DDA4220 · Fall 2026

Lecture 2

Deep Learning and Applications

Neural networks: representation and objectives

Zhen Liu

CUHK-Shenzhen

Lecture outline

01 Neurons and representations

02 Capacity and activations

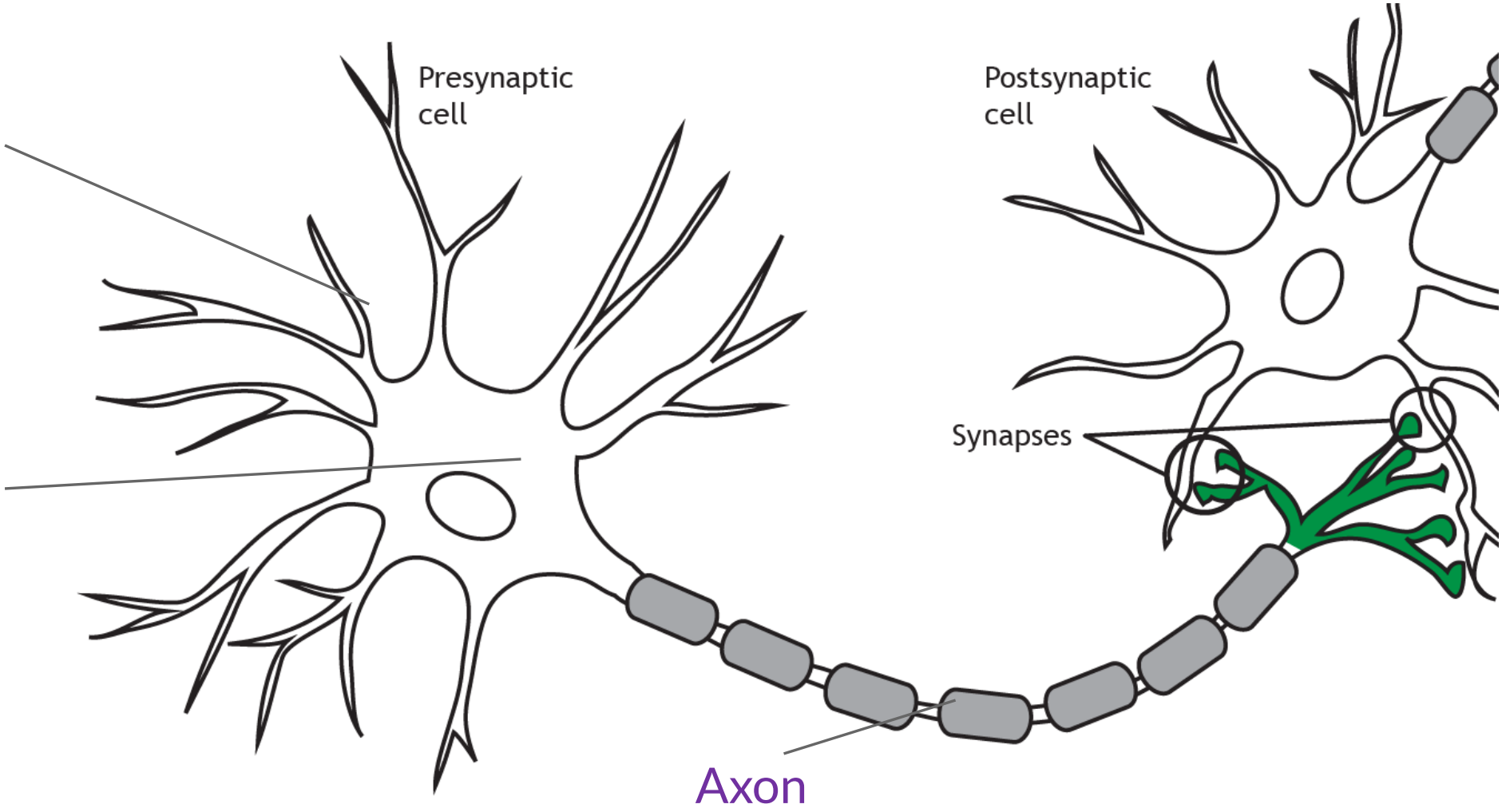
03 Predictions and objectives

* Optional

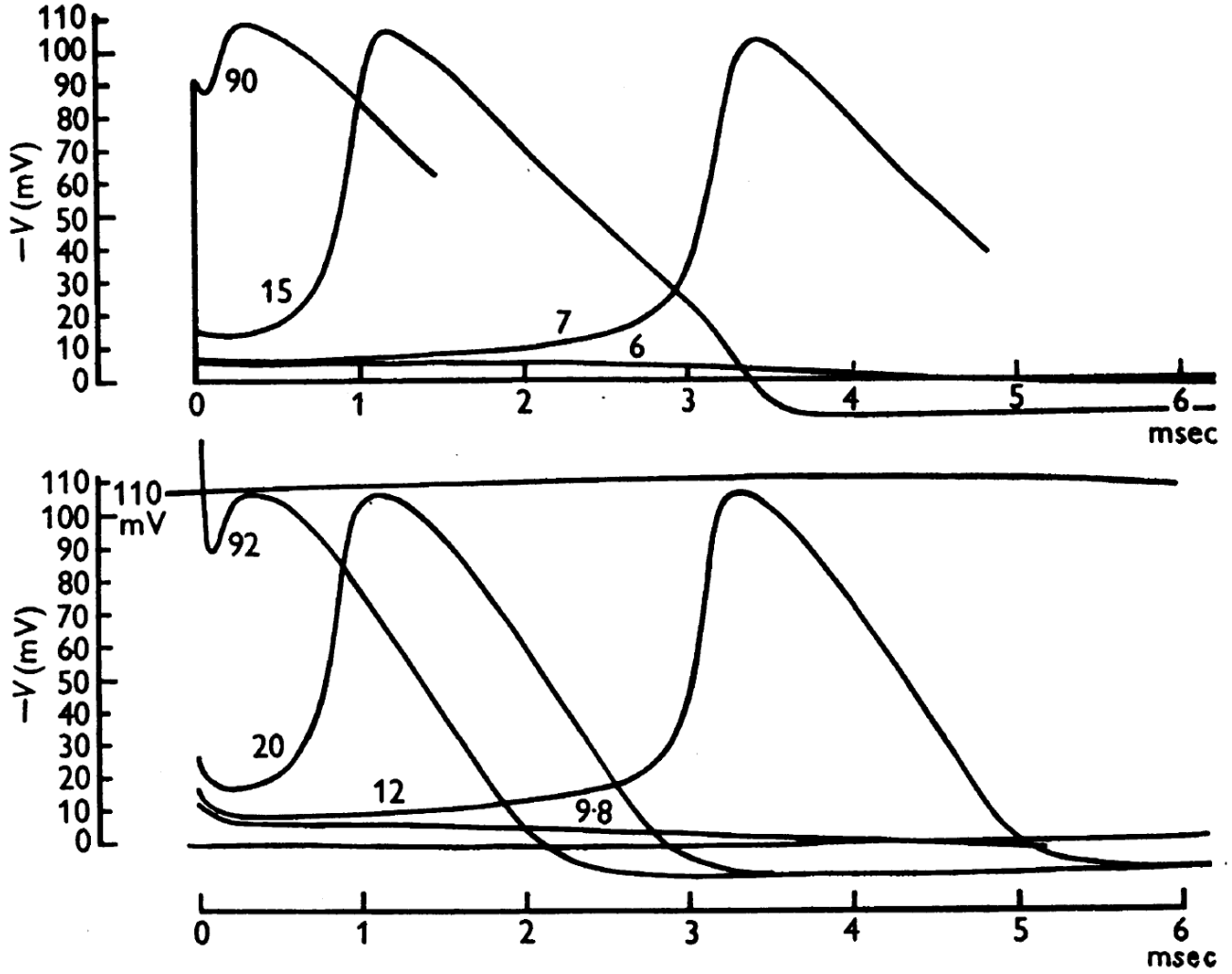
Biological neurons

Dendrites

Cell body



Hodgkin–Huxley model



1952

Calculated

Model equations

Recorded

Squid giant axon

Figure: Hodgkin & Huxley (1952)

Hodgkin–Huxley dynamics

Four coupled differential equations

$$C_m \frac{dV}{dt} = I(t) - \bar{g}_{\mathrm{Na}} m^3 h (V - E_{\mathrm{Na}}) - \bar{g}_{\mathrm{K}} n^4 (V - E_{\mathrm{K}}) - g_{\mathrm{L}} (V - E_{\mathrm{L}})$$

$$\frac{dm}{dt} = \alpha_m(V)(1-m) - \beta_m(V) m$$

$$\frac{dh}{dt} = \alpha_h(V)(1-h) - \beta_h(V) h$$

$$\frac{dn}{dt} = \alpha_n(V)(1-n) - \beta_n(V) n$$

V

Membrane voltage

m, h, n

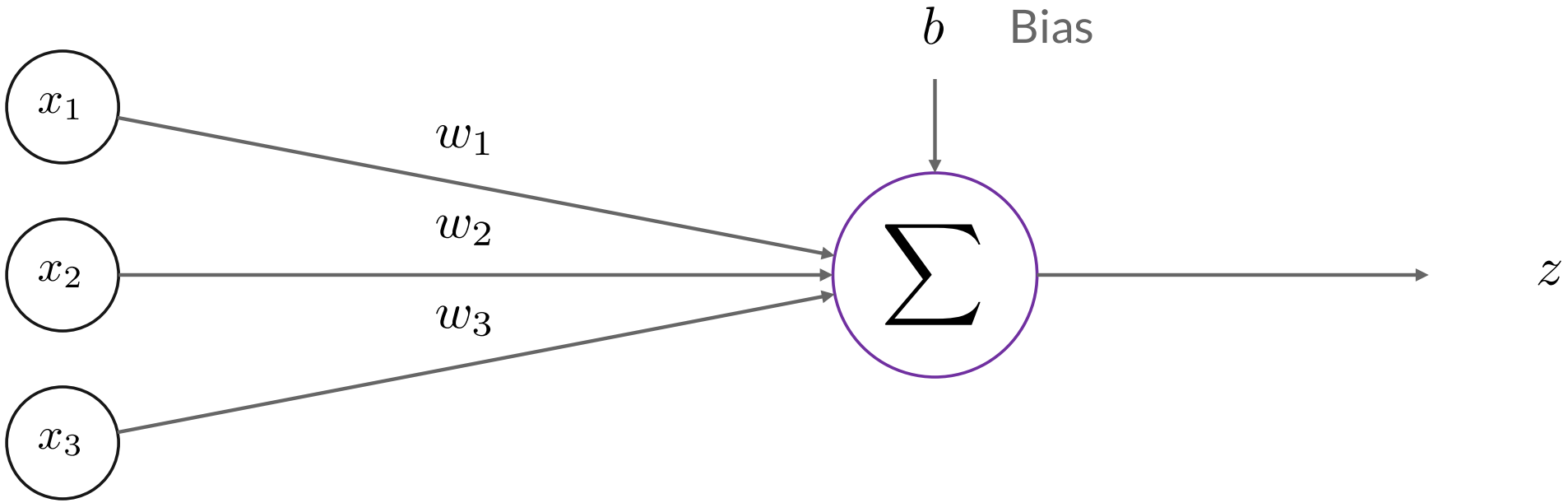
Channel gating

Biological inspiration and mathematical models

$$z = w^{\mathsf{T}} x + b, \qquad h = \phi(z)$$

- Biology motivated weighted connections and nonlinear responses.
- Our goal is to learn useful functions from data.
- These networks do not simulate ion currents or spike timing.

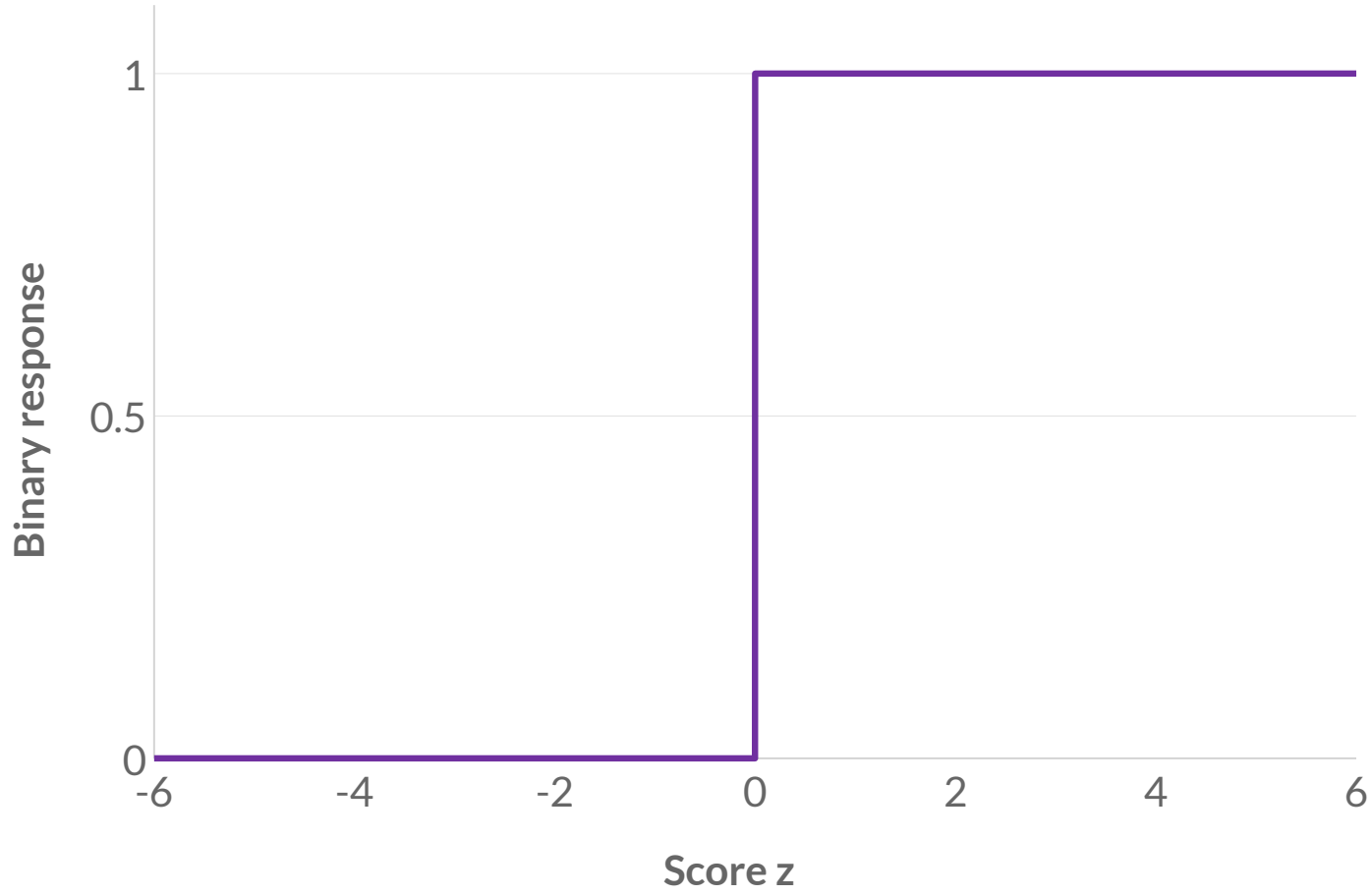
Weighted inputs



$$z = w^{\mathsf{T}} x + b = \sum_{j=1}^{d} w_j x_j + b$$

- Each weight scales its input's contribution.
- The bias adds an offset to the weighted sum.

Threshold neuron



$$s = \mathbf{1}\{z \geq 0\}$$

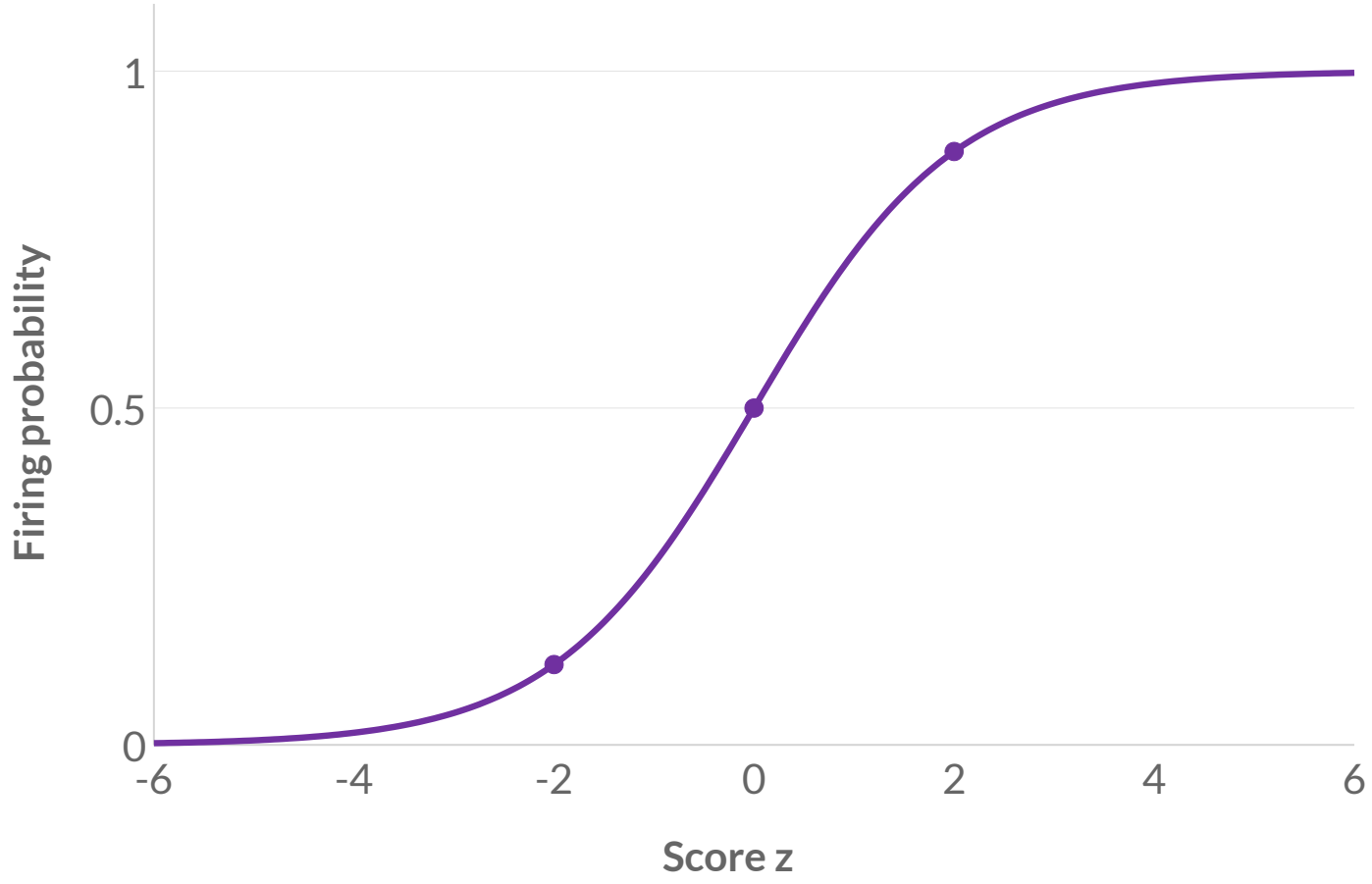
$$z = \sum_i w_i x_i + b$$

Indicator function

- True condition: 1
- False condition: 0

Sigmoid: a simple firing model

A larger input score raises the firing probability gradually.



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

$$\Pr(s = 1 \mid x) = p$$

$$\sigma(-2) \approx 0.12$$

$$\sigma(0) = 0.50$$

$$\sigma(2) \approx 0.88$$

Binary and continuous activations

Stochastic binary unit

$$s \sim \text{Bernoulli}(p)$$

$$\mathbb{E}[s \mid x] = p$$

- Each sampled event is 0 or 1.

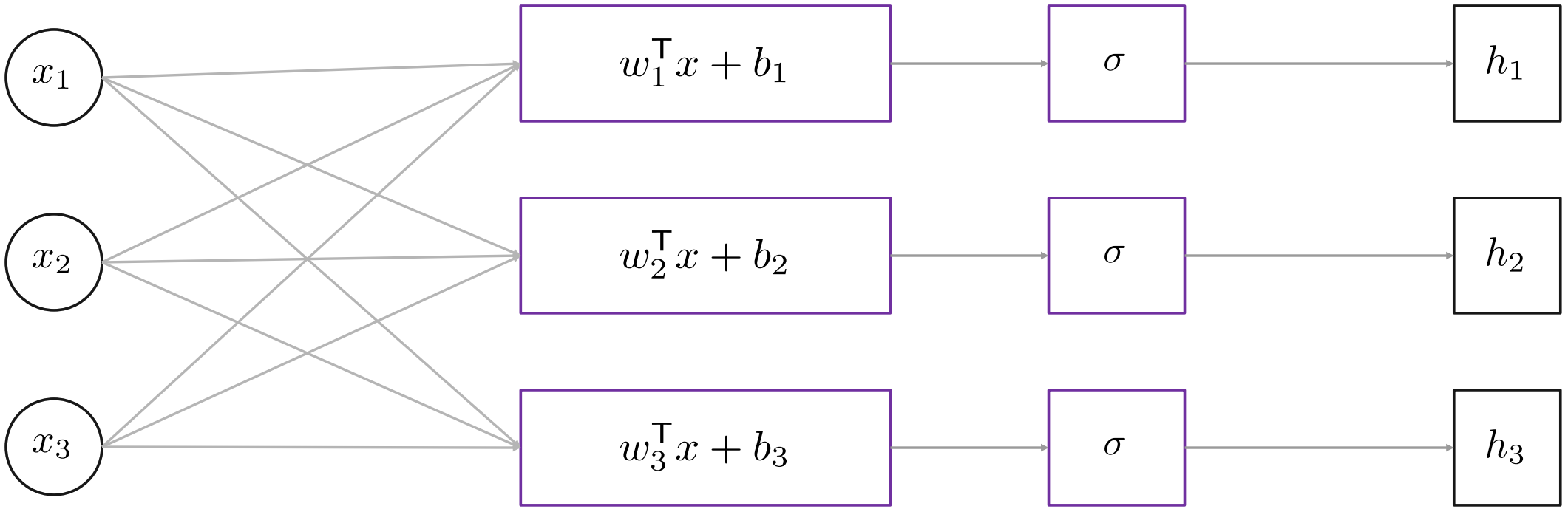
Ordinary sigmoid MLP

$$h = \sigma(z) = p$$

$$\text{For } p = 0.8: \quad h = 0.8$$

- Pass the continuous response.

A layer of neurons



$$h_j = \sigma(w_j^\top x + b_j), \qquad j = 1, 2, 3$$

- Same input; different weights and biases.

Layer notation

$$z_j = w_j^\mathsf{T} x + b_j$$

T transposes the weight vector into a row.

$$z = Wx + b, \qquad W = \begin{bmatrix} w_1^\mathsf{T} \\ w_2^\mathsf{T} \\ \vdots \\ w_m^\mathsf{T} \end{bmatrix}$$

- One row of W belongs to one output unit.
- A linear layer computes weighted sums plus biases.

Vectors, matrices and dimensions

$$x \in \mathbb{R}^d, \qquad W \in \mathbb{R}^{m \times d}, \qquad Wx \in \mathbb{R}^m$$

d input entries | m rows, d columns | m output entries

$$W = \begin{bmatrix} 1 & -1 \\ 2 & 0 \\ 0 & 1 \end{bmatrix}, \quad x = \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \quad Wx = \begin{bmatrix} 1 \\ 4 \\ 1 \end{bmatrix}$$

- Each column holds one example, with one entry per feature.
- Rows of W specify different weighted sums.

Elementwise activation

$$h = \sigma(z) = \begin{bmatrix} \sigma(z_1) \\ \sigma(z_2) \\ \sigma(z_3) \end{bmatrix}$$

$$\begin{bmatrix} -2 \\ 0 \\ 2 \end{bmatrix} \xrightarrow{\sigma} \begin{bmatrix} 0.12 \\ 0.50 \\ 0.88 \end{bmatrix}$$

- Apply the scalar function separately to each entry.

