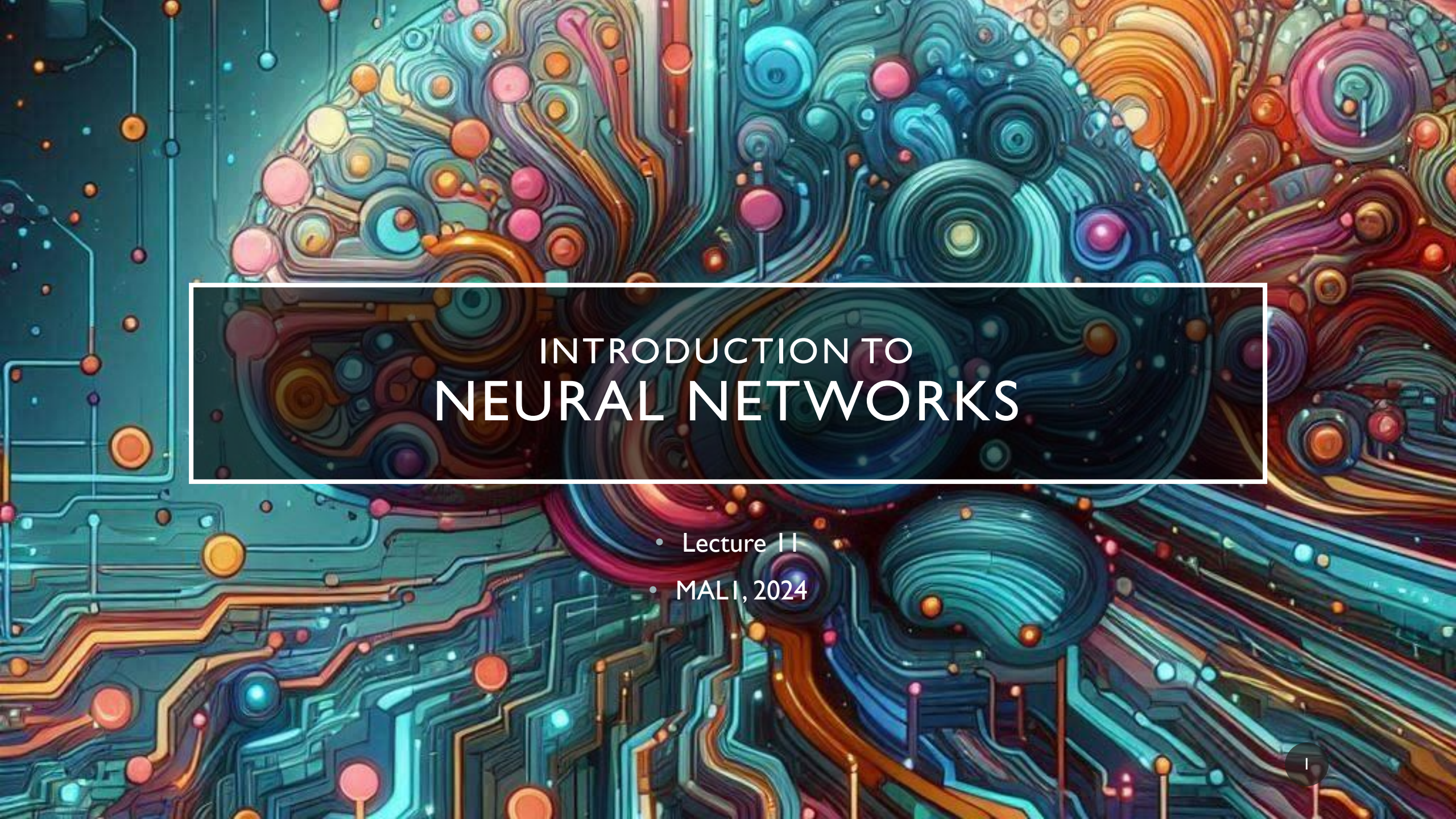
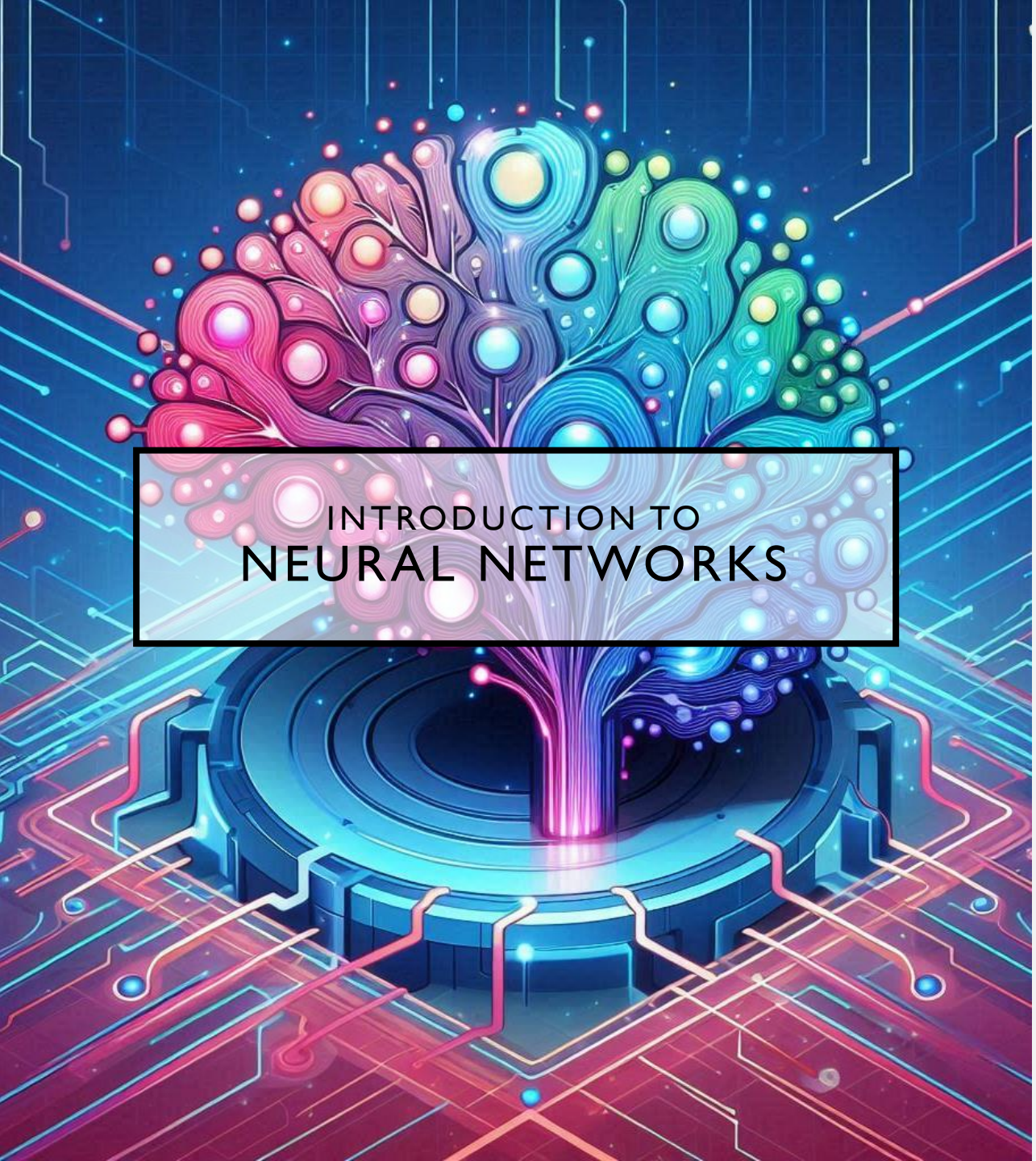


