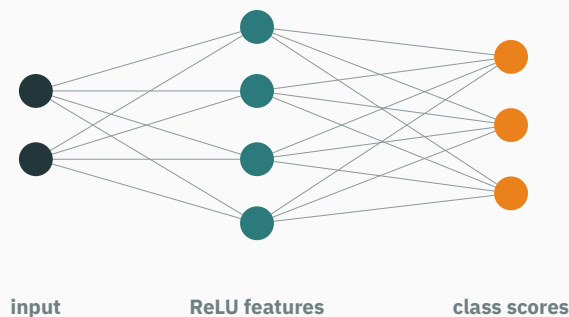
output surface: flat plane folded into a ramp



**The zero line is the hinge: one side stays flat, the other rises linearly.
More units add folds in different positions and directions.**

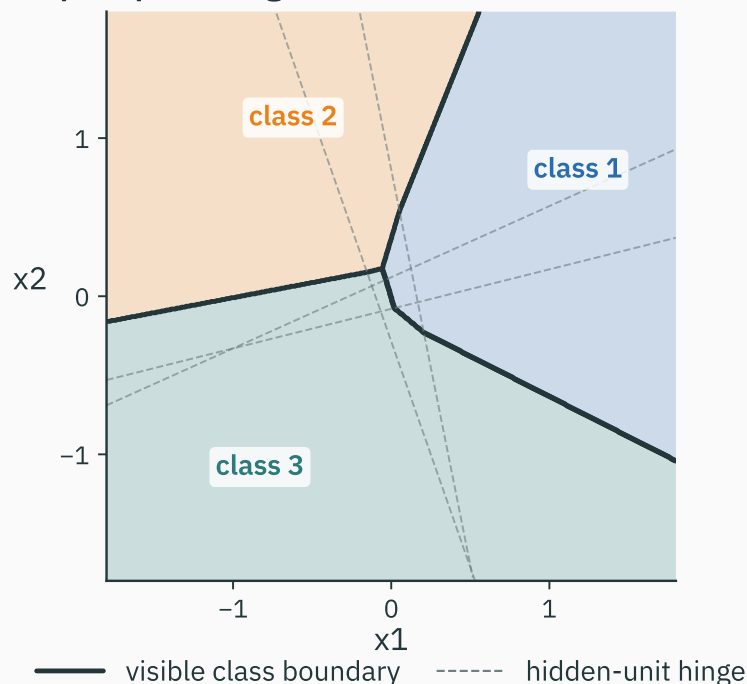
ReLU features create piecewise-linear class boundaries

FEATURES FEED CLASS SCORES



$$h_{j(x)} = \text{ReLU}(w_j^\top x + b_j)$$
$$z_k = c_k + \sum_j v_{kj} h_{j(x)}$$
$$\hat{y} = \arg \max_k z_k$$

input space: largest of three ReLU-based scores



Inside one activation pattern, pairwise score ties are straight; across ReLU hinges they join into piecewise-linear boundaries in input space.

INTERACTIVE

↗ nipunbatra.github.io/interactive-articles/relu-function-approximation

Open the **ReLU function lab**. Construction and training answer different questions, so the lab keeps them in separate lanes.