Matrix products and elementwise products

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \qquad B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

Matrix product

$$AB = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$

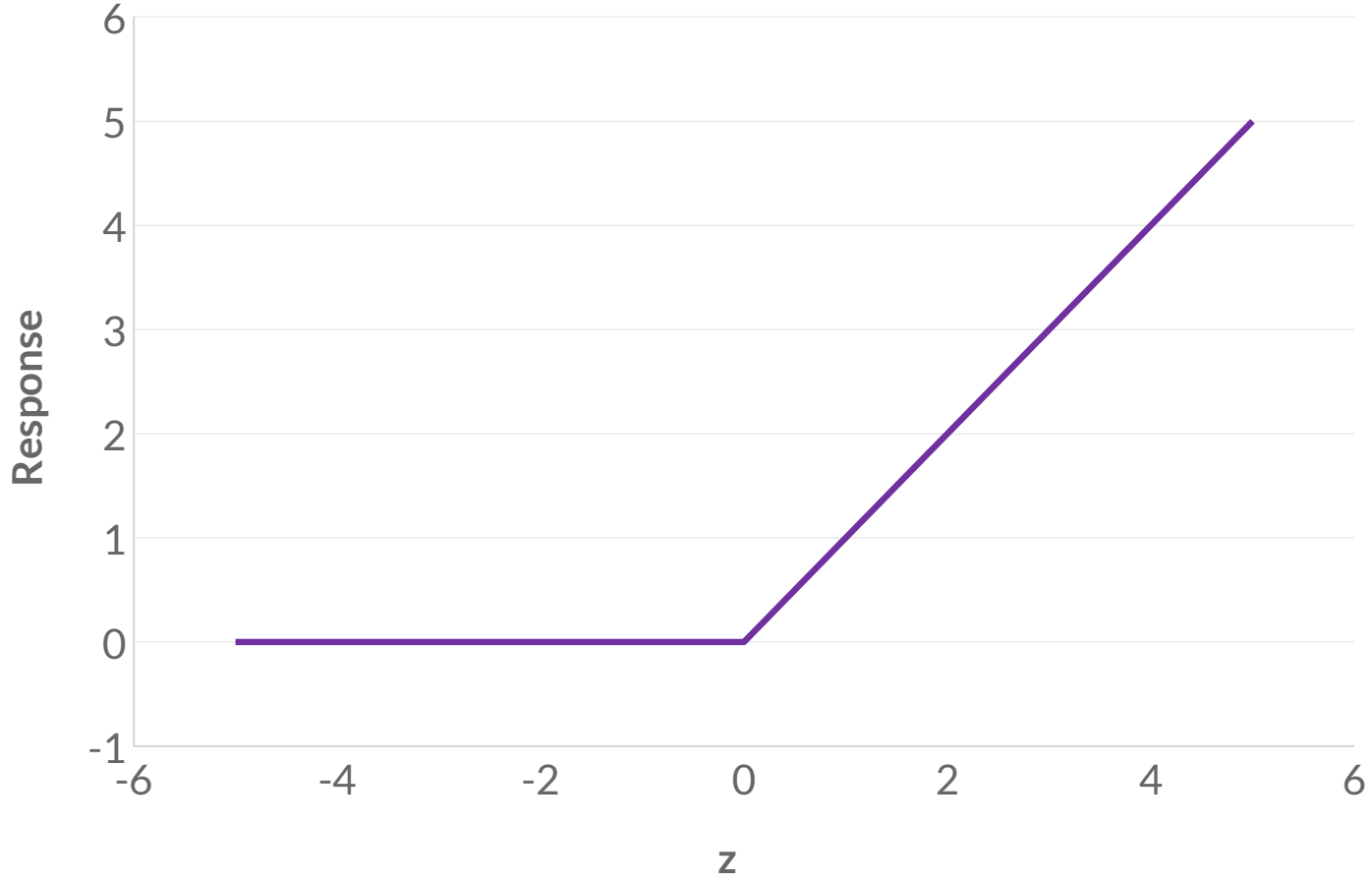
- Combines rows with columns.

Elementwise product

$$A \odot B = \begin{bmatrix} 5 & 12 \\ 21 & 32 \end{bmatrix}$$

- Multiplies matching entries.

ReLU activation

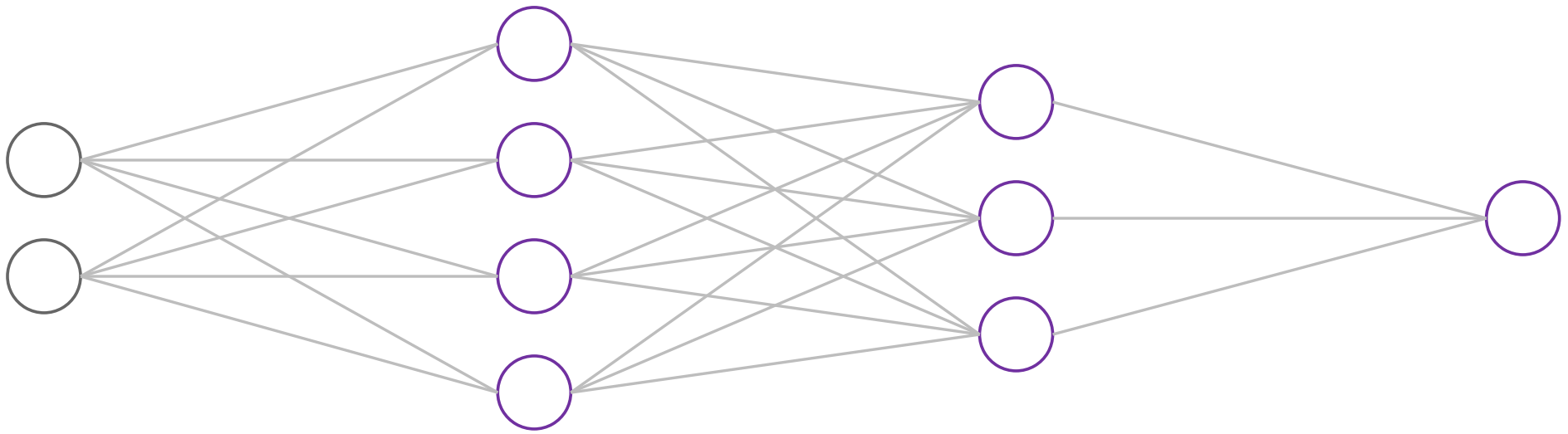


$$h = \phi(z)$$

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

- Negative input $\rightarrow 0$
- Positive input $\rightarrow$ unchanged response

Multilayer perceptron

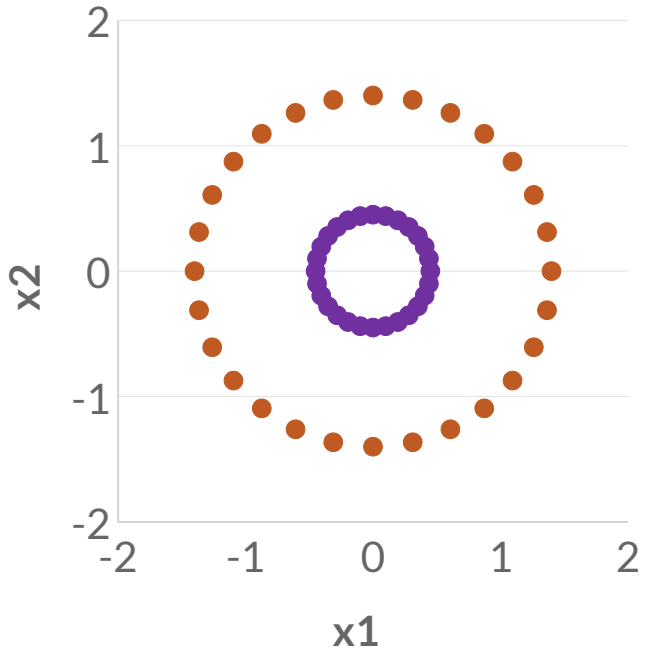


Input Hidden layer 1 Hidden layer 2 Output

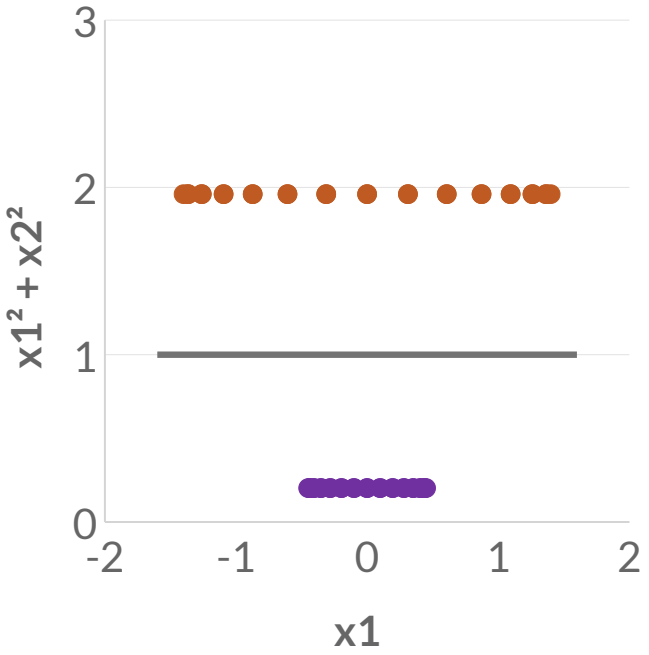
- Hidden layers form new feature vectors (representations).
- The output layer produces the task's prediction or scores.

Nonlinear feature maps

Input coordinates



Chosen nonlinear coordinates



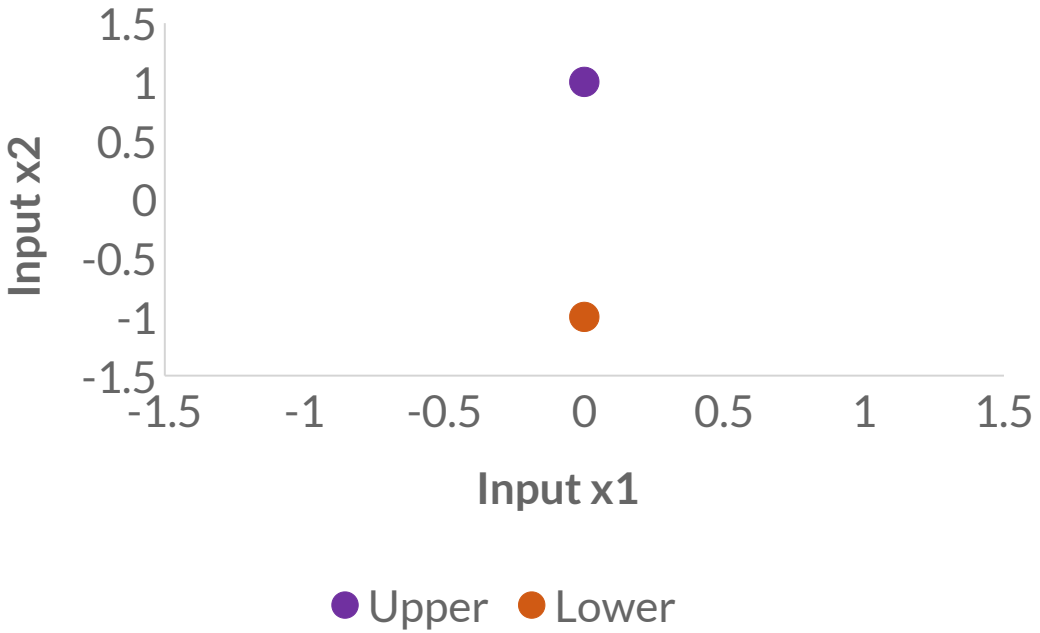
Inner class Outer class Linear boundary

$$s = w^\mathsf{T} x + b$$

$$s = v^\mathsf{T} \phi(x) + c$$

Information in a representation

A different task: upper or lower



$$\phi(x) = (x_1,\ x_1^2 + x_2^2)$$

The same fixed feature map

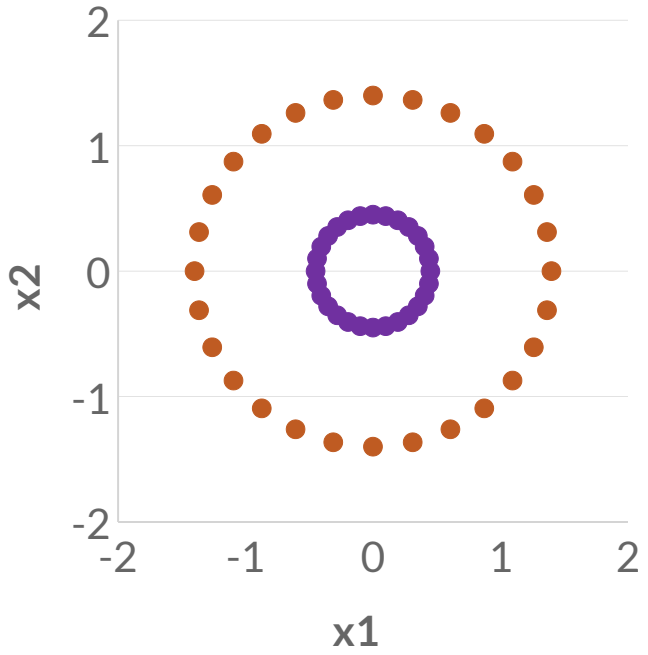
$$\phi(0,1) = \phi(0,-1) = (0,1)$$

Once two inputs have the same representation, the output cannot distinguish them.

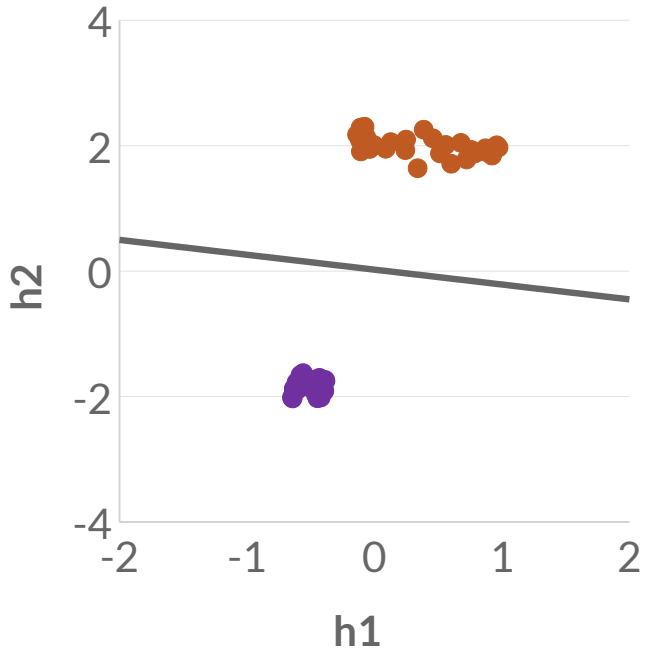
Learned representations

The hidden-layer weights and biases θ are learned from the labels.

Input coordinates



Two learned coordinates



Inner class Outer class Linear boundary

$$h_\theta(x) \in \mathbb{R}^2, \qquad s = v^\mathsf{T} h_\theta(x) + c$$

MLP equations

$$h^{[1]} = \phi(W^{[1]}x + b^{[1]})$$

$$h^{[2]} = \phi(W^{[2]}h^{[1]} + b^{[2]})$$

$$s = W^{[3]}h^{[2]} + b^{[3]}$$

- Each layer uses the previous layer's responses.

Forward computation

Parameters and input

$$x = 1.5$$

$$w = (1, 1, 1)^\mathsf{T}$$

$$b = (0, -1, -2)^\mathsf{T}$$

$$v = (1, -1, 1)^\mathsf{T}, \quad c = 0$$

Computed values

$$z = wx + b = (1.5, 0.5, -0.5)^\mathsf{T}$$

$$h = \mathrm{ReLU}(z) = (1.5, 0.5, 0)^\mathsf{T}$$

$$\widehat{y} = v^\mathsf{T} h + c = 1$$

Batched computation

B examples are grouped as columns of X.

$$X = [x_1 \; x_2 \; \cdots \; x_B] \in \mathbb{R}^{d \times B}$$

$$WX = [Wx_1 \; Wx_2 \; \cdots \; Wx_B] \in \mathbb{R}^{m \times B}$$

- The same weights are applied to every example.
- A batch is a collection of examples, not a new layer.

Biases and broadcasting

$$Z = WX + \underbrace{\begin{bmatrix} b & b & \cdots & b \end{bmatrix}}_{\text{one bias column per example}}$$

$$b = \begin{bmatrix} 1 \\ -2 \end{bmatrix} \quad \longrightarrow \quad \begin{bmatrix} 1 & 1 & 1 \\ -2 & -2 & -2 \end{bmatrix}$$

- Each output unit shares its bias across the batch.
- Broadcasting expresses the repeated addition.

What is PyTorch?

PyTorch is an open-source Python framework
for building and training neural networks.

- Models and training are written as Python code.
- PyTorch supports training neural networks on GPUs to speed up computation.

PyTorch

pytorch.org

Tensors, axes and shapes

A torch.Tensor stores numbers with zero, one, two or more axes.

An axis (dimension) is one direction used to index the array.

0 axes: scalar

$$5$$

shape $()$

1 axis: vector

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$$

shape $(3,)$

2 axes: matrix

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

shape $(2, 3)$

3 axes: two matrices

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \quad \begin{bmatrix} 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix}$$

shape $(2, 2, 3)$

```
import torch
x = torch.tensor([[1., 2., 3.], [4., 5., 6.]])
x.shape    # torch.Size([2, 3]): arrangement of values
x.ndim     # 2 axes: rows and columns
```

Matrix and elementwise multiplication

Matrix multiplication

```
a @ b
torch.matmul(a, b)
```

$$\underbrace{\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}}_{a}\ \underbrace{\begin{bmatrix}5 & 6\\ 7 & 8\end{bmatrix}}_{b} = \begin{bmatrix}19 & 22\\ 43 & 50\end{bmatrix}$$

Each output is a row–column dot product.

Elementwise multiplication can broadcast.

```
scale * values  # shapes (3, 1) and (3, 5)
```

$$\underbrace{\begin{bmatrix}2\\ 3\\ 4\end{bmatrix}}_{(3,1)} * \underbrace{\begin{bmatrix}1 & 2 & 3 & 4 & 5\\ 1 & 2 & 3 & 4 & 5\\ 1 & 2 & 3 & 4 & 5\end{bmatrix}}_{(3,5)} = \underbrace{\begin{bmatrix}2 & 4 & 6 & 8 & 10\\ 3 & 6 & 9 & 12 & 15\\ 4 & 8 & 12 & 16 & 20\end{bmatrix}}_{(3,5)}$$

Each row's value is reused across its five columns.

Align axes from the right. Each size must match or be 1.

Reshape, view and flatten

Reshape keeps the value order. Use -1 to infer one dimension.

$$x = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

```
x.reshape(3, -1)
x.view(3, -1)
# infer 2 columns
```

Six values in three rows gives two columns.

Flatten the non-batch axes into one feature row per example.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \quad \begin{bmatrix} 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 7 & 8 & 9 & 10 & 11 & 12 \end{bmatrix}$$

$\text{shape } (2, 2, 3)$ $\text{shape } (2, 6)$

```
flat = batch.flatten(start_dim=1)  # keep axis 0: examples
```

Transpose and permute

Transpose swaps two axes. Permute chooses the full axis order.

```
x.transpose(0, 1)  # rows and columns swap
```

$$x = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

Axis 0: rows
Axis 1: columns

Batch axes: 0 = examples, 1 = rows, 2 = columns.

```
batch.permute(0, 2, 1)
batch.permute(0, -1, 1)  # same axis order
```

$$(2, 2, 3) \longrightarrow (2, 3, 2)$$

Here -1 means the last axis. It does not infer a dimension size.

After permute, view may fail. Reshape is simpler for regrouping.

An MLP in PyTorch

```
import torch
from torch import nn

model = nn.Sequential(
    nn.Linear(1, 32), nn.ReLU(),
    nn.Linear(32, 32), nn.ReLU(),
    nn.Linear(32, 1),
)
x = torch.tensor([[1.5], [2.0]])
prediction = model(x)
```

- PyTorch puts examples in rows, features in columns.
- Two hidden layers, each of width 32
- One numerical output per example

Function composition

$$f_\theta = f_L \circ f_{L-1} \circ \cdots \circ f_1$$

Apply f_1 first, then f_2, and so on.

$$h^{[l]} = f_l(h^{[l-1]})$$

- Width: several responses computed in parallel
- Depth: responses transformed repeatedly

Lecture outline

01 Neurons and representations

02 Capacity and activations

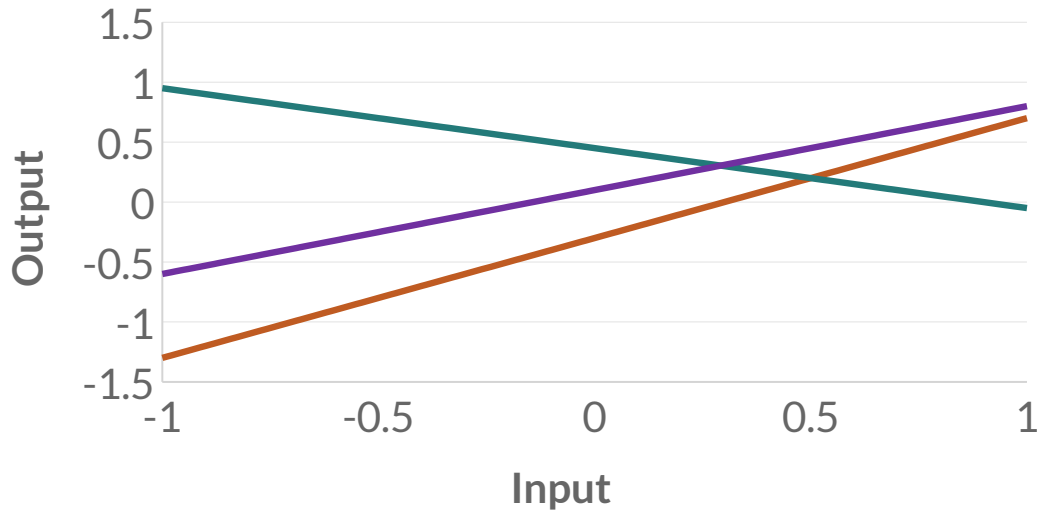
03 Predictions and objectives

Hypothesis class

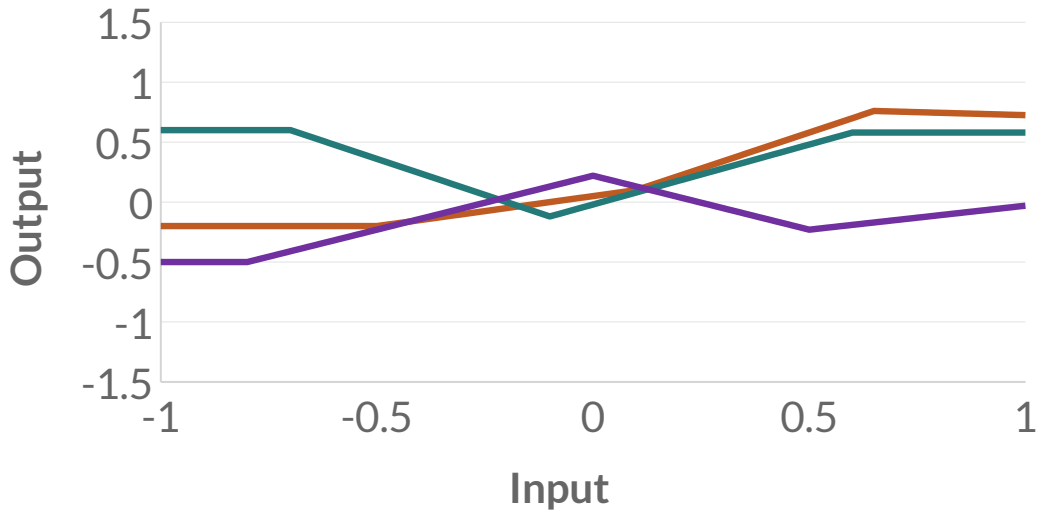
The family of prediction functions the model can represent

θ collects the P trainable weights and biases.

Linear model



MLP with 3 ReLU units



$$\mathcal{H} = \{f_\theta : \theta \in \mathbb{R}^P\}$$

Composition of linear layers

Without nonlinear activations

$$W_2(W_1x + b_1) + b_2$$

$$= (W_2W_1)x + (W_2b_1 + b_2)$$

Stacking linear layers without activations gives one linear layer.