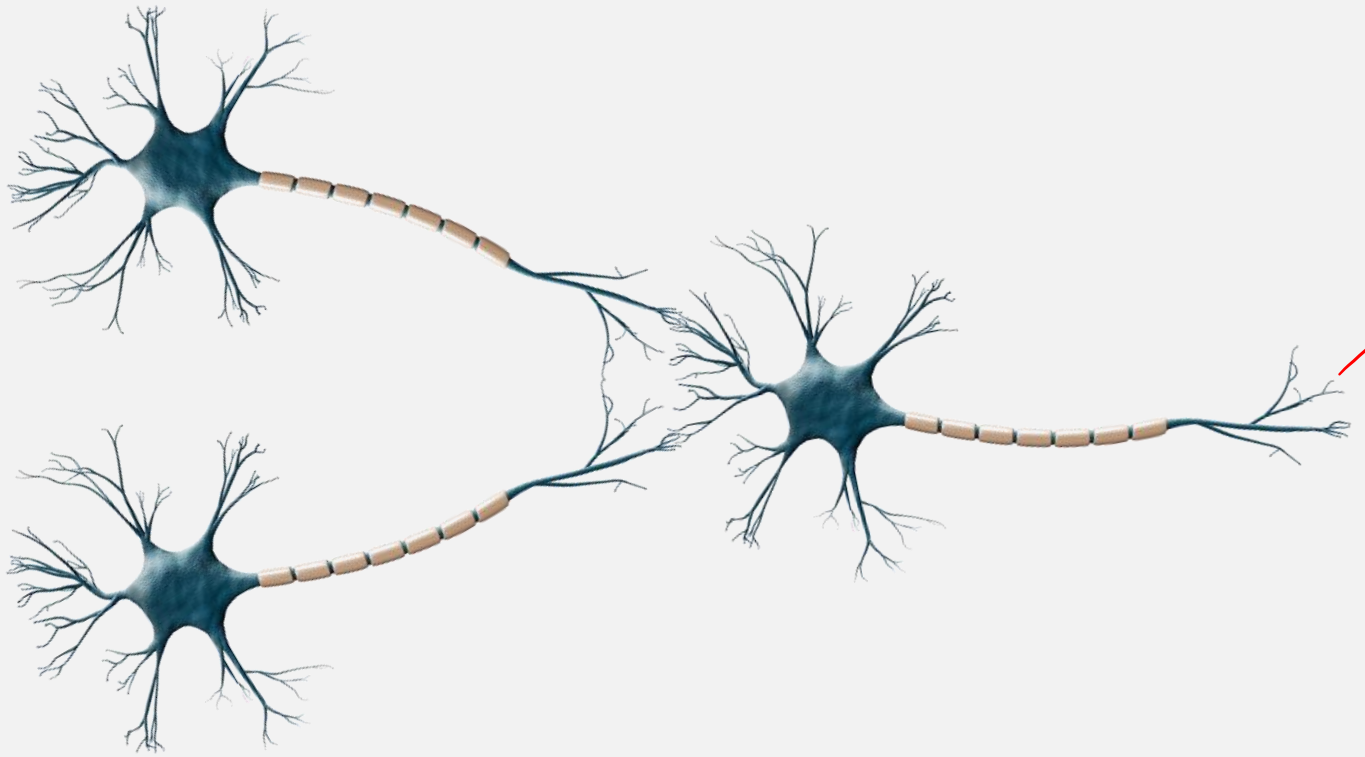
INTRODUCTION TO NEURAL NETWORKS

- Lecture 11
- MAL1, 2024



- What is a neural network?
- How do we structure it?
- How do we train it?
- How do we implement it?