CONSTRUCT

choose a target and
drag the hinge
locations

TRAIN

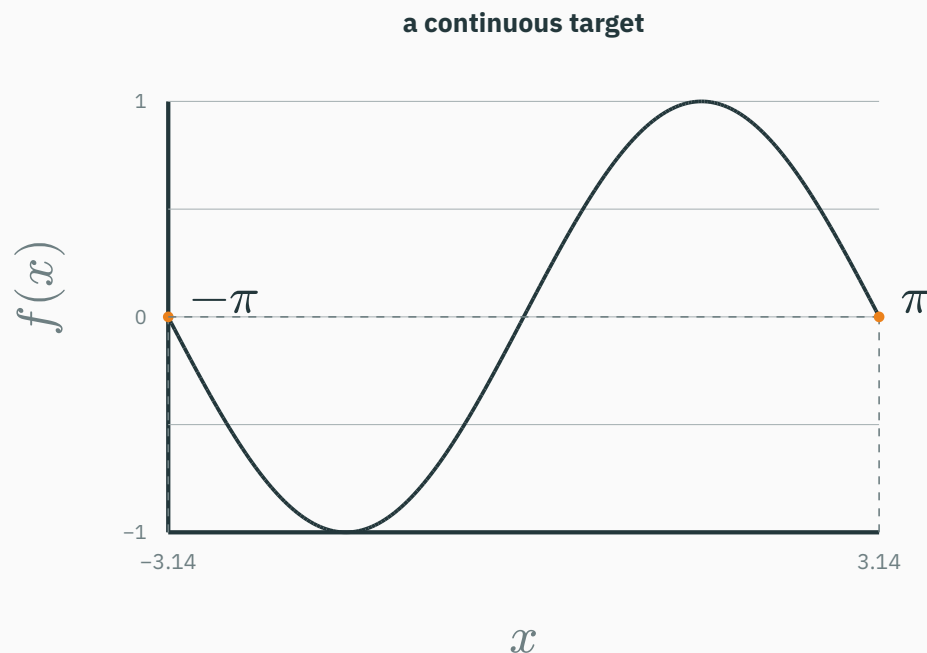
keep the width
fixed, reset the
seed, and optimize

INSPECT

learned function,
residual, grid gap,
and signed ReLU
terms

More hinges increase representational freedom; the residual shows what the chosen construction or training run still misses.

First fix the target and the input region



Continuous means that the target has no jumps.

In one dimension, a **compact domain** can be a closed, bounded interval such as

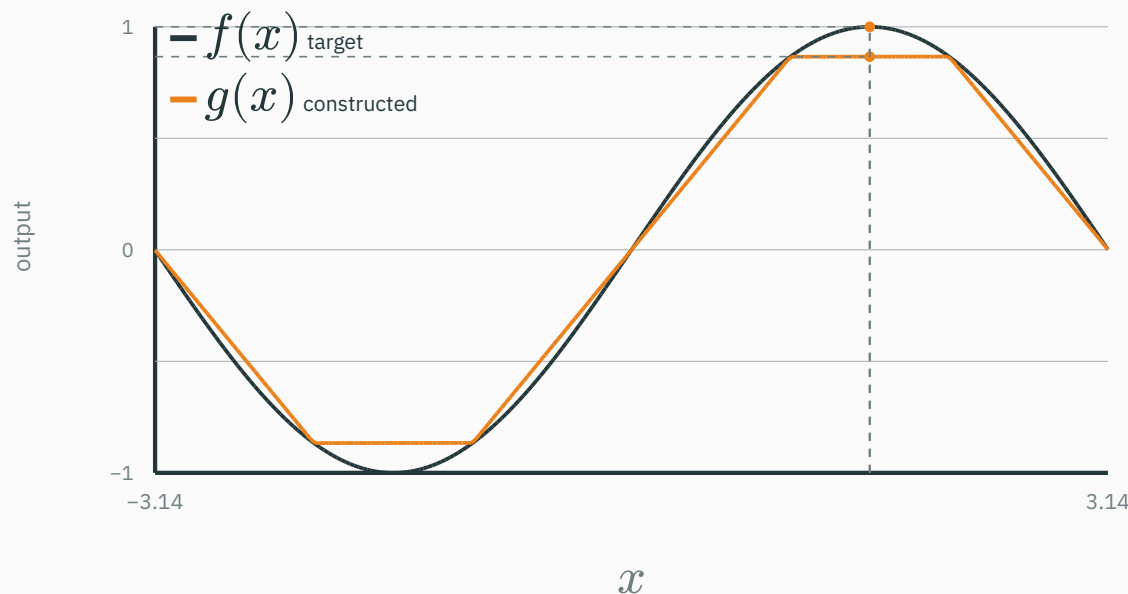
$$D = [-\pi, \pi].$$

We ask the network to match f **on** D ; the theorem makes no claim outside that chosen region.