Logical operations with threshold units

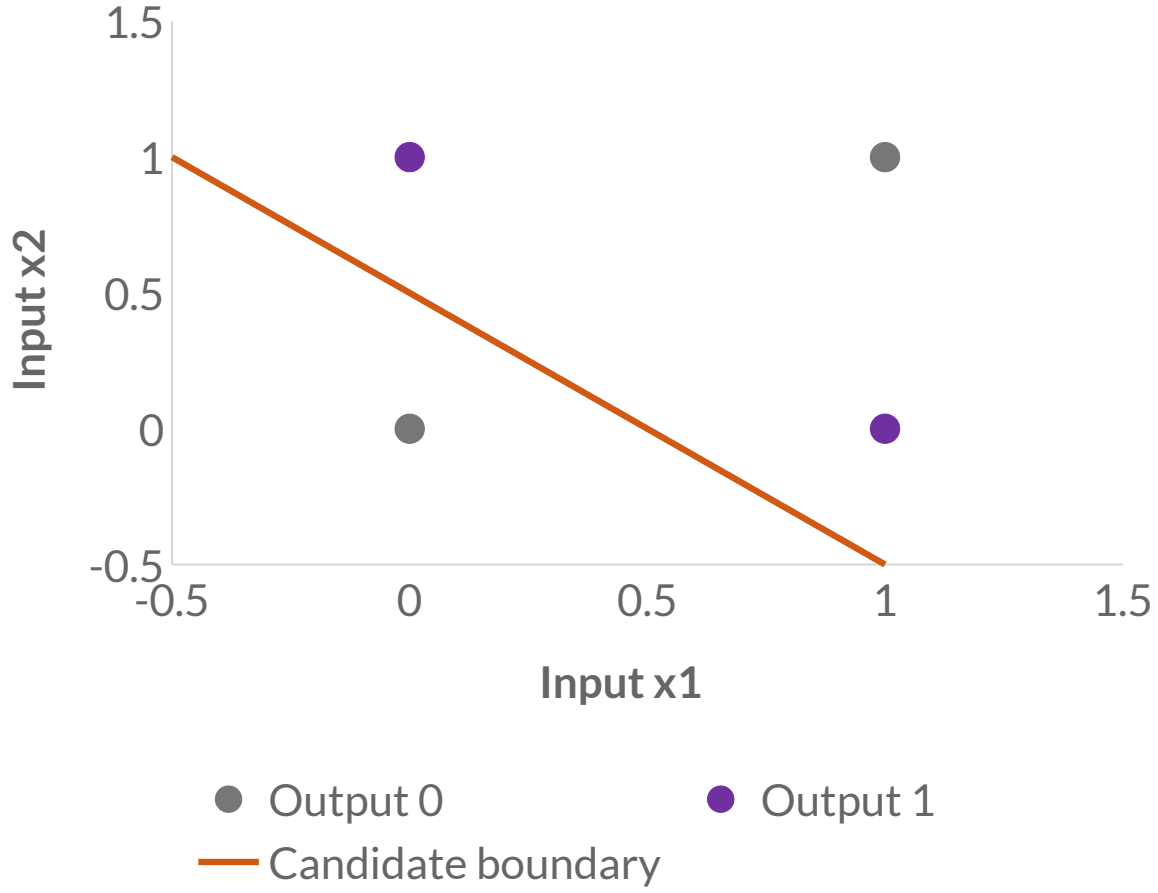
Binary inputs: 0 = false, 1 = true.
A unit fires when its weighted sum reaches the threshold.

$$\mathrm{AND}(x_1, x_2) = \mathbf{1}\{x_1 + x_2 \geq 2\}$$

$$\mathrm{OR}(x_1, x_2) = \mathbf{1}\{x_1 + x_2 \geq 1\}$$

$$\mathrm{NOT}(x) = \mathbf{1}\{-x \geq 0\}$$

XOR



Exclusive OR

Output 1 when exactly one input is 1.

$$y = x_1 \oplus x_2$$

This line also predicts 1 for the input (1, 1).

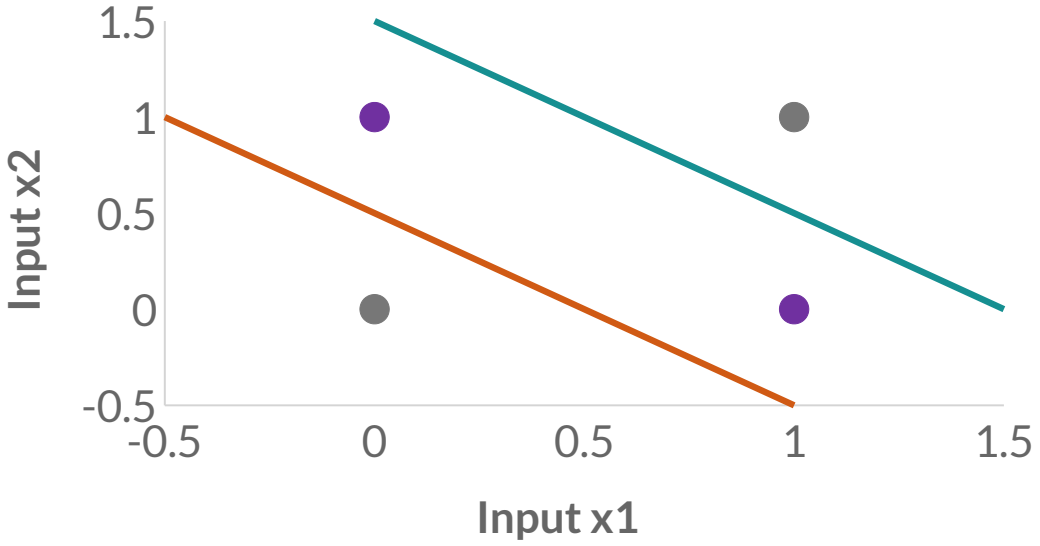
No single straight line separates both pairs.

XOR in hidden coordinates

Two hidden threshold units

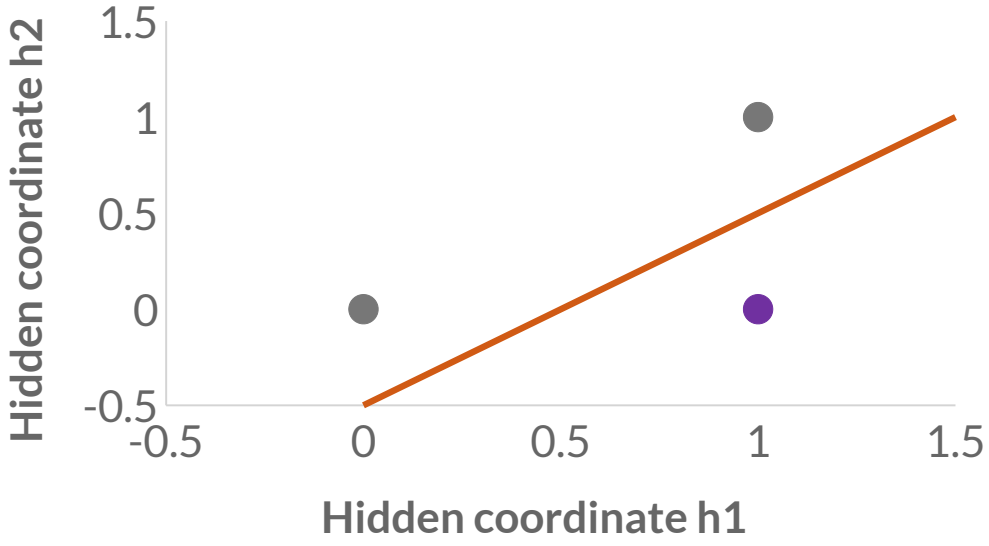
$$h_1 = \mathbf{1}\{x_1 + x_2 \geq \tfrac{1}{2}\}$$

$$h_2 = \mathbf{1}\{x_1 + x_2 \geq \tfrac{3}{2}\}$$

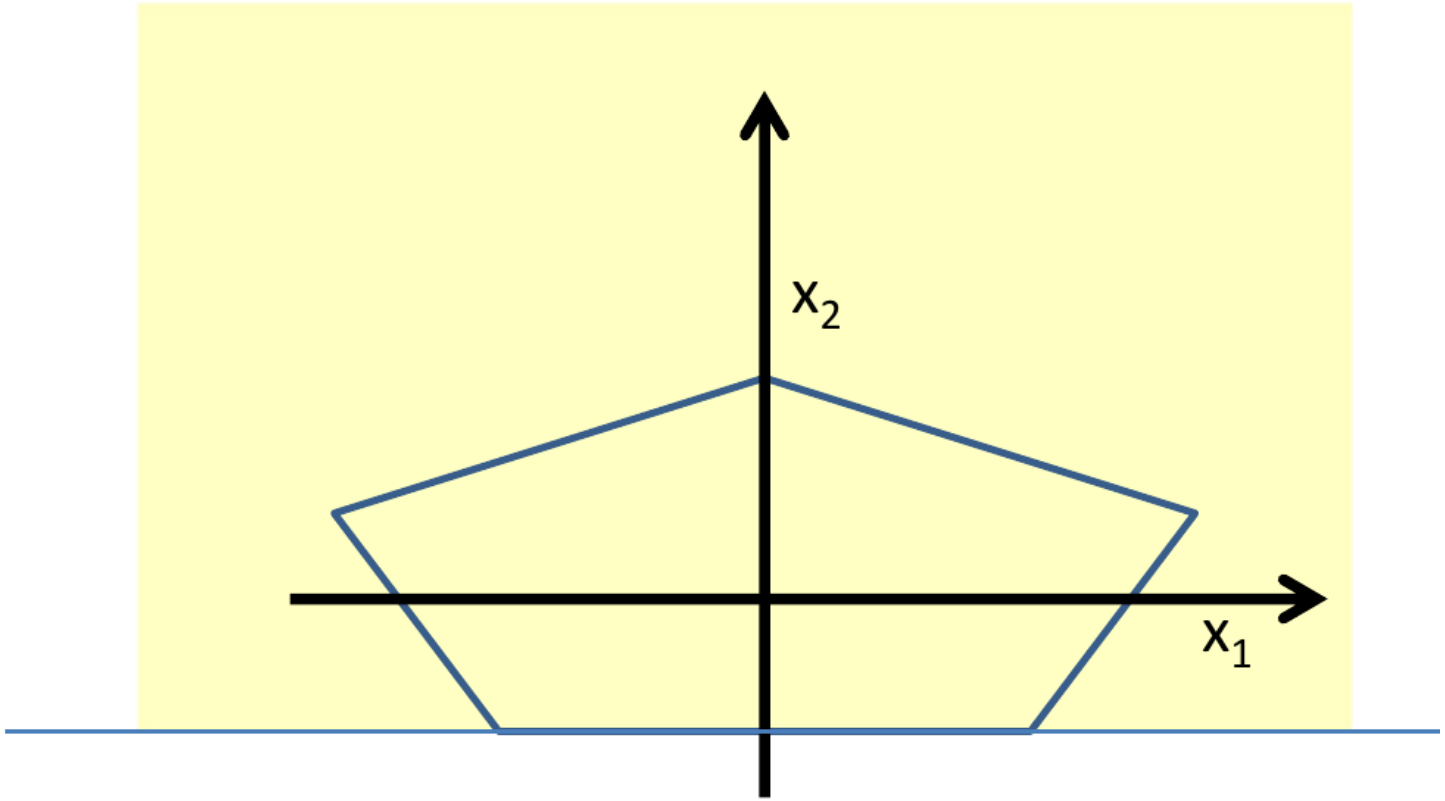


Linear output

$$\widehat{y} = h_1 - h_2$$



Half-plane tests

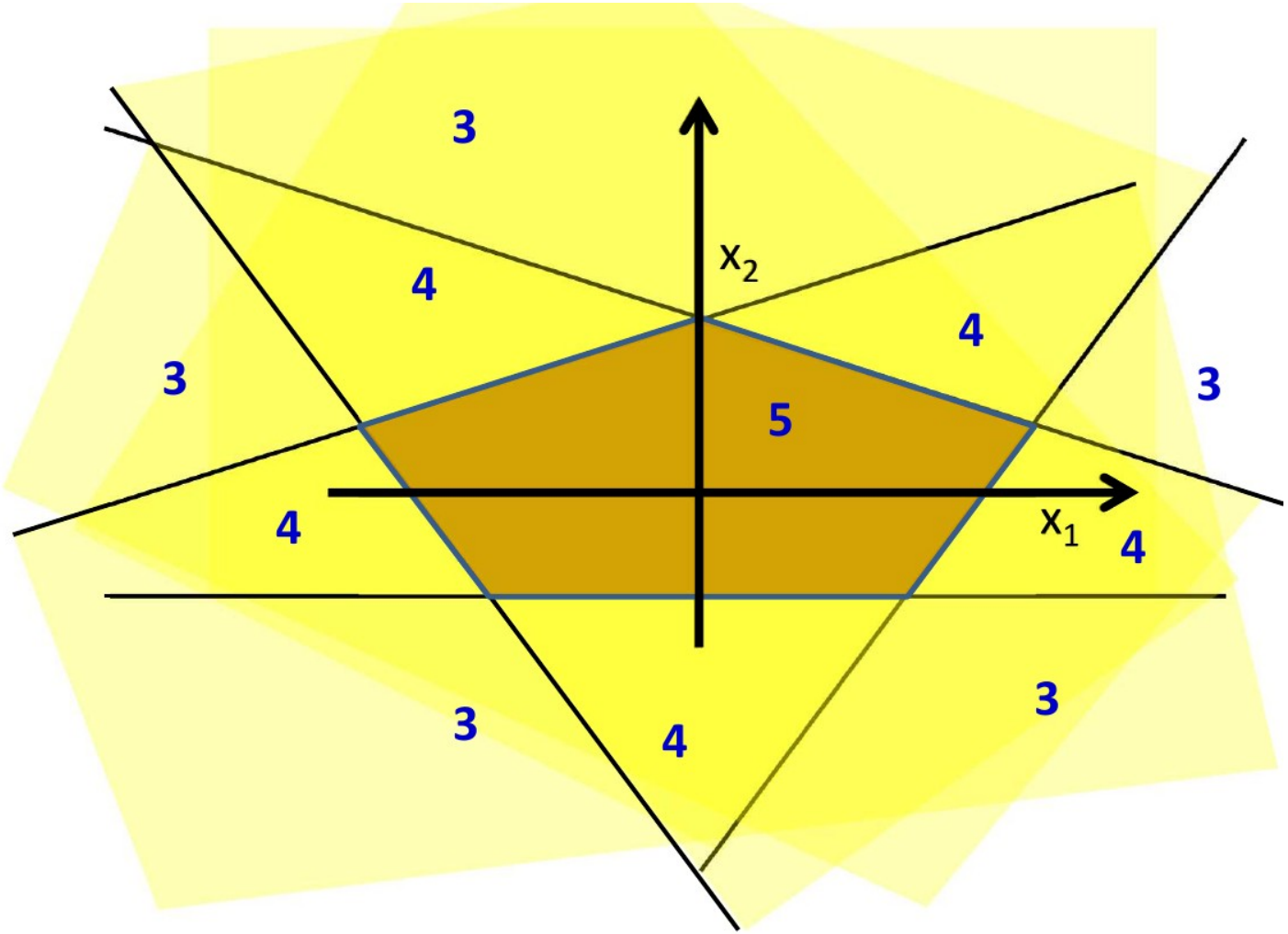


The inputs are coordinates.
A hidden unit tests one side of a line.

$$h_i(x) = \mathbf{1}\{w_i^\mathsf{T} x + b_i \geq 0\}$$

The first test selects points above the bottom edge of the polygon.

Intersecting half-planes



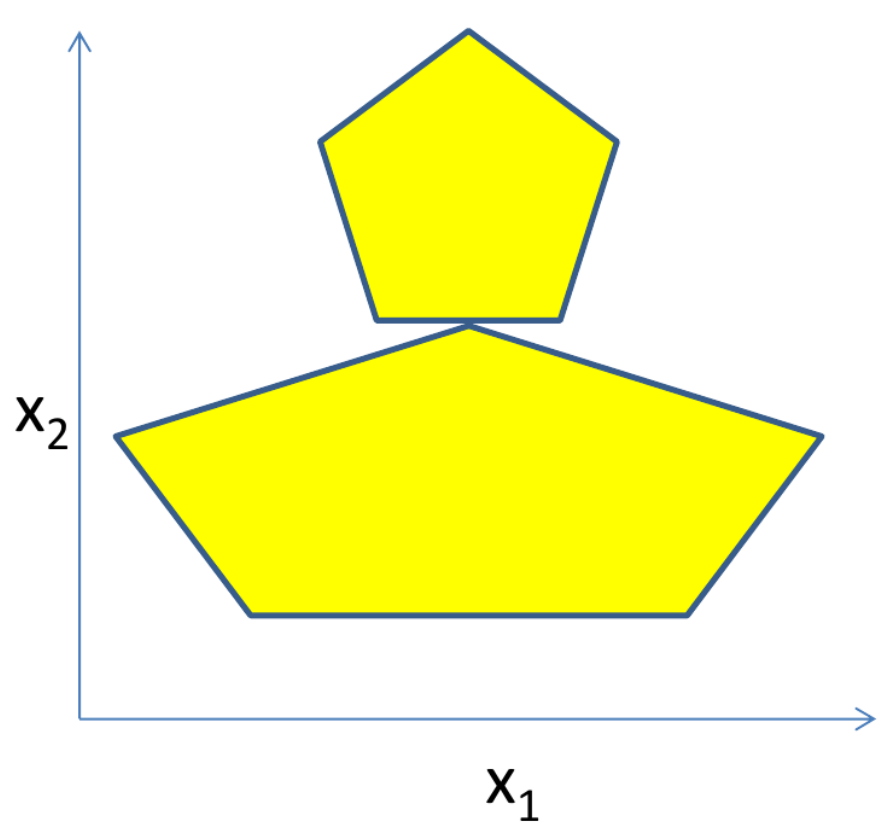
One hidden unit per edge. The numbers count how many tests return 1.

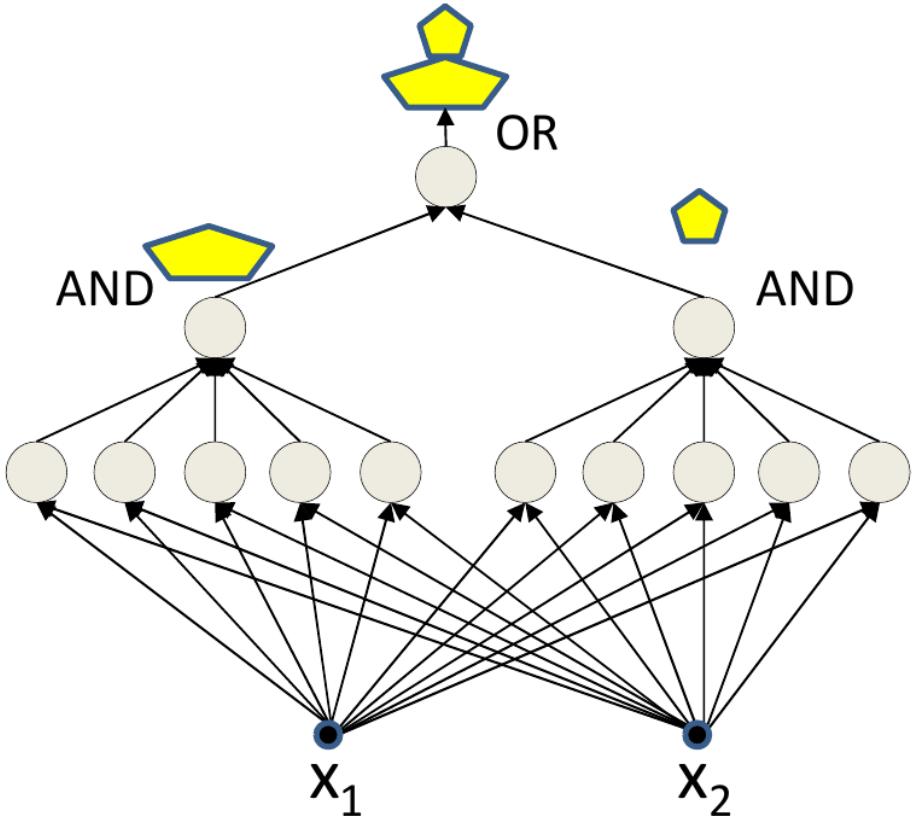
$$y = \mathbf{1}\left\{\sum_{i=1}^{5} h_i(x) \geq 5\right\}$$

AND keeps the intersection: all five tests pass inside the pentagon.

Combining polygons with OR

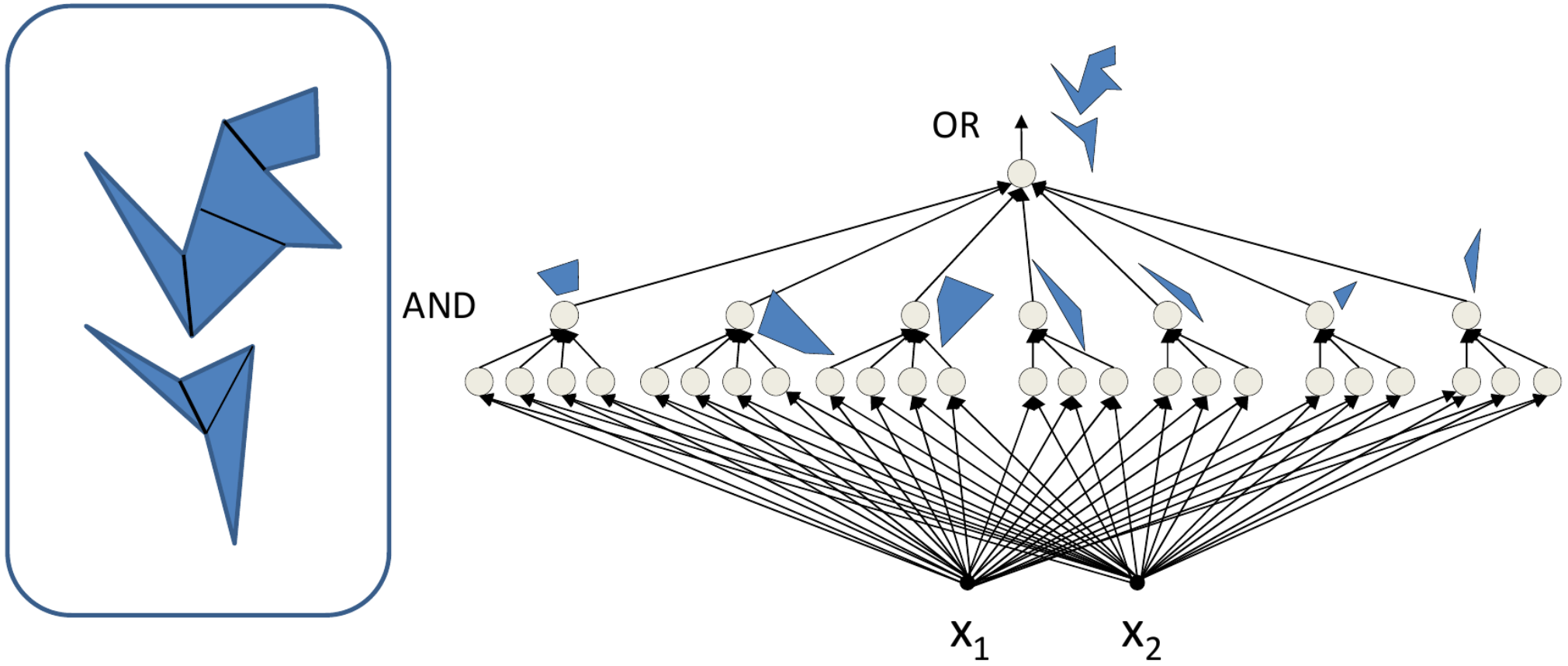
Each AND unit recognizes one polygon. The OR unit accepts either one.



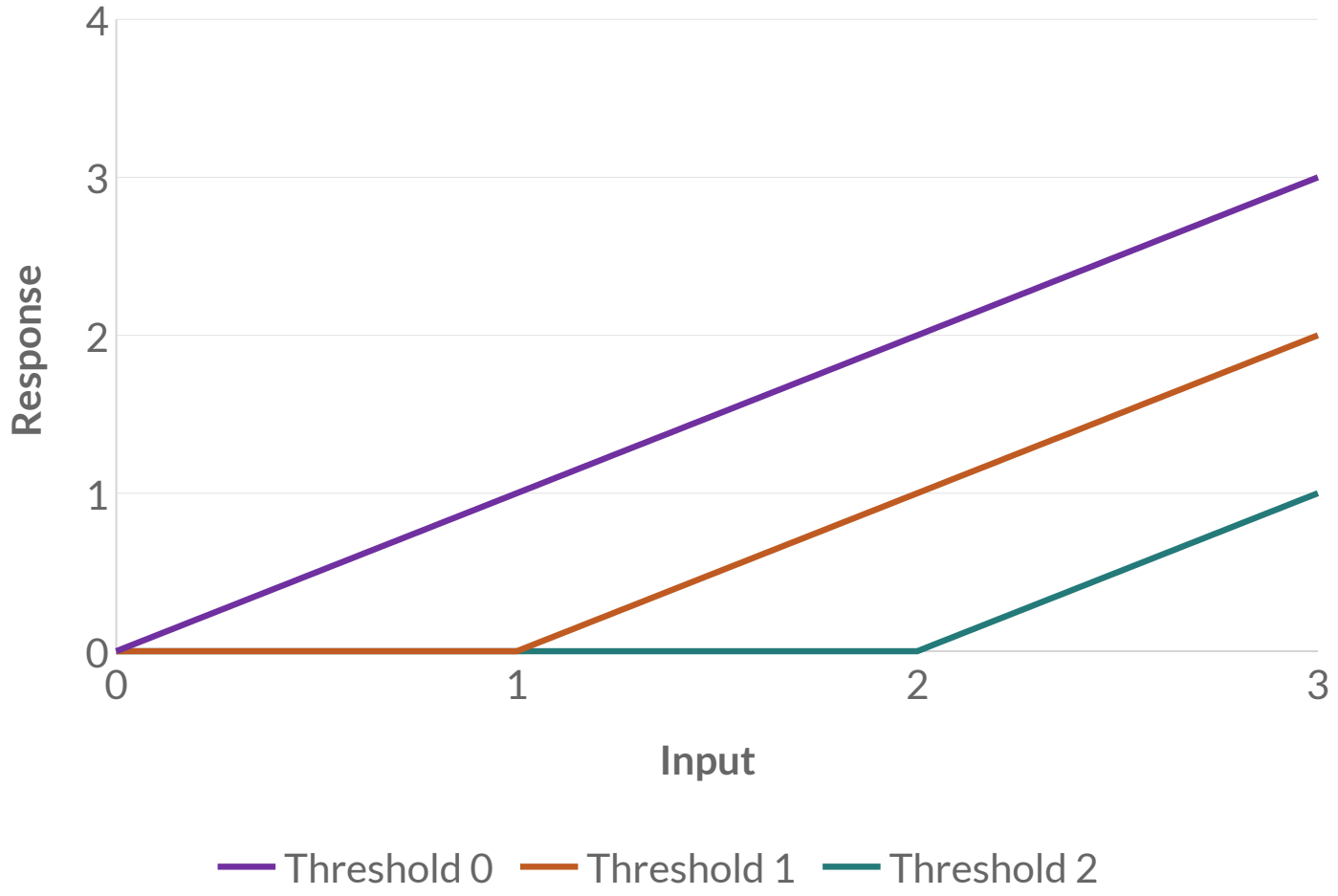


Complex decision regions

Recognize the convex pieces, then combine them.



