Artificial neural networks with Python

Juan Eloy Arco Martín, jearco@ugr.es

SEPNECA

cimcyc

Centro de Investigación Mente, Cerebro y Comportamiento

Why?

- Great impact on our daily lives:
  - Voice recognition (Siri, Alexa).
  - Recommendation systems on streaming platforms.
  - Collision prevention in the automotive industry.

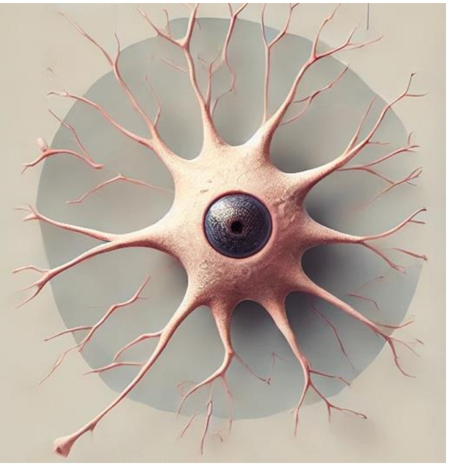
- Connection between Neuroscience and Artificial Intelligence:
  - Biological inspiration in the design of these types of networks.
  - Potential to use these models to explore brain patterns.

What will we learn today?

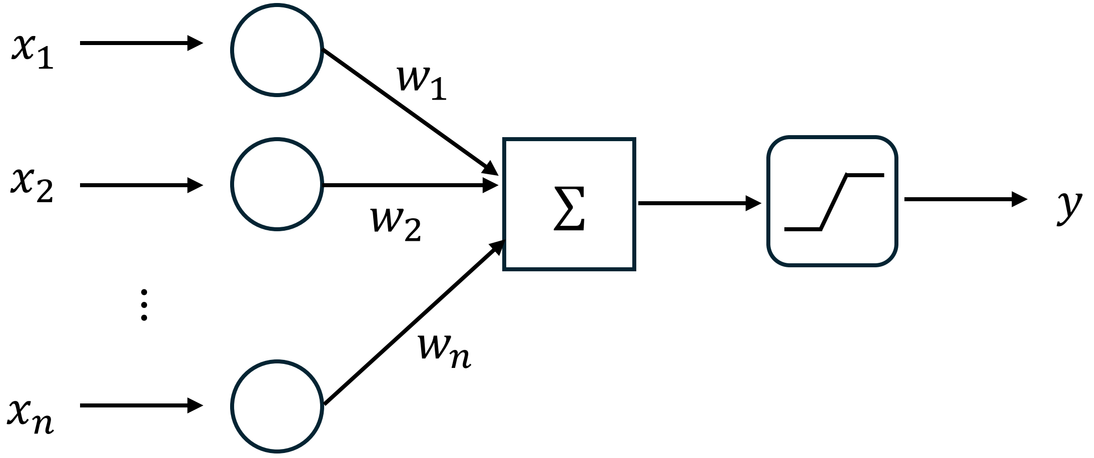
- What are Artificial Neural Networks?
- How do they work?
- How do they “learn”?
- Examples of network types
  - Multilayer Perceptron (MLP).
  - Convolutional Neural Network (CNN).
  - Autoencoders
- Applications

Artificial Neural Networks

- They are mathematical models inspired by biological neural networks, designed to **process data** and **learn patterns**.
- Artificial neurons are much simpler and operate using mathematical operations instead of bioelectrical signals.



