



Introduction to Data Science

The Neural Network

29 Sep 2026

The Neural Network

Roadmap

1. Connect neurons and compute a layer

Read a feed-forward network, then calculate the output of a small layer.

2. Understand nonlinearity

Use the meal-price example and compare sigmoid, tanh, and ReLU.

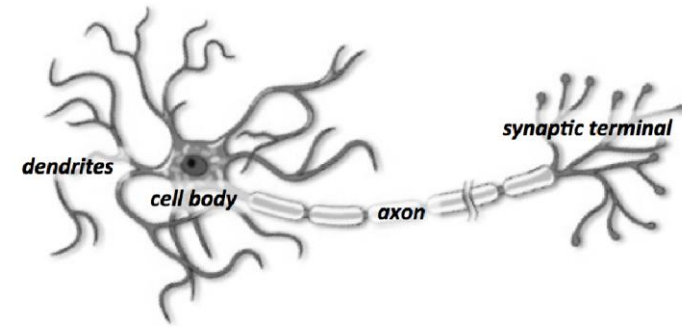
3. Build class probabilities and review

Work through Softmax numerically and connect it to handwritten-digit prediction.

We will follow information through a network and interpret the final outputs.

The Neuron

Recall the Single-Neuron Computation



$$z = x \cdot w + b$$

$$y = f(z)$$



The weighted sum and bias produce one score.

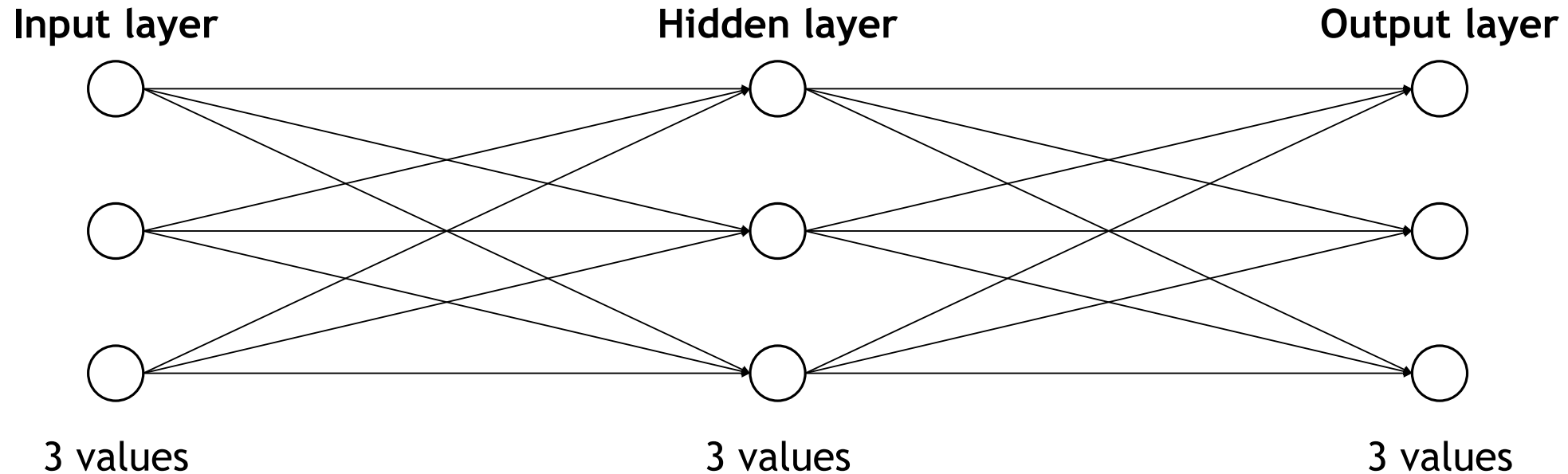
The activation converts that score into the neuron's output.

A network will repeat this computation with many neurons.

A network is easier to read when the computation inside one neuron is clear.

Feed-Forward Neural Networks

Connect Neurons into Layers



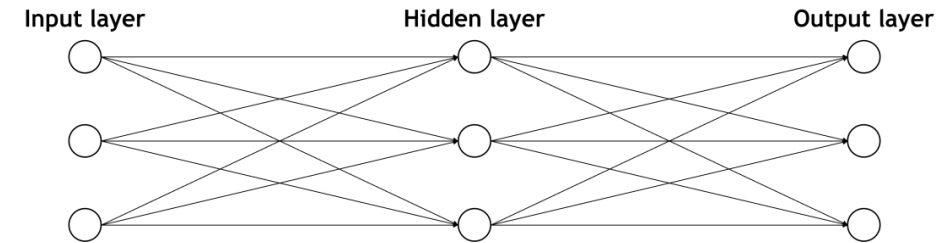
One neuron's output becomes input to the next layer.

This example has input, hidden, and output layers.

Each layer transforms the values received from the preceding layer.

Feed-Forward Neural Networks

Input, Hidden, and Output Layers



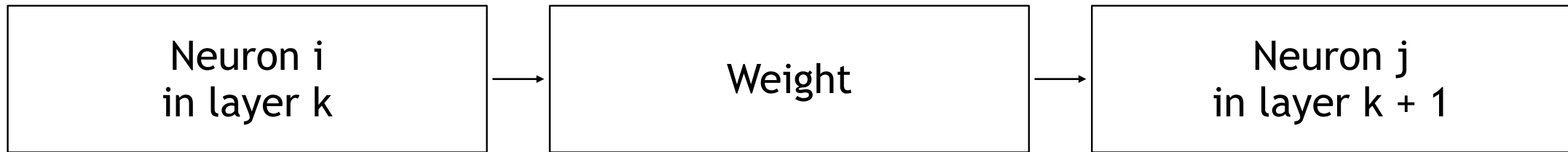
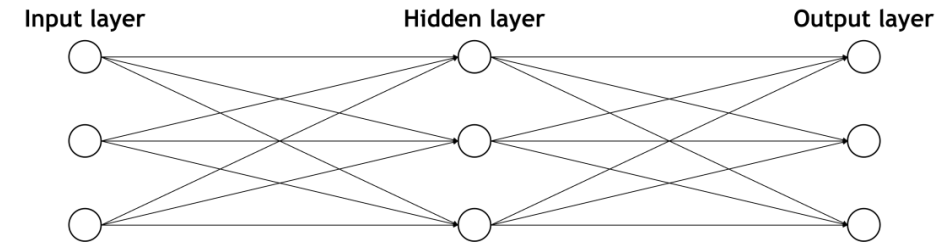
Layer	Role in the example
Input layer	Holds the numerical description of the current example.
Hidden layer	Computes intermediate values used by later neurons.
Output layer	Produces the network's answer to the task.

“Three layers” here counts the input, hidden, and output layers.

Layer names tell us where the values enter, change, and leave.

Feed-Forward Neural Networks

What Does One Connection Mean?



$$W_{i,j}^{(k)}$$

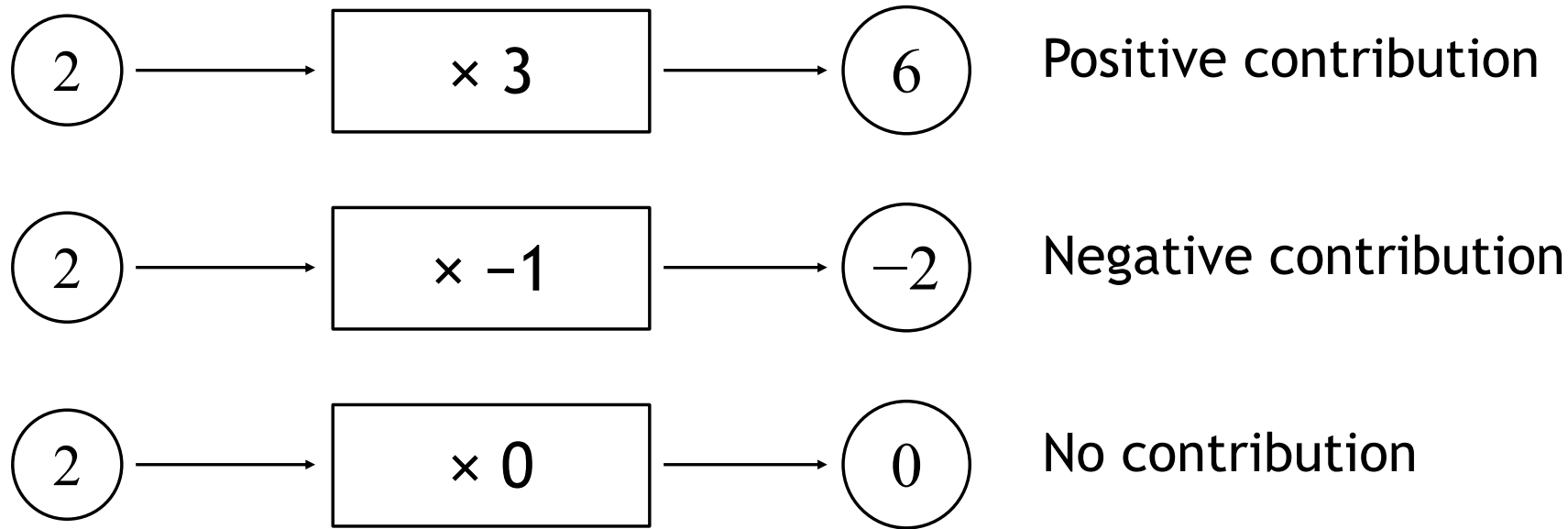
The first index identifies the sending neuron. The second identifies the receiving neuron.