ReLU hinges

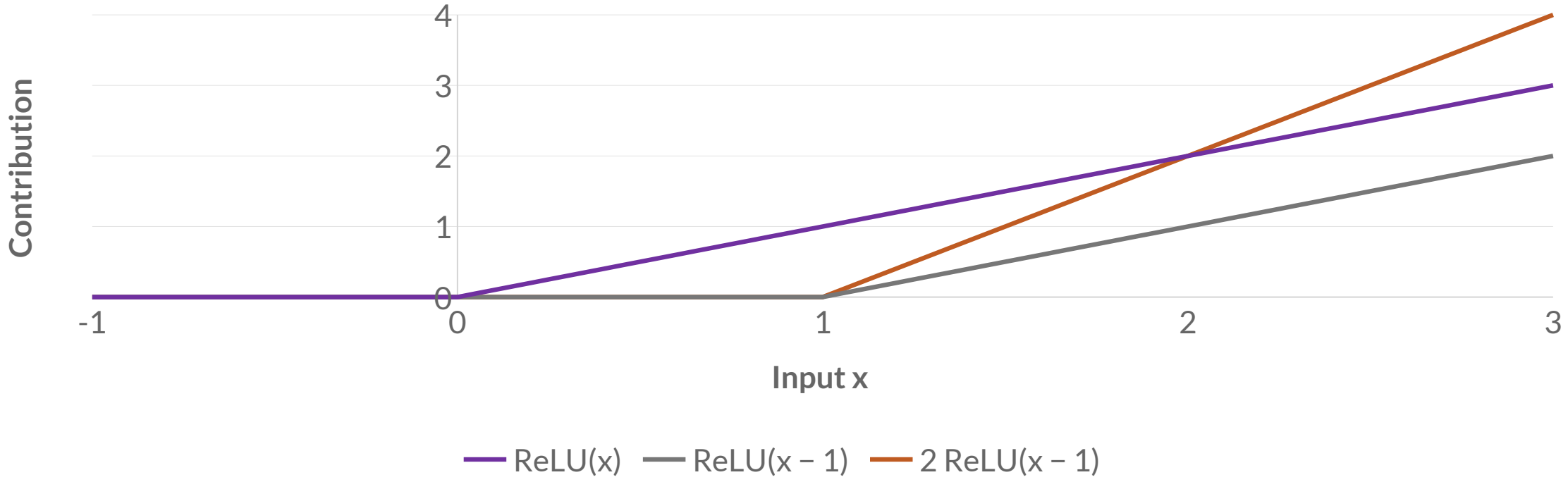


$$h_j(x) = \max(0, x - t_j)$$

- The bias sets the location of a bend.
- A weighted unit changes the slope after that point.

Location and slope of a ReLU hinge

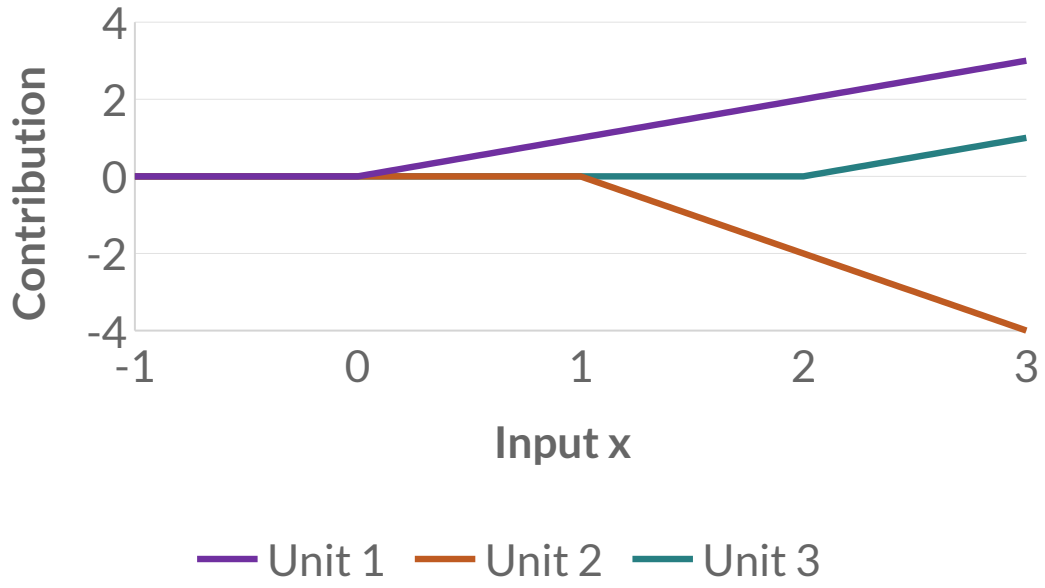
The bias places the bend at t. The output weight v sets the slope.



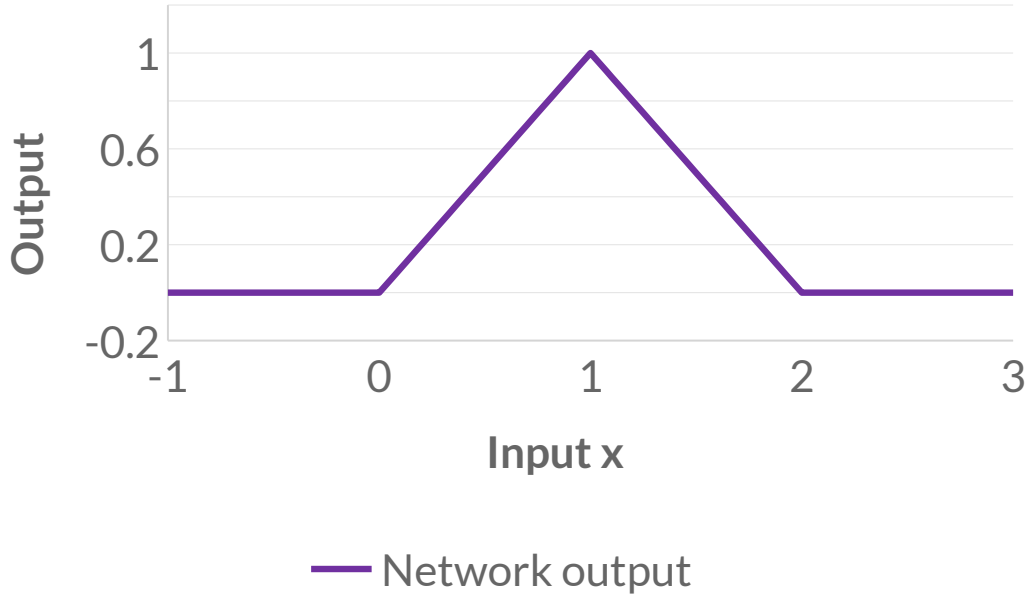
$$f(x) = v\ \mathrm{ReLU}(x - t)$$

Combining ReLU units

Weighted hidden responses



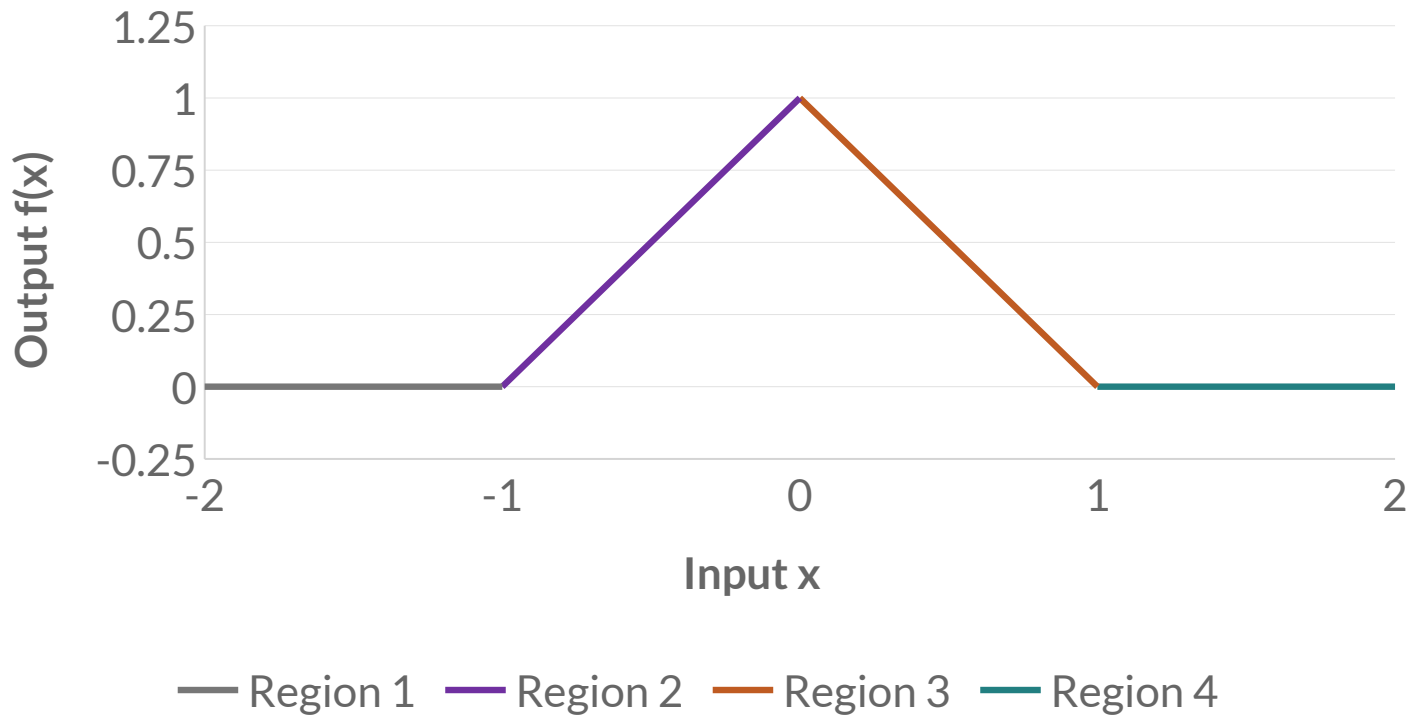
Output: their sum



$$f(x) = \mathrm{ReLU}(x) - 2\,\mathrm{ReLU}(x-1) + \mathrm{ReLU}(x-2)$$

Linear regions in ReLU networks

$$f(x) = \mathrm{ReLU}(x+1) - 2\,\mathrm{ReLU}(x) + \mathrm{ReLU}(x-1)$$

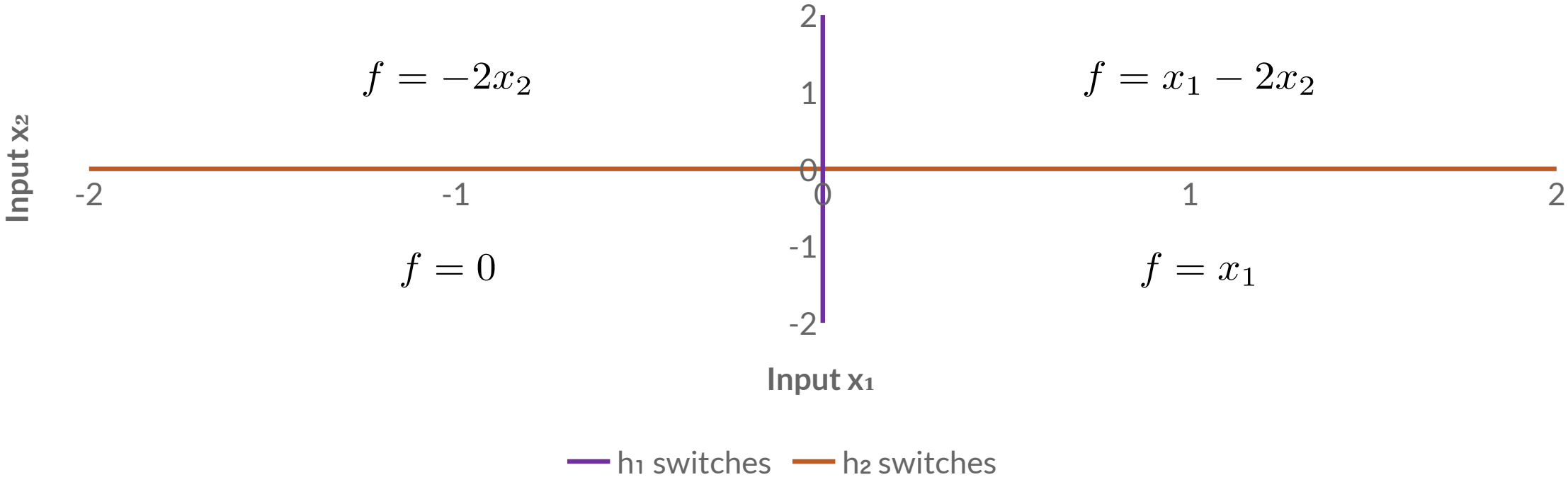


Within a region, each ReLU passes its input through or outputs zero. The network follows one linear rule.

At a boundary, a ReLU changes state and the linear rule can change.

ReLU regions in two dimensions

The lines mark where a hidden unit switches between zero and positive.



$$h_1 = \mathrm{ReLU}(x_1), \quad h_2 = \mathrm{ReLU}(x_2), \quad f = h_1 - 2h_2$$

Universal approximation

- Each added ReLU can introduce another bend.
- More bends give a closer match to the target.

Universal approximation theorem

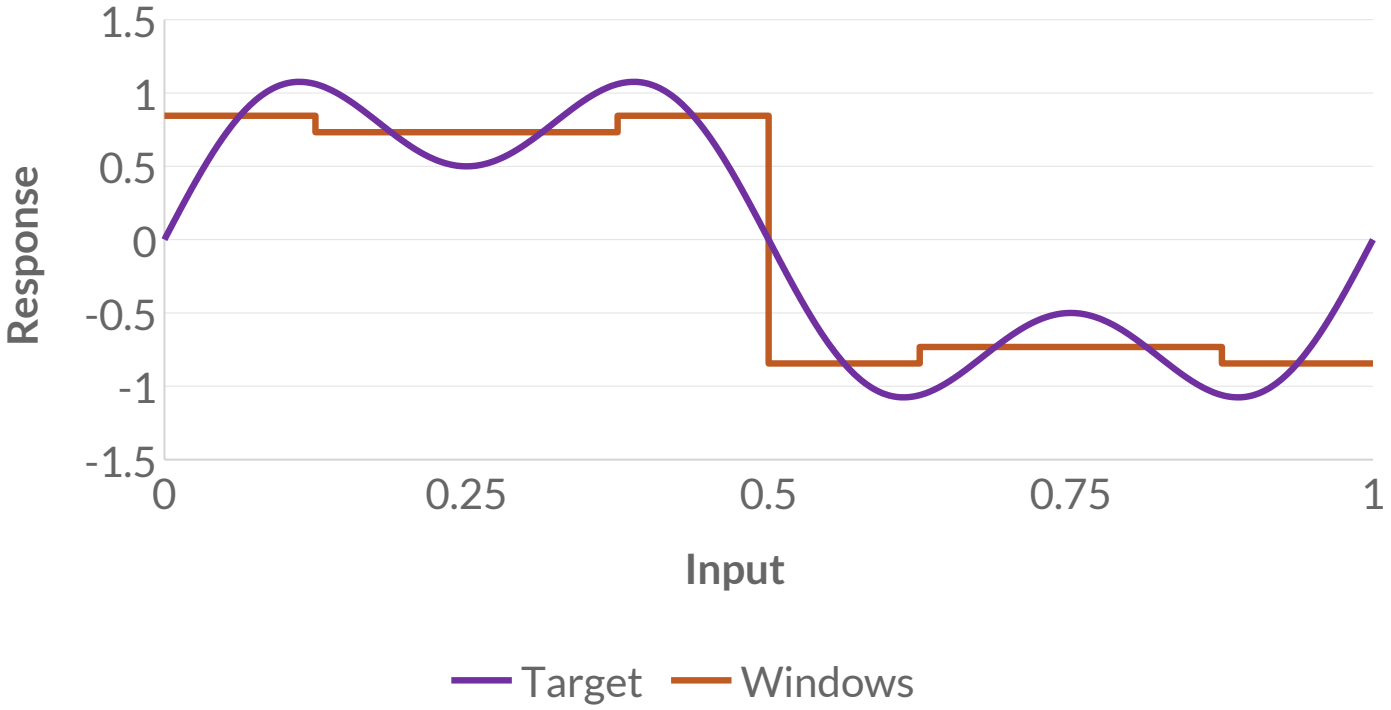
On a closed, bounded input region, a sufficiently wide network with one ReLU hidden layer can approximate any continuous function as closely as desired.

Limits of the result

- The required network may be very large.
- The theorem does not tell us how to find its weights.

Approximation with local windows

Different building blocks can require very different numbers of units.

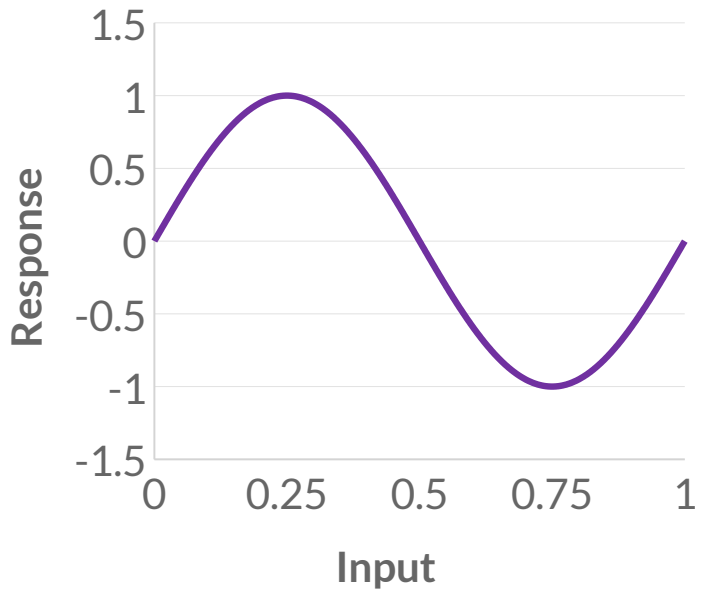


- A window has a constant value on one interval.
- More pieces can describe finer variation.

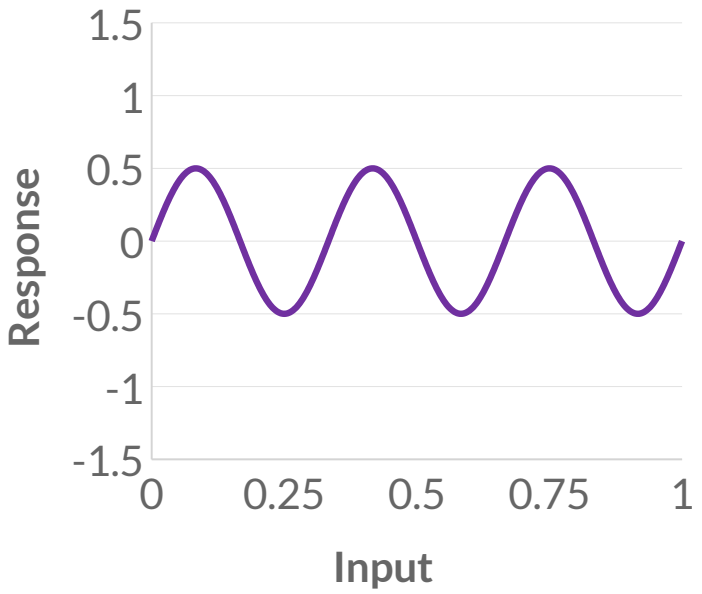
Adding the windows gives a stepwise approximation to the target.

A sinusoidal expansion

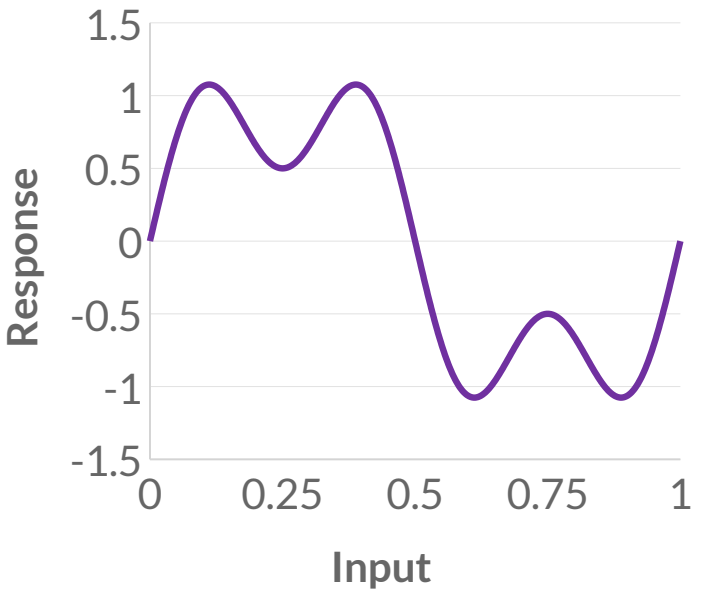
One cycle



Three cycles



Their sum



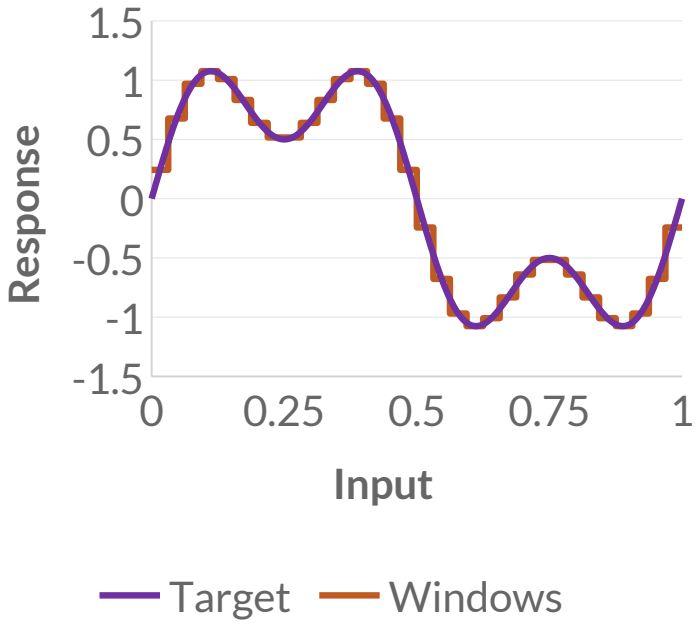
$$f(x) = \sin(2\pi x) + \tfrac{1}{2}\sin(6\pi x)$$

Basis choice and approximation

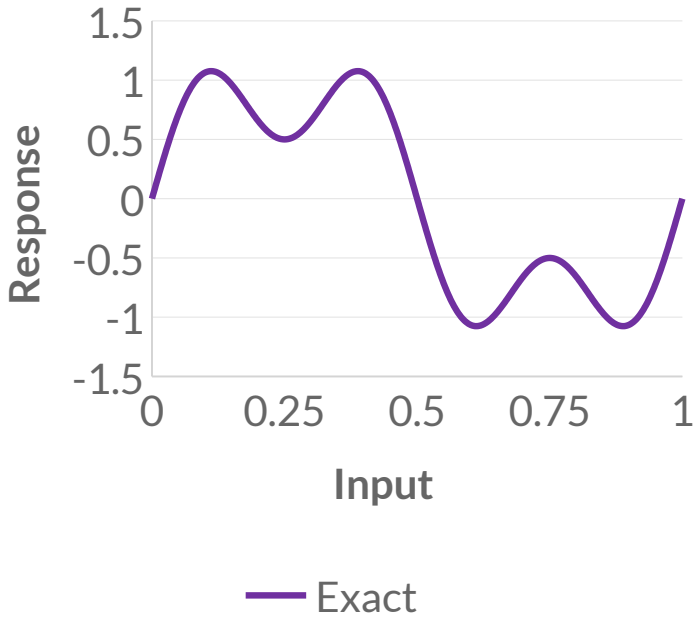
Same target, different building blocks: local windows or sinusoids.