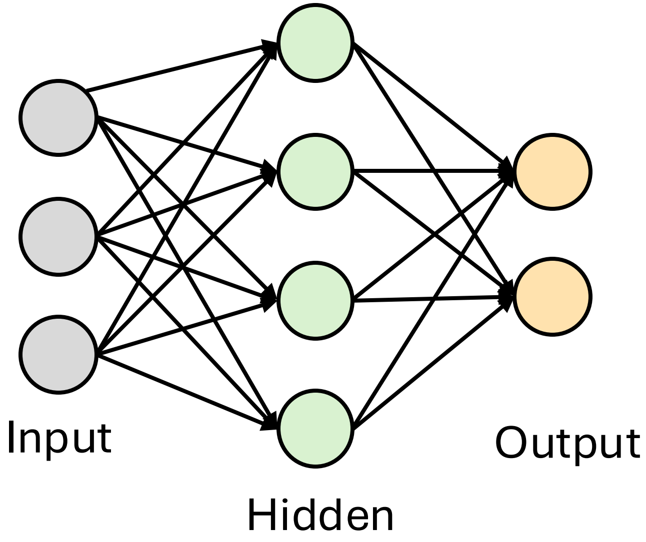
Biological neuron



Artificial neuron

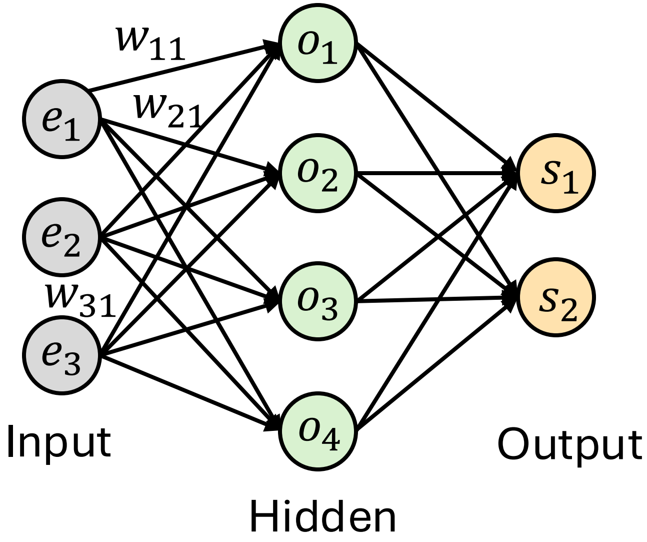
Artificial Neural Networks

- A group of interconnected neurons forms a **neural network**.
- There are three different types of layers:
  - Input layer: receives the initial data.
  - Hidden layers: perform internal transformations and extract patterns.
  - Output layer: produces the final response (classification, prediction, etc).



Artificial Neural Networks

- Neurons in one layer process information from the neurons in the previous layer through a weighted sum: all inputs are added together, but not all have the same weight.



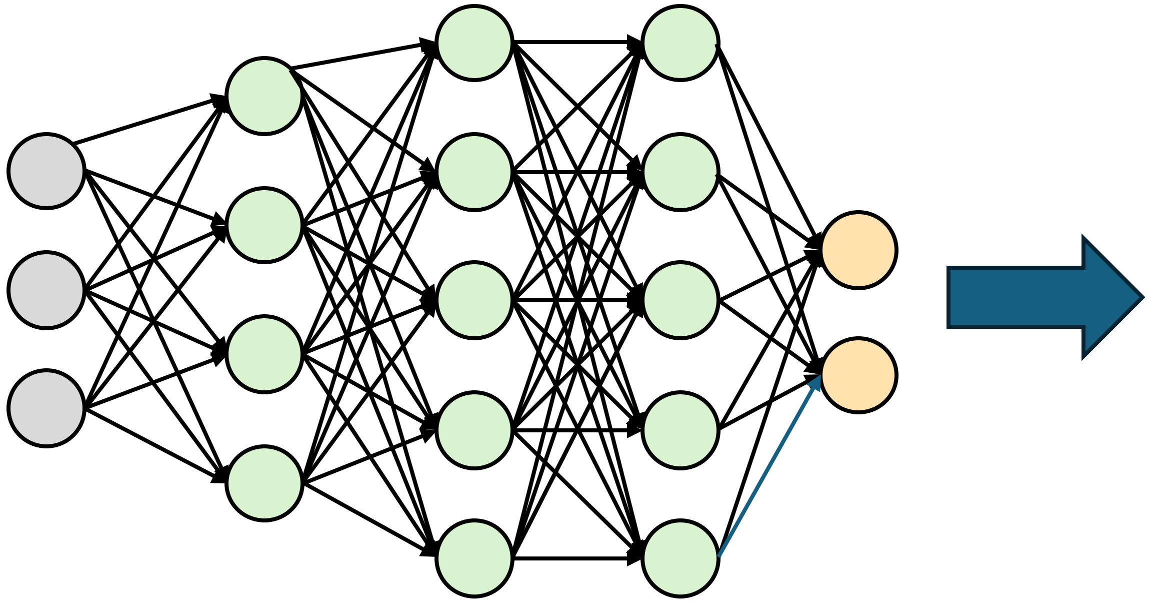
$$\left.\begin{aligned}
o_1 &= w_{11}e_1 + w_{21}e_2 + w_{31}e_3 \\
o_2 &= w_{12}e_1 + w_{22}e_2 + w_{32}e_3 \\
o_3 &= w_{13}e_1 + w_{23}e_2 + w_{33}e_3 \\
o_4 &= w_{14}e_1 + w_{24}e_2 + w_{34}e_3
\end{aligned}\right\}$$

The **goal** is to **optimize the weights** to maximize the network's performance

Artificial Neural Networks

- In the hidden layer is where the “magic” happens:
  - Extraction of relevant information
  - Difficult to access or interpret what happens inside.

- This becomes especially concerning as the number of hidden layers increases.



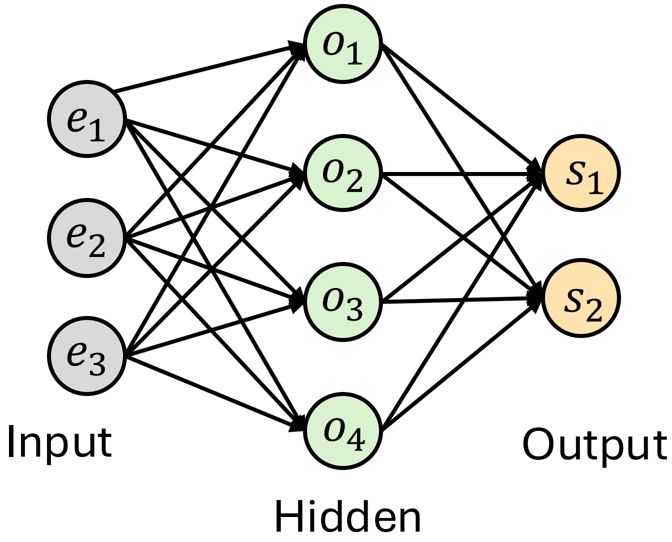
12 weights in the first hidden layer.
20 in the second.
25 in the third.