The layer index identifies which pair of layers the connection joins.

A connection weight scales one neuron's contribution to the next neuron.

Feed-Forward Neural Networks

One Input, Three Different Contributions



The input stays at 2. Only the weight changes.

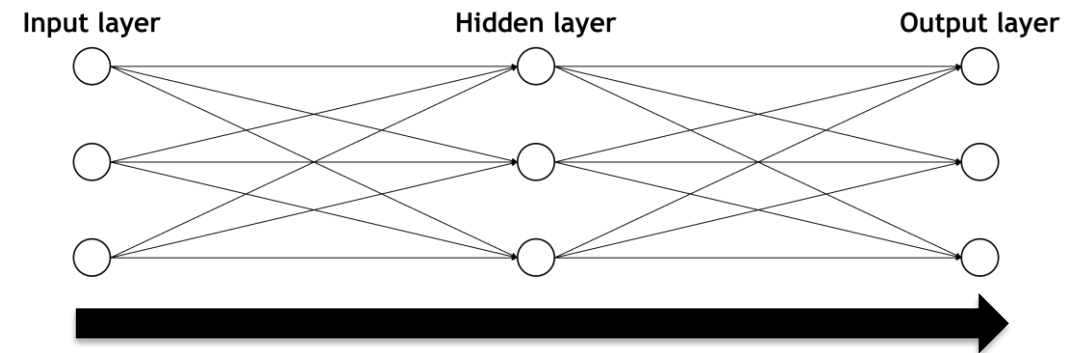
A weight can increase, reverse, or remove an input's contribution.

Feed-Forward Neural Networks

Why Is This Network Called Feed-Forward?

Forward between layers

Inputs are used to calculate hidden values, which are then used to calculate outputs.



No same-layer connections in this diagram

A neuron does not receive an output from another neuron in its own layer.

No backward connections in this diagram

A later layer does not send its output back to an earlier layer.

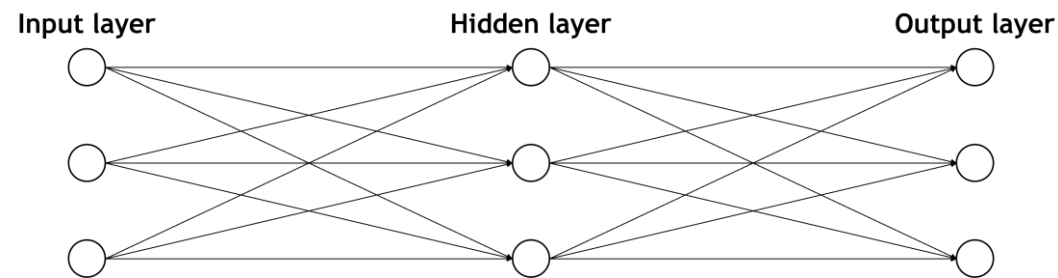
In this example, information moves forward through the layers.

Feed-Forward Neural Networks

Why Is a Hidden Layer “Hidden”?

We supply the input example and compare the final output with the task’s answer.

The hidden layer lies between them and produces intermediate values.



Those values are computed by the network rather than given as the final target labels.

“Hidden” describes its position and role; it does not mean we cannot inspect its values.

Hidden values are intermediate computations, not the externally supplied answers.

Feed-Forward Neural Networks

From Hand-Designed Features to Hidden Representations

EARLIER: HAND-WRITTEN FEATURES

We tried to describe digits using loops and manually chosen rules.

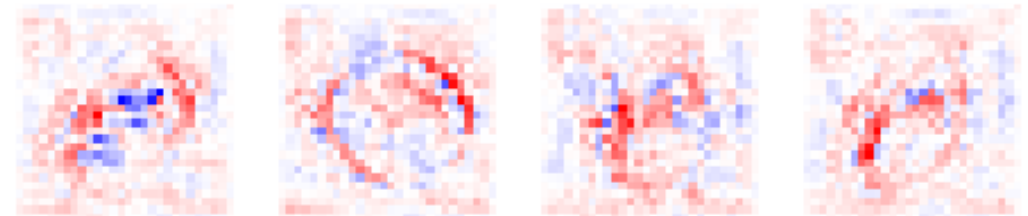
The programmer had to decide which visual patterns to test.



NOW: HIDDEN REPRESENTATIONS

Hidden neurons transform the input into intermediate values.

The book uses edges and contours as intuitive examples of useful visual representations.

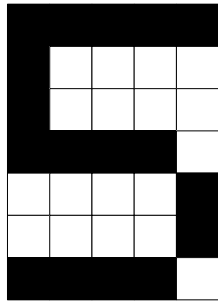


The chapter's next step is to let hidden layers form useful intermediate representations.

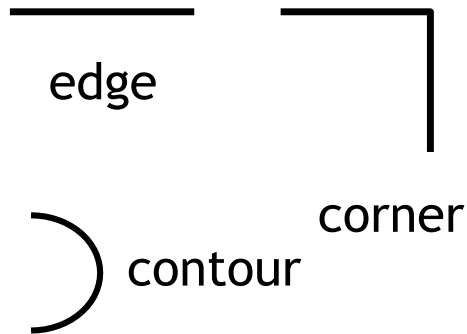
Feed-Forward Neural Networks

A Picture of an Intermediate Representation

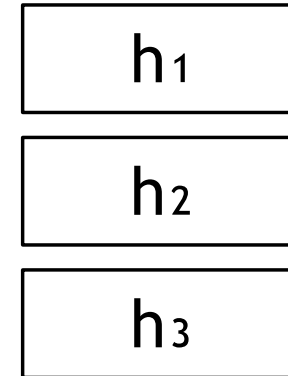
PIXEL INPUT



VISUAL PATTERNS



HIDDEN VALUES



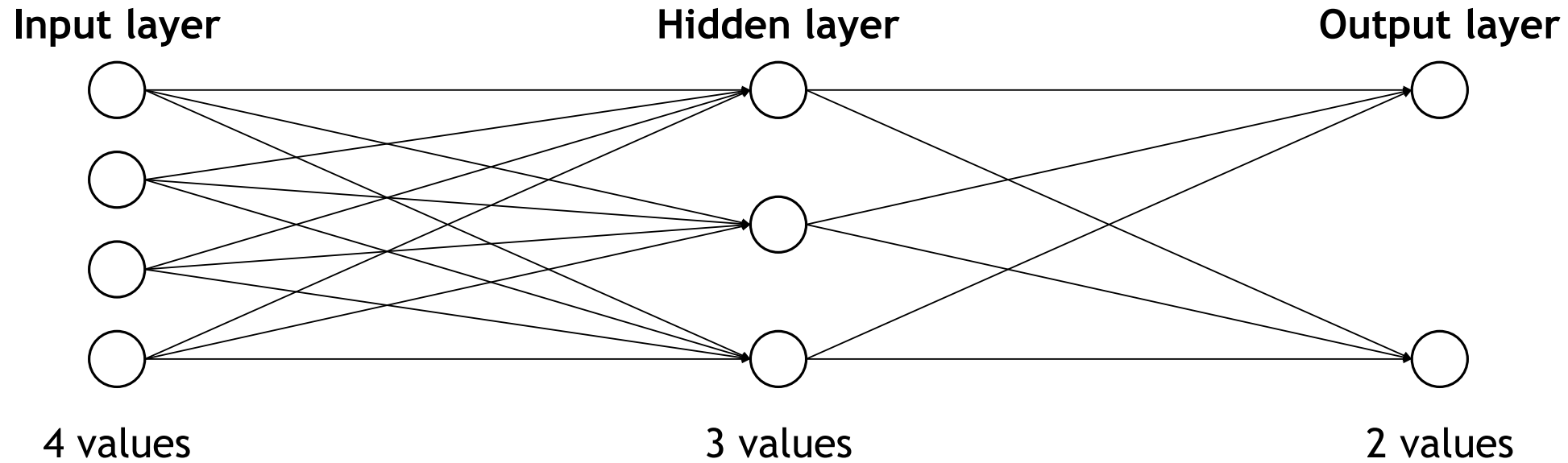
Inputs to the next layer

These patterns illustrate the idea; they are not measured features of a trained model.

Hidden features are new numerical descriptions, not the final class labels.

Feed-Forward Neural Networks

Must Every Layer Have the Same Width?



Layer widths can differ; a narrower hidden layer uses fewer intermediate values.

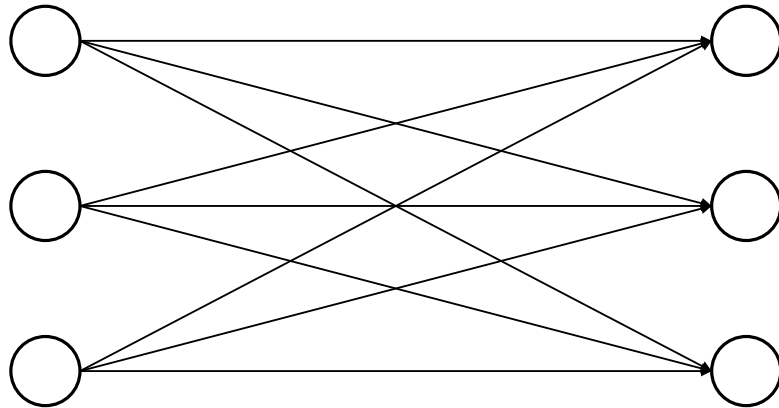
The 4-3-2 sizes are illustrative, not a required architecture.