8 local windows

32 local windows



2 sinusoidal units

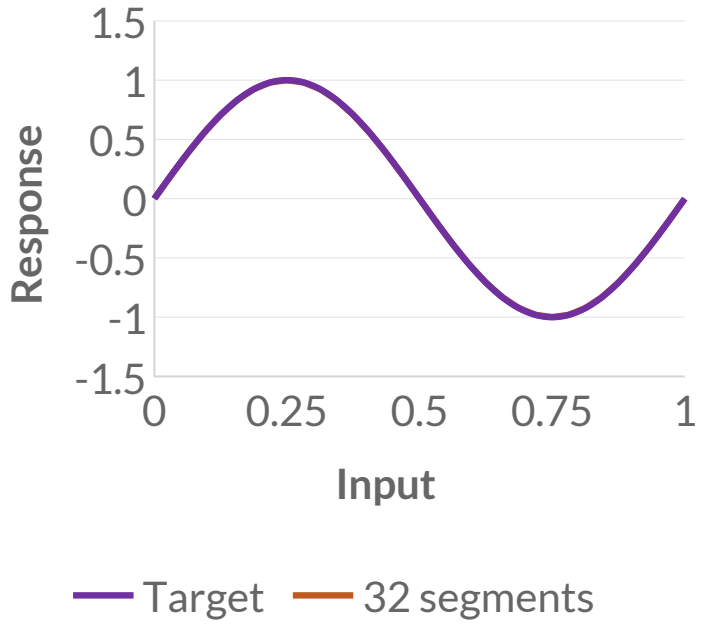


$$f(x) = \sin(2\pi x) + \tfrac{1}{2}\sin(6\pi x)$$

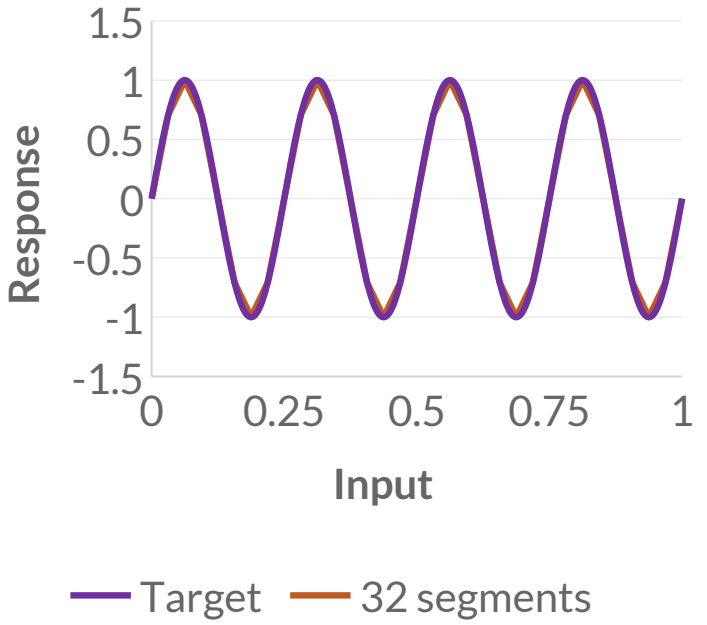
Approximation on a fixed grid

Straight segments join values sampled at equal intervals.

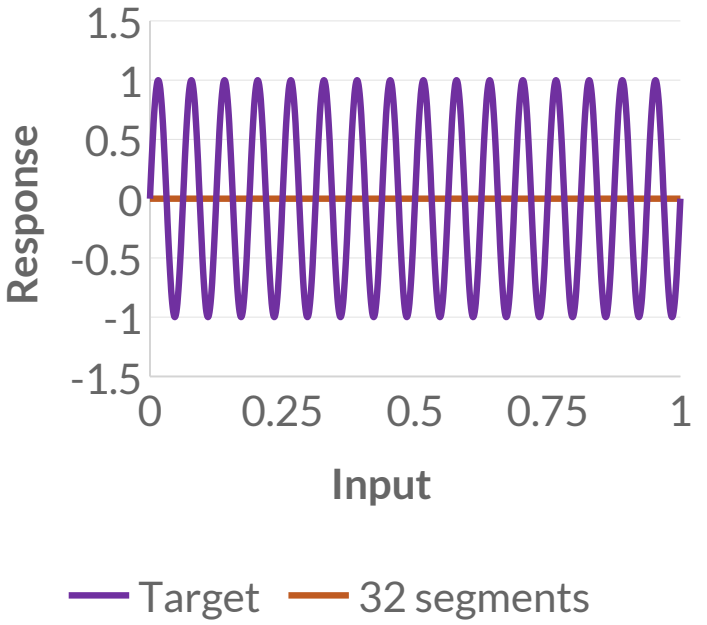
1 cycle



4 cycles



16 cycles



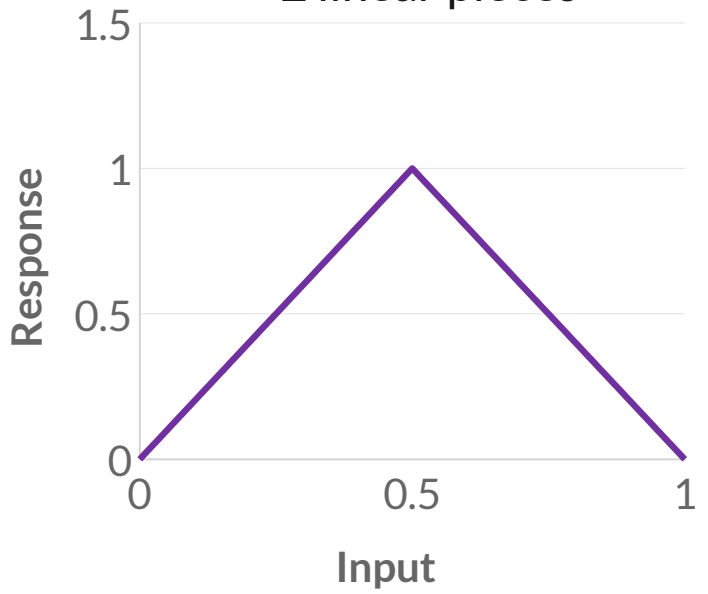
$$f(x) = \sin(2\pi\omega x), \qquad 32 \text{ linear segments in every panel}$$

Repeated composition

Each stage feeds its output through another copy of the same function.

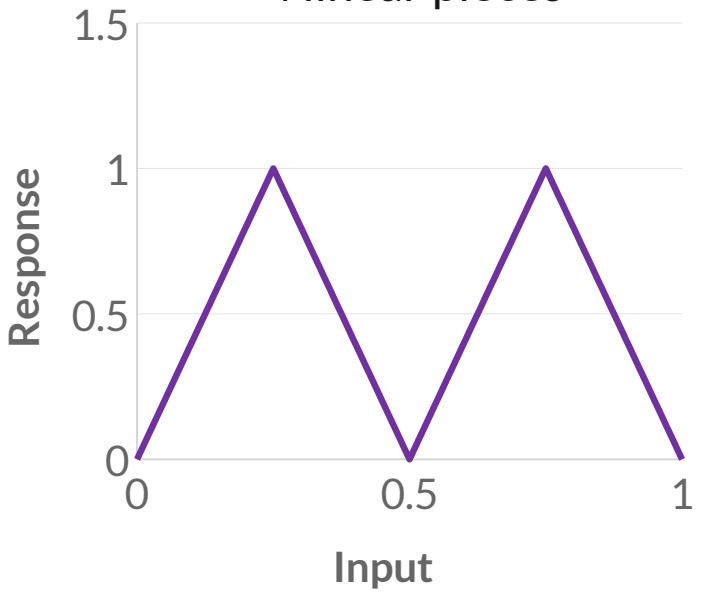
g(x)

2 linear pieces



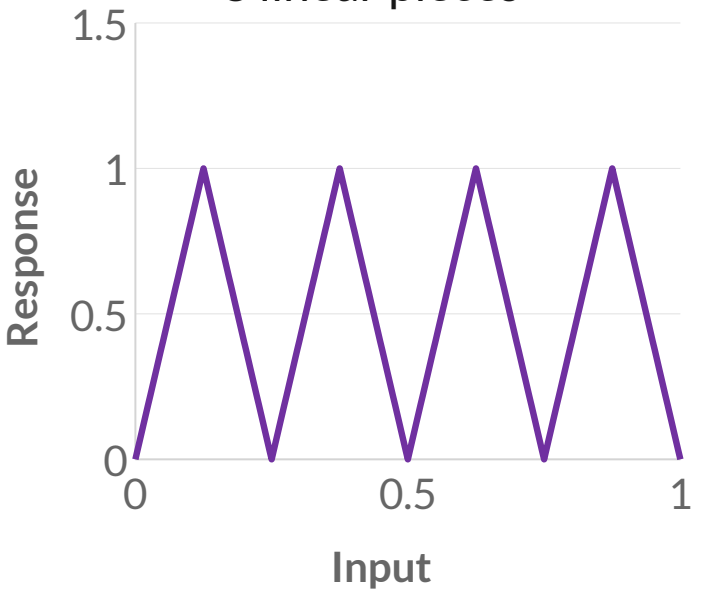
g(g(x))

4 linear pieces



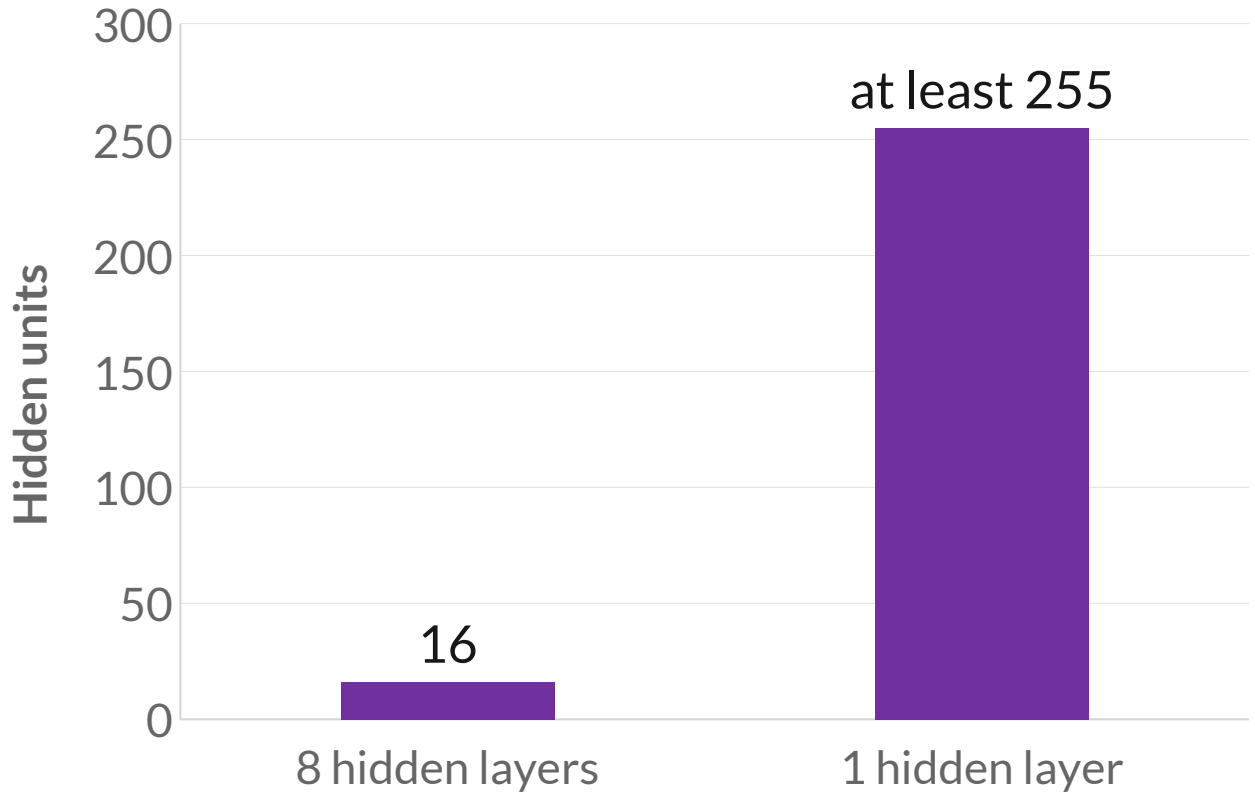
g(g(g(x)))

8 linear pieces



$$g(x) = 2\,\mathrm{ReLU}(x) - 4\,\mathrm{ReLU}(x - \tfrac{1}{2}), \quad x \in [0, 1]$$

Representing the same folded function



Eight repetitions of g

Depth 8, width 2

16 hidden units
Two units per fold

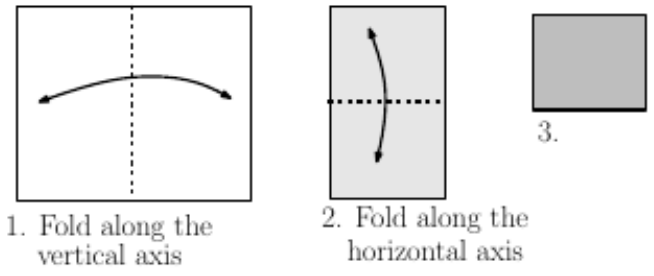
One hidden layer

At least 255 hidden units
One unit adds at most one bend

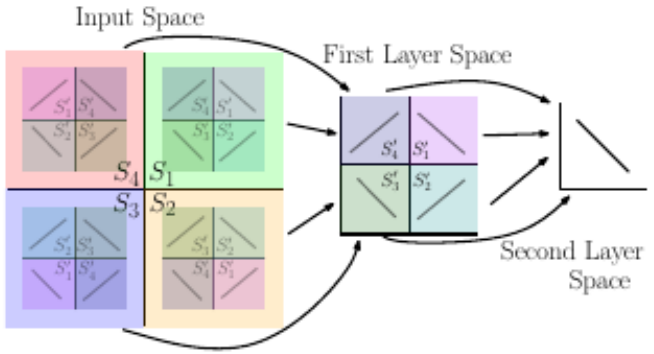
Each fold doubles the pieces: eight folds give 256 pieces.

Matching this exact function with one hidden layer needs at least 255 bends.

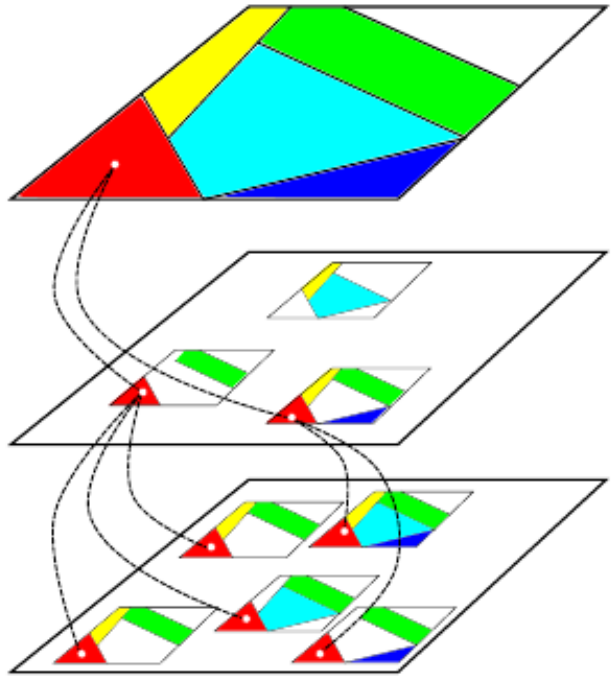
Depth and input-space folding



(a)



(b)



(c)

Different input regions can map to the same hidden representation.

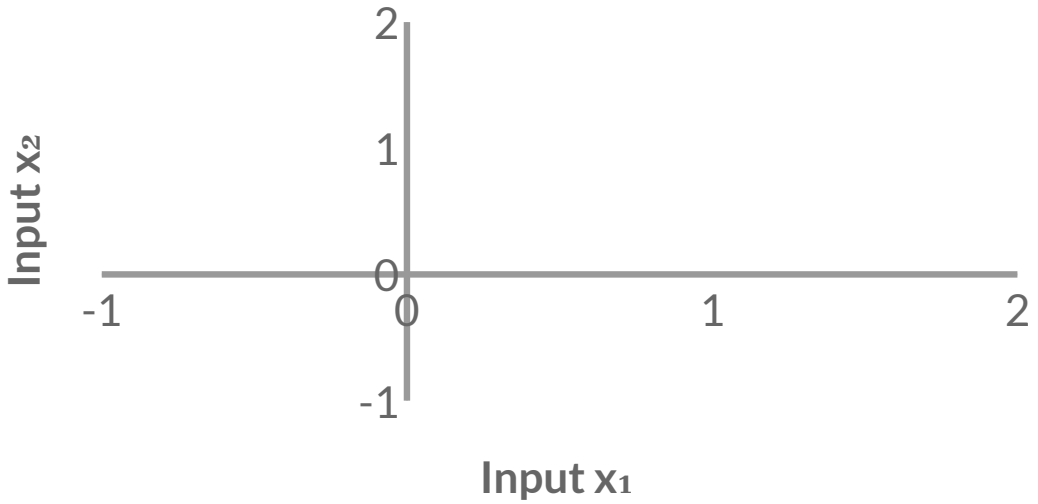
A later partition then appears in several regions of the original input space.

Composition can multiply the number of linear pieces.

Refining regions through depth

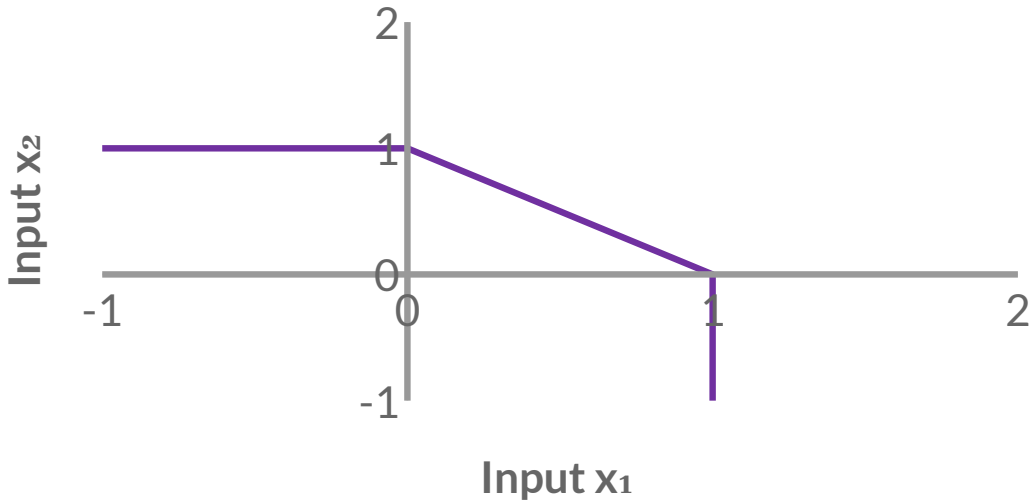
A later ReLU can split regions formed by earlier units.

First hidden layer



$$h_1 = \mathrm{ReLU}(x_1), \quad h_2 = \mathrm{ReLU}(x_2)$$

Add one ReLU unit

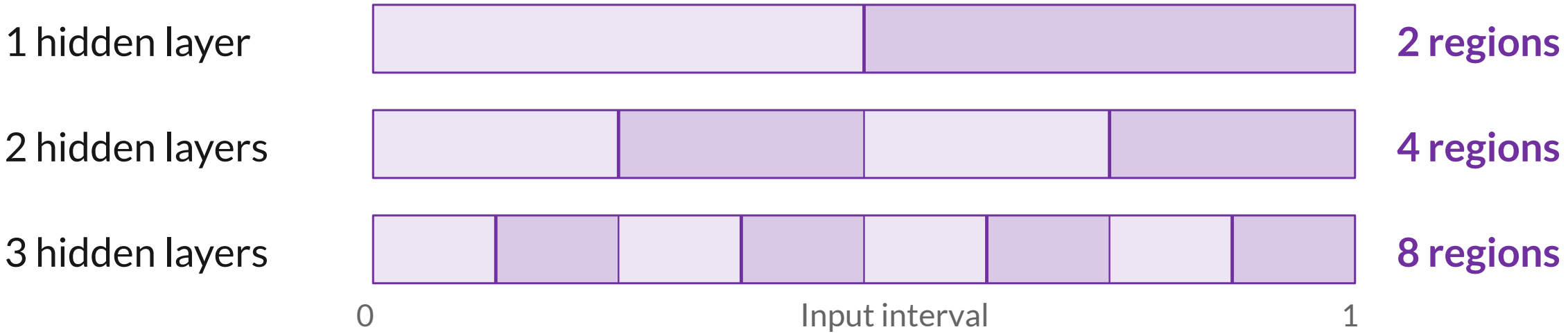


$$u = \mathrm{ReLU}(h_1 + h_2 - 1)$$

The new boundary bends across earlier regions.
Repeating these splits can create many regions.

Exponential growth with depth

For the repeated tent function, keep two ReLU units per hidden layer.

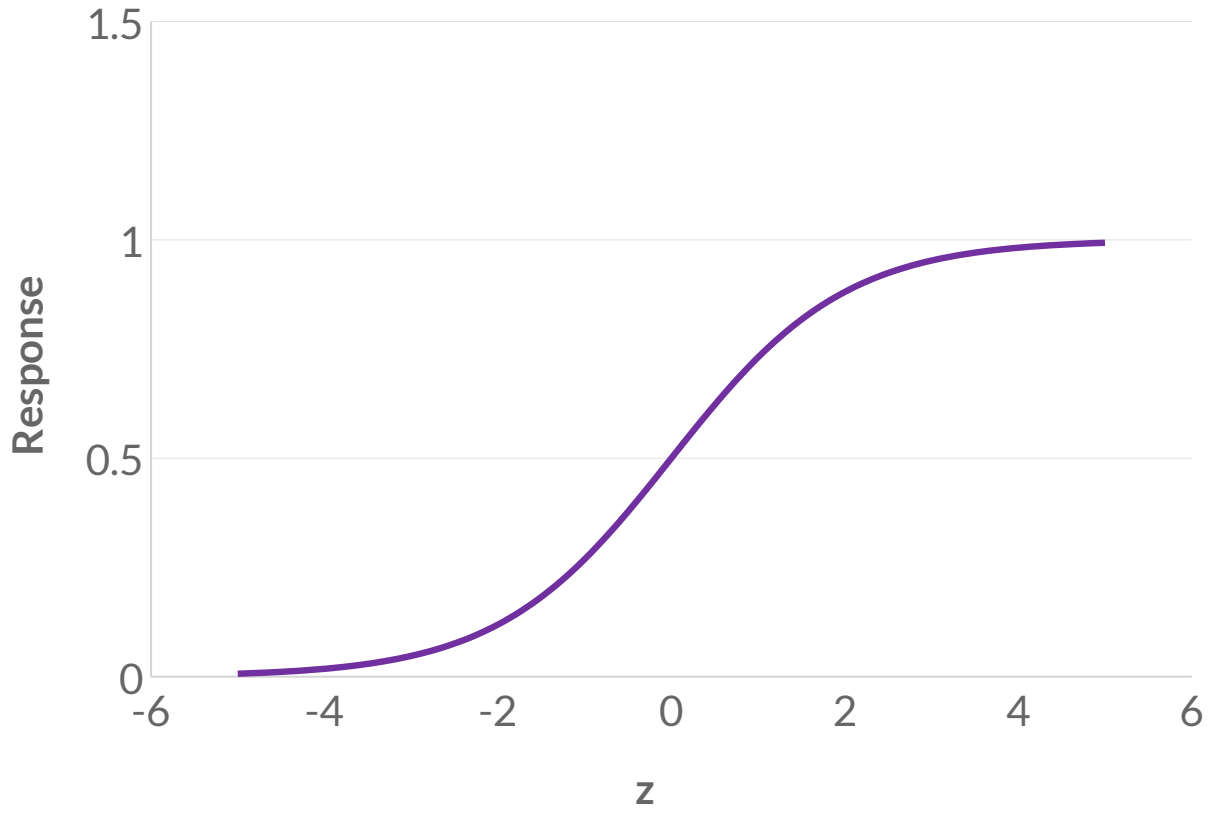


Each piece's output spans 0 to 1. The next fold splits it in two.

$$\underbrace{2 \times 2 \times \cdots \times 2}_{L \text{ hidden layers}} = 2^L \text{ linear regions}$$

General case: a rough count that may double-count regions.