Approximation begins with one target function on one specified input region.

$$e(x_0) = |f(x_0) - g(x_0)|$$

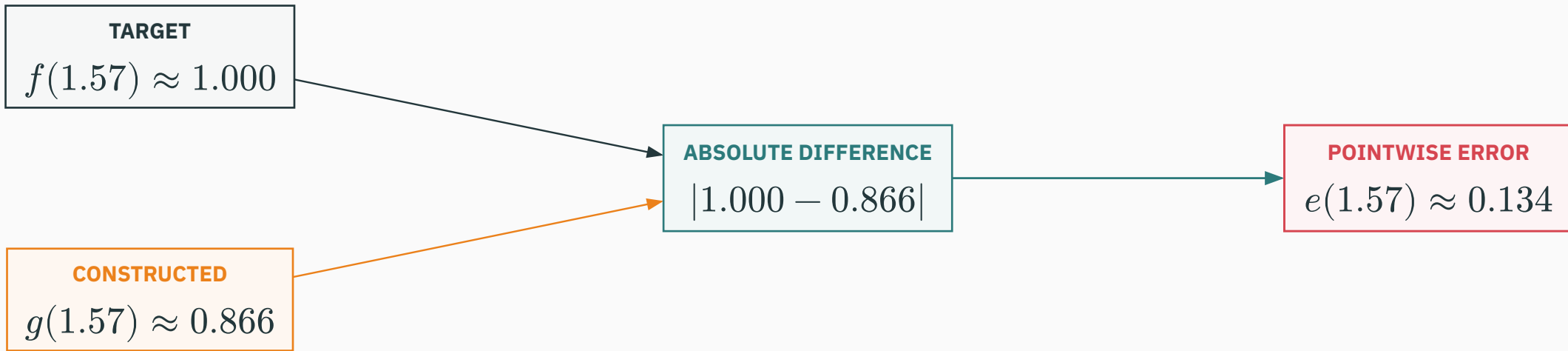
vertical slice at $x_0 = 1.57$



Pointwise error is the vertical distance between the target and its constructed approximation at the same input.

Calculate the gap at $x_0 = 1.57$

$$e(x_0) = |f(x_0) - g(x_0)|$$



One point is not enough: universal approximation asks for the largest gap over the whole region.

“sup” means the worst gap over the whole region

$$\sup_{x \in D} |f(x) - g(x)|$$

sup means **supremum**: the smallest number at least as large as every pointwise error.

Here the error is continuous and $D = [-\pi, \pi]$ is closed and bounded, so a highest error exists. In this plot, read **sup** as **maximum**.

sup $< \epsilon$: no input in D crosses the tolerance.



The entire orange error curve stays below $\epsilon = 0.05$, so the uniform guarantee is satisfied.

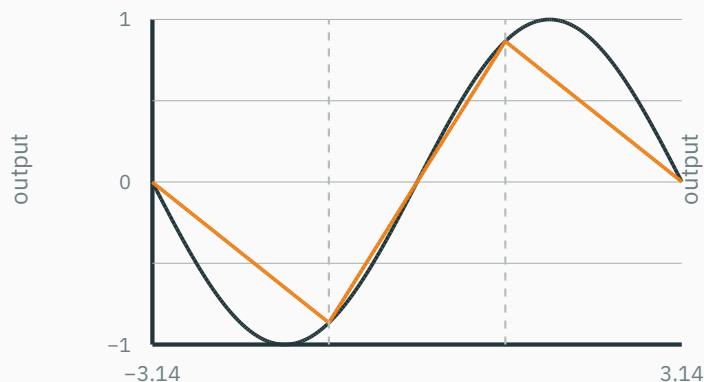
More hinges reduce the largest approximation error

CONSTRUCTED INTERPOLANTS · FIXED KNOTS · NOT TRAINED solid: target $\sin x$ dashed: $g_m(x)$

Each g_m matches $\sin x$ at fixed knots and is linear between adjacent knots.