Layer sizes are design choices; the 3-3-3 figure is only one example.

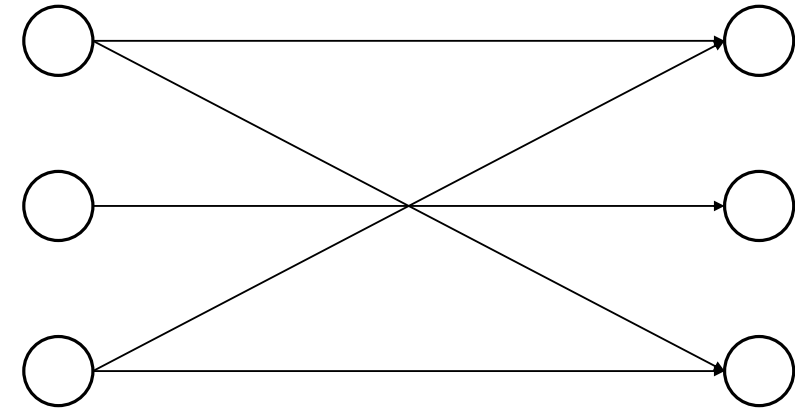
Feed-Forward Neural Networks

Must Every Neuron Connect to Every Next-Layer Neuron?

ALL CONNECTIONS



SELECTED CONNECTIONS



Not every neuron must connect to every neuron in the next layer.

Connection patterns are part of the model's structure.

Connectivity and layer width are separate design choices.

Feed-Forward Neural Networks

What Do Multiple Output Values Represent?

It gives two output indicators: dog present and cat present.

Image contents	Output (dog, cat)
Dog only	(1, 0)
Cat only	(0, 1)
Both	(1, 1)
Neither	(0, 0)

These are two presence indicators, not mutually exclusive classes.

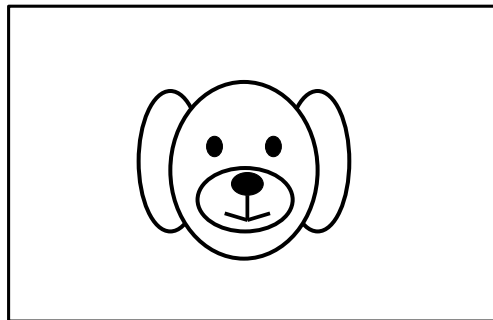
An output vector must be interpreted using the task's label convention.

Feed-Forward Neural Networks

Read the Picture Before the Output

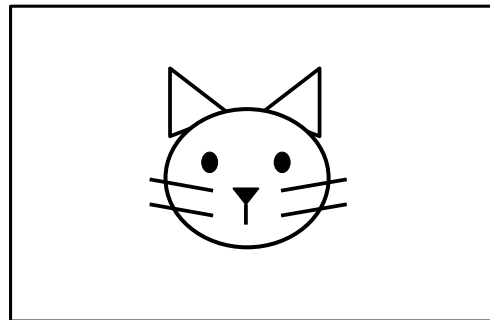
Output order: (dog present, cat present)

Dog only



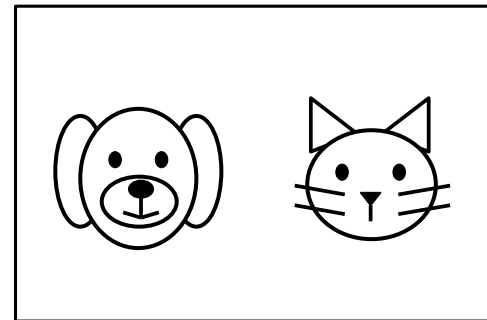
$(1, 0)$

Cat only



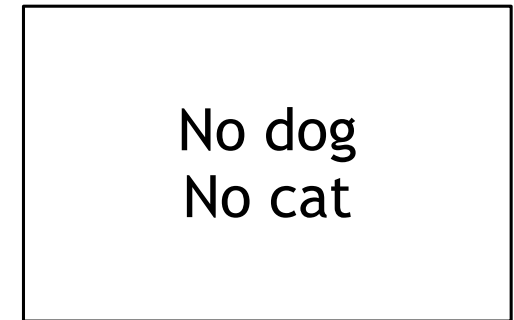
$(0, 1)$

Both



$(1, 1)$

Neither



$(0, 0)$

“Both” is a valid answer, so these indicators do not have to sum to 1.

The two output positions answer two separate presence questions.

Feed-Forward Neural Networks

One Neuron, One Layer, One Network

One neuron

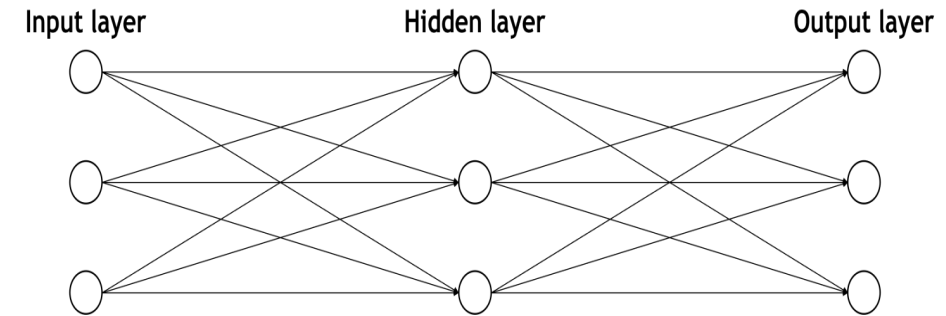
Computes one weighted score and transforms it into one output value.

One layer

Computes several neuron outputs from a shared input vector.

One network

Connects these layer computations from the input to the task output.



Do not confuse a single output value with the full output vector.

Feed-Forward Neural Networks

Read the Matrix Convention

$$y = f(W^T x + b)$$

This chapter stores the weights of one receiving neuron in one column of W .

If the layer has n inputs and m output neurons, W has shape $n \times m$.

Therefore, $W^T x$ produces m scores. Add the m bias values, then apply f .

Using $W^T x$ is required by this chapter's matrix convention, not by neural networks.

The transpose follows the storage convention; it is not an extra learning step.

The columns of W contain neuron weights, so this chapter multiplies by W^T .

Feed-Forward Neural Networks

A Small Layer We Can Calculate by Hand

$$\mathbf{x} = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \quad \mathbf{W} = \begin{bmatrix} 1 & 3 \\ 2 & -1 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

FIRST COLUMN

Weights for neuron 1: (1, 2).

Its bias is 0.

SECOND COLUMN

Weights for neuron 2: (3, -1).

Its bias is 1.

Both neurons see the same input, but use different weights and biases.

Feed-Forward Neural Networks

Calculate the First Neuron

Neuron 1: $w_1 = (1, 2)$, $b_1 = 0$

$$z_1 = 1 \times 2 + 2 \times 1 + 0$$

$$z_1 = 2 + 2 = 4$$

$$\mathbf{x} = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \quad \mathbf{W} = \begin{bmatrix} 1 & 3 \\ 2 & -1 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Use the first column of $\mathbf{W}$.

This is a score before activation, not yet the final transformed output.

Each receiving neuron has its own dot product and bias.

Feed-Forward Neural Networks

Calculate the Second Neuron

Neuron 2: $w_2 = (3, -1)$, $b_2 = 1$

$$z_2 = 3 \times 2 + (-1) \times 1 + 1$$

$$z_2 = 6 - 1 + 1 = 6$$

$$\mathbf{x} = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \quad \mathbf{W} = \begin{bmatrix} 1 & 3 \\ 2 & -1 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Use the second column of $\mathbf{W}$.

The negative weight subtracts the second input's contribution from the score.

The same input can contribute positively to one neuron and negatively to another.

Feed-Forward Neural Networks

Collect the Scores in One Vector

$$Z = \begin{bmatrix} 1 & 2 \\ 3 & -1 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 4 \\ 6 \end{bmatrix}$$

The first matrix is W^T : its rows are the two neurons' weight vectors.

The score entries are 4 and 6, one for each neuron.

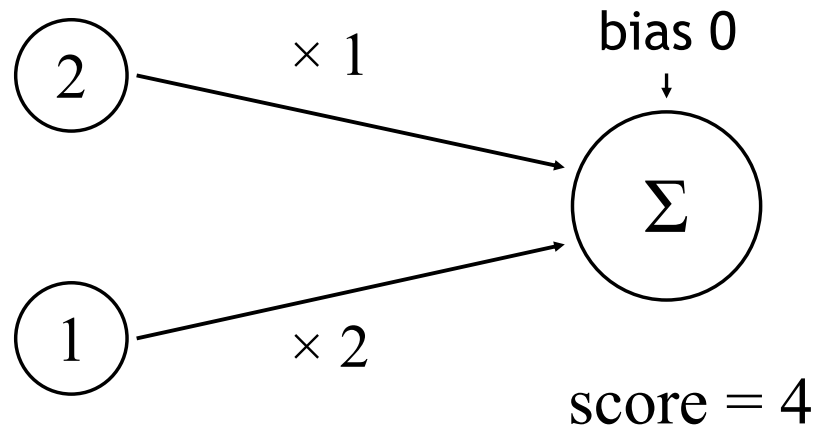
Matrix multiplication performs both dot products together.

$W^T x + b$ is the vector form of the separate neuron calculations.