Sigmoid



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

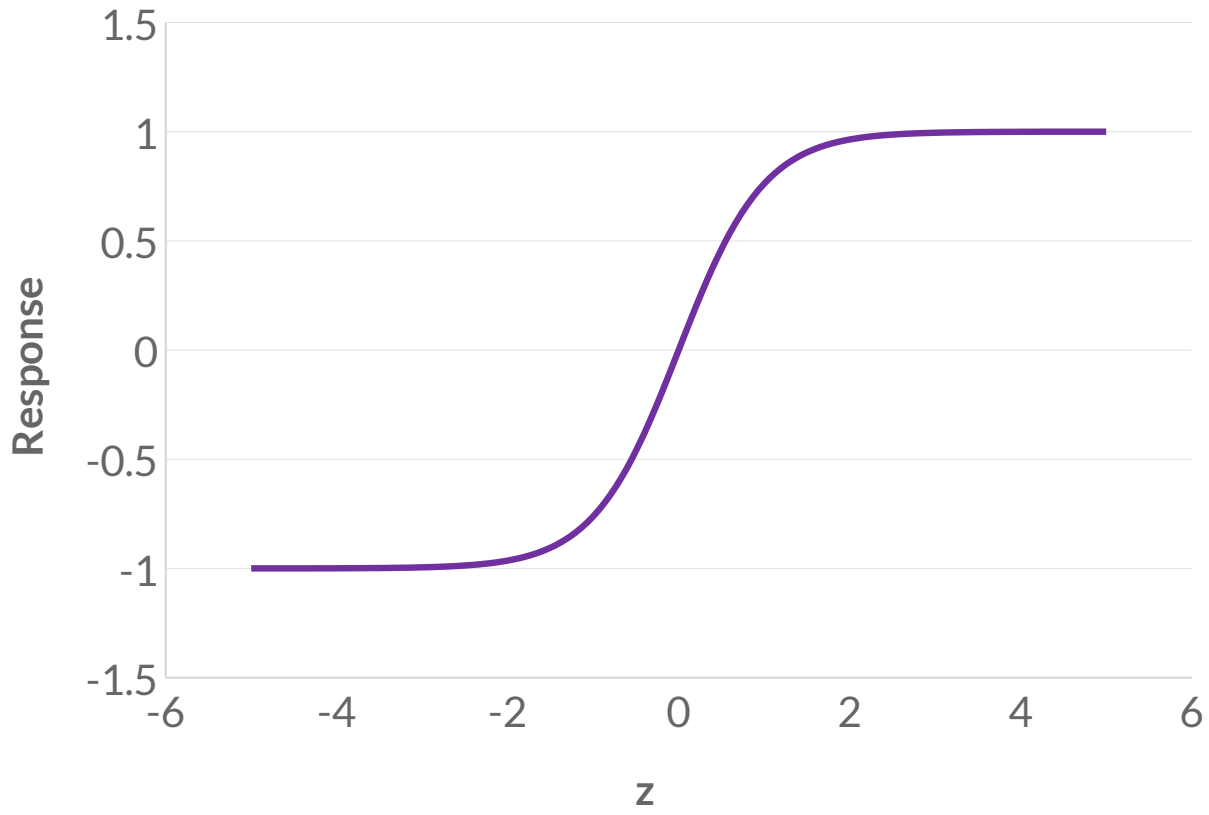
$$\sigma'(z) = \sigma(z)(1 - \sigma(z))$$

- Smooth, with range (0, 1)
- Saturation gives small derivatives in both tails

Used for

Binary probabilities and gates

Tanh



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

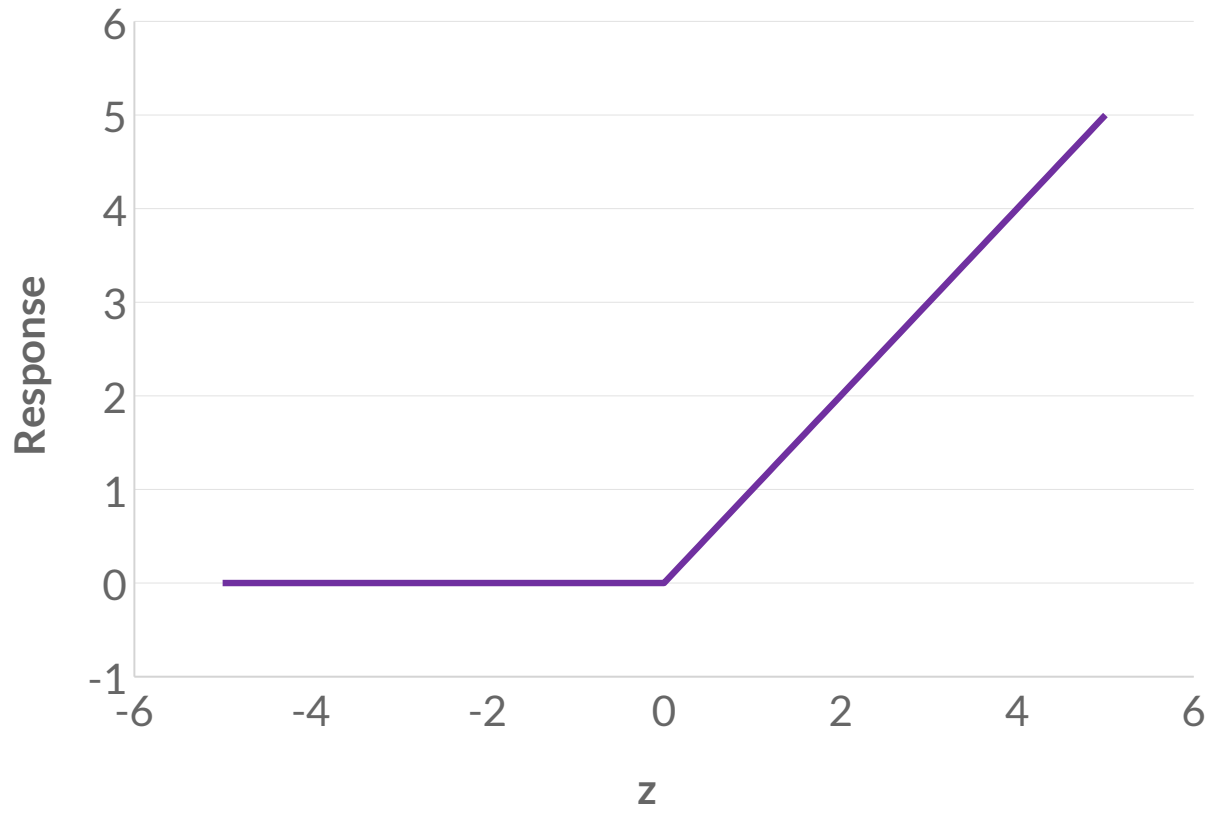
$$\tanh'(z) = 1 - \tanh^2(z)$$

- Signed range (−1, 1)
- Zero-centered, but saturating

Used for

Bounded outputs, recurrent states

ReLU



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

$$\phi'(z) = \begin{cases} 0, & z < 0 \\ 1, & z > 0 \end{cases}$$

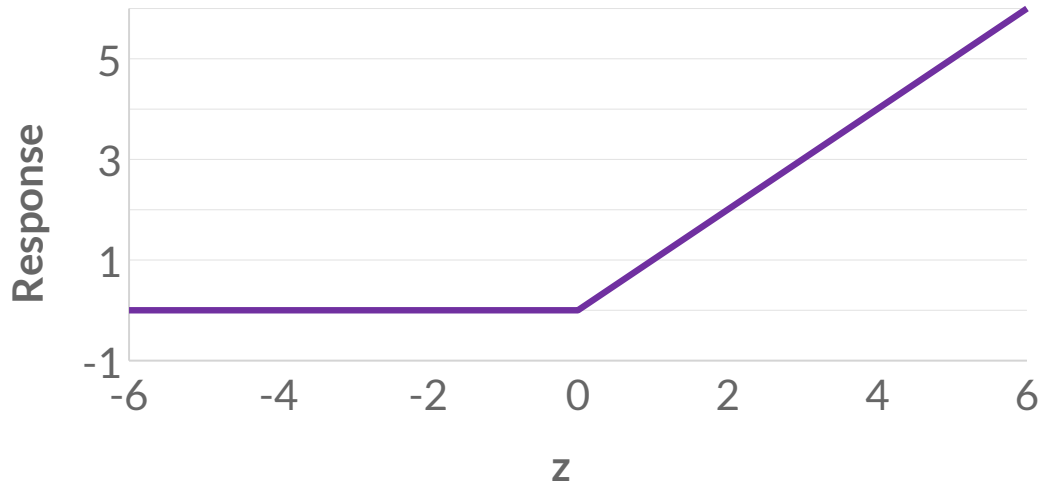
- Cheap; no saturation on the positive branch
- Zero output and local gradient for negative inputs

Used for

MLPs and convolutional networks

ReLU and Leaky ReLU

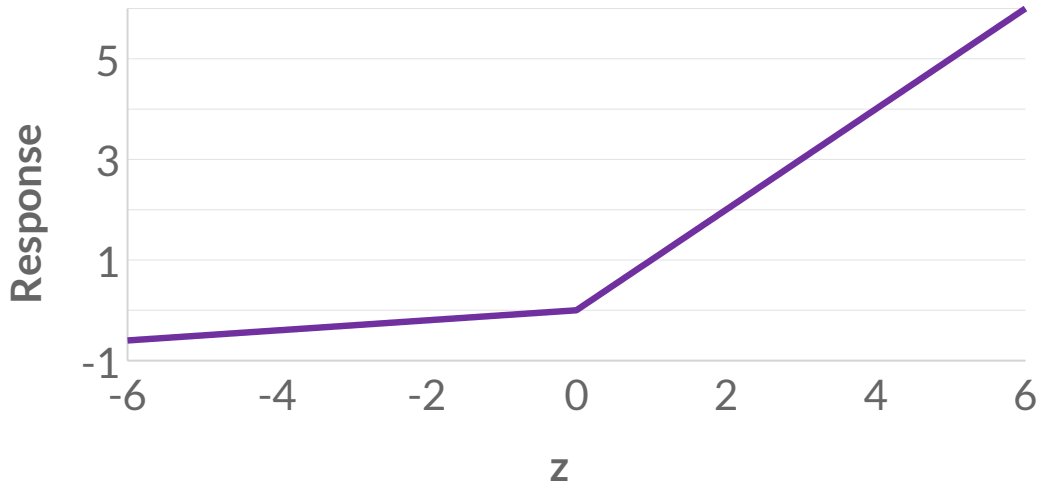
ReLU



$$\max(0, z)$$

- Zero response for negative inputs
- A unit may stay inactive across the data

Leaky ReLU

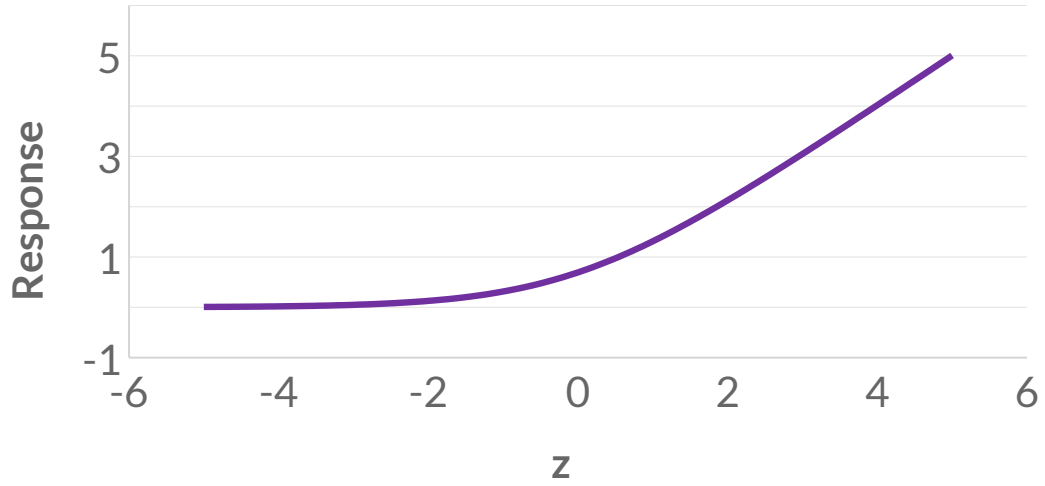


$$\max(\alpha z, z), \quad 0 < \alpha < 1$$

- Nonzero slope for negative inputs
- PReLU learns this slope

Softplus and ELU

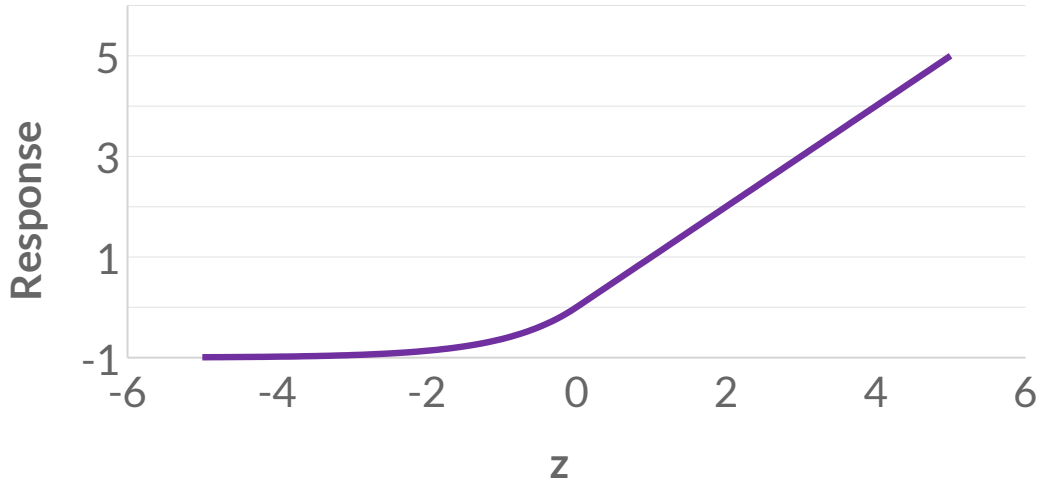
Softplus



$$\log(1 + e^z)$$

- Smooth and strictly positive
- Useful for positive scales or rates

ELU, $\alpha = 1$

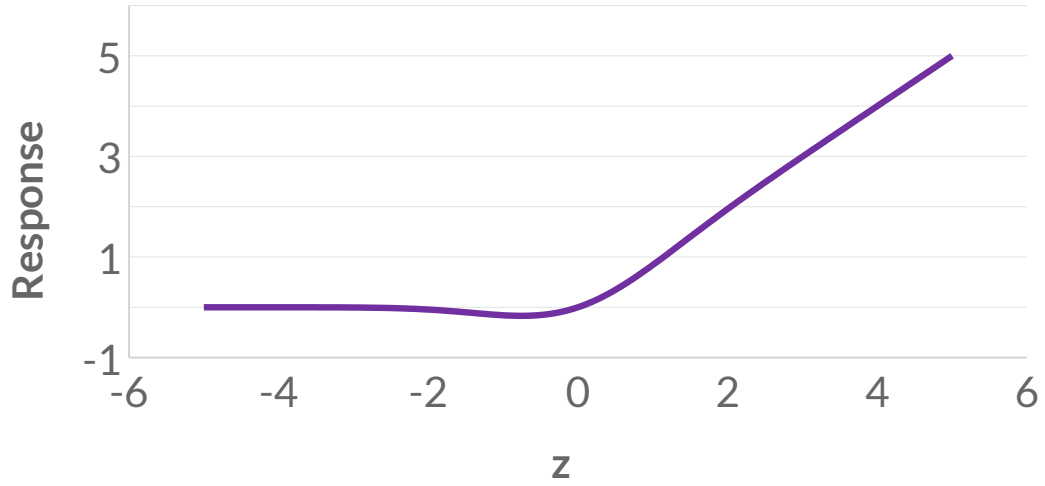


$$\begin{cases} z, & z > 0 \\ e^z - 1, & z \leq 0 \end{cases}$$

- Negative values; positive linear branch
- Alternative hidden activation

GELU and SiLU

GELU

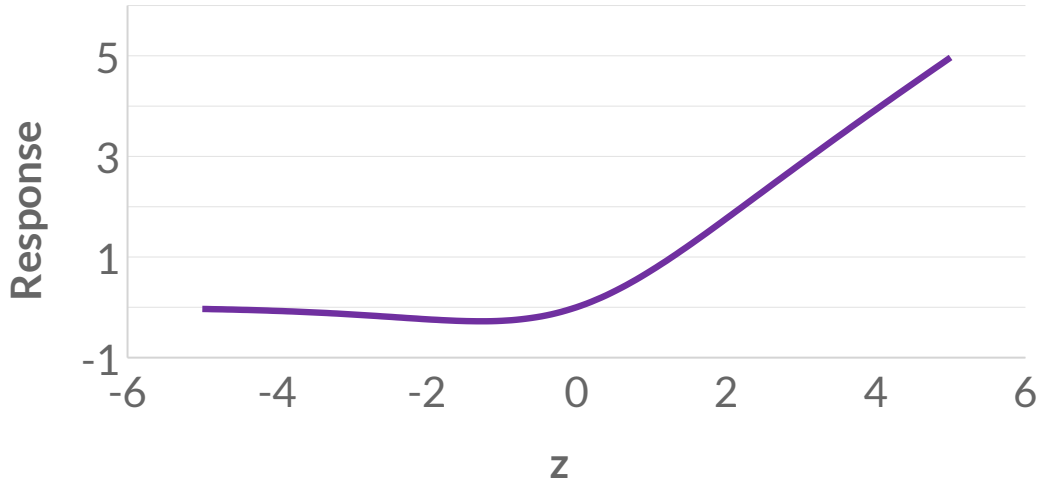


$$\mathrm{GELU}(z) = z\Phi(z)$$

Φ(z): probability a standard normal value is ≤ z

- Smooth weighting by a normal CDF
- Used in BERT, a language model

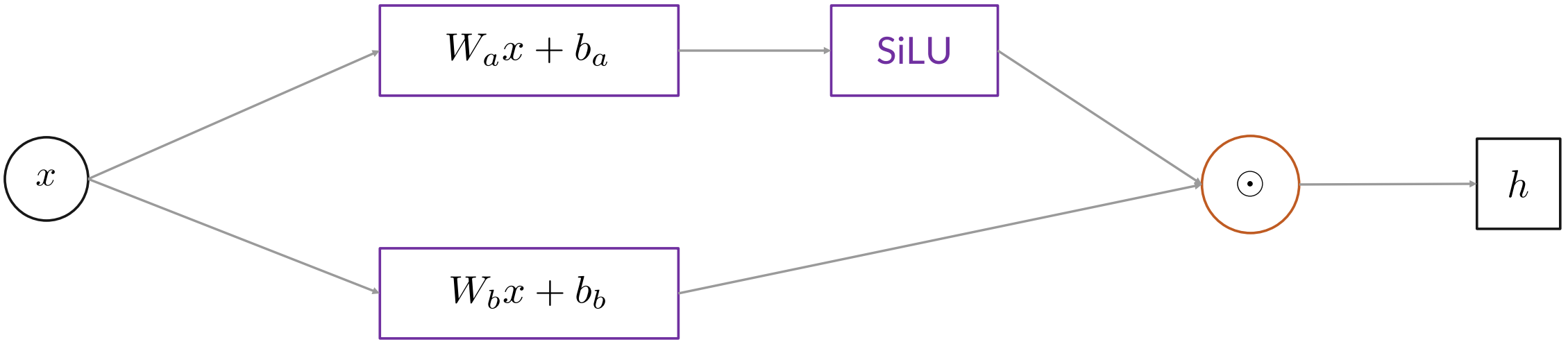
SiLU / Swish with $\beta = 1$



$$\mathrm{SiLU}(z) = z\sigma(z)$$

- Smooth weighting by a sigmoid
- Used within SwiGLU layers

SwiGLU: gated activation



One branch scales the other element by element.

$$h = \mathrm{SiLU}(W_a x + b_a) \odot (W_b x + b_b)$$

$$y = W_o h + b_o$$

Lecture outline

01 Neurons and representations

02 Capacity and activations

03 Predictions and objectives

Match the output to the task

Regression

$$\widehat{y} \in \mathbb{R}$$

- Predict a numerical quantity
- Use a numerical error

Classification

$$p_k = P_\theta(y = k \mid x)$$

- Predict probabilities of categories
- Use the observed category

Recap: multiclass classification

One raw score (logit) for each mutually exclusive class.

$$s = W_{\text{out}} h + b_{\text{out}} \in \mathbb{R}^K$$

$$\widehat{y} = \arg\max_k s_k$$

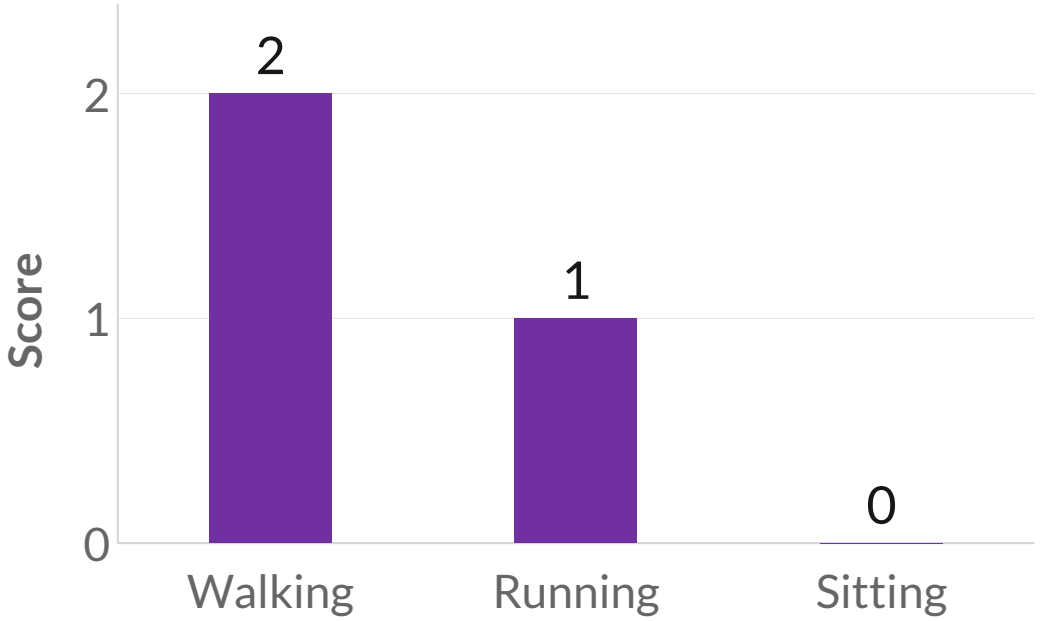
argmax chooses the class with the largest score.

$$\ell_{0/1}(\widehat{y}, y) = \mathbf{1}\{\widehat{y} \neq y\}$$

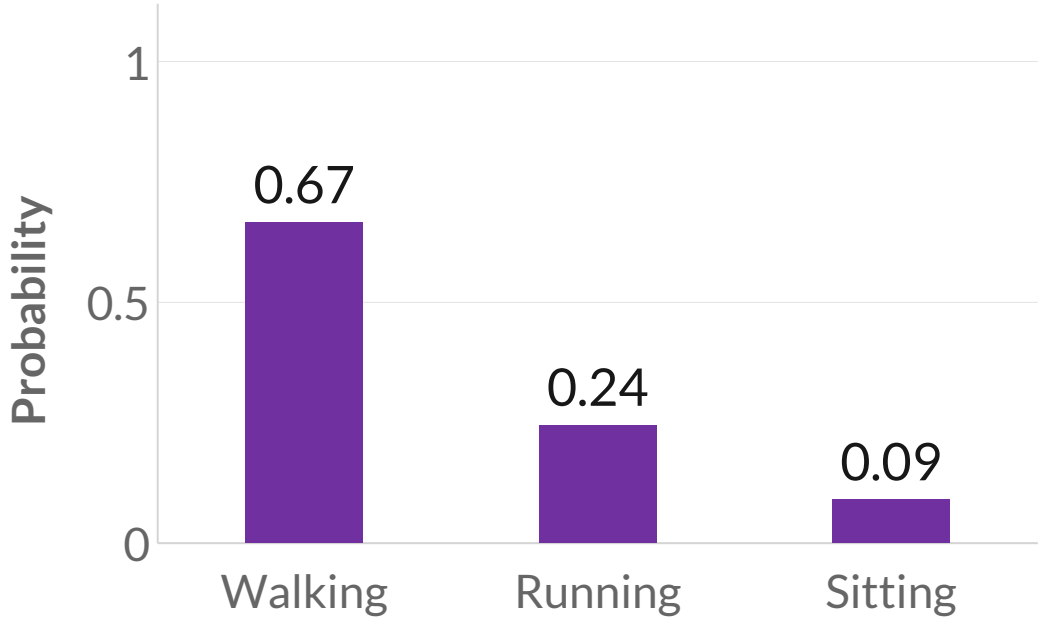
0/1 loss
Prediction $\hat{y}$, true label y
1 if different, 0 if equal