How do they “learn”?

- Forward propagation.
- Error computation through a loss function.
- Optimization via Gradient Descent.
- Backpropagation.

Forward propagation

- Information flows from the input layer to the hidden layer, and finally to the output layer.
- A bias term is added to the weighted sum.



$$o_1 = w_{11}e_1 + w_{21}e_2 + w_{31}e_3 + b_1$$

$$o_2 = w_{12}e_1 + w_{22}e_2 + w_{32}e_3 + b_2$$

$$o_3 = w_{13}e_1 + w_{23}e_2 + w_{33}e_3 + b_3$$

$$o_4 = w_{14}e_1 + w_{24}e_2 + w_{34}e_3 + b_4$$

Forward propagation

- In addition, an activation function is applied to introduce non-linearities into the model, allowing it to learn complex patterns.

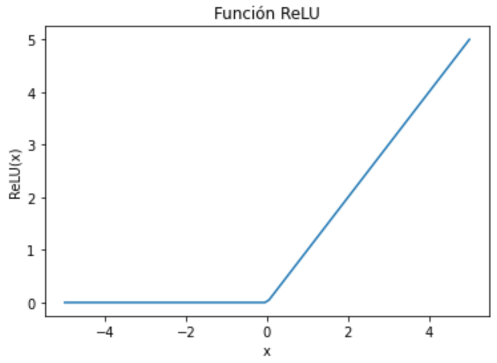
$$o_1 = f(w_{11}e_1 + w_{21}e_2 + w_{31}e_3 + b_1)$$

$$o_2 = f(w_{12}e_1 + w_{22}e_2 + w_{32}e_3 + b_2)$$

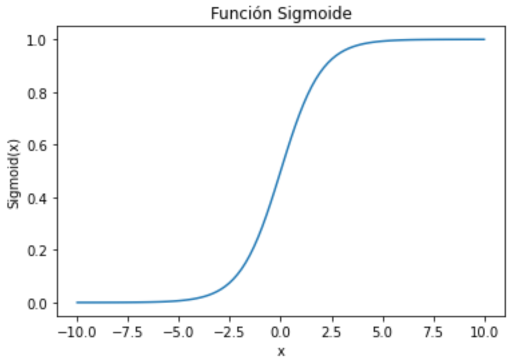
$$o_3 = f(w_{13}e_1 + w_{23}e_2 + w_{33}e_3 + b_3)$$

$$o_4 = f(w_{14}e_1 + w_{24}e_2 + w_{34}e_3 + b_4)$$

Examples of functions used in this context:



ReLU function



Sigmoid function

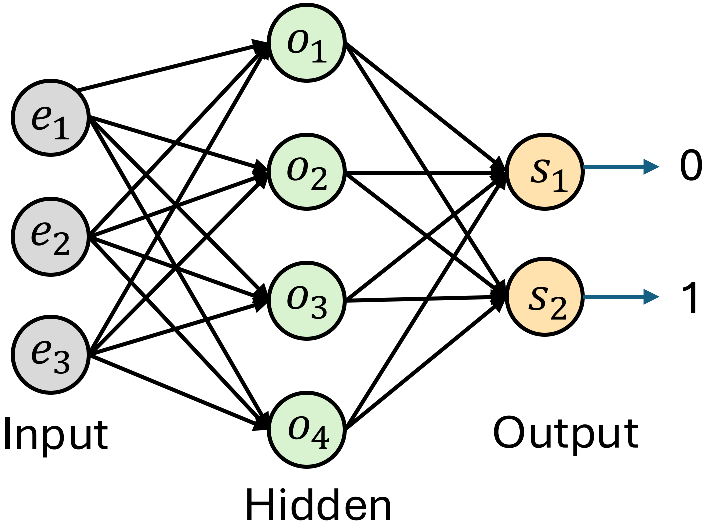
- Important: the weights are initialized randomly.