NEURONS



hand hot

input

hand hurts

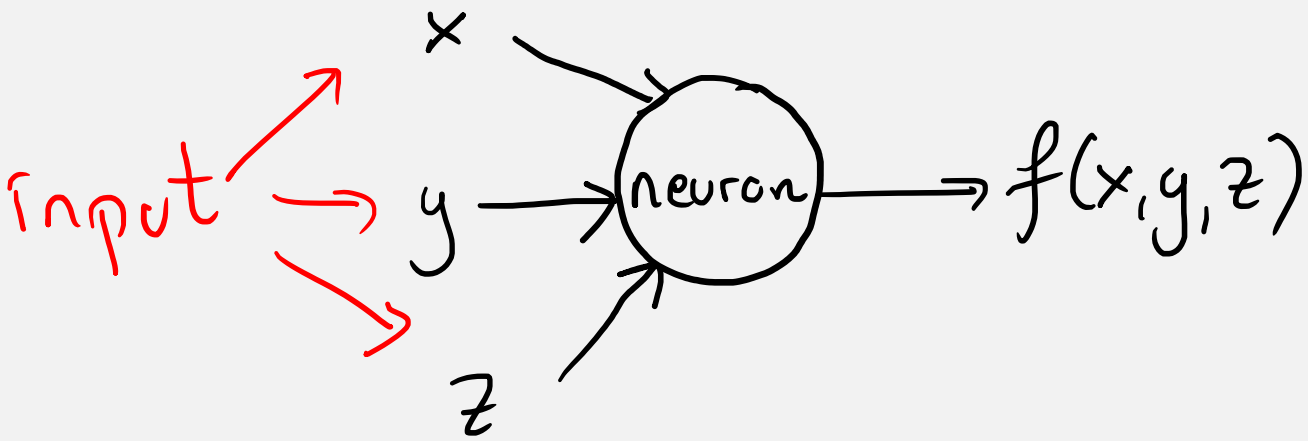
remove hand

output

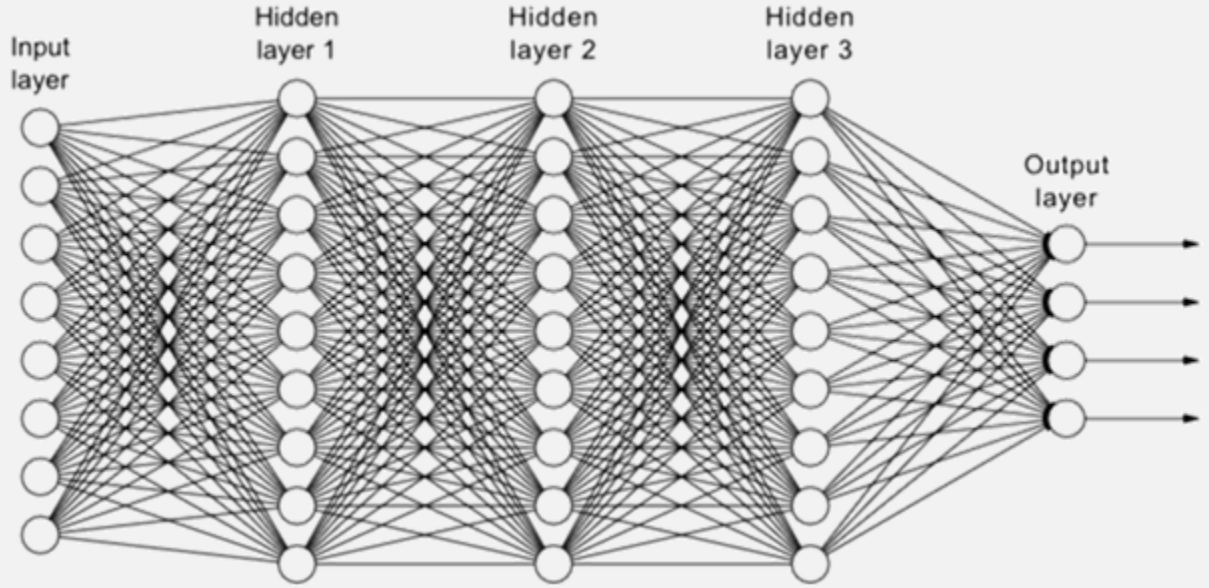
neural network tries to mimic this idea

ARTIFICIAL NEURONS

are really just functions



ARTIFICIAL NEURAL NETWORKS



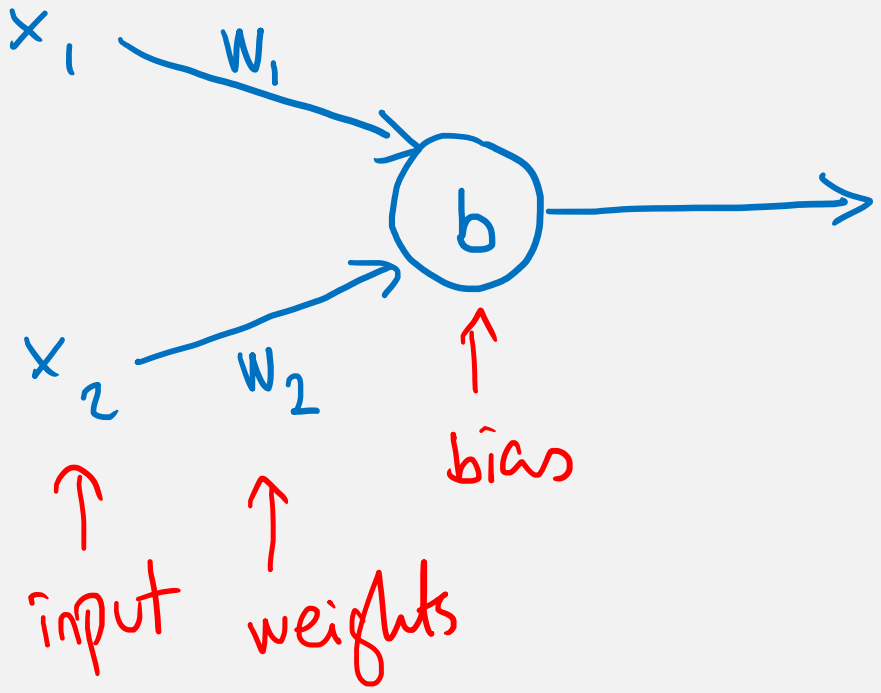
But how do we train a neural network to make good decisions?

HOWEVER …

… a neural network has absolutely nothing to do with a brain.

Rather, it allows for arbitrarily complex decision boundaries / regression functions

NOTATION: WEIGHTS AND BIASES



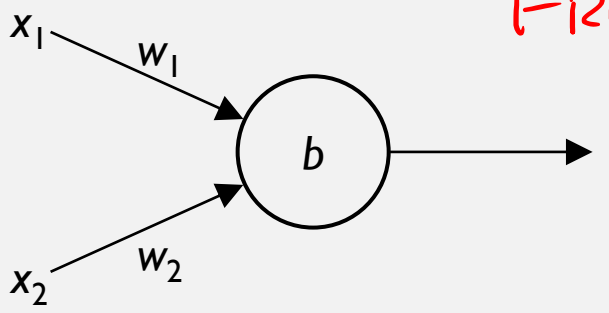
a neuron $\Rightarrow$

a linear combination

$w_1x_1+w_2x_1+b=\vec{w}\cdot\vec{x}+b$

PERCEPTRONS

↳ MAKES THE TRANSFORMATION FROM INPUT → OUTPUT NONLINEAR

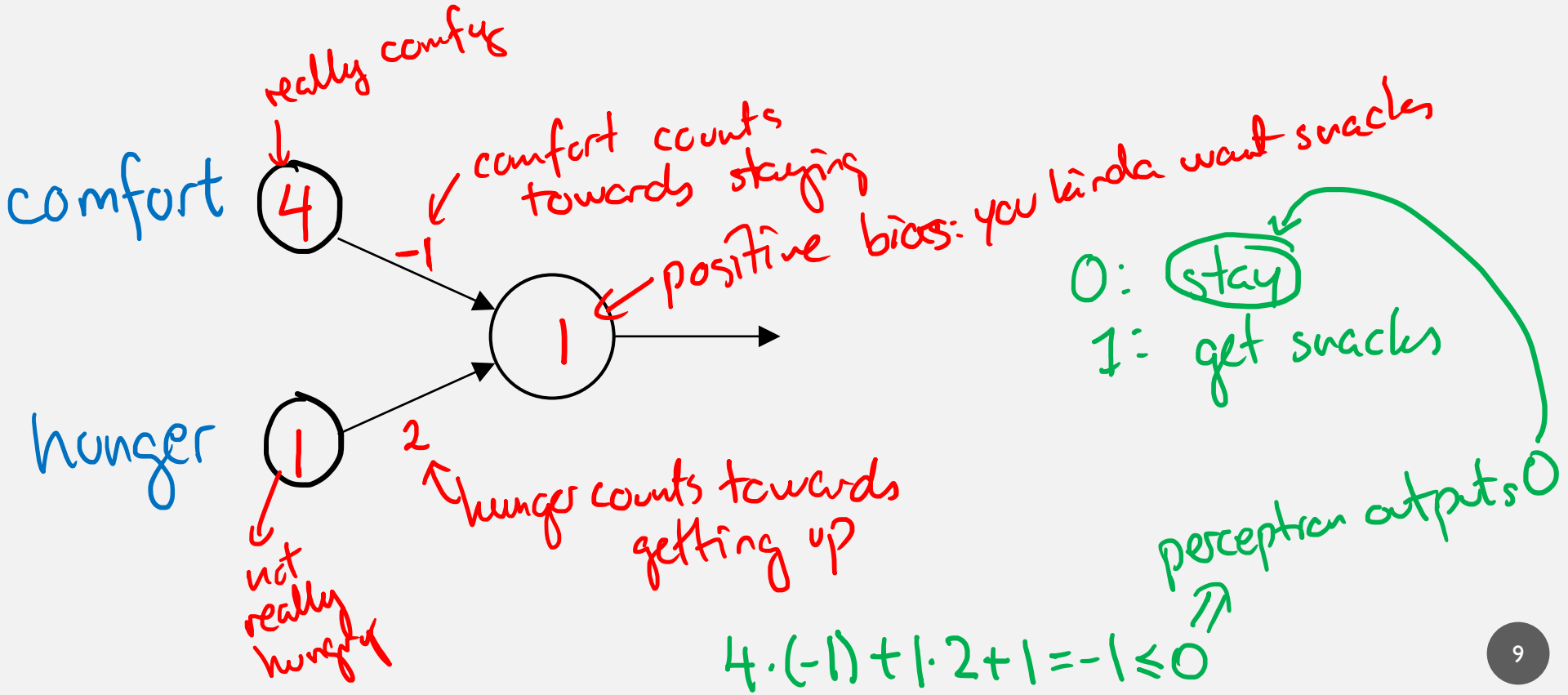


$$\text{output} = \begin{cases} 0 & \text{if } \vec{w} \cdot \vec{x} + b \leq 0 \\ 1 & \text{if } \vec{w} \cdot \vec{x} + b > 0 \end{cases}$$

THE SNACK EXAMPLE

You just sat down in the couch to watch your favorite tv show!