Feed-Forward Neural Networks

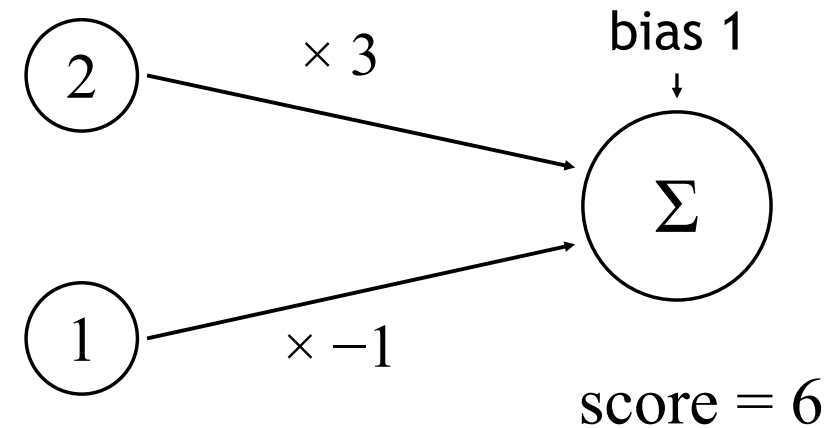
See the Two Dot Products in the Wiring

NEURON 1



$$2 \times 1 + 1 \times 2 + 0 = 4$$

NEURON 2



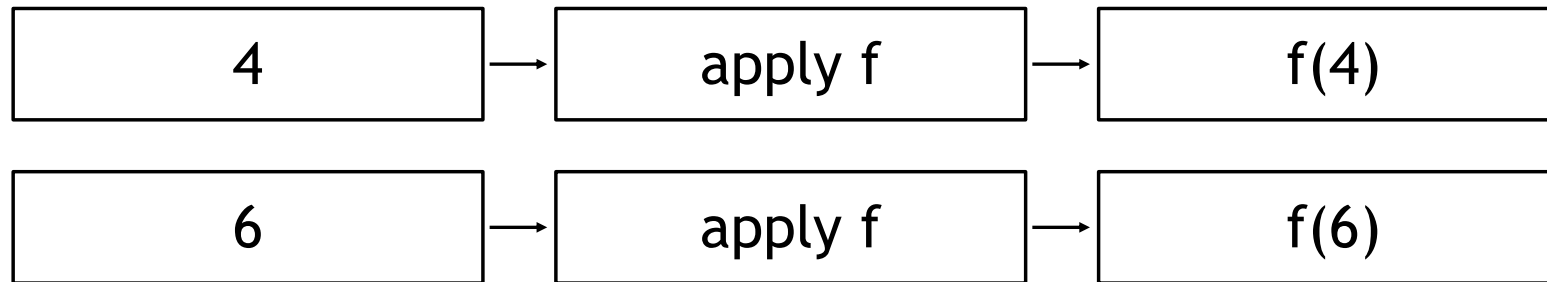
$$2 \times 3 + 1 \times (-1) + 1 = 6$$

The diagram and $W^T x + b$ perform exactly the same calculations.

Feed-Forward Neural Networks

What Does “Apply f Element-Wise” Mean?

$$z = (4, 6) \rightarrow y = (f(4), f(6))$$



“Element-wise” means that f is applied separately to each score.

The first output uses $f(4)$, and the second uses $f(6)$.

We must know which activation function f was chosen before assigning the final numerical outputs.

Transform each coordinate separately; do not add the scores together.

Feed-Forward Neural Networks

Check the Dimensions Before Calculating

For a layer with 3 inputs and 2 output neurons:

Quantity	Shape	Why?
W	3×2	One column per output neuron.
W^T	2×3	One row per output calculation.
x	3×1	Three input values.
$W^T x$, b , and y	2×1	Two scores, two biases, two outputs.

The output length equals the number of receiving neurons.

Feed-Forward Neural Networks

Follow the Values Through the Whole Network



Calculate a layer's output before passing it onward.

That output vector becomes the input vector for the next layer.

A forward pass is the sequence of these calculations with the current weights and biases.

The output of one layer becomes the input to the next.

Linear Neurons

Linear Neurons and Their Limitations

It first considers a linear activation and uses $f(z) = z$ as a simple case.

With $f(z) = z$, the neuron returns its score unchanged.

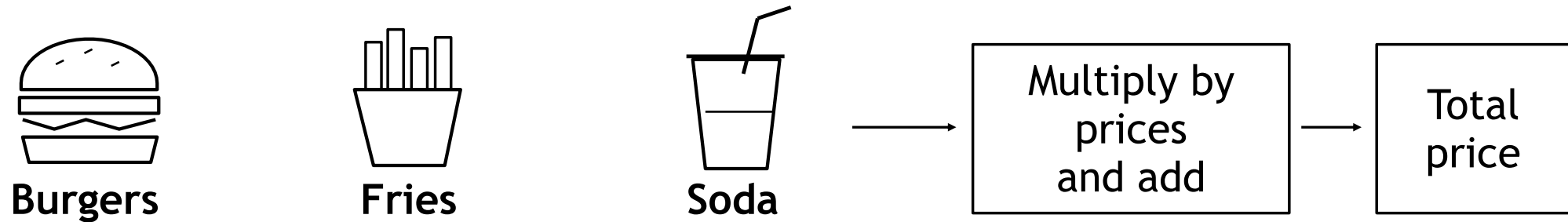
The meal-price example shows a task where this is exactly what we want.

We will then ask why stacking only such computations is limited.

The activation function determines what happens after the weighted sum.

Linear Neurons

The Meal-Price Neuron



$$y = w_1x_1 + w_2x_2 + w_3x_3$$

Inputs are item quantities; weights are item prices.

The additive total follows the paragraph describing the meal-price example.

For a meal total, the output should add the cost of every ordered item.

Linear Neurons

Work Through One Meal Order

Item	Quantity	Price per item	Contribution
Burger	2	\$5	\$10
Fries	1	\$2	\$2
Soda	3	\$1	\$3

$$y = 2 \times 5 + 1 \times 2 + 3 \times 1 = 15$$

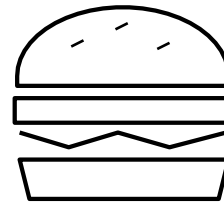
A linear neuron can compute a weighted total with a direct numerical meaning.

Linear Neurons

One More Burger: What Changes?

CURRENT ORDER

2 burgers	\$10
1 fries	\$2
3 sodas	\$3
Total	\$15



+1 burger

+ \$5



NEW TOTAL
\$20

Only the burger quantity changes: $15 + 1 \times 5 = 20$.

For this linear price model, one extra burger always adds its unit price.

Linear Neurons

Why Use $f(z) = z$ for the Meal?

If the weighted sum is 15 dollars, returning 15 gives the desired total.

For this task, forcing the result between 0 and 1 would lose the direct price meaning.

The activation $f(z) = z$ leaves the score unchanged.

An output does not have to be a probability; its meaning depends on the task.

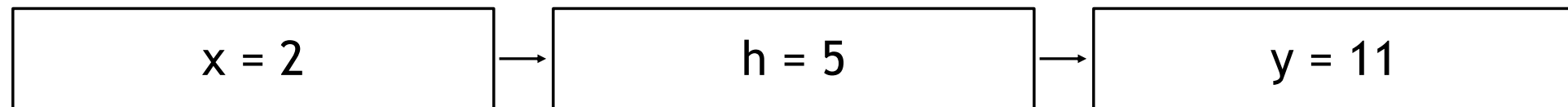
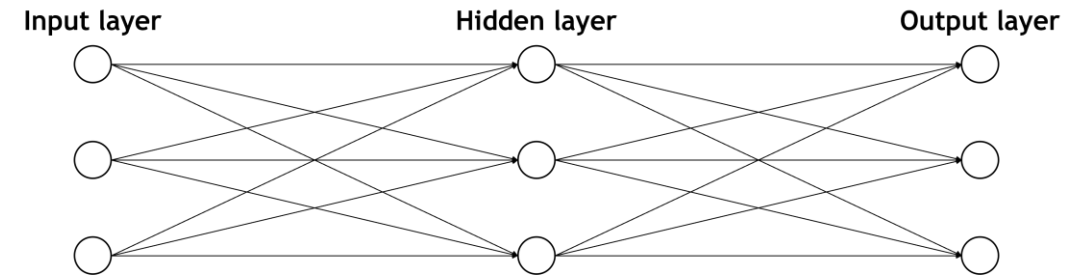
Linear Neurons

Two Linear Stages Can Collapse into One

First layer: $h = 2x + 1$

Second layer: $y = 3h - 4$

Substitute h : $y = 3(2x + 1) - 4 = 6x - 1$



Two stages have become one expression of the same form.

With biases, this is affine. It is still not nonlinear.

More layers do not automatically create a more complex kind of function.