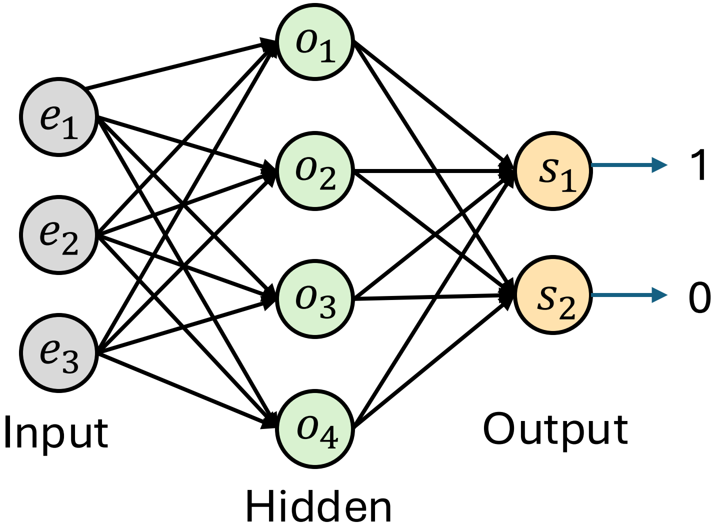
Forward propagation

- When the network generates an output, it is necessary to evaluate **how good** that output is.
- In binary classification, the target outputs are typically 0-1 for one class and 1-0 for the other.
- However, what the network actually produces at the output are **probabilities** indicating the likelihood that the input data belongs to one class or the other.

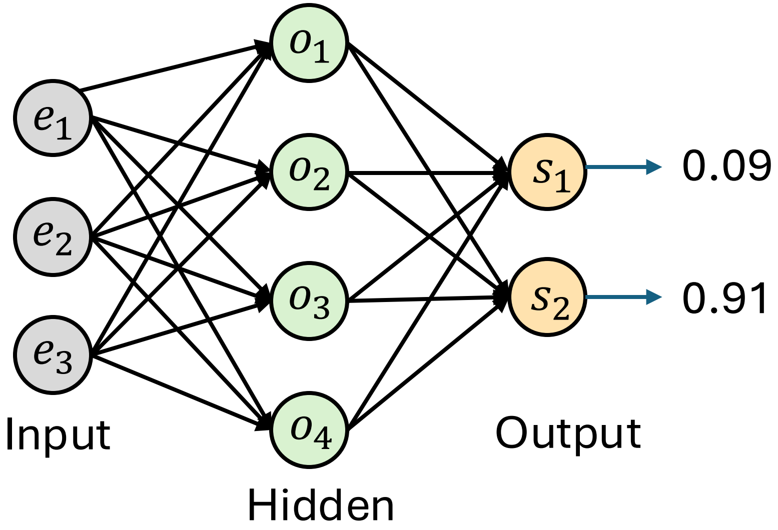
Example: A network to distinguish between images of dogs and cats.

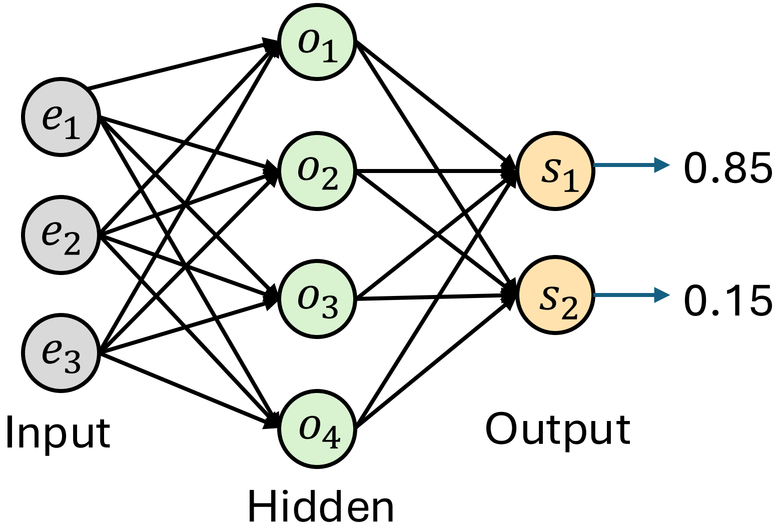
Forward propagation





Forward propagation





Loss function

- Since no network is perfect, how can we evaluate how wrong it is?
- Based on that evaluation, more or fewer adjustments need to be made to the network.
- **Loss function:** it measures the error between the network's output and ideal output.
- In binary classification, a common choice is **binary cross entropy**:

$$L_{BCE} = -\frac{1}{n}\sum_{i=1}^{n}\left(Y_i \cdot \log \hat{Y}_i + (1 - Y_i) \cdot \log\left(1 - \hat{Y}_i\right)\right)$$

Loss function

- Since no network is perfect, how can we evaluate how wrong it is?
- Based on that evaluation, more or fewer adjustments need to be made to the network.
- **Loss function:** it measures the error between the network's output and ideal output.
- In binary classification, a common choice is **binary cross entropy**:

$$L_{BCE} = -\frac{1}{n}\sum_{i=1}^{n}\left(Y_i \cdot \log \hat{Y}_i + (1 - Y_i) \cdot \log\left(1 - \hat{Y}_i\right)\right)$$