Softmax converts logits into probabilities

Logits



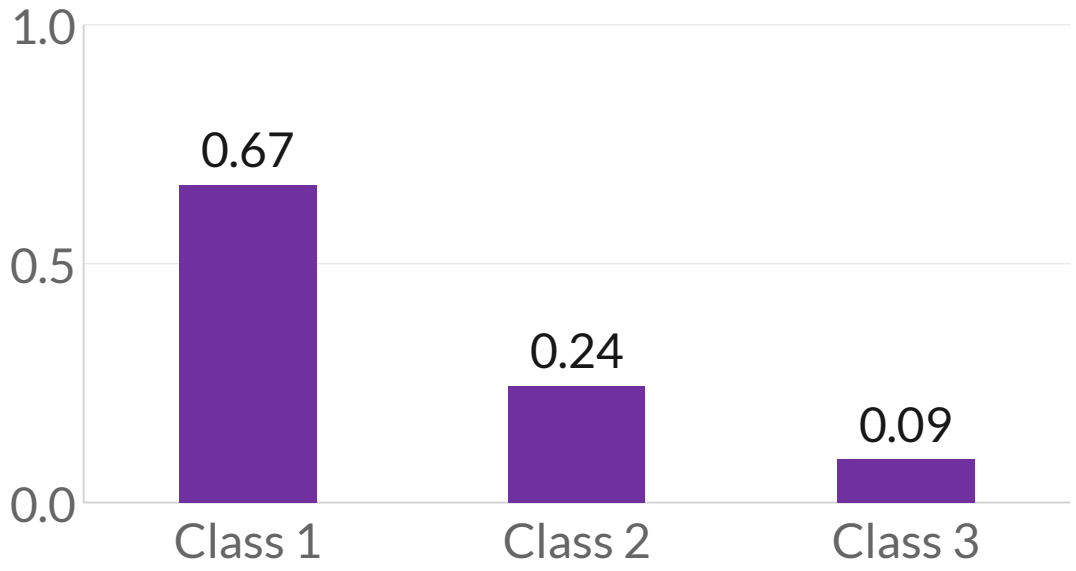
Probabilities



$$p_k = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

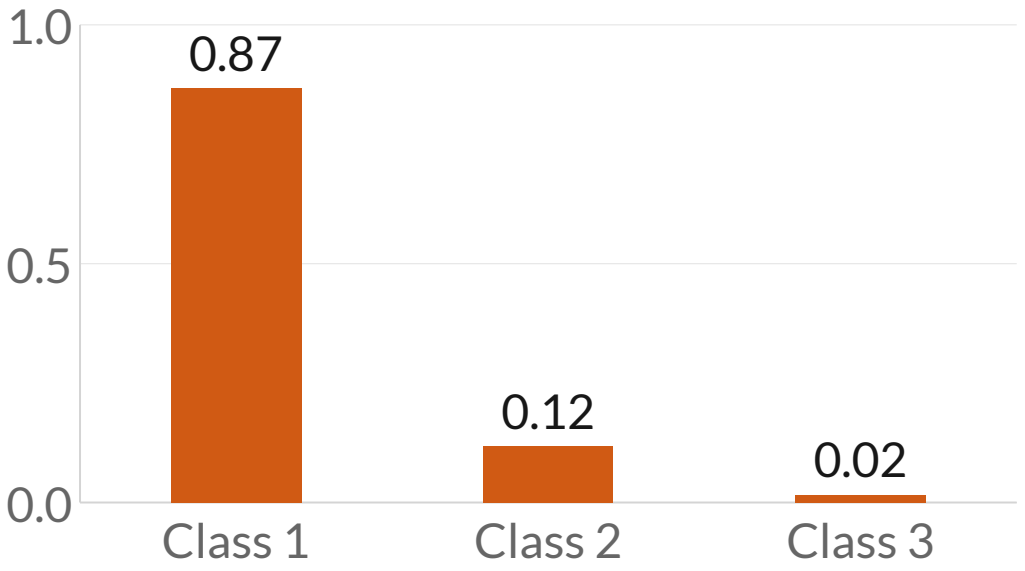
Logit differences and probabilities

Logits: (2, 1, 0)



$$\frac{p_i}{p_j} = e^{s_i - s_j}$$

Logits: (4, 2, 0)



$$\log \frac{p_i}{p_j} = s_i - s_j$$

The winning class is unchanged; the predicted probabilities are different.

Surrogate loss for classification

Prediction and evaluation

$$\widehat{y} = \arg\max_k s_k, \qquad \ell_{0-1} = \mathbf{1}\{\widehat{y} \neq y\}$$

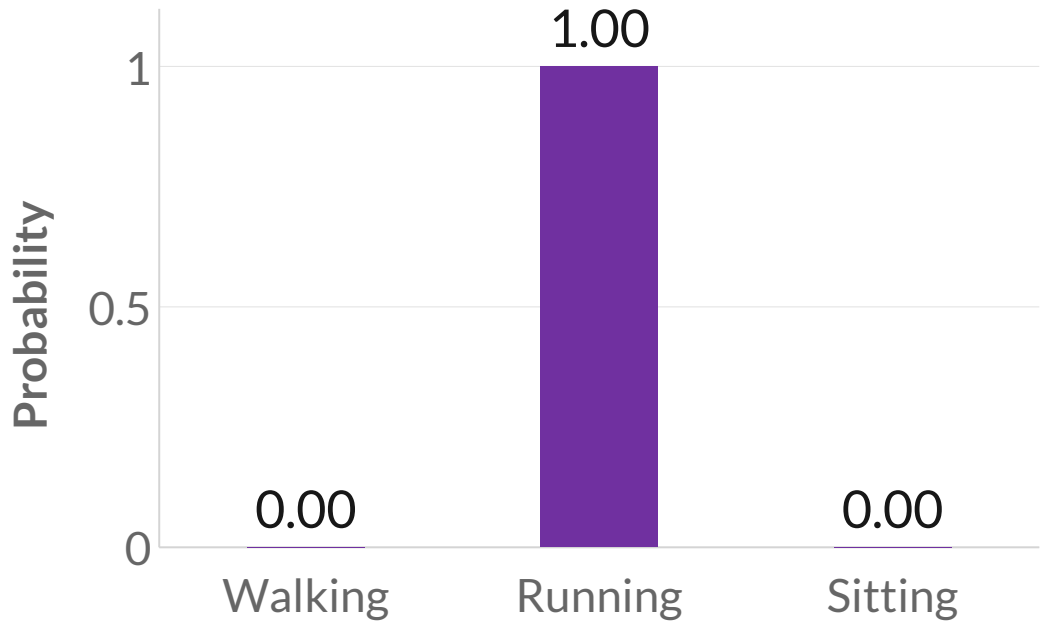
Training

Improving the correct-class score may leave 0–1 loss unchanged.

Cross entropy is a smooth training proxy (a surrogate loss).

Labels as target distributions

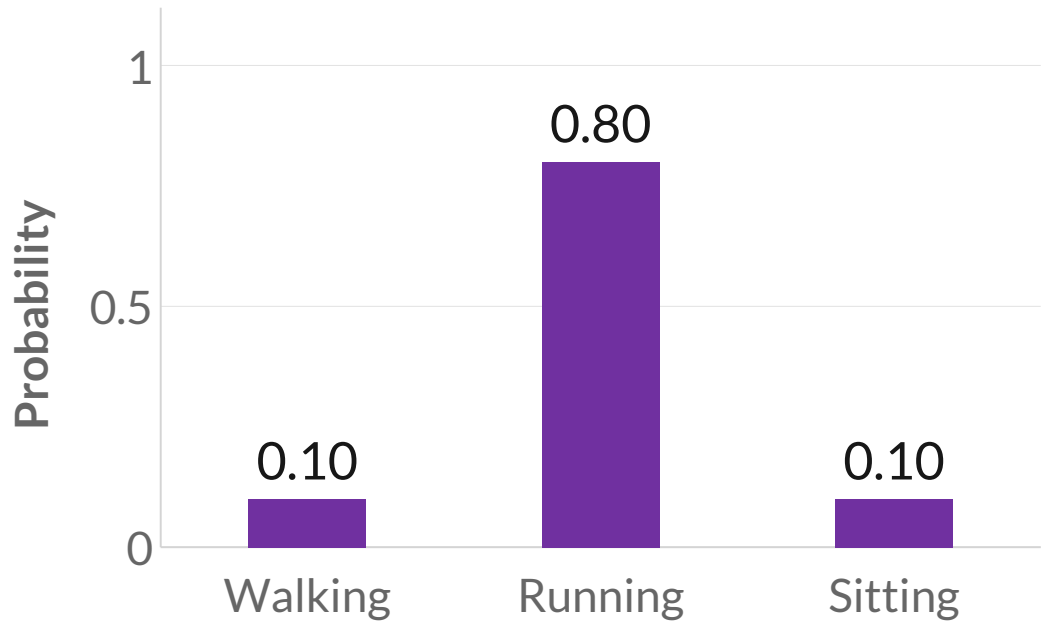
One-hot target



All target probability is on class y.

Both target distributions sum to one.

Soft target



Target probability is shared across classes.

Maximum likelihood and negative log likelihood

$$\max_{\theta} \prod_{i=1}^{N} p_\theta(y_i \mid x_i)$$

$$\iff \quad \min_{\theta} -\sum_{i=1}^{N} \log p_\theta(y_i \mid x_i)$$

- For independent examples, ∏ multiplies model probabilities of their labels.
- Logarithms turn this product into a sum.

Cross entropy loss

For a correct class label y and predicted class probabilities p:

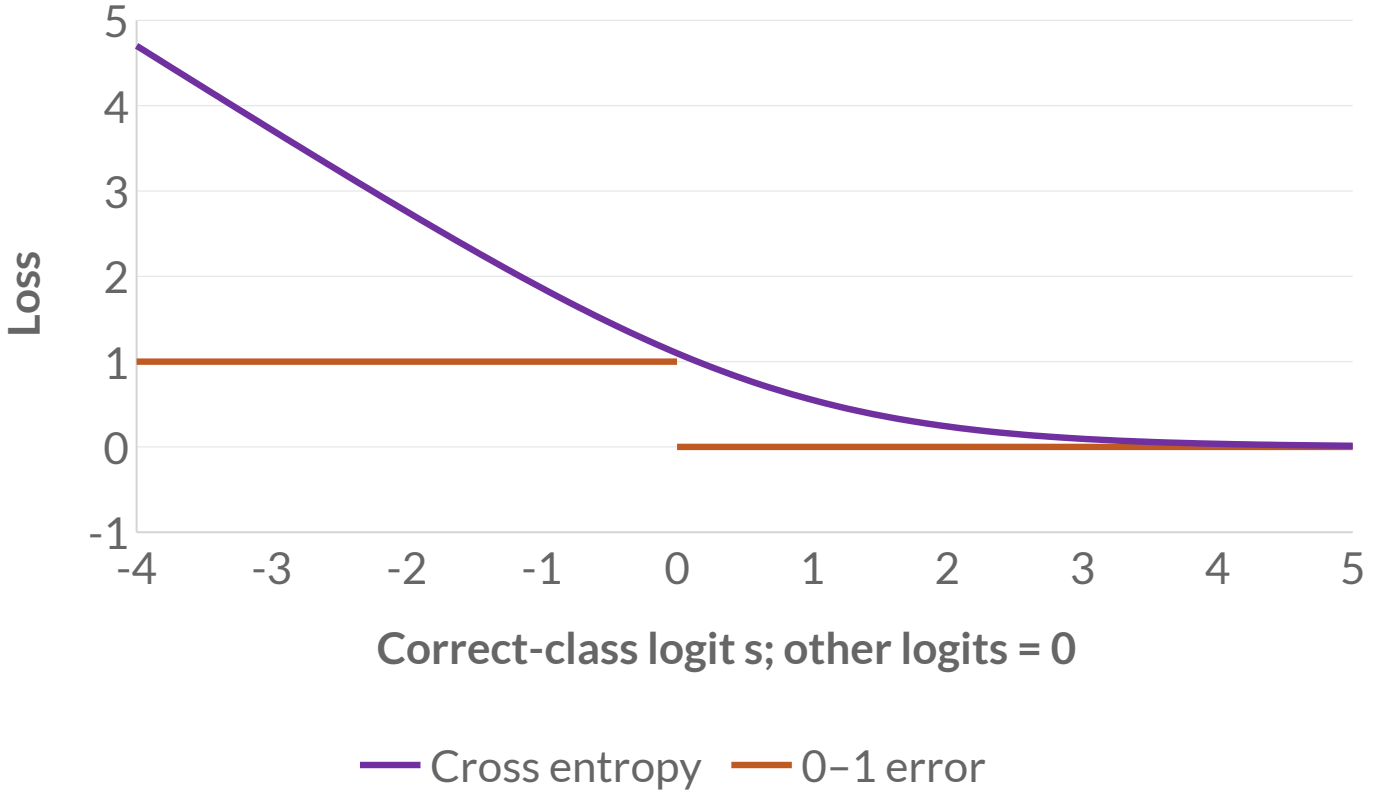
$$\ell_{\mathrm{CE}}(y, p) = -\log p_y$$

For a target distribution q over the classes:

$$H(q, p) = -\sum_{k=1}^{K} q_k \log p_k$$

A one-hot label places all target probability on class y.

Cross entropy and the correct-class logit



$$\ell_{\mathrm{CE}} = -s_y + \log \sum_j e^{s_j}$$

Increasing the correct logit relative to the others raises its probability and lowers the loss.

The loss keeps decreasing after the predicted class becomes correct.

Numerical overflow in softmax

Three logits: 1000, 999, 998. Direct exponentiation in 32-bit floating point gives:

$$(e^{1000}, e^{999}, e^{998}) \longrightarrow (\text{inf}, \text{inf}, \text{inf})$$

$$\frac{\text{inf}}{\text{inf}} \longrightarrow \text{NaN}$$

The intended probabilities are finite: approximately 0.665, 0.245, 0.090.

inf: overflow beyond the representable range · NaN: undefined numerical result

Stable softmax by shifting logits

$$(1000, 999, 998) - 1000 = (0, -1, -2)$$

$$(e^0, e^{-1}, e^{-2}) \approx (1, 0.368, 0.135)$$

$$\frac{e^{s_k - c}}{\sum_j e^{s_j - c}} = \frac{e^{s_k}}{\sum_j e^{s_j}}, \qquad c = \max_j s_j$$

Every exponential is at most 1; the denominator is at least 1.

Stable cross entropy

Compute log probabilities directly from the logits.

$$\log p_k = (s_k - m) - \log \sum_j e^{s_j - m}, \qquad m = \max_j s_j$$

$$\ell_{\mathrm{CE}} = -\log p_y$$

PyTorch combines these operations in its cross-entropy loss.

Cross entropy loss in PyTorch

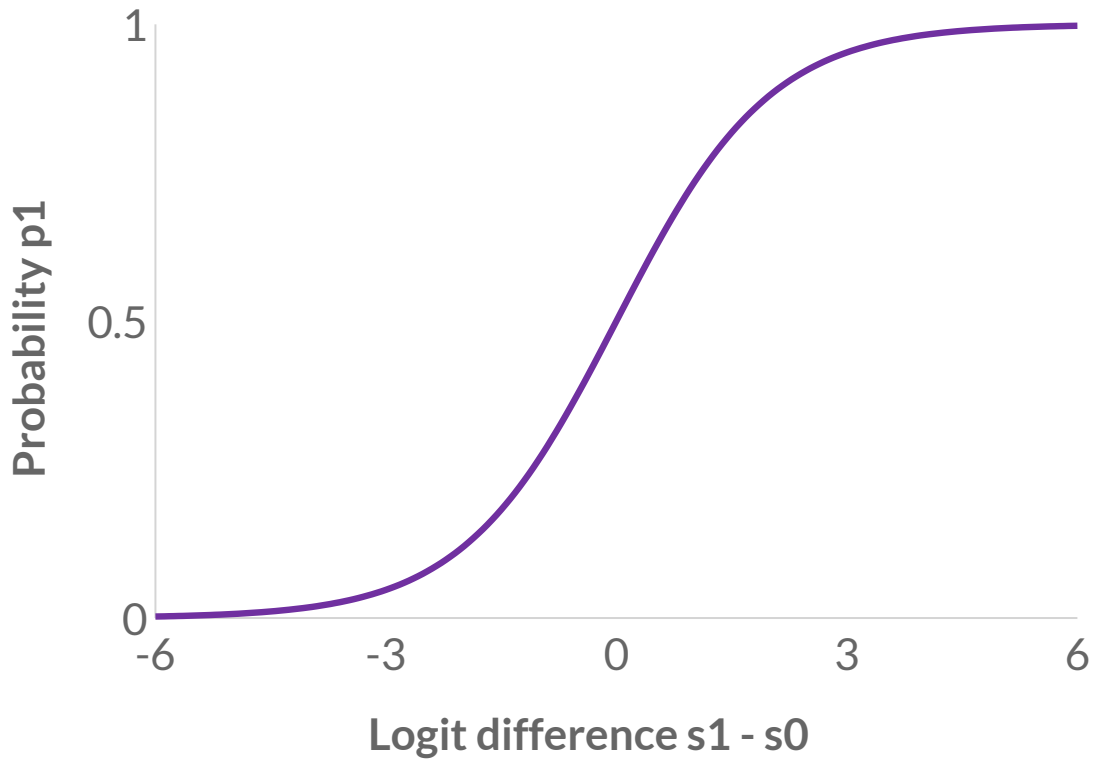
```
import torch
import torch.nn.functional as F

logits = torch.tensor([[1000., 999., 998.]])
labels = torch.tensor([0])  # the first class

F.cross_entropy(logits, labels)
# tensor(0.4076)
```

cross_entropy takes raw logits and includes log-softmax and negative log likelihood.

Two-class softmax and sigmoid



Two mutually exclusive classes

$$p_1 = \frac{e^{s_1}}{e^{s_0} + e^{s_1}}$$

$$= \frac{1}{1 + e^{-(s_1 - s_0)}}$$

One logit can store the difference.

$$z = s_1 - s_0, \qquad p_1 = \sigma(z), \quad p_0 = 1 - p_1$$

Binary cross entropy loss

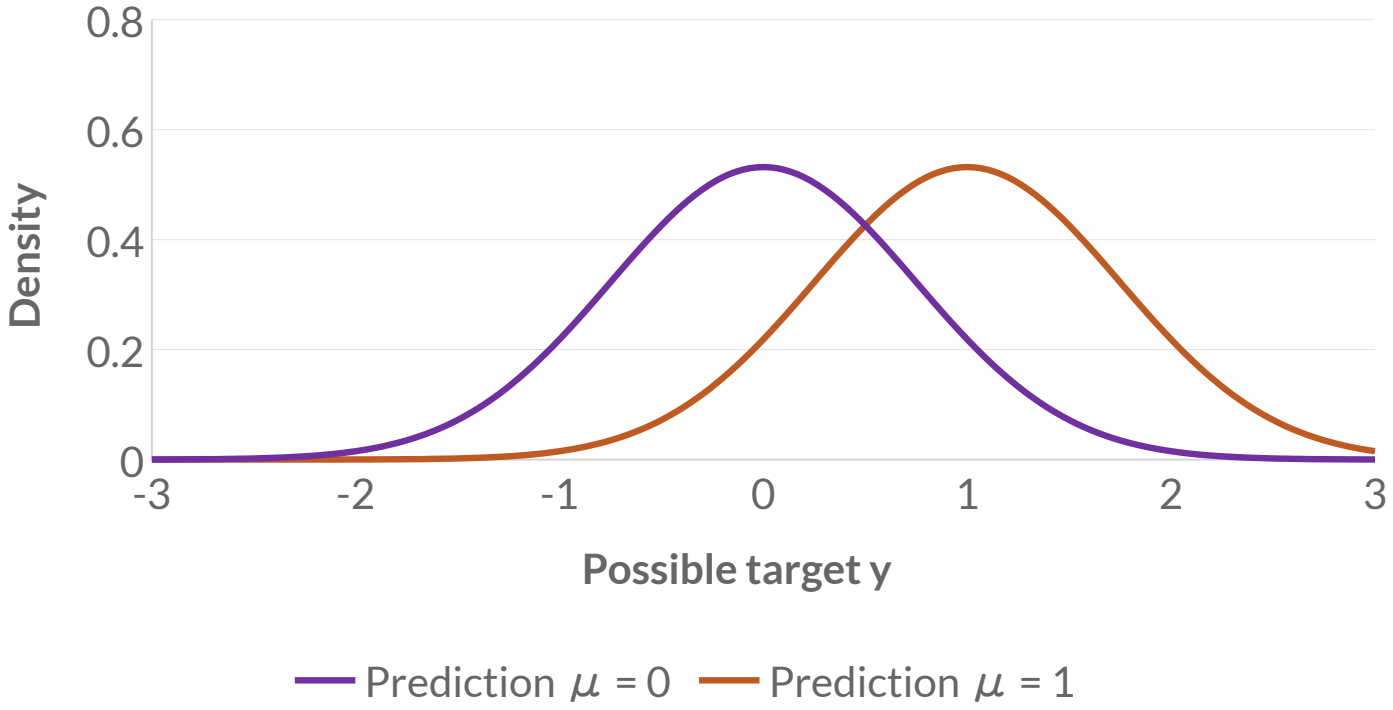
One logit z describes the probability p of outcome 1.

$$p = \sigma(z), \qquad P(y) = p^y (1-p)^{1-y}, \quad y \in \{0, 1\}$$

$$\ell_{\mathrm{BCE}} = -y \log p - (1-y) \log(1-p)$$

Use the probability of the observed outcome: p for label 1, or 1 − p for label 0.

Squared error and a Gaussian model



$$y \mid x \sim \mathcal{N}(\mu_\theta(x), \sigma^2)$$

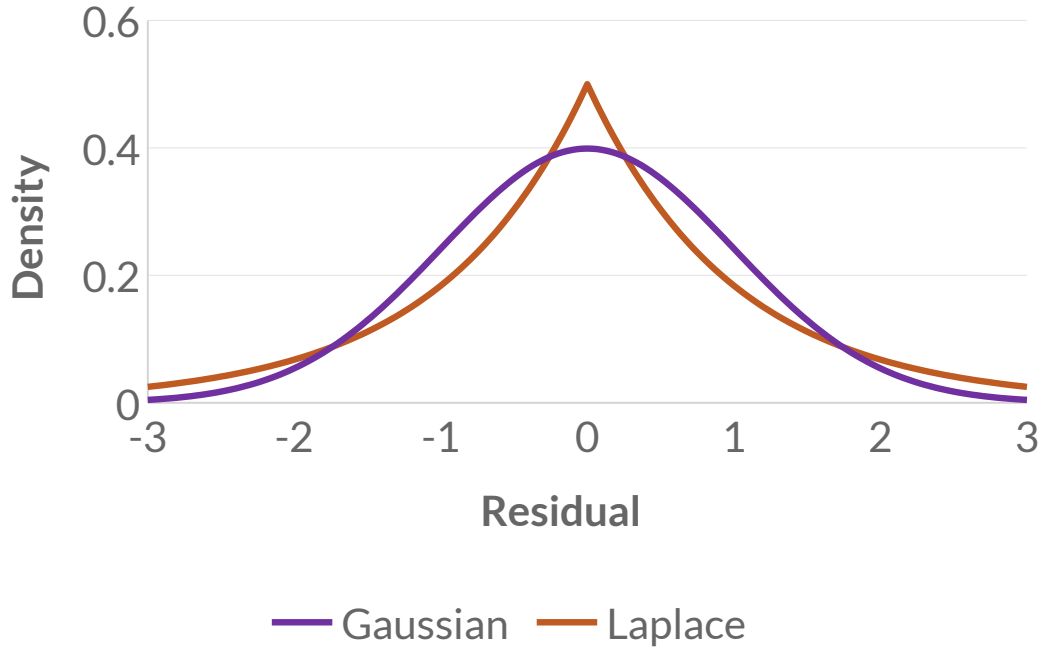
- Assume a fixed variance.
- The prediction is the conditional mean.
- C is independent of the prediction.

$$-\log p_\theta(y \mid x) = \frac{(y - \mu_\theta(x))^2}{2\sigma^2} + C$$

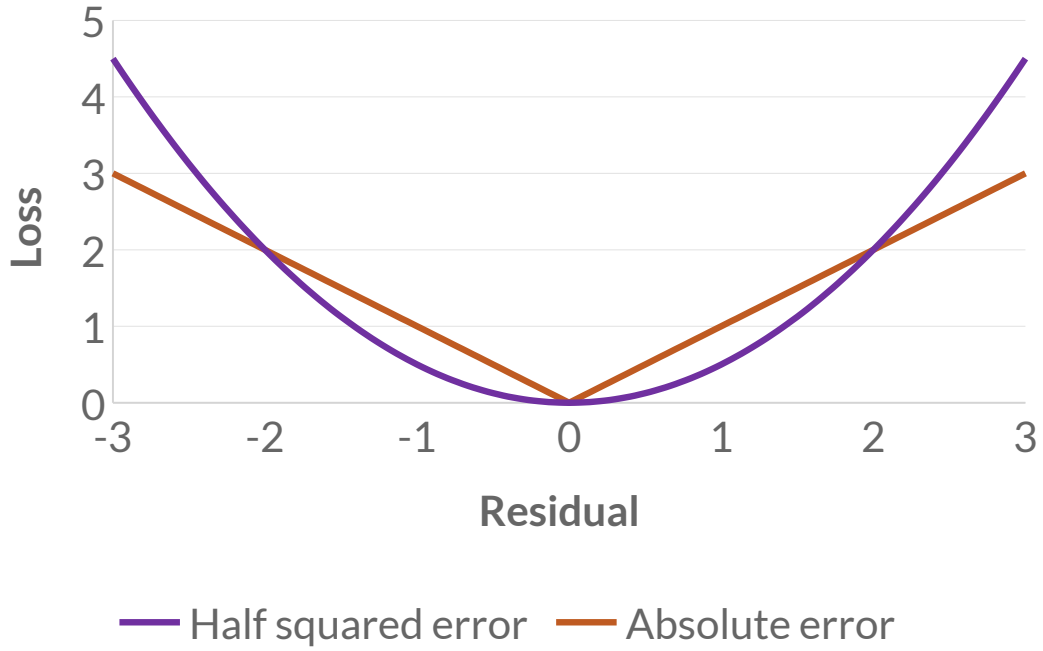
L1 loss and a Laplace model

The scale parameter b controls the width of the Laplace distribution.