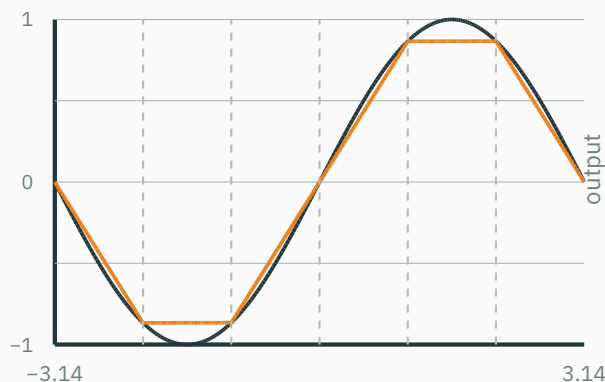
2 hinges

$$\varepsilon_m \approx 0.437$$



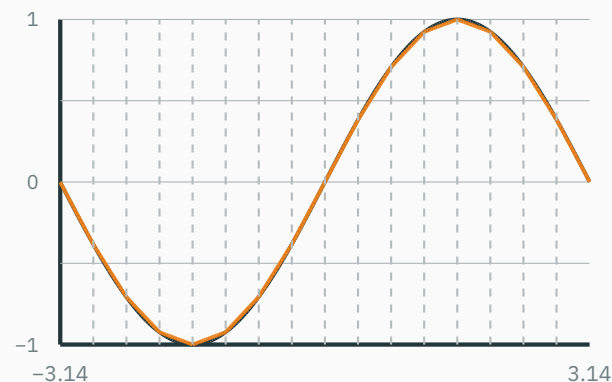
5 hinges

$$\varepsilon_m \approx 0.134$$



15 hinges

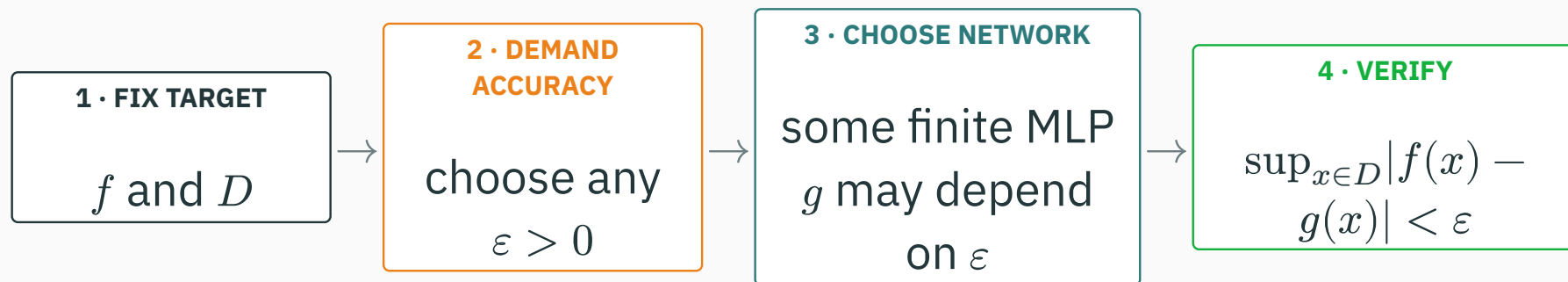
$$\varepsilon_m \approx 0.019$$



More hinges shorten each linear segment, so the worst gap falls from 0.437 to 0.019.

Read the universal-approximation claim in order

D DERIVATION



$\forall \varepsilon > 0, \exists$ finite MLP g such that $\sup_{x \in D} |f(x) - g(x)| < \varepsilon$.

For every ε : we may request any positive tolerance.

There exists g : the network may change when the tolerance changes.

This is a statement about representational capacity—not yet about how to find the network.