"What actually comes out"
0.85-0.15

"What should come out"
0-1

Loss function

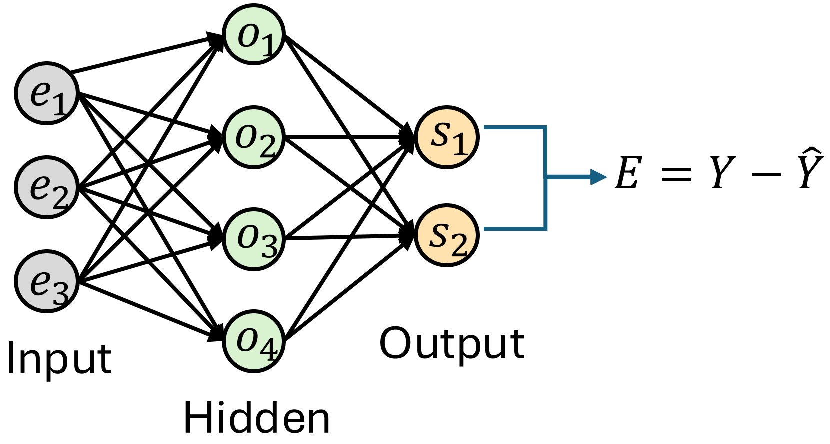
- We now know the output of our model and how close it is to the desired result.
- The goal in training any neural network is to **minimize the difference** between the predicted output and the actual output.
- Knowing this, we adjust the network's parameters (weights and biases) to try to reduce the error.
- But how?
  - Should we increase the weights?
  - Should we decrease them?
  - And most importantly, which ones? All of them? Just random ones?
- The optimization of the weights is done through a process called **backpropagation**.

Backpropagation

- At first, we only know the influence on the error of the neurons in **the last layer** of the network.

$$\frac{\partial E}{\partial \hat{Y}} = (Y - \hat{Y})$$

- However, there are mathematical concepts such as the **chain rule** that can be used to calculate how the error changes across the rest of the neurons in the network.

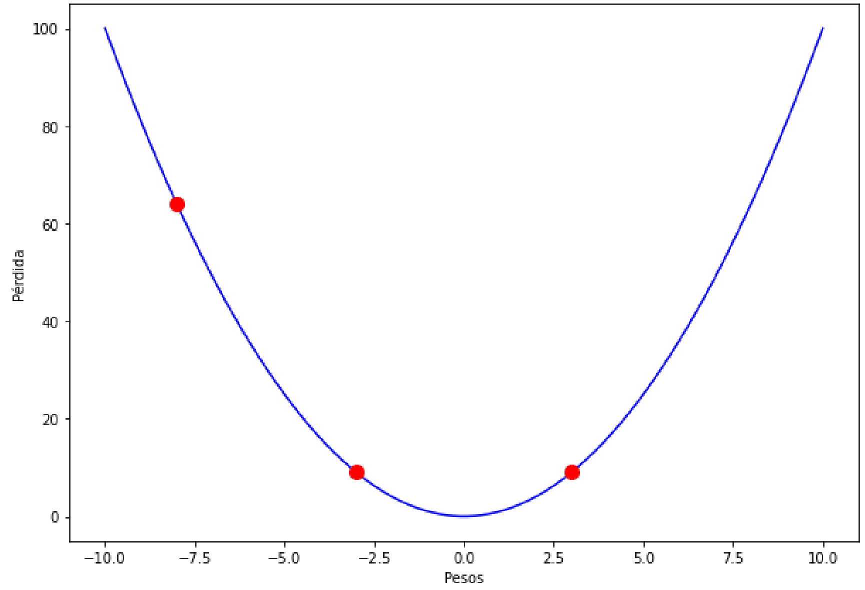


$$\frac{\partial E}{\partial o_1} = \frac{\partial E}{\partial \hat{Y}} \cdot \frac{\partial \hat{Y}}{\partial o_1}$$

$$\frac{\partial E}{\partial o_2} = \frac{\partial E}{\partial \hat{Y}} \cdot \frac{\partial \hat{Y}}{\partial o_2}$$

Gradient descent

- The gradient tells us how the error changes with respect to each weight.
  - If the loss increases as the weight increase ---> the gradient is positive.
  - It the loss decreases as the weight increases ---> the gradient is negative.
- This tells us **which direction** to move in, while the **learning rate** determines **how much** to move.



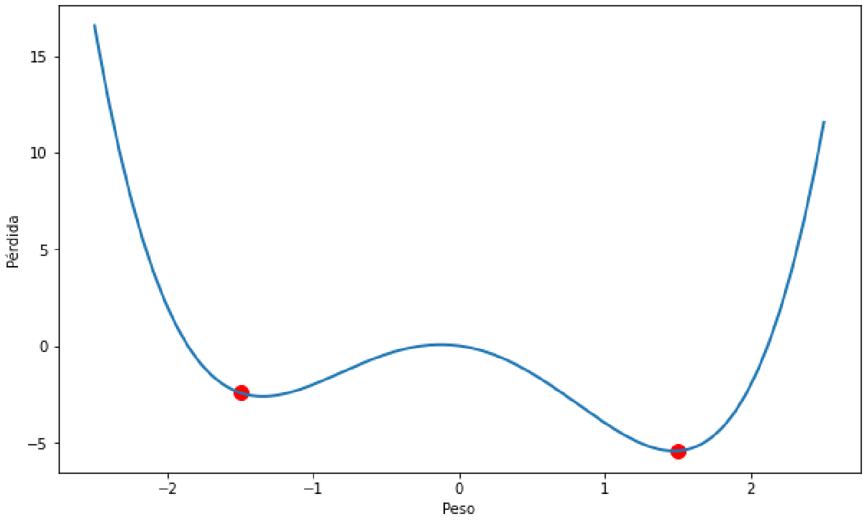
It is important to set a value that is **neither too small nor too large**