Linear Neurons

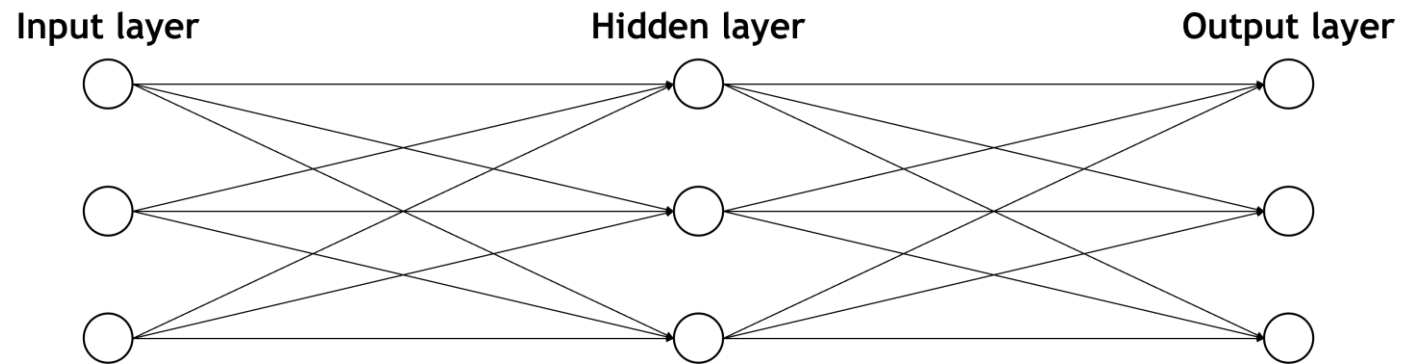
The Same Limitation in Matrix Form

$$h = Ax + b$$

$$y = Bh + c$$

$$y = B(Ax + b) + c$$

$$y = (BA)x + (Bb + c)$$

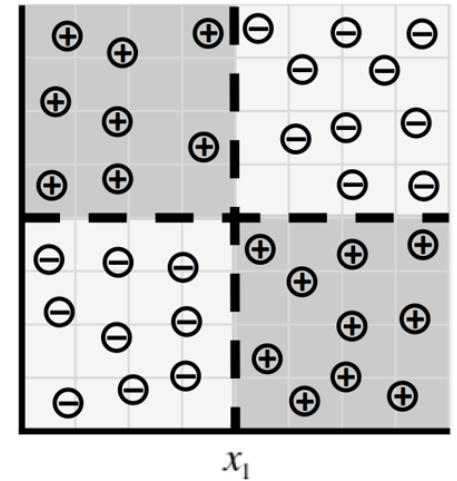
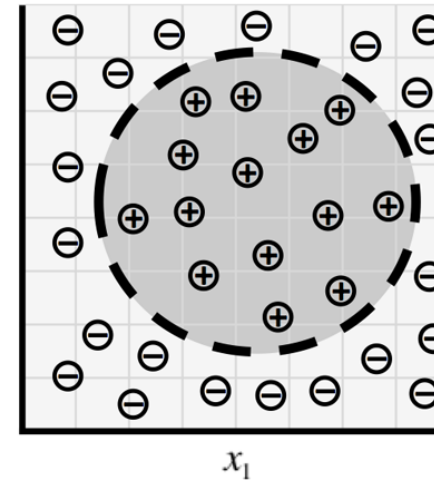
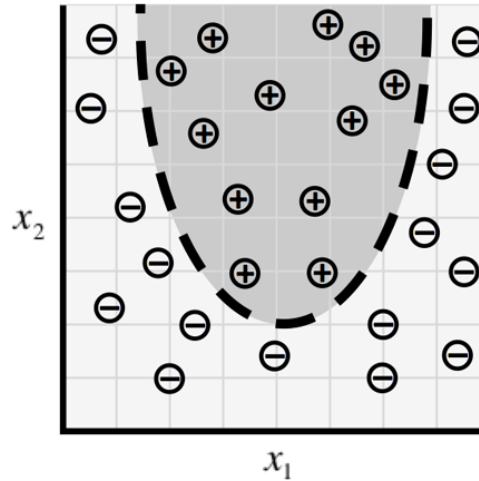
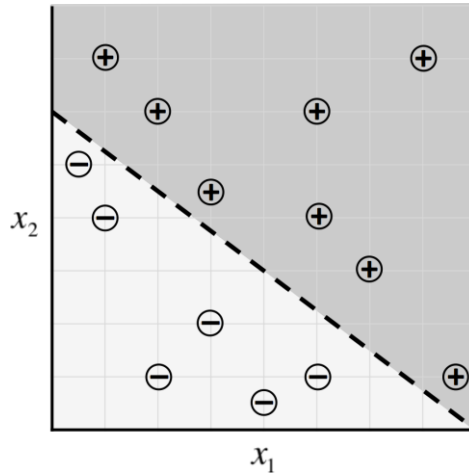
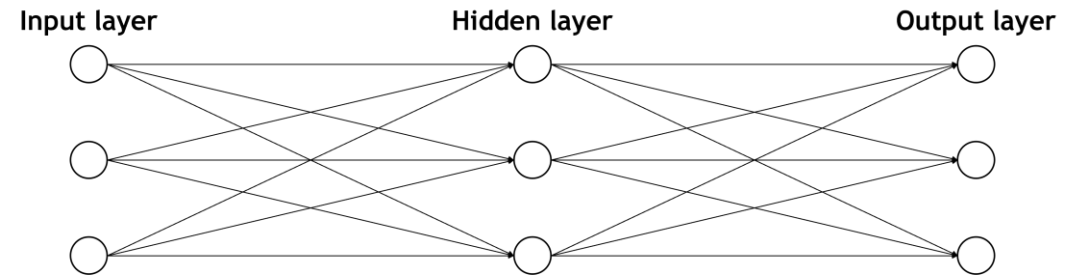


BA is one combined matrix, and $Bb + c$ is one combined bias vector.

Stacked affine layers can be replaced by one affine layer.

Linear Neurons

Why We Need a Nonlinear Step



Recall the curved and multi-region patterns in Figure.

More linear layers still collapse into one affine calculation.

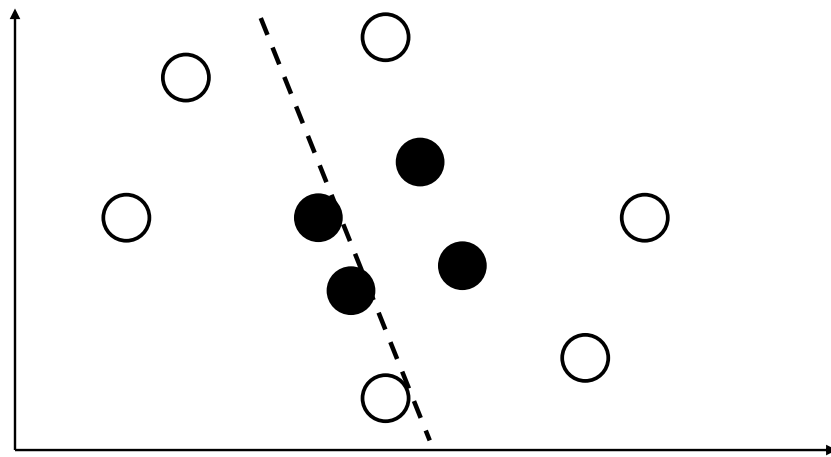
Nonlinear activations let us go beyond that calculation.

To go beyond a single affine transformation, introduce a nonlinearity.

Linear Neurons

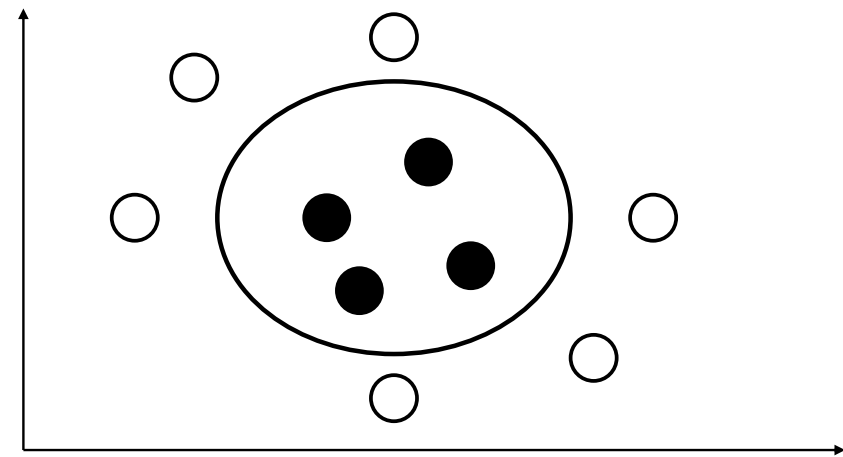
When a Straight Boundary Is Not Enough

ONE STRAIGHT LINE



Filled = class A Open = class B

AN ENCLOSED REGION



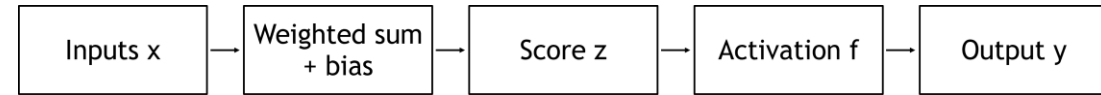
Filled = class A Open = class B

A line splits the plane into two sides; it cannot enclose the center.

Some label patterns require a boundary that one straight line cannot provide.

Activation Functions

Sigmoid, Tanh, and ReLU Neurons



Activation	Main output behavior
Sigmoid	Maps a score between 0 and 1.
Tanh	Maps a score between -1 and 1.
ReLU	Returns zero for negative scores and keeps nonnegative scores.

The same score z can produce different outputs under different functions.

Keep the weighted-score calculation separate from the activation choice.

Activation Functions

Sigmoid: Read the Formula

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

The denominator is 1 plus e raised to the power $-z$.