Existence does not guarantee learnability

Universal approximation does **not** promise:

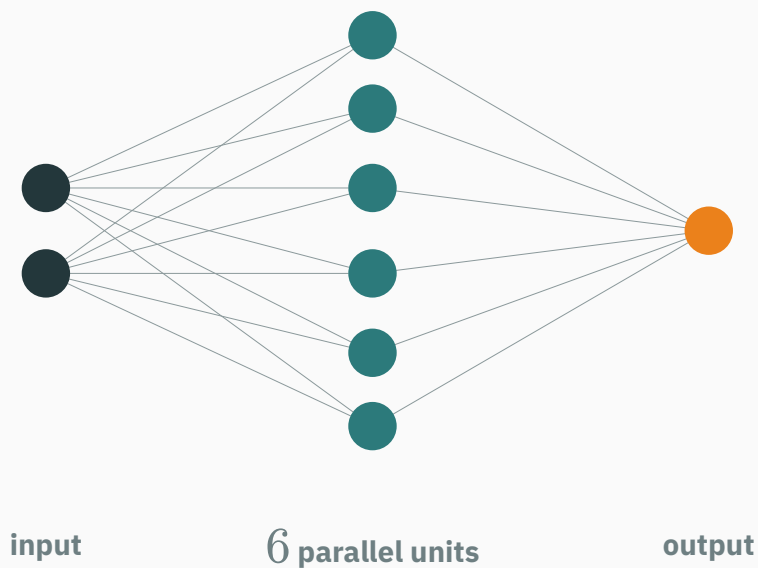
- that training will **find** that function;
- that **finite data** is enough;
- that a **small** network suffices;
- that it will **generalize**.