Gradient descent

- This process is repeated for each sample in our dataset (e.g. dog/cat images in our examples).
- Once all samples have been processed, we say that **one epoch** has passed.
- This is another parameter that must be set in advance.
- After each epoch, the network's parameters are updated. So, if there are 100 epochs, each sample passes through the network 100 times, and the parameters are updated 100 times.

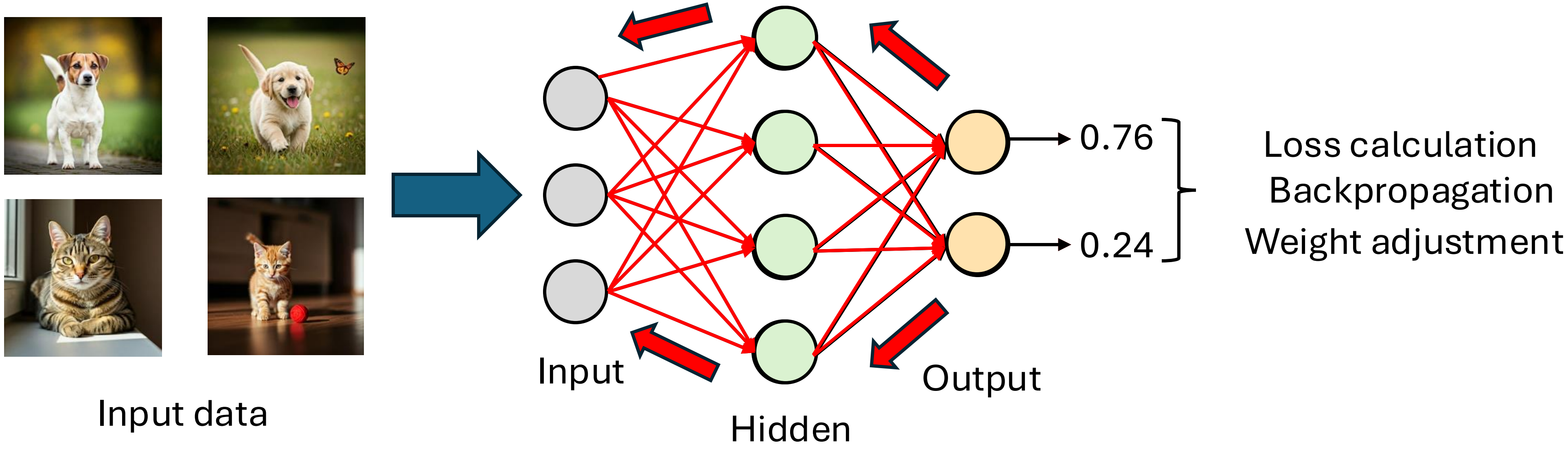
Gradient descent

- The shape of the loss function is never that simple.
- It is possible to reach a **suboptimal result** because the algorithm may find a **local minimum** rather than the global **minimum** of the function.



With the use of **optimizers**, the learning is adjusted **automatically**, helping to prevent the optimization process from getting stuck in a **local minimum.**

Summary

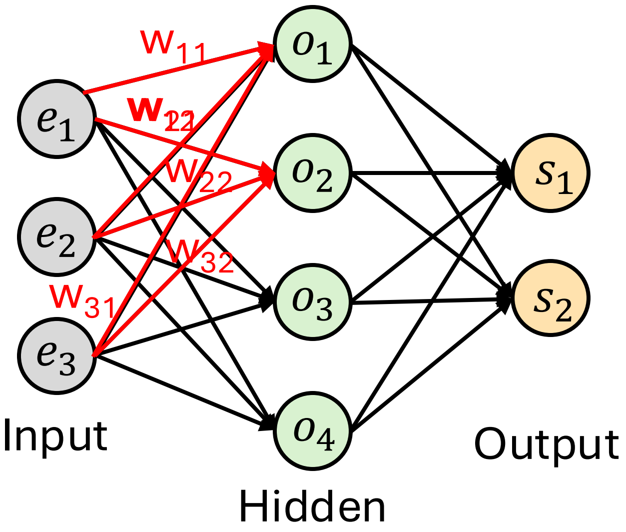

Input data
Input
Hidden
Output
0.76
0.24
Loss calculation
Backpropagation
Weight adjustment

What will we be learn today?

