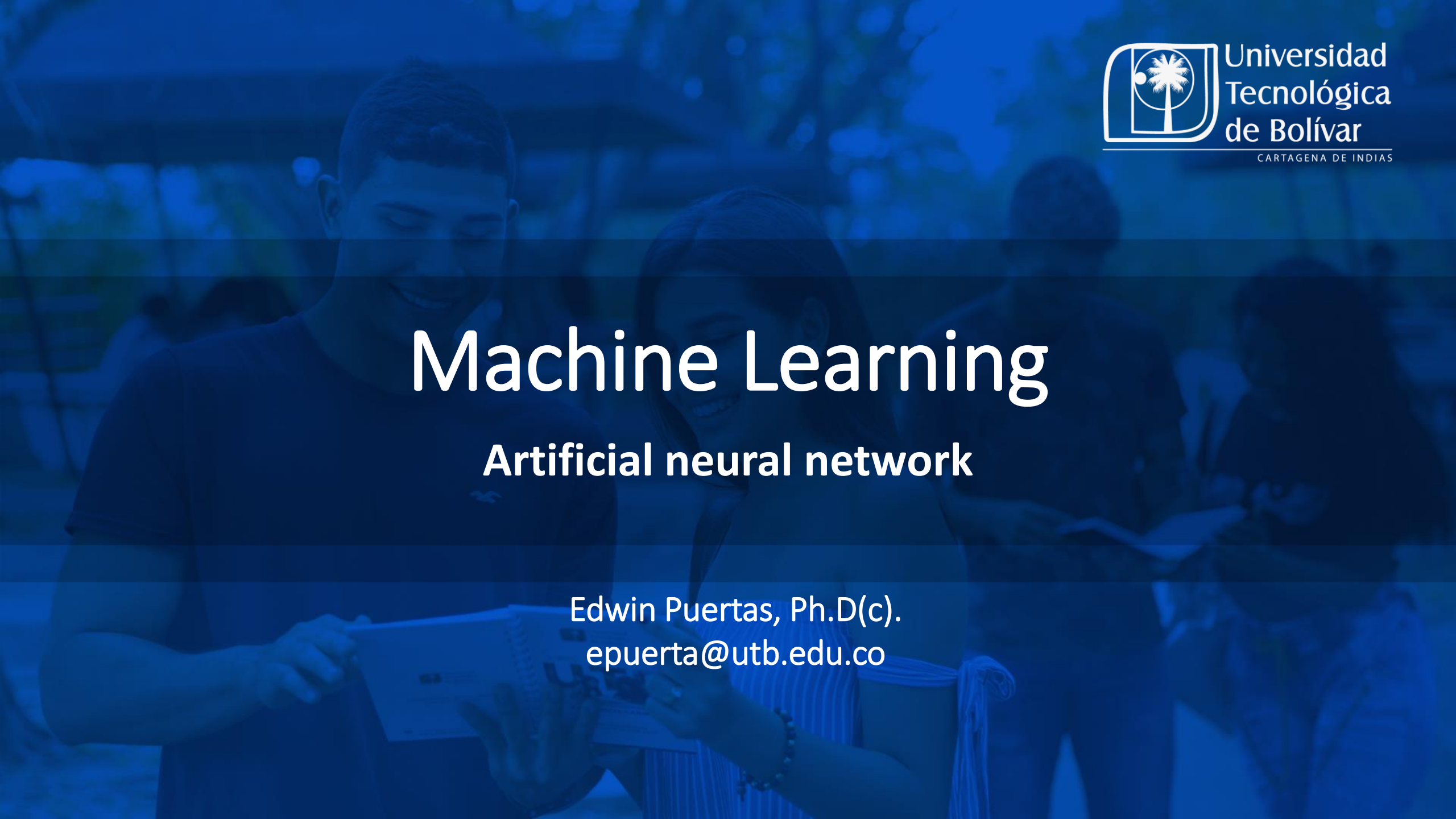


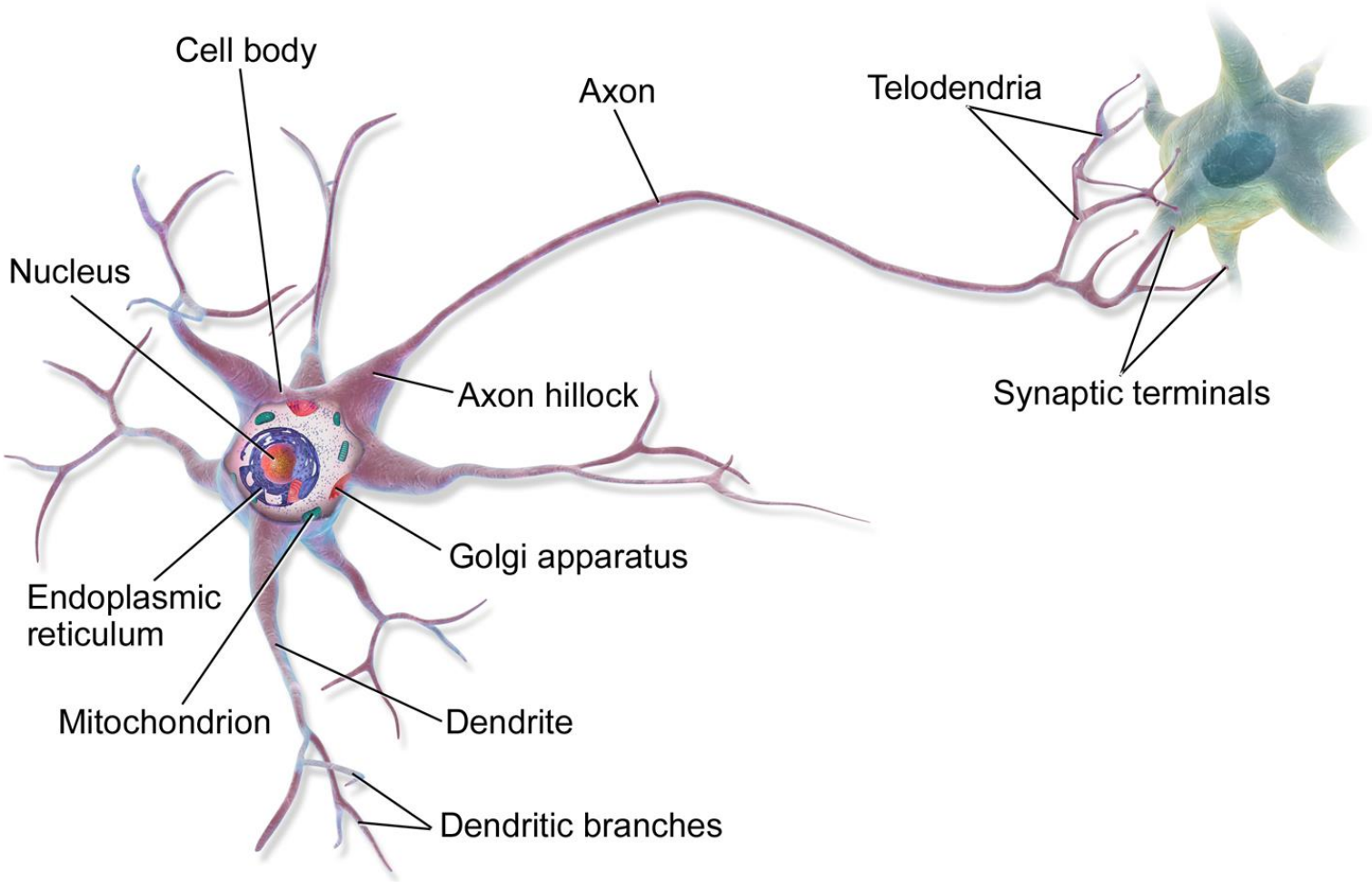
Universidad Tecnológica de Bolívar
CARTAGENA DE INDIAS

Machine Learning

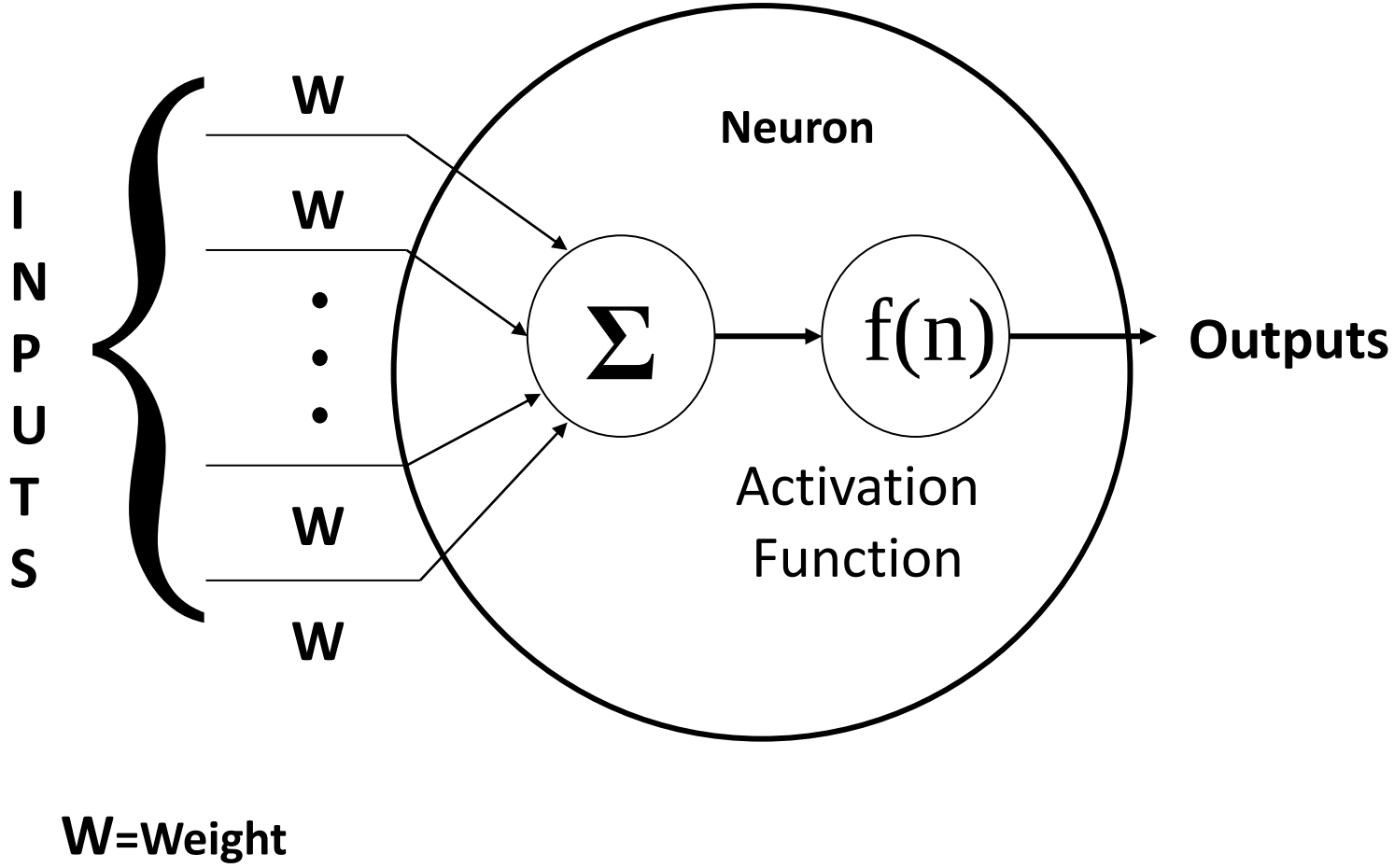
Artificial neural network

Edwin Puertas, Ph.D(c).
epuerta@utb.edu.co

Biological Neuron