expressivity $\neq$ trainability $\neq$ generalization

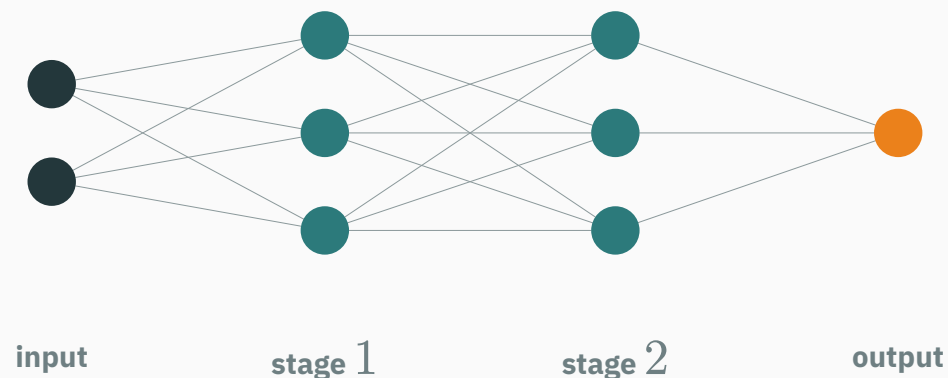
Depth versus width

Node view: width adds rows; depth adds columns

WIDER



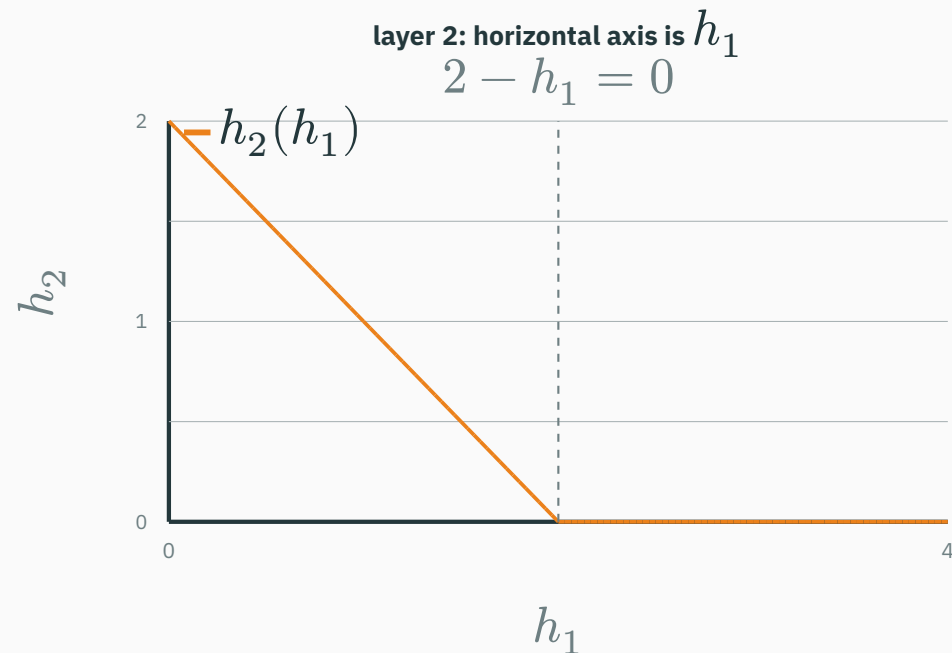
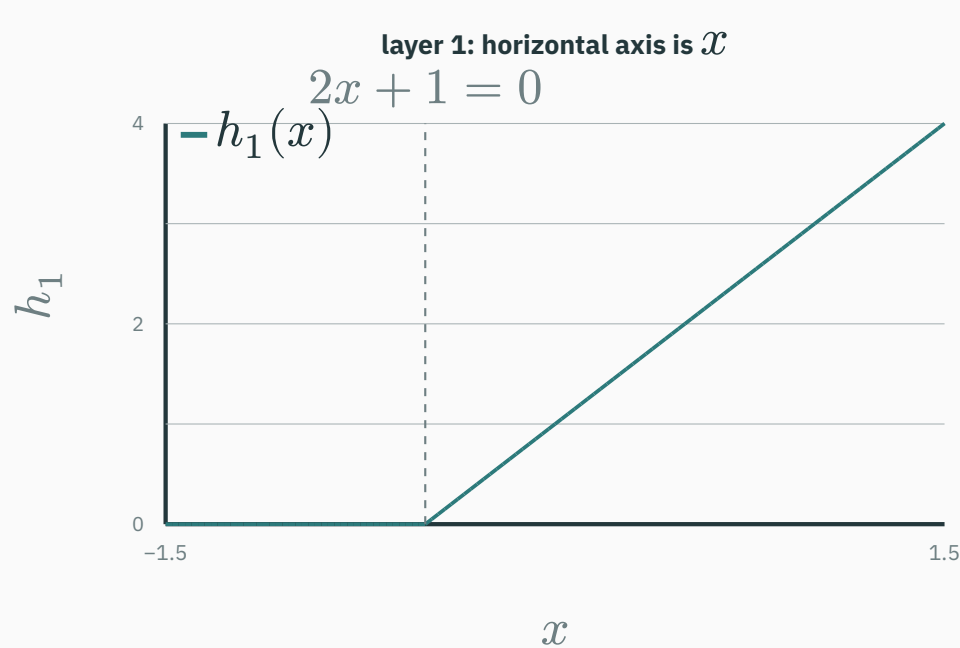
DEEPER



Count nodes vertically to read width; count hidden columns horizontally to read depth.

Depth means composition: layer 2 receives h_1 , not x

$$x \rightarrow h_1(x) = \text{ReLU}(2x + 1) \rightarrow h_2(h_1) = \text{ReLU}(2 - h_1)$$