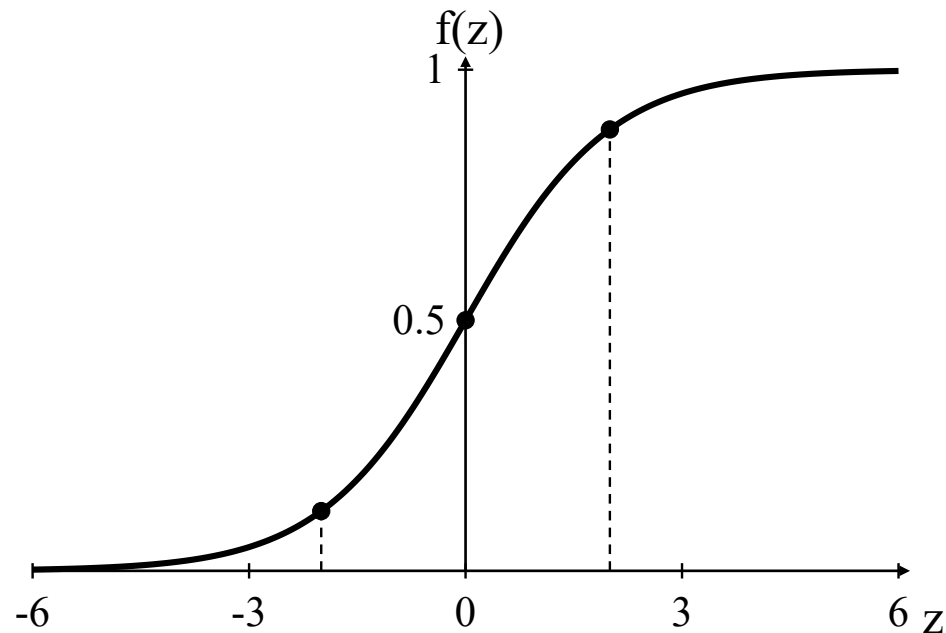
At $z = 0$, $e^0 = 1$, so $f(0) = 1 \div 2 = 0.5$.

Negative scores give outputs below 0.5; positive scores give outputs above 0.5.

Sigmoid turns a real-valued score into a value strictly between 0 and 1.

Activation Functions

Sigmoid: Read the S-Shaped Curve



Marked inputs: -2, 0, and 2.

A very negative score produces an output close to 0.

A score of zero produces 0.5.

A very positive score produces an output close to 1.

The curve approaches 0 and 1 without reaching them for finite z .

Read the input on the horizontal axis and the output on the vertical axis.

Activation Functions

Sigmoid: Three Concrete Inputs

Score z	Sigmoid output	Interpret the number
-2	0.119	Closer to 0 than to 1.
0	0.500	The center of the curve.
2	0.881	Closer to 1 than to 0.

Values are rounded to three decimal places.

The activation output changes smoothly as the score increases.

Activation Functions

Tanh: What Changes from Sigmoid?

$$f(z) = \tanh(z)$$

Tanh also has an S-shaped curve, but its output range is different.

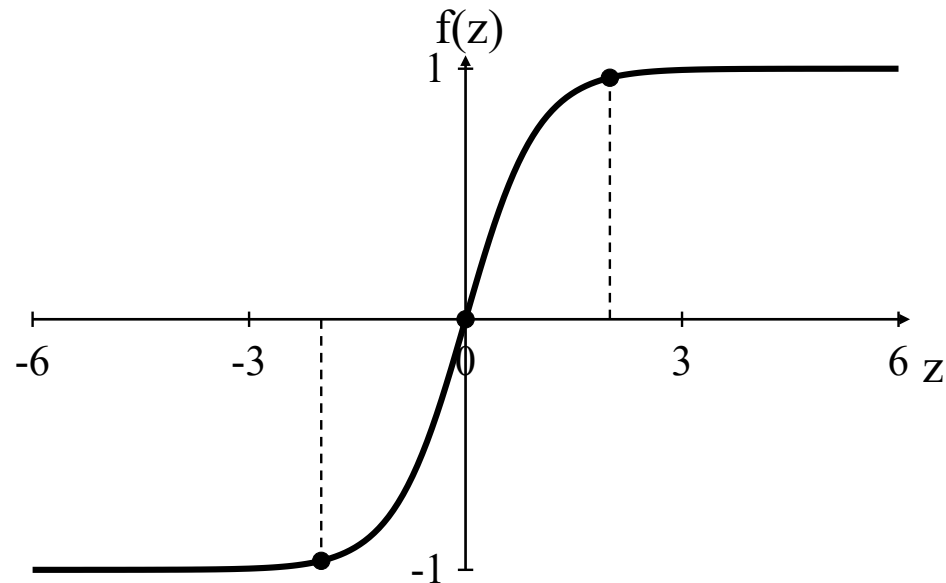
A negative score gives a negative output, and a positive score gives a positive output.

At $z = 0$, the output is 0. This is the meaning of “zero-centered” in this comparison.

Tanh is centered at zero and produces values between -1 and 1 .

Activation Functions

Tanh: Read the Curve



Marked inputs: -2, 0, and 2.

Far to the left, the curve is close to -1.

The curve passes through the origin:
 $\tanh(0) = 0$.

Far to the right, the curve
is close to +1.

For finite scores, the output remains
strictly between -1 and +1.

Unlike sigmoid, tanh can return a negative output.

Activation Functions

Tanh: Three Concrete Inputs

Score z	Tanh output	Interpretation
-2	-0.964	Near -1.
0	0.000	Exactly at the center.
2	0.964	Near +1.

The values are rounded. The outputs at +2 and -2 have opposite signs.

A zero-centered activation preserves the sign of these scores.

Activation Functions

ReLU: Choose the Larger of 0 and z

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

WHEN z IS NEGATIVE

Compare 0 and the negative number.

The larger number is 0, so the output is 0.

WHEN z IS NONNEGATIVE

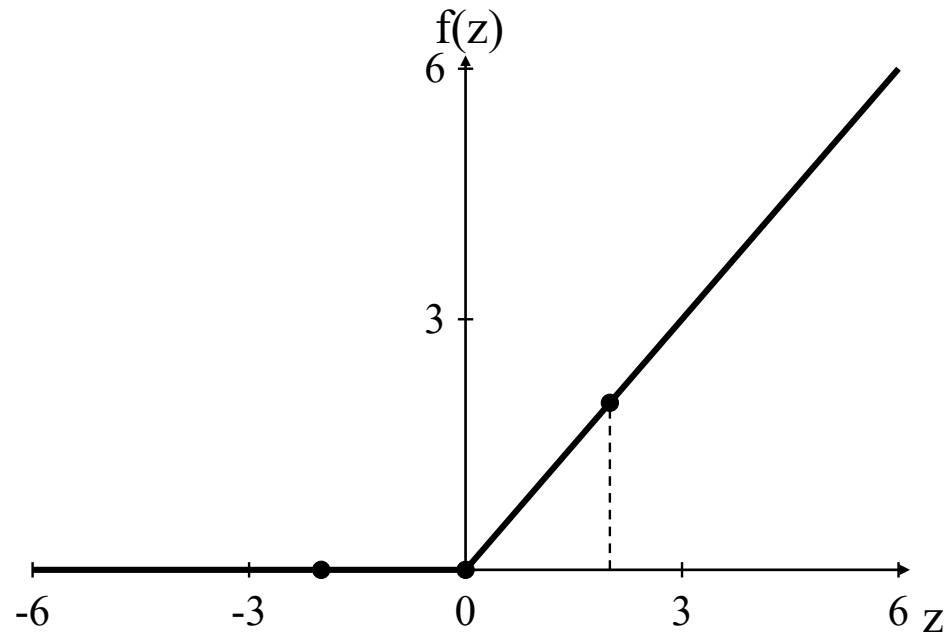
Compare 0 and z .

The output is z itself.

“max” means choose the larger value, not add the two values.

Activation Functions

ReLU: Read the Hockey-Stick Shape



Marked inputs: -2, 0, and 2.

The negative side is flat at zero.

The nonnegative side follows the line $y = z$.

For example, $f(-2) = 0$ and $f(2) = 2$.

There is no upper bound at 1; a ReLU output is not automatically a probability.

ReLU removes negative scores and leaves nonnegative scores unchanged.

Activation Functions

Compare the Same Three Scores

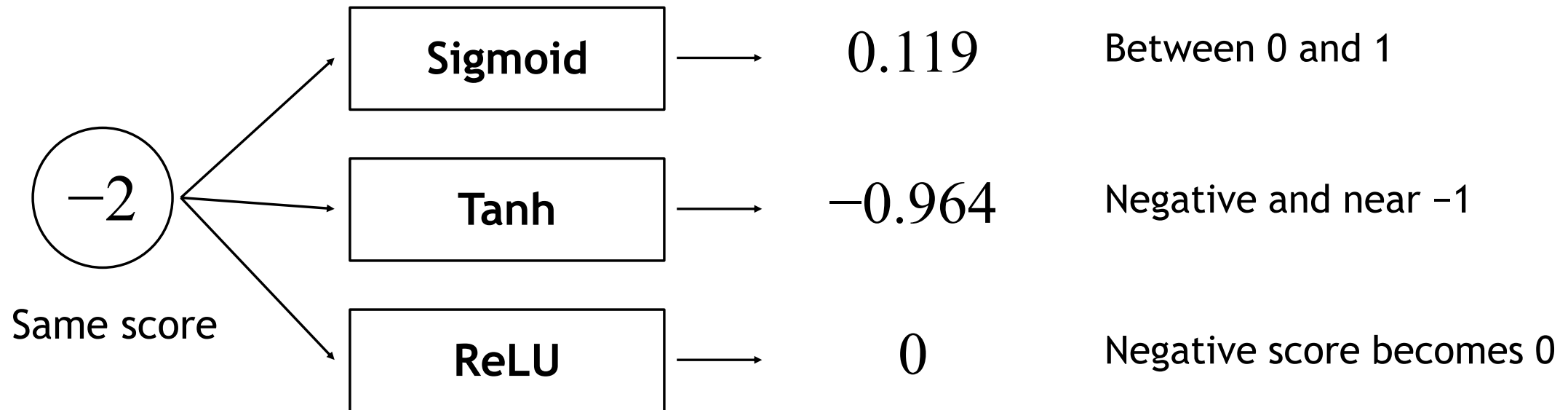
Same score z	Sigmoid	Tanh	ReLU
-2	0.119	-0.964	0
0	0.500	0.000	0
2	0.881	0.964	2

Each row uses one score; only the activation changes.