Error distributions



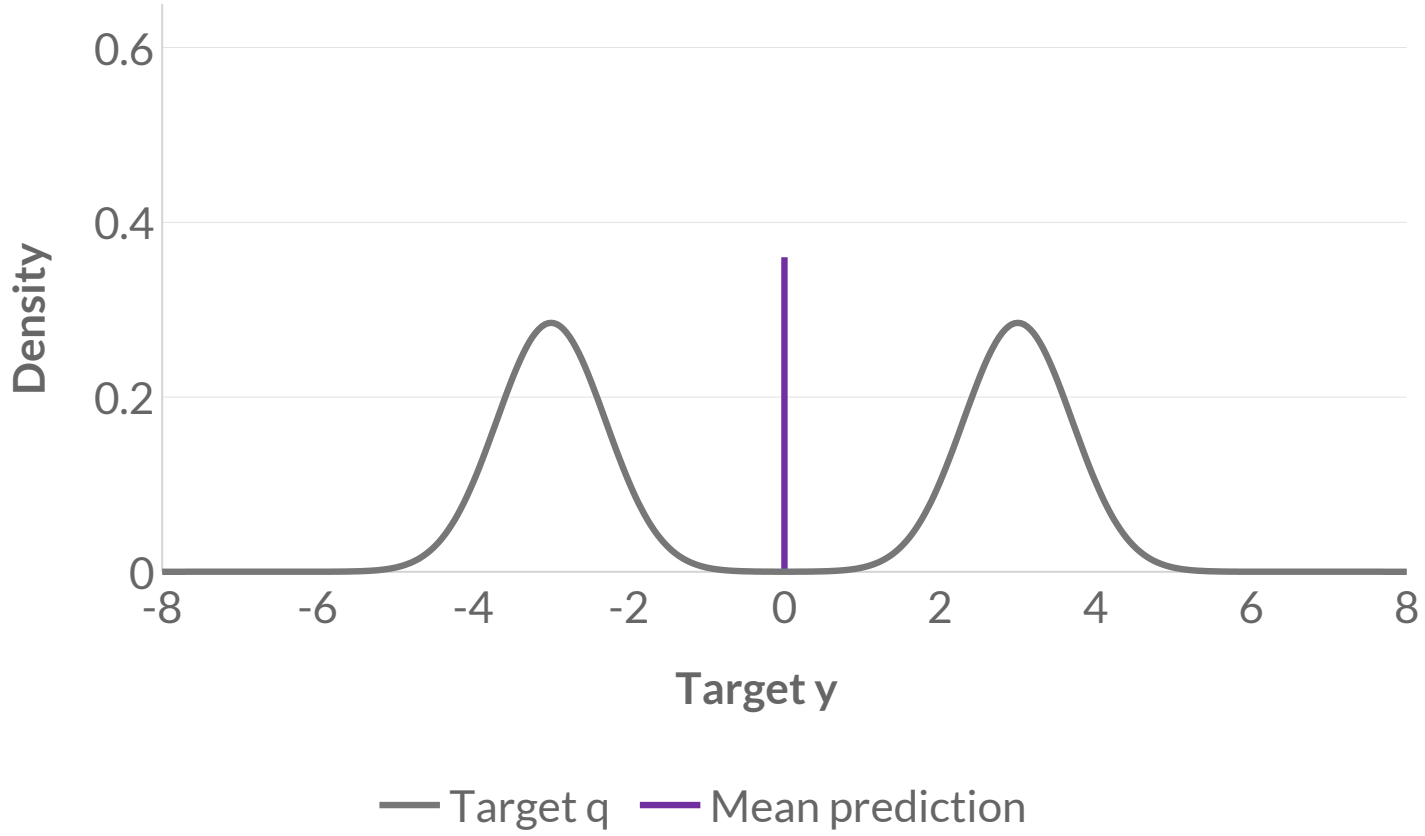
Loss as the residual grows



$$p(y \mid \mu) = \frac{1}{2b} e^{-|y-\mu|/b}$$

$$-\log p(y \mid \mu) = \frac{|y-\mu|}{b} + C$$

Squared error and ambiguous outcomes



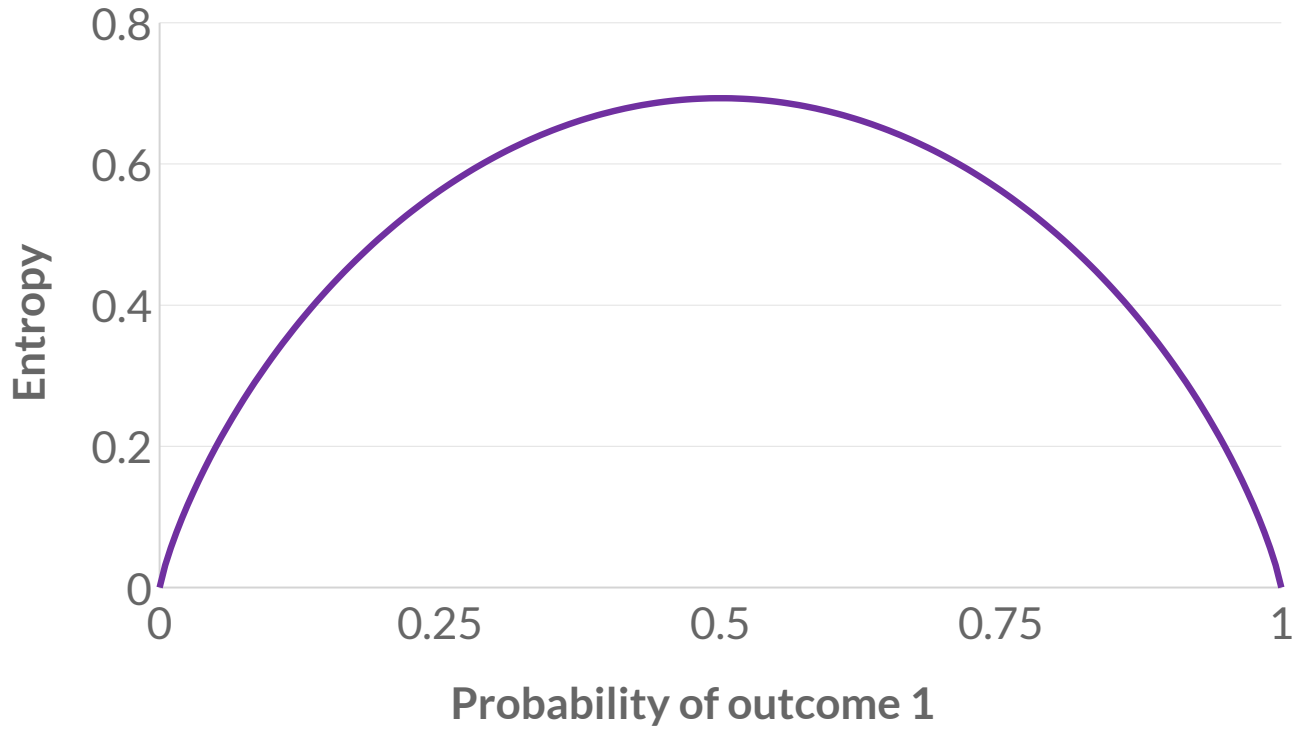
Both peaks describe plausible targets for the same input.

$$\widehat{y}_{\mathrm{MSE}} = \mathbb{E}[Y \mid x] = 0$$

The best mean prediction lies between the peaks, where outcomes are rare.

Entropy

Entropy measures the average uncertainty in an outcome.



$$H(q) = -\sum_k q_k \log q_k$$

A predictable outcome has low entropy.

For two outcomes, uncertainty is greatest when both are equally likely.

Cross entropy and KL divergence

$$D_{\mathrm{KL}}(q\|p) = \sum_k q_k \log \frac{q_k}{p_k}$$

$$H(q,p) = \underbrace{H(q)}_{\text{target uncertainty}} + \underbrace{D_{\mathrm{KL}}(q\|p)}_{\text{excess expected loss}}$$

For a fixed target q, minimizing cross entropy minimizes forward KL.

KL is nonnegative; it is zero when the distributions match.

One-hot labels: cross entropy equals KL

$$q_k = \mathbf{1}\{k = y\}, \qquad H(q) = 0$$

$$D_{\mathrm{KL}}(q\|p) = \log \frac{1}{p_y} = -\log p_y = \ell_{\mathrm{CE}}$$

Only the observed class contributes to the loss.

Maximum likelihood and forward KL

At a fixed input, q describes the target labels and p gives the model probabilities.

$$D_{\mathrm{KL}}(q \| p_\theta) = \underbrace{\mathbb{E}_q[\log q]}_{\text{independent of } \theta} - \mathbb{E}_q[\log p_\theta]$$

$$L(\theta) = -\frac{1}{N} \sum_{i=1}^{N} \log p_\theta(y_i \mid x_i)$$

Average the log loss over observed input–label pairs.

Forward and reverse KL

The target distribution q is fixed; the model distribution p can change.

Forward KL

$$\mathbb{E}_{x\sim q}\left[\log \frac{q(x)}{p_\theta(x)}\right]$$

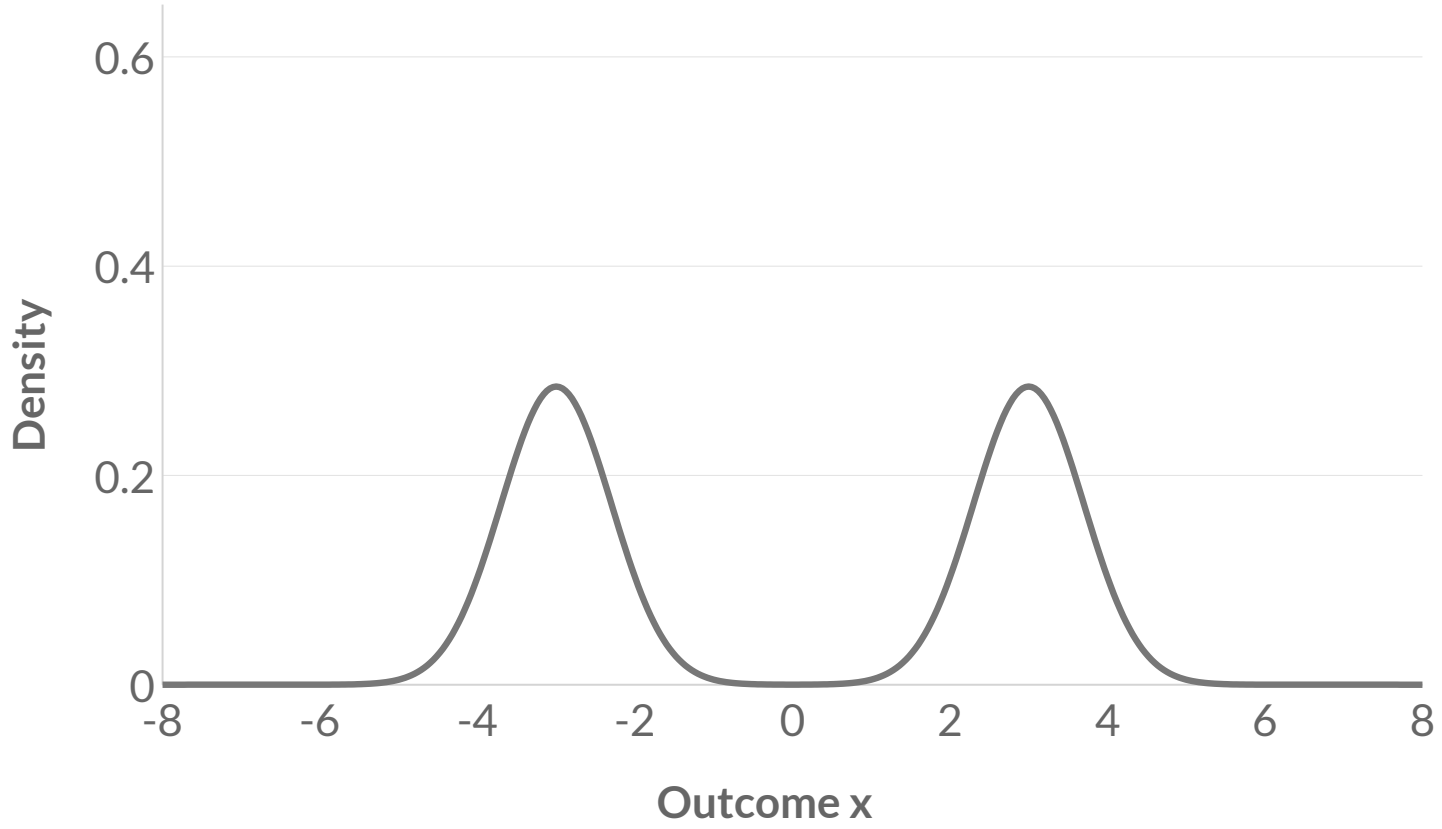
Averages over target outcomes. Strongly penalizes missing their probability mass.

Reverse KL

$$\mathbb{E}_{x\sim p_\theta}\left[\log \frac{p_\theta(x)}{q(x)}\right]$$

Averages over model outcomes. Strongly penalizes placing mass where the target is tiny.

Fitting a two-mode distribution



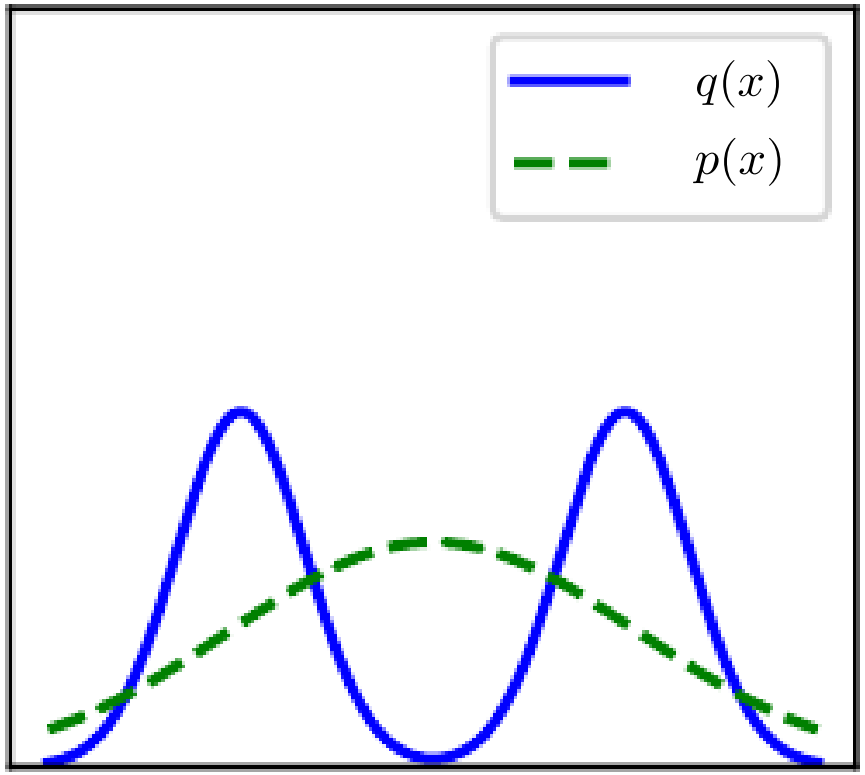
The target has two separated peaks.

The model has one peak. Its mean and width can change.

$$p_\theta(x) = \mathcal{N}(x; \mu, \sigma^2)$$

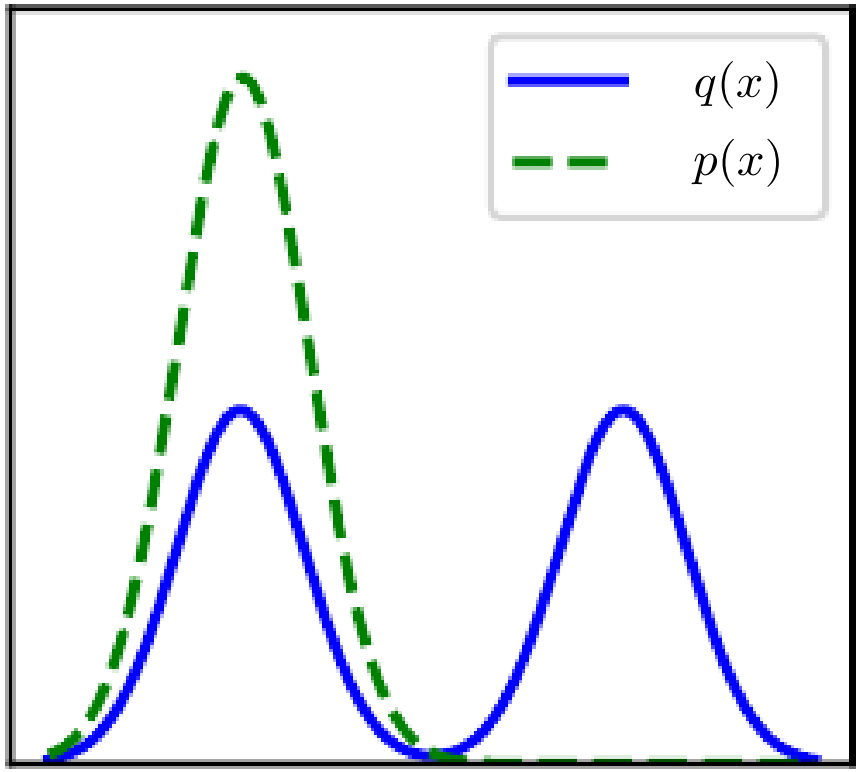
Mode covering and mode seeking

Forward KL: mode covering



Missing either target peak is costly.
The model broadens to cover both.

Reverse KL: mode seeking



Probability in the gap is costly.
The model concentrates on one peak.

Zero probabilities and KL direction

Forward KL: a target outcome must receive model probability.

$$q_k > 0,\ p_k \to 0 \quad \Longrightarrow \quad q_k \log(q_k/p_k) \to +\infty$$

Reverse KL: model probability on an impossible target outcome is forbidden.

$$p_k > 0,\ q_k = 0 \quad \Longrightarrow \quad D_{\mathrm{KL}}(p\|q) = +\infty$$

Empirical loss

$$\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$$

N training examples, each with an input x and target y

$$L_{\text{train}}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \ell(f_\theta(x_i), y_i)$$

- Empirical risk: average loss on the training data.
- Empirical: measured on the observed samples.

Empirical risk minimization

Choose the lowest-loss function in the hypothesis class

$$\min_{f \in \mathcal{H}} \; L_{\text{train}}(f)$$

Choose weights and biases that minimize the training loss

$$\min_{\theta} \; L_{\text{train}}(\theta)$$

ERM defines the objective. An optimizer searches for a solution.

Empirical and population risk

Empirical risk: average over the N training examples

$$\widehat{R}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \ell\big(f_\theta(x_i), y_i\big)$$

Population risk: average over the data distribution P

$$R(\theta) = \mathbb{E}_{(x,y)\sim P}\big[\ell\big(f_\theta(x), y\big)\big]$$

Low training loss alone does not ensure low loss on future data.

Capacity, optimization and generalization

- Capacity: does the model family contain a useful function?
- Optimization: can training find suitable parameters?
- Generalization: do the predictions work on unseen data?

Finding a good neural network

Fix the architecture and loss.
How should we choose the weights and biases?

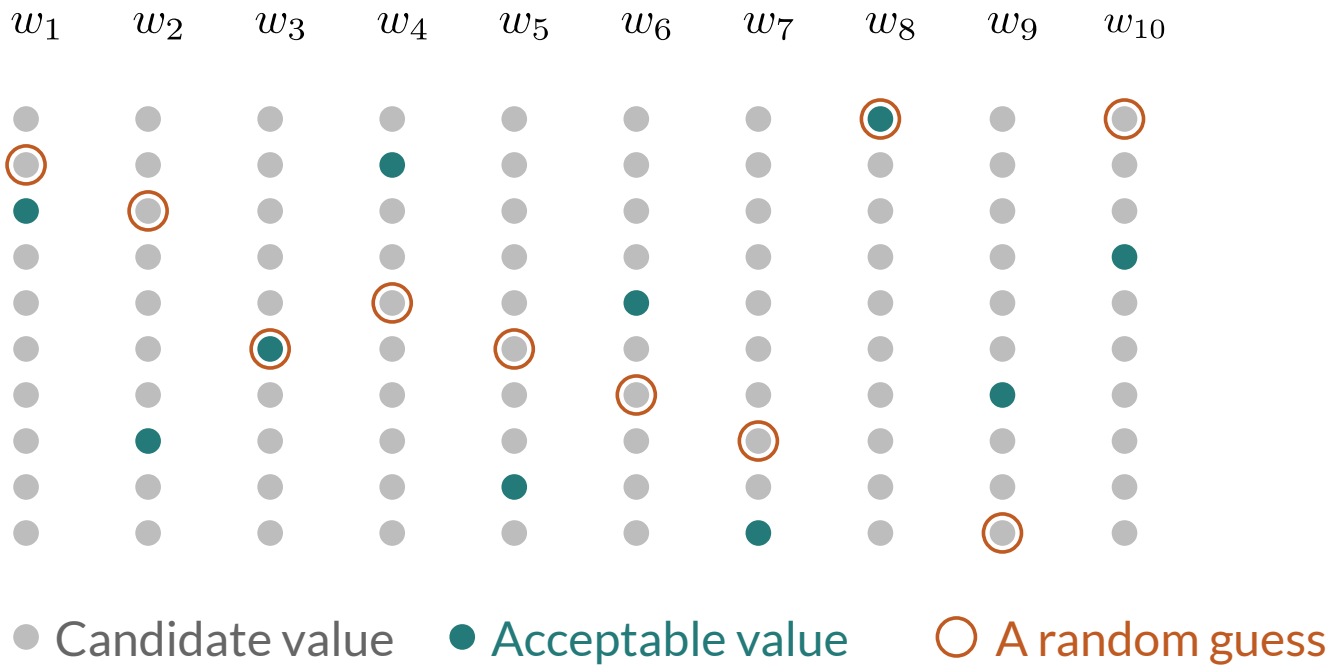
First attempt: random guessing

1. Draw a random set of parameters.
2. Evaluate its loss on the training data.
3. Repeat and keep the best candidate.

Random guessing in high dimensions

Ten candidate values for each weight

Choose one value from every column.



Possible combinations

1 weight

10

2 weights

100

10 weights

10 billion

Assume one acceptable combination out of 10 billion.

What else can we do?

- Use local information about the loss.
- Compute it efficiently for millions of parameters.
- Next: gradients, backpropagation and optimizers.