FIRST BEND

$$2x + 1 = 0 \rightarrow x = -0.5$$

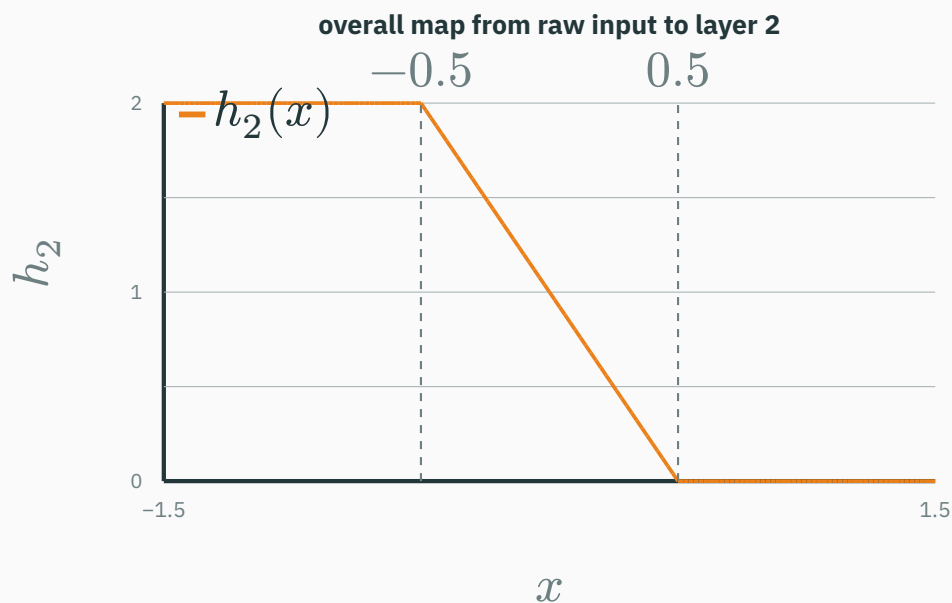
SECOND BEND IN INPUT SPACE

$$h_1 = 2 \rightarrow 2x + 1 = 2 \rightarrow x = 0.5$$

Composing the two maps gives three input regions

$$h_2(x) = f_2(f_1(x)) = \text{ReLU}(2 - \text{ReLU}(2x + 1))$$

This illustrates sequential composition; it does not claim that this simple function requires depth.



LEFT PLATEAU

$$x \leq -0.5$$
$$h_1 = 0 \rightarrow h_2 = 2$$

MIDDLE RAMP

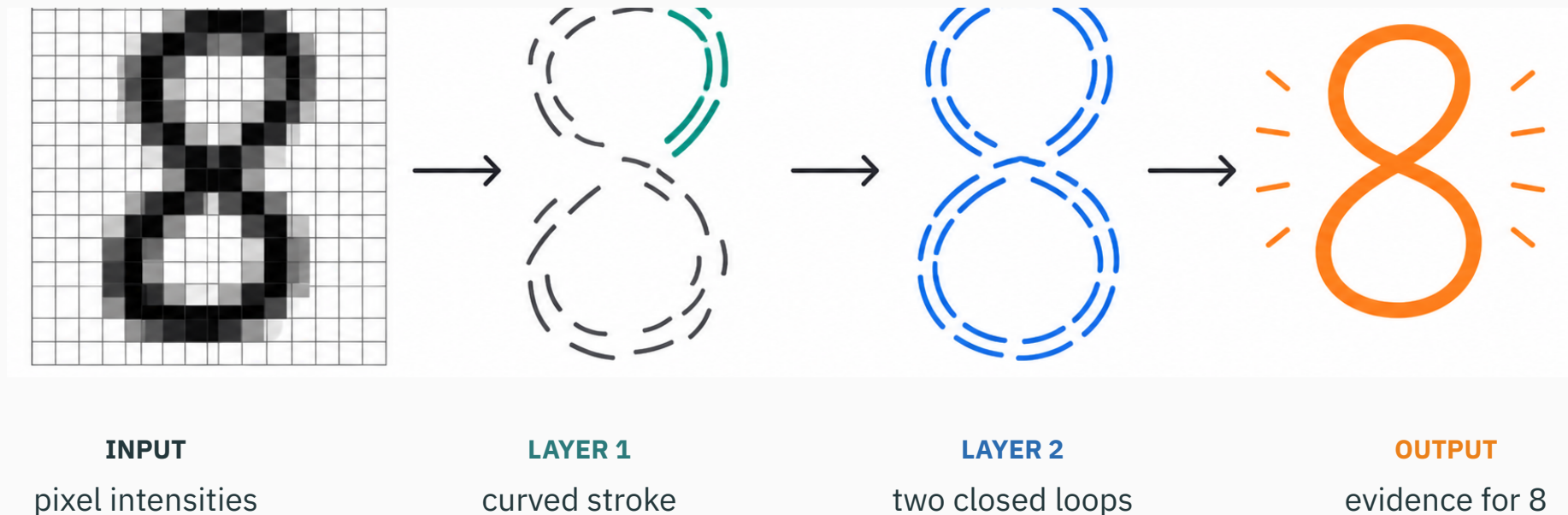
$$-0.5 < x < 0.5$$
$$h_1 = 2x + 1 \rightarrow h_2 = 1 - 2x$$

RIGHT PLATEAU

$$x \geq 0.5$$
$$h_1 \geq 2 \rightarrow h_2 = 0$$

Vision: depth builds strokes, then loops

CONCEPTUAL SCHEMATIC · illustrates a possible hierarchy; not measured model activations



Early units **can** respond to local edges; later units can combine them into shapes that support the final class.