Different nonlinearities do different things to the same input score.

Activation Functions

One Score, Three Different Responses



All three outputs come from the same input value, not from different weights.

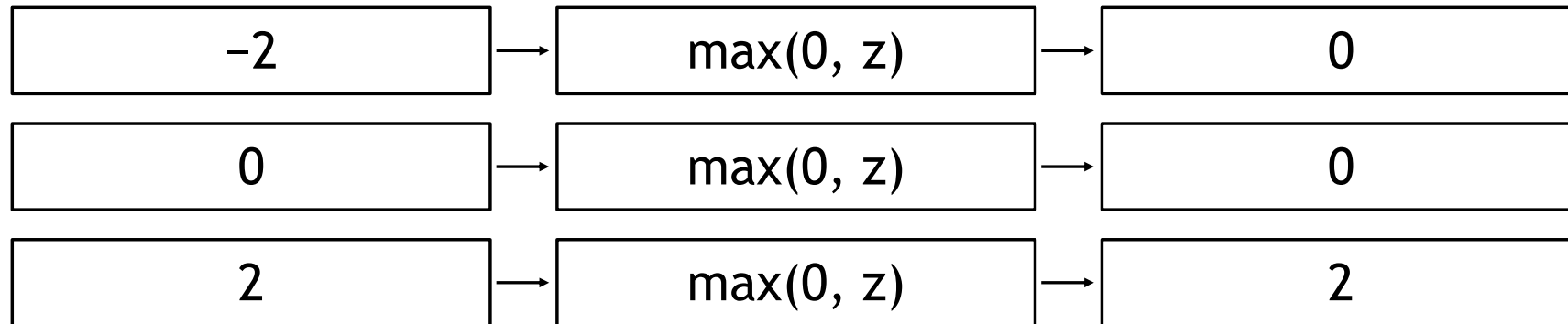
The input score stays fixed; the activation changes its numerical interpretation.

Activation Functions

Apply an Activation to a Layer

$$z = (-2, 0, 2)$$

$$\text{ReLU}(z) = (0, 0, 2)$$



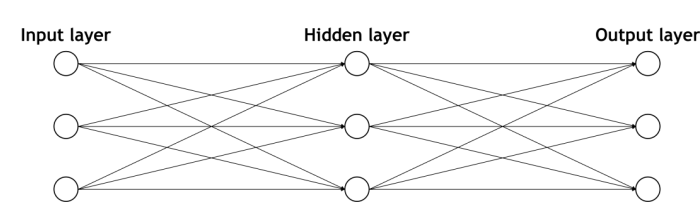
Apply the same function separately to each entry.

The first score is negative, so it becomes 0. The other two are unchanged.

“Element-wise” does not mean choose one output for the whole vector.

A layer can transform each score without mixing the coordinates at this step.

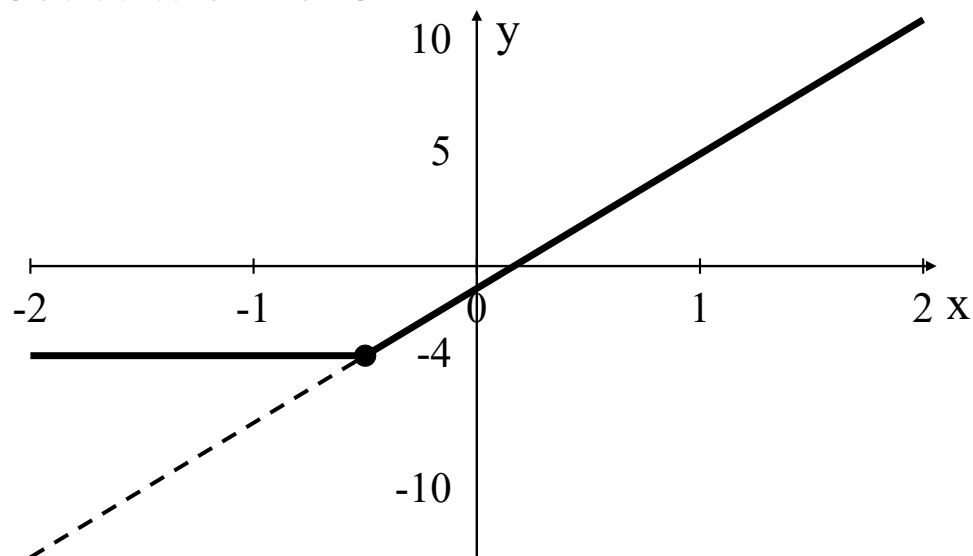
Activation Functions



A ReLU Adds a Bend to the Whole Computation

Dashed: without ReLU

Solid: with ReLU



WITHOUT ReLU

$$y = 3(2x + 1) - 4 = 6x - 1$$

WITH ReLU

$$y = 3 \max(0, 2x + 1) - 4$$

At $x = -1$, the middle score is -1 .
ReLU changes it to 0 , so y becomes -4 instead of -7 .

The solid curve is flat when $2x + 1$ is negative, then rises after the bend.

A nonlinear step can prevent two layers from collapsing into one affine formula.

Activation Functions

What Is Chosen, and What Is Learned?

MODEL DESIGN IN THIS CHAPTER

Choose the layers and their connections.

Choose an activation such as sigmoid, tanh, or ReLU.

PARAMETER VALUES

Weights and biases determine the scores.

Learning selects these values using examples.

An activation is part of the model definition;
the parameters set the numerical contributions.

Softmax Output Layers

When We Need One Class Distribution

For one handwritten-digit image, the candidate labels are 0 through 9.

The labels are mutually exclusive: the image has one target digit in this task.

We want one probability value for each candidate label.

Those values must be nonnegative and add up to 1.



For one MNIST digit, the output should describe alternatives that sum to one.

Softmax Output Layers

A Score Is Not Yet a Probability

raw scores: $z = (2, 1, 0)$

A layer's raw scores are not automatically a probability distribution.

Here, 2 is larger than 1 and the three scores add up to 3, not 1.

Softmax transforms the whole score vector into values that add up to 1.

We use three candidate classes to make the arithmetic visible.

The final output needs an explicit normalization step.

Softmax Output Layers

Step 1: Exponentiate Each Score

Score	Exponential	Approximate value
$z_1 = 2$	$\exp(2)$	7.389
$z_2 = 1$	$\exp(1)$	2.718
$z_3 = 0$	$\exp(0)$	1.000

$\exp(z)$ means e raised to z . These positive values still do not sum to 1.

Exponentiation makes each contribution positive before normalization.

Softmax Output Layers

Step 2: Divide by the Sum

$$\text{total} = 7.389 + 2.718 + 1.000 = 11.107$$

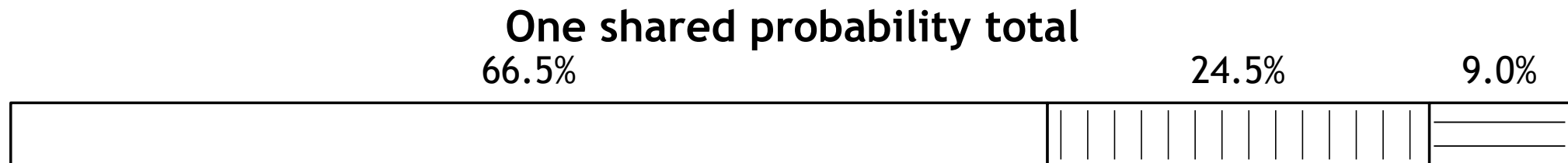
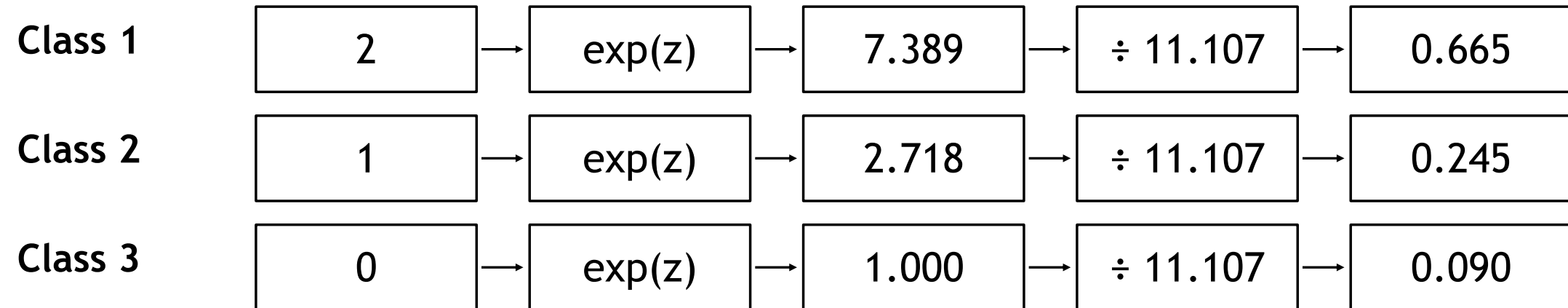
Class	Divide by the common total	Probability
1	$7.389 \div 11.107$	0.665
2	$2.718 \div 11.107$	0.245
3	$1.000 \div 11.107$	0.090

The rounded probabilities add up to 1.000.