- What are Artificial Neural Networks?
- How do they work?
- How do they “learn”?
- **Examples of network types**
  - **Multilayer Perceptron (MLP).**
  - **Convolutional Neural Network (CNN).**
  - **Autoencoders**
- Applications

Multilayer Perceptron

- It is a fully connected network: all neurons in one layer are connected to every neuron in the next layer.
- The output of each neuron is calculated based on the **dot product** of the inputs and the weights of each connection.



$$o_1 = \begin{bmatrix} e_1 & e_2 & e_3 \end{bmatrix} \bullet \begin{bmatrix} w_{11} \\ w_{21} \\ w_{31} \end{bmatrix} = e_1 w_{11} + e_2 w_{21} + e_3 w_{31} = \text{x}$$

$$o_2 = \begin{bmatrix} e_1 & e_2 & e_3 \end{bmatrix} \bullet \begin{bmatrix} w_{12} \\ w_{22} \\ w_{32} \end{bmatrix} = e_1 w_{12} + e_2 w_{22} + e_3 w_{32} = \text{y}$$

Convolutional Neural Network

- In this case, not all neurons in one layer are connected to the neurons in the next layer.
- This greatly **reduces the number of weights per layer**, making this type of network especially effective for **image processing.**

1	4	15	8	5	12
19	12	0	1	17	29
54	2	33	36	20	7
35	17	9	10	19	40
23	2	18	6	35	11
3	44	16	20	37	48

Convolutional Neural Network

- Convolution is similar to a dot product, but it's a **sliding operation**, where a **kernel size** must be defined in advance.

Kernel

I	II
III	IV

Input image

1	4	15	8	5	12
19	12	0	1	17	29
54	2	33	36	20	7
35	17	9	10	19	40
23	2	18	6	35	11
3	44	16	20	37	48

$\mathrm{I} \cdot 1 + \mathrm{II} \cdot 4 + \mathrm{III} \cdot 19 + \mathrm{IV} \cdot 12 = 25$

$\mathrm{I} \cdot 15 + \mathrm{II} \cdot 8 + \mathrm{III} \cdot 0 + \mathrm{IV} \cdot 1 = 8$

$\mathrm{I} \cdot 5 + \mathrm{II} \cdot 12 + \mathrm{III} \cdot 17 + \mathrm{IV} \cdot 29 = 19$

$\vdots$

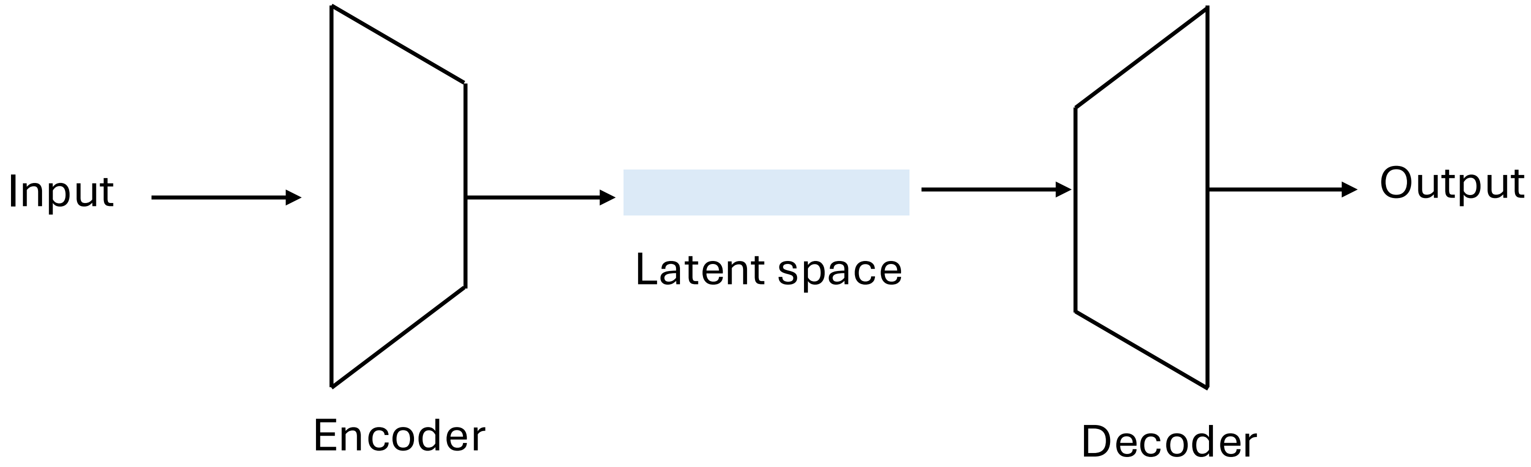
$\mathrm{I} \cdot 35 + \mathrm{II} \cdot 11 + \mathrm{III} \cdot 37 + \mathrm{IV} \cdot 48 = 15$