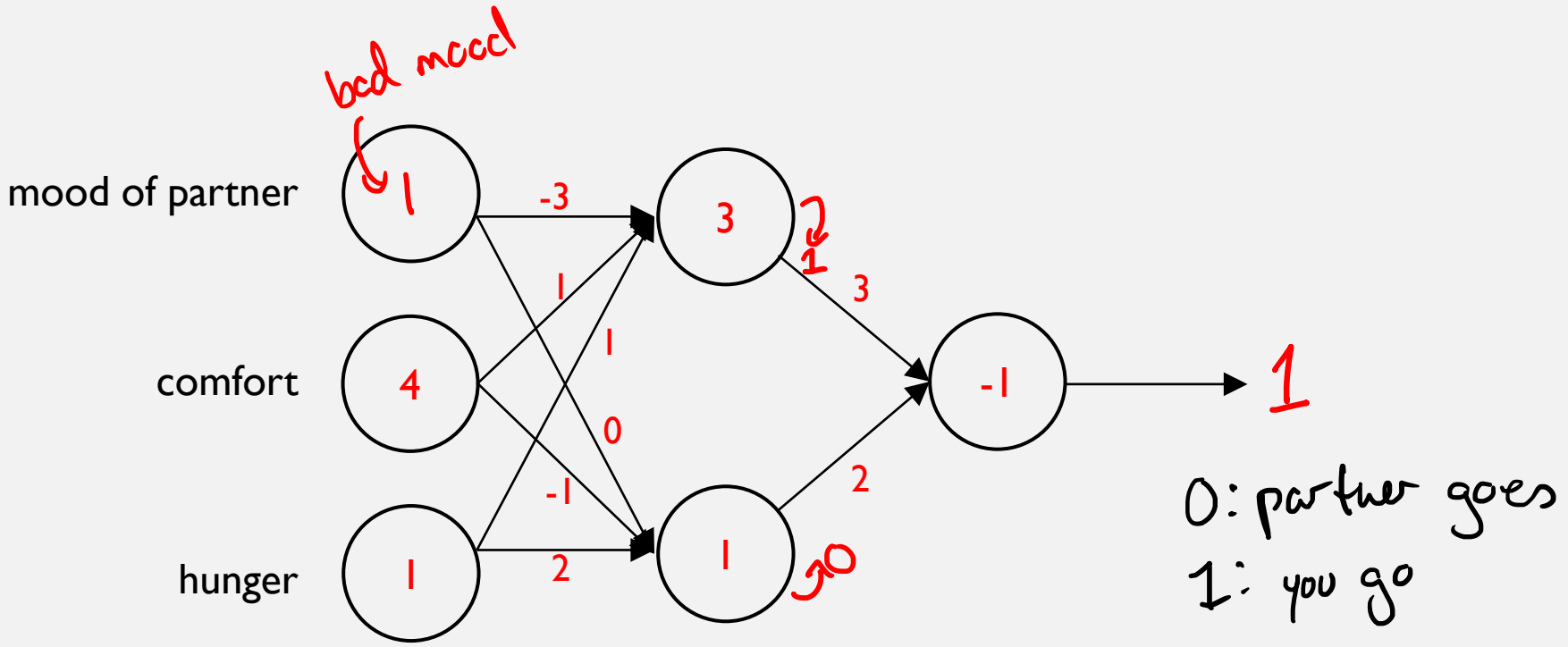
Is it really worth it to get up again to get some snacks?



THE SNACK EXAMPLE II

You decide that you absolutely want snacks.

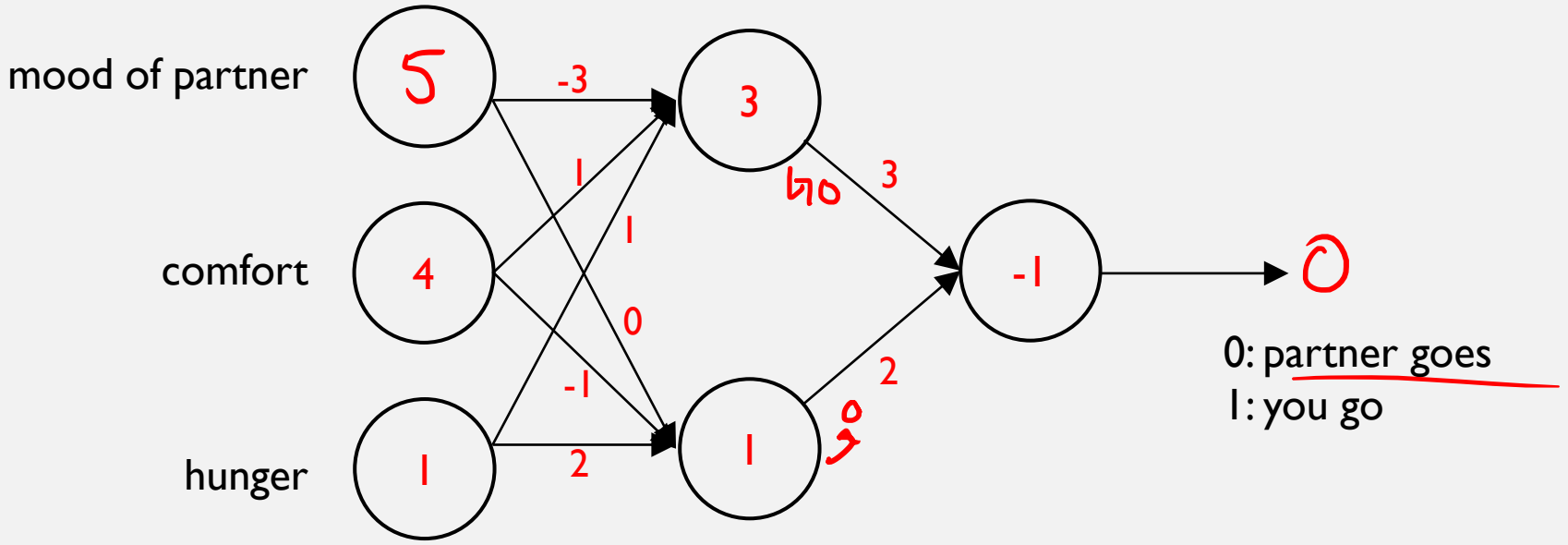
Luckily, your partner is not sitting in the couch!



THE SNACK EXAMPLE II

You decide that you absolutely want snacks.

Luckily, your partner is not sitting in the couch!



TRAINING A NEURAL NETWORK

means finding the best set of weights and biases for a given network architecture

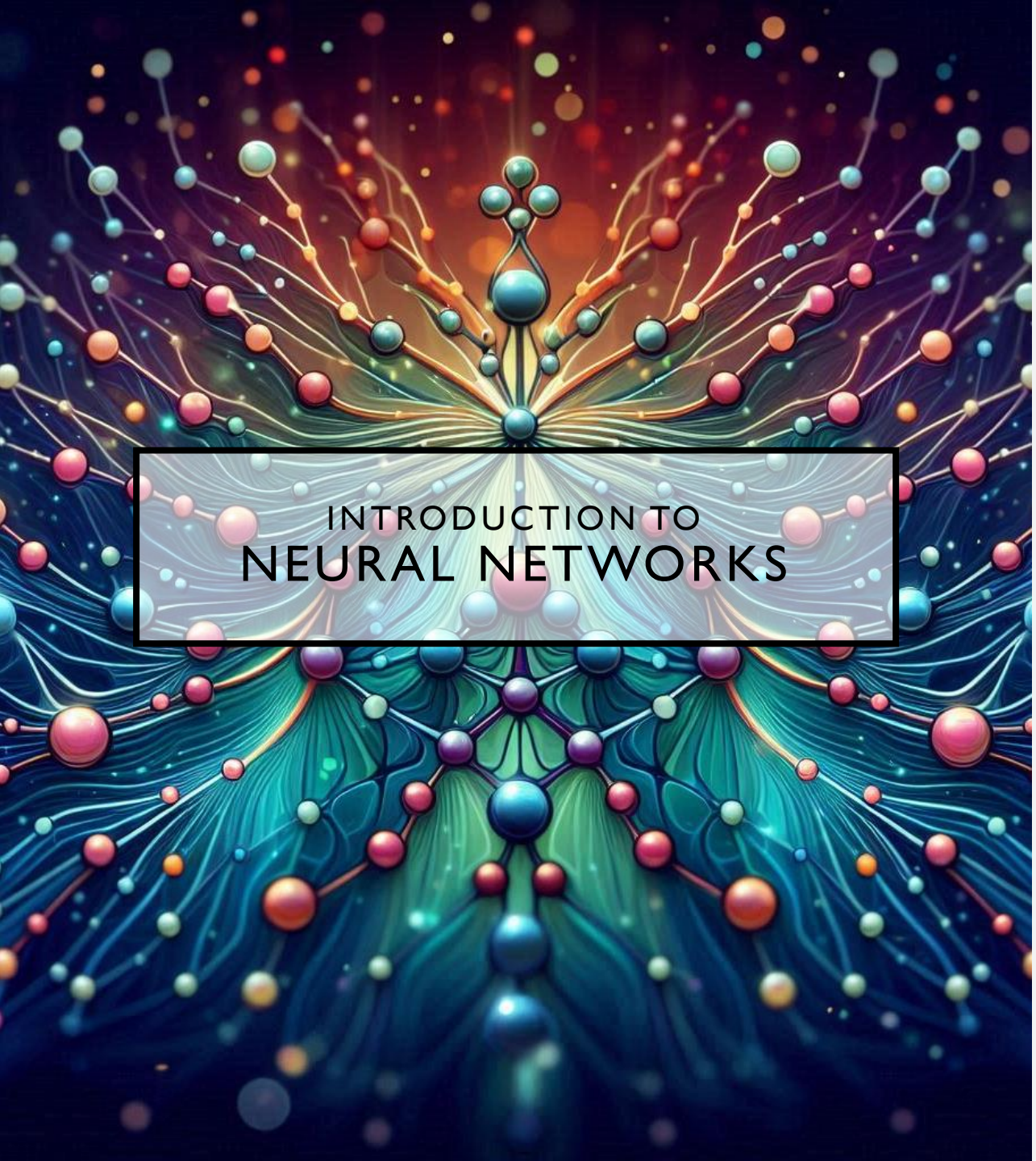
THIS LEAVES TWO QUESTIONS

1. How to structure the network

2. How to optimize weights & biases

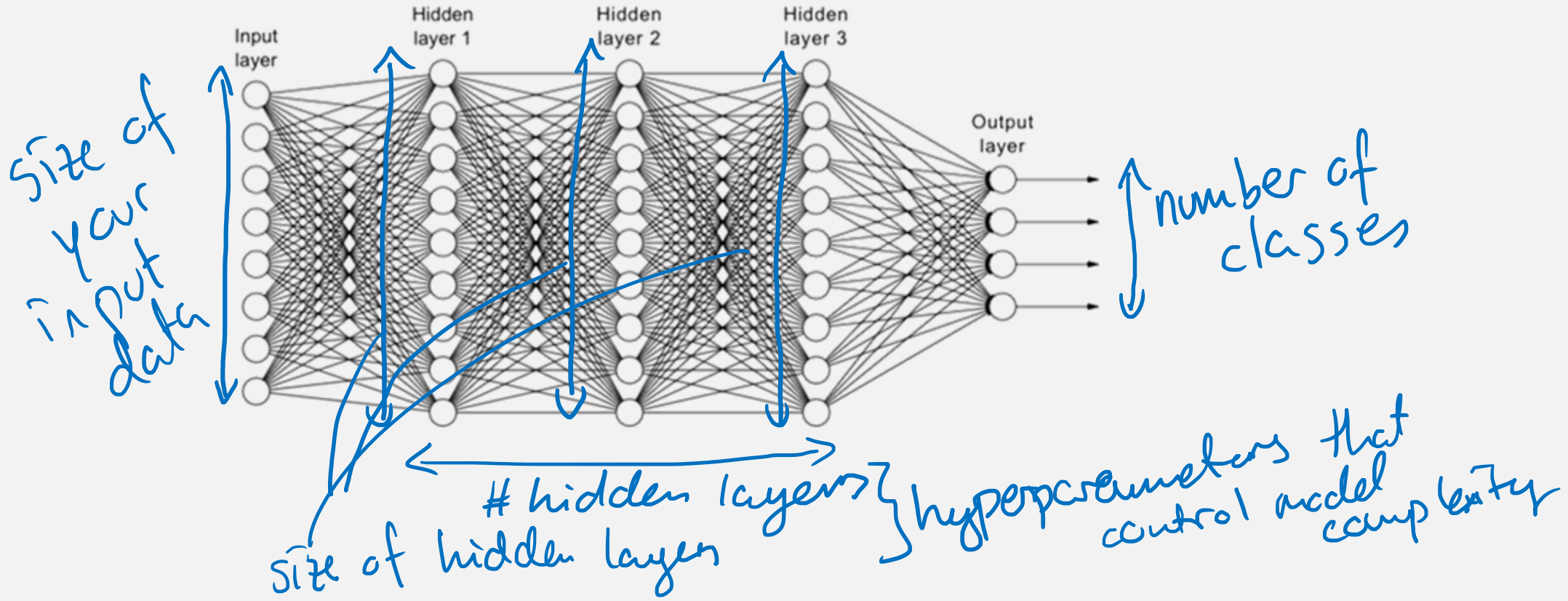
THE TENSORFLOW PLAYGROUND

playground.tensorflow.org



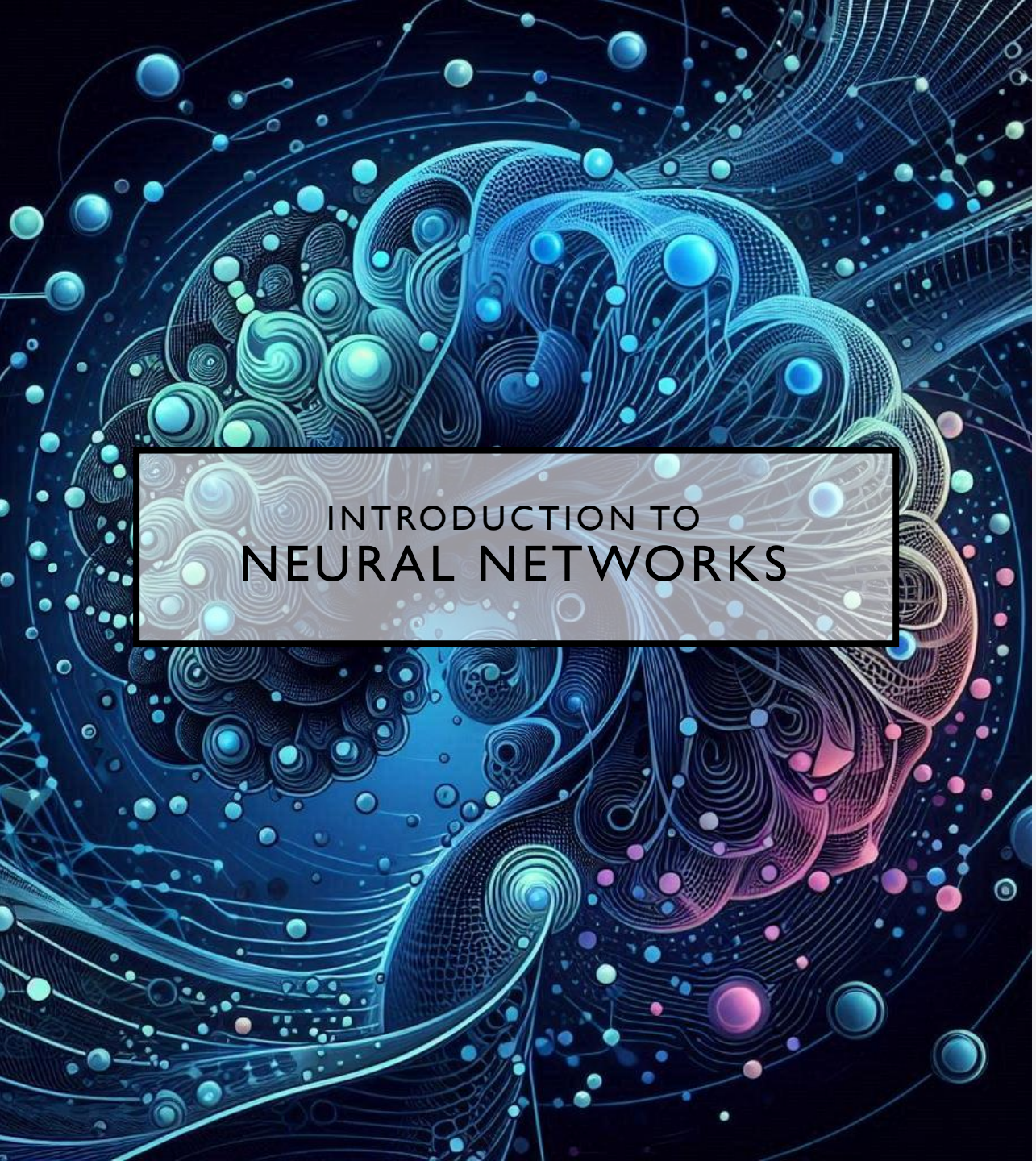
- What is a neural network?
- How do we structure it?
- How do we train it?
- How do we implement it?

HOW TO STRUCTURE THE NETWORK



neuron in input size > hidden size > output size

Rules of thumb: usually, one hidden layer is enough



- What is a neural network?
- How do we structure it?
- How do we train it?
- How do we implement it?

HOW TO OPTIMIZE WEIGHTS AND BIASES

Minimization problem

"Searching for the lowest point in a landscape"

Longitude & latitude: weights, biases

Altitude: 1-accuracy or something similar

GRADIENT DESCENT

1. Find the direction in which the descent is steepest

$$\nabla L(\beta) = \left[\frac{\partial L}{\partial \beta_0}(\beta), \frac{\partial L}{\partial \beta_1}(\beta), \ldots\right]$$

gradient "altitude" ↳ points ↳ weights & biases

2. Take a step in that direction

$$\beta \leftarrow \beta - \eta \nabla L(\beta)$$

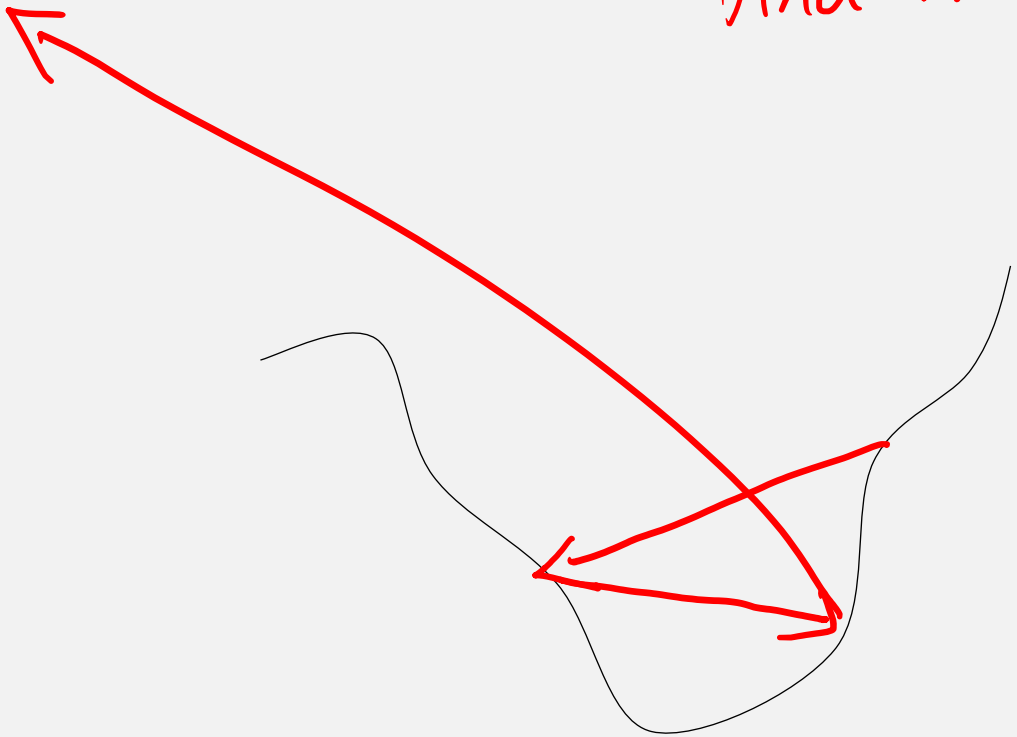
↳ learning rate (step size)

3. Repeat until you reach the bottom

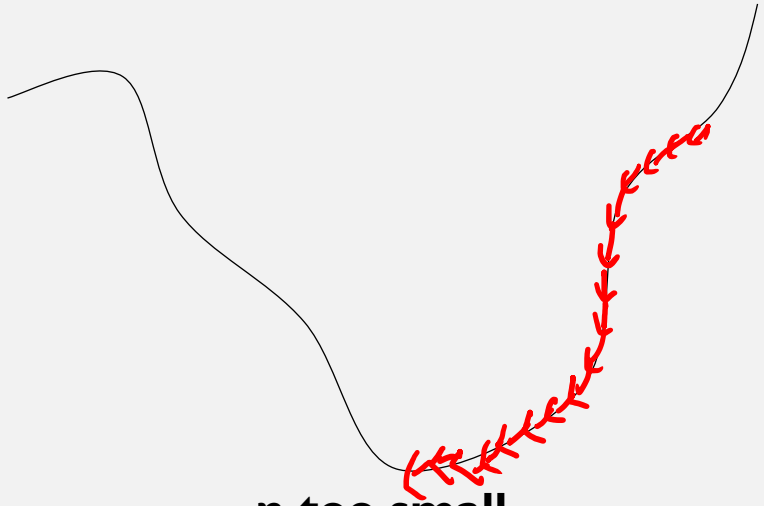
$$\nabla L(\beta) = [0, 0, 0, \ldots 0]$$

THE LEARNING RATE

find it by trial-and-error



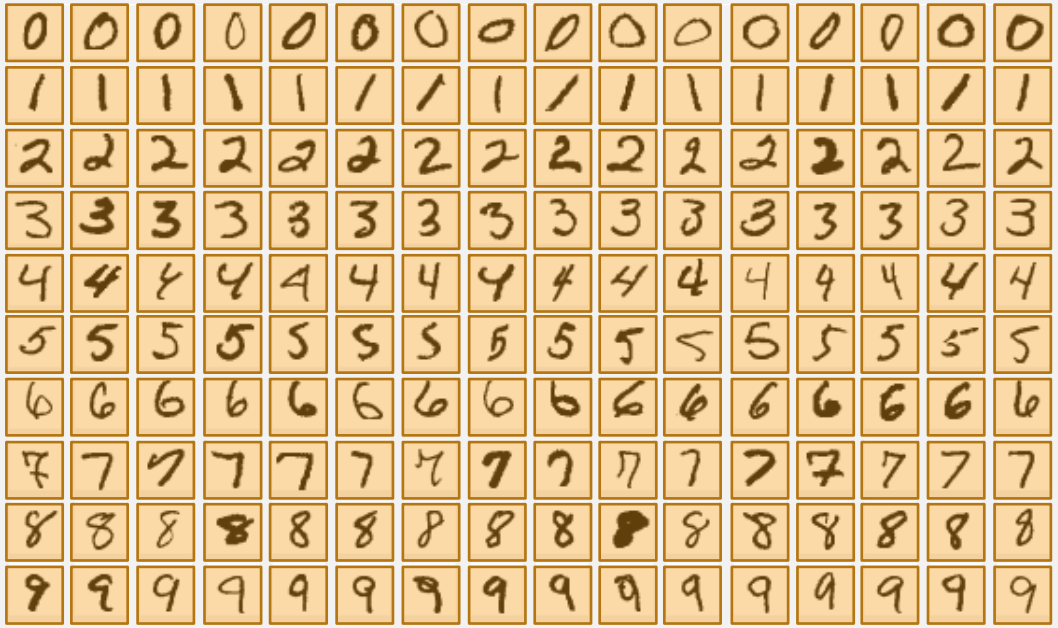
η too big



η too small

STOCHASTIC GRADIENT DESCENT

Don't update weights and biases based on **all** your data every time. Instead,



When you have gone through all your samples, you finish a **training epoch.**

BUT WHAT IF …

landscapes are mostly flat but a small step leads to a sudden change

SUDDEN CHANGES

1. Perceptrons suddenly change from 0 to 1

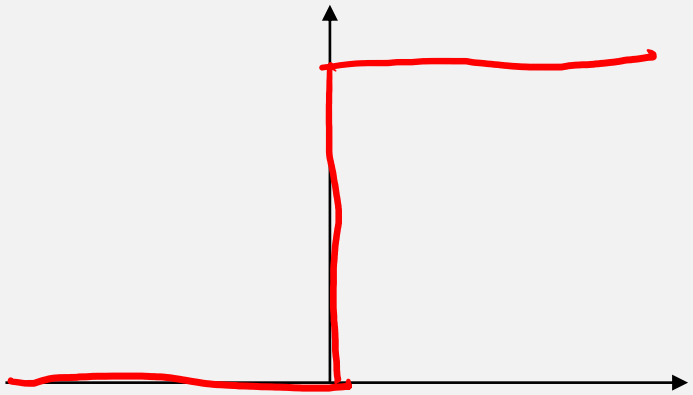
⇒ don't use perceptrons

2. The number of misclassifications (1-accuracy) suddenly change when a perceptron changes its mind

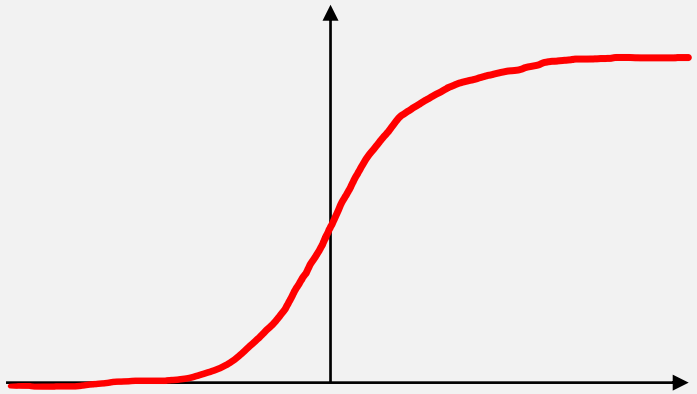
⇒ don't use accuracy

FIXING THE PERCEPTRON PROBLEM

$$\text{output} = \begin{cases} 0 & \text{if} \quad \boldsymbol{wx} + b \leq 0 \\ 1 & \text{if} \quad \boldsymbol{wx} + b > 0 \end{cases}$$



$$\text{output} = \frac{1}{1 + e^{-(\boldsymbol{wx}+b)}}$$

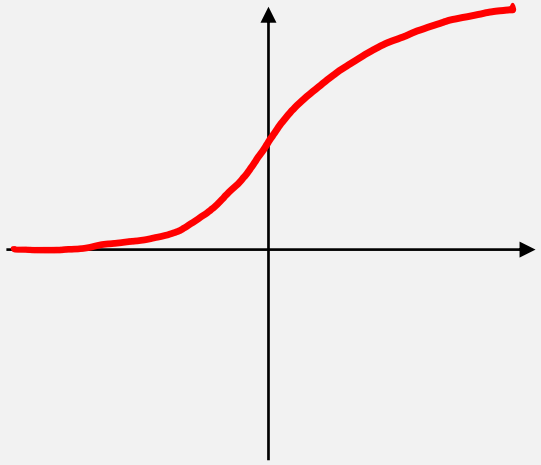


Changes smoothly

DIFFERENT ACTIVATION FUNCTIONS

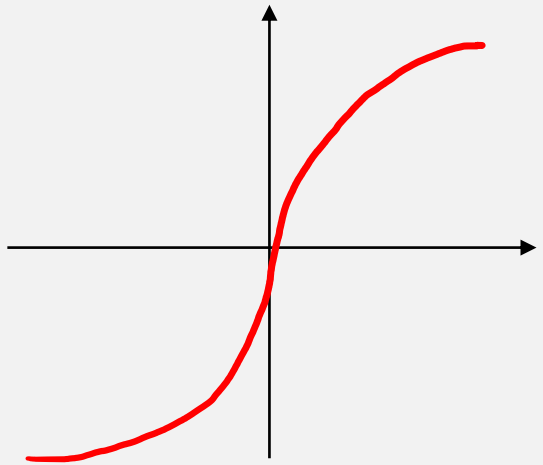
sigmoid

$$\frac{1}{1+e^{-z}}$$



tanh

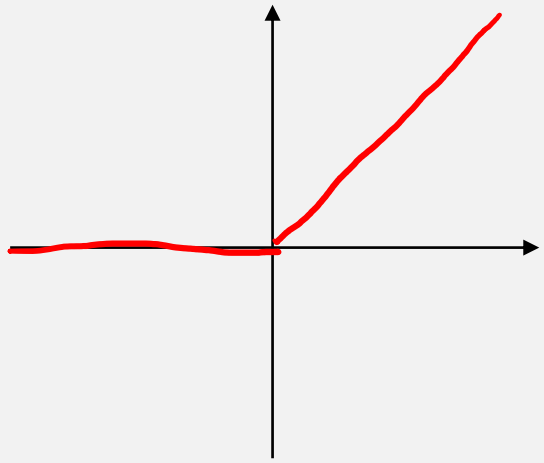
$\tanh(z)$



ReLU

"Rectified linear unit"

$\max(0,z)$



ACTIVATION IN THE OUTPUT LAYER

$z = \vec{w} \cdot \vec{x} + b$

$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

output layer

class A	5
class B	2.5
class C	0.5

softmax ⟹

0.92	→ probability of belonging to each class
0.07	sums to 1
0.01	

FIXING THE ACCURACY PROBLEM

Define a loss function L

true label (class 2) → $y(x_i) = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$

output of network → $a(x_i, w, b) = \begin{bmatrix} 0.1 \\ 0.8 \\ 0.07 \\ 0.03 \end{bmatrix}$

$$L(w,b) = \frac{1}{2n} \sum_i \| y(x_i) - a(x_i, w, b) \|^2$$

training data points

THE LOSS FUNCTION

The quadratic loss function captures the idea

$$L(\boldsymbol{w}, \boldsymbol{b}) = \frac{1}{2n} \sum_{x} \left|\left| y(x) - a(x, \boldsymbol{w}, \boldsymbol{b}) \right|\right|^2$$

but usually we use the cross-entropy loss function

$$L(w,b) = -\frac{1}{n} \sum_{j} y(x_j) \cdot \ln\left(a(x_j, w, b)\right)$$

(dot product)

Example from before

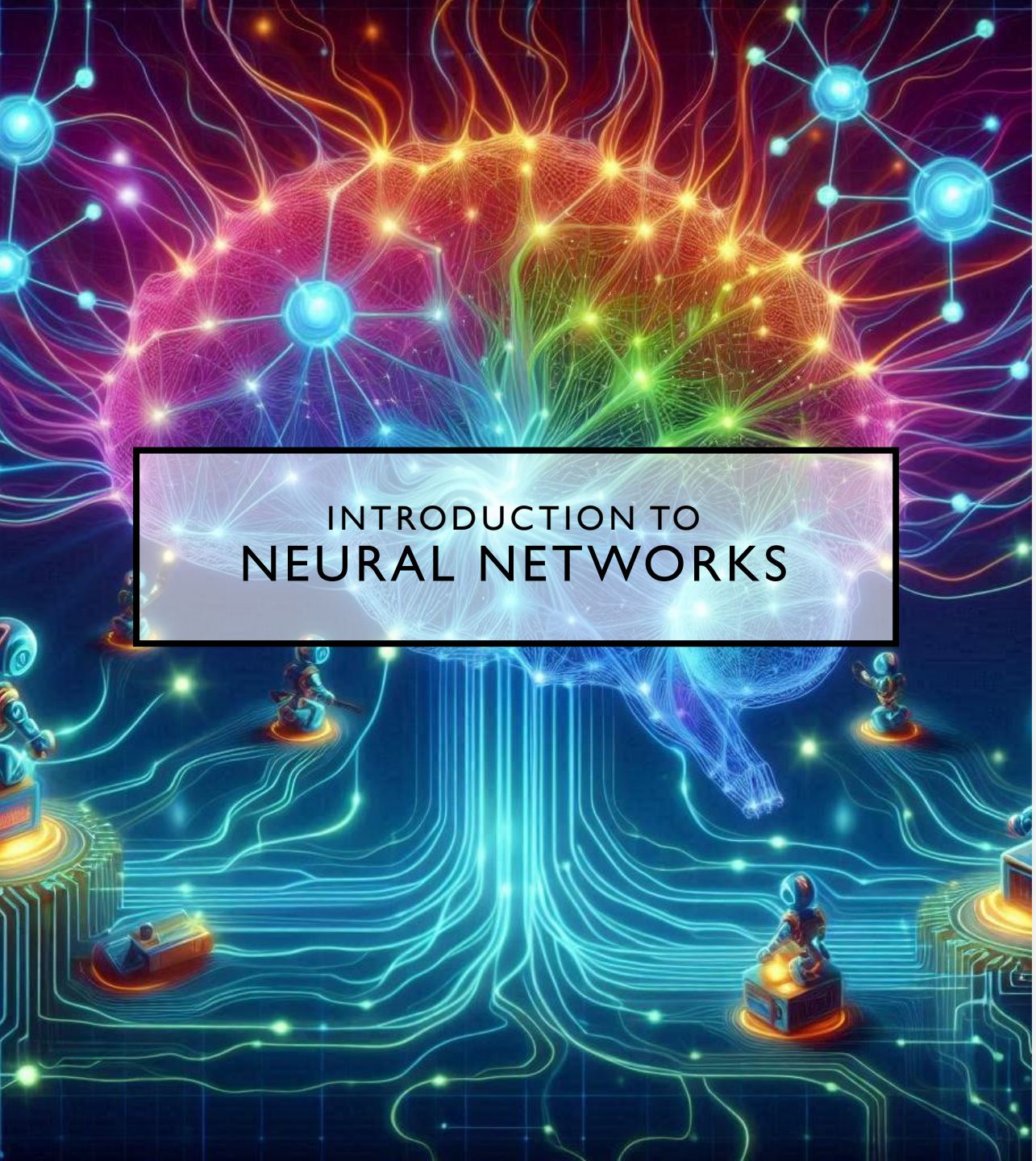
$$y(x_1) = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

$$a(x_1, \boldsymbol{w}, \boldsymbol{b}) = \begin{bmatrix} 0.1 \\ 0.8 \\ 0.07 \\ 0.03 \end{bmatrix}$$

$$y(x_1) \cdot \ln\left(a(x_1, w, b)\right) = 0 \times \ln 0{,}1 + 1 \times \ln 0{,}8 + 0 \times \ldots + 0 \times \ldots$$

$$= 1 \times \ln 0{,}8 = -0{,}223$$

proportional to "how wrong we are"



- What is a neural network?
- How do we structure it?
- How do we train it?
- How do we implement it?

LET'S TRY TO MAKE ONE

Jupyter Notebook **Neural networks - Digits**