ADD WIDTH

- adds more features **in parallel**;
- helps when many patterns live at the same abstraction level;
- costs wider activations and weight matrices.

Example: add hinge locations to follow a detailed curve.

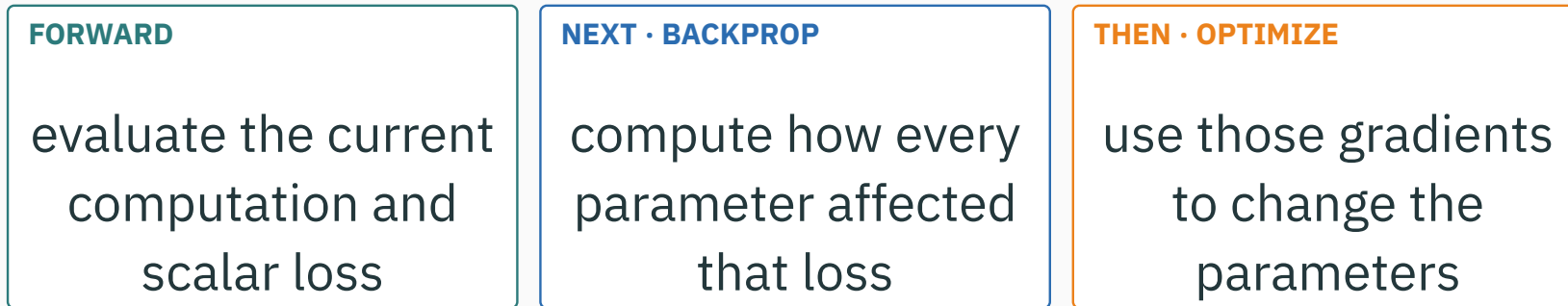
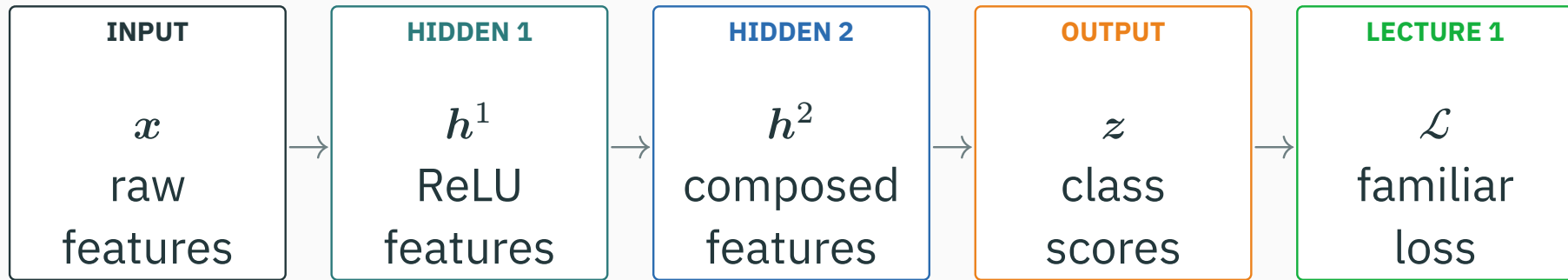
ADD DEPTH

- adds more transformations **in sequence**;
- helps when the target is compositional or hierarchical;
- costs longer paths and harder optimization.

Example: strokes → loops → digit identity.

From representation to learning

A richer representation; the same learning loop



We now have $x \rightarrow f_{\theta}(x) \rightarrow \mathcal{L}$.

Next: how does the loss send credit backward through that computation?

computation graphs · local derivatives · backpropagation · automatic differentiation