Output image

25	8	19
89	35	10
25	3	15

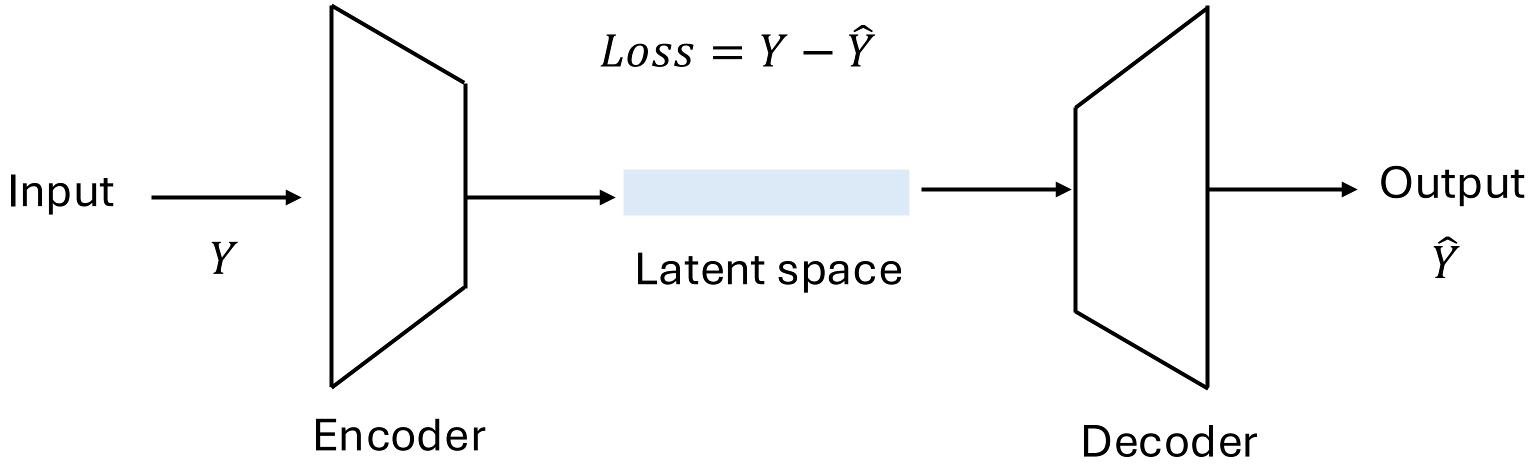
Autoencoder

- Originally designed for data compression, its structure consists of an encoder followed by a decoder.
- Inside the encoder, there are layers similar to those previously discussed: convolutional, fully connected, etc.
- The decoder **reverses** the operations performed by the encoder.



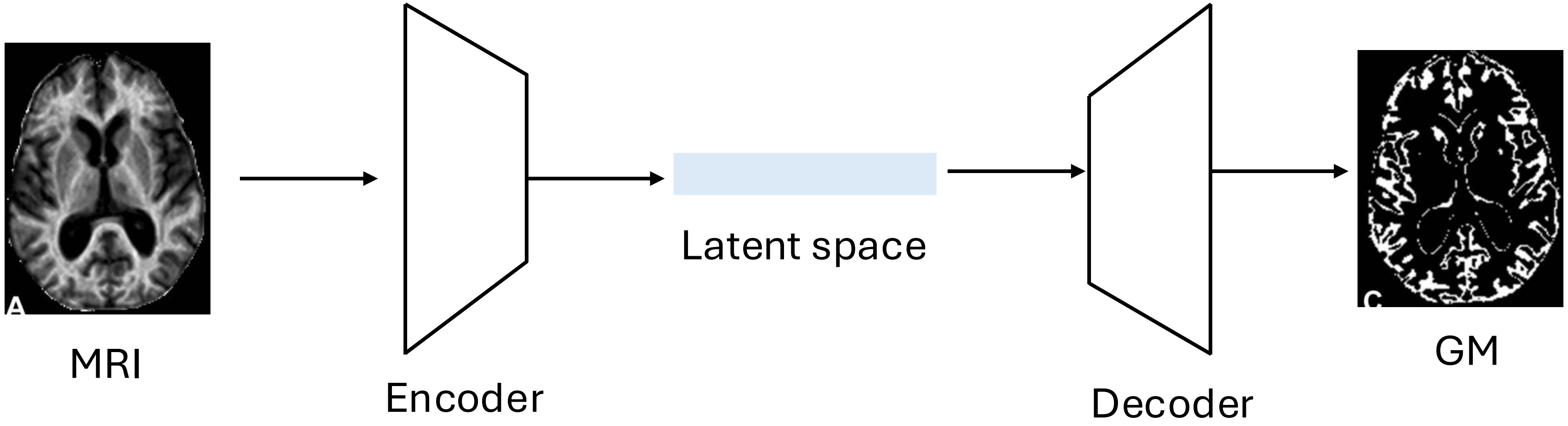
Autoencoder

- Its usefulness depends entirely on what the loss function is applied to. Example: compression for reducing file size.
- In this case, the loss function simply aims for the output to match the input.
- After training the network, the **latent space** contains a **much lower-dimensional representation** that still allows the input to be reconstructed.



Autoencoder

- Multiple applications
  - Gray Matter segmentation in Magnetic Resonance Imaging.



Autoencoder

- Modelling of complex relationships between brain features and non-neural behavioural measures.