Every class uses the same denominator so the outputs form one distribution.

Softmax Output Layers

Softmax as Shares of One Whole



Exponentiate first, then divide the positive contributions by their total.

Softmax Output Layers

Now Read the General Formula

$$y_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

The numerator uses the score for the class we are calculating.

The denominator adds the exponentials of all class scores.

Each output is a share of that common total.

The numerator is one class's positive contribution; the denominator is everyone's total.

Softmax Output Layers

Changing One Score Affects All Outputs

Logits	Class 1	Class 2	Class 3
(2, 1, 0)	0.665	0.245	0.090
(2, 3, 0)	0.259	0.705	0.035

Only the second score changes, but all probabilities change.

Its exponential changes the common denominator.

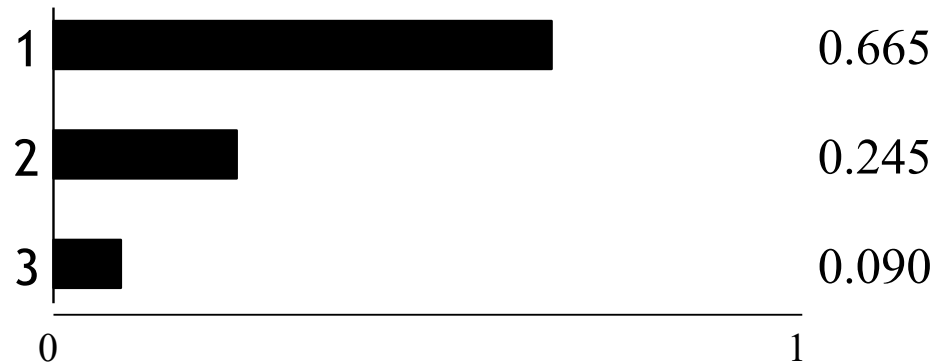
Softmax couples the outputs through their shared total.

Softmax Output Layers

Watch the Probability Shares Change

BEFORE

Scores: (2, 1, 0)



Exponential total = 11.107

AFTER

Scores: (2, 3, 0)



Exponential total = 28.475

Class 1 keeps score 2, but its probability drops from 0.665 to 0.259.

A larger competing score changes the common total, so every share changes.

Softmax Output Layers

Read a Strong or a Weak Prediction

Concentrated

$(0.95, 0.03, 0.02)$



One class receives almost all the mass.

Spread out

$(0.34, 0.33, 0.33)$



Several classes receive similar mass.

A confident-looking prediction is not a guarantee that the chosen class is correct.

The distribution shows how the model allocates probability across the alternatives.

Softmax Output Layers

What If All Scores Are Equal?

$$\mathbf{z} = (0, 0, 0)$$

$$\exp(\mathbf{z}) = (1, 1, 1)$$

$$\text{Softmax}(\mathbf{z}) = (1/3, 1/3, 1/3)$$

Equal scores give equal positive contributions.

Dividing by their common sum gives equal probabilities.

Softmax does not favor a class when all logits are equal.

Softmax Output Layers

Return to MNIST: Ten Candidate Digits

An illustrative output for one image with target digit 5:

Digit	Probability	Digit	Probability
0	0.01	5	0.86
1	0.02	6	0.01
2	0.03	7	0.02
3	0.01	8	0.02
4	0.01	9	0.01

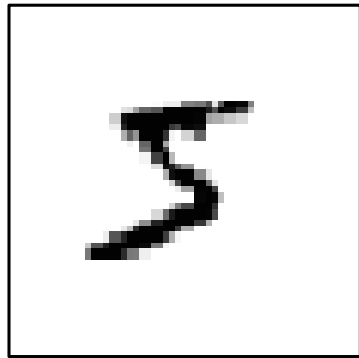
The ten values sum to 1, and digit 5 has the largest probability.

The same normalization works with ten classes instead of three.

Softmax Output Layers

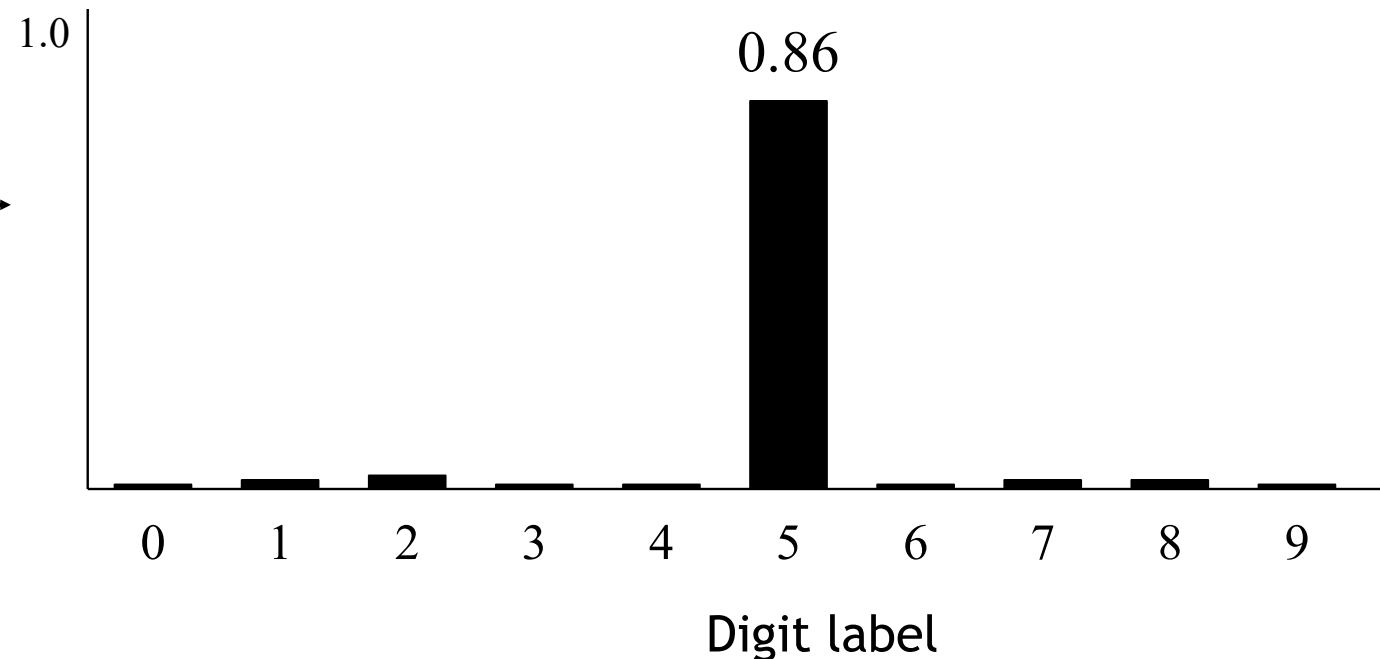
One Image, Ten Competing Labels

ONE INPUT IMAGE



Target digit: 5

NETWORK OUTPUT



Ten probabilities sum to 1. Digit 5 has the largest share.

The predicted label is the position of the largest probability.

Softmax Output Layers

Do Not Confuse Two Different Output Tasks

ONE MNIST DIGIT

The classes 0 through 9 are mutually exclusive.

Softmax assigns probability across these competing alternatives.

DOG AND CAT PRESENCE

The image may contain both or neither.

The textbook's two presence indicators describe a different task, not one exclusive distribution.

Choose the output interpretation before choosing a normalization.

The Neural Network

Forward Computation Is Not Training

FORWARD COMPUTATION

Take an input vector.

Compute layer values using current weights and biases.

Read the final output.

PARAMETER LEARNING

Compare predictions with examples.

Adjust the parameters to improve the model.

A forward pass uses parameters; training is how suitable parameters are found.

The Neural Network

Put the Pieces Together



Early layers construct intermediate representations of the input.

For exclusive classification, Softmax converts logits to probabilities.

Hidden layers build features. The output layer expresses the answer.

A network combines repeated neuron computations with a task-appropriate output layer.

Summary

Quick Check

1. A layer has 3 inputs and 2 neurons. In this chapter's convention, what is the shape of W ?
2. What are $\text{sigmoid}(0)$, $\text{tanh}(0)$, and $\text{ReLU}(-2)$?
3. Why can changing one logit change all Softmax probabilities?

Test the meaning of the computation as well as the final number.

Summary

Explanations

1. Weight-matrix shape

There is one column for each of 2 neurons and one row for each of 3 inputs: 3×2 .

2. Activation values

Sigmoid maps zero to 0.5. Tanh maps zero to 0. ReLU maps -2 to 0.

3. Coupled Softmax outputs

Changing one exponential changes the shared total used by every class.

Read the weight convention, activation rule, and shared denominator carefully.

Summary

What Did We Learn?

Connect neurons

A feed-forward network sends values through input, hidden, and output layers.

Introduce nonlinear transformations

Linear layers alone collapse into one affine map. Sigmoid, tanh, and ReLU change that computation.

Interpret the final output

Softmax turns class logits into a distribution. The next chapter explains parameter learning.

Network structure defines the computation; learning selects the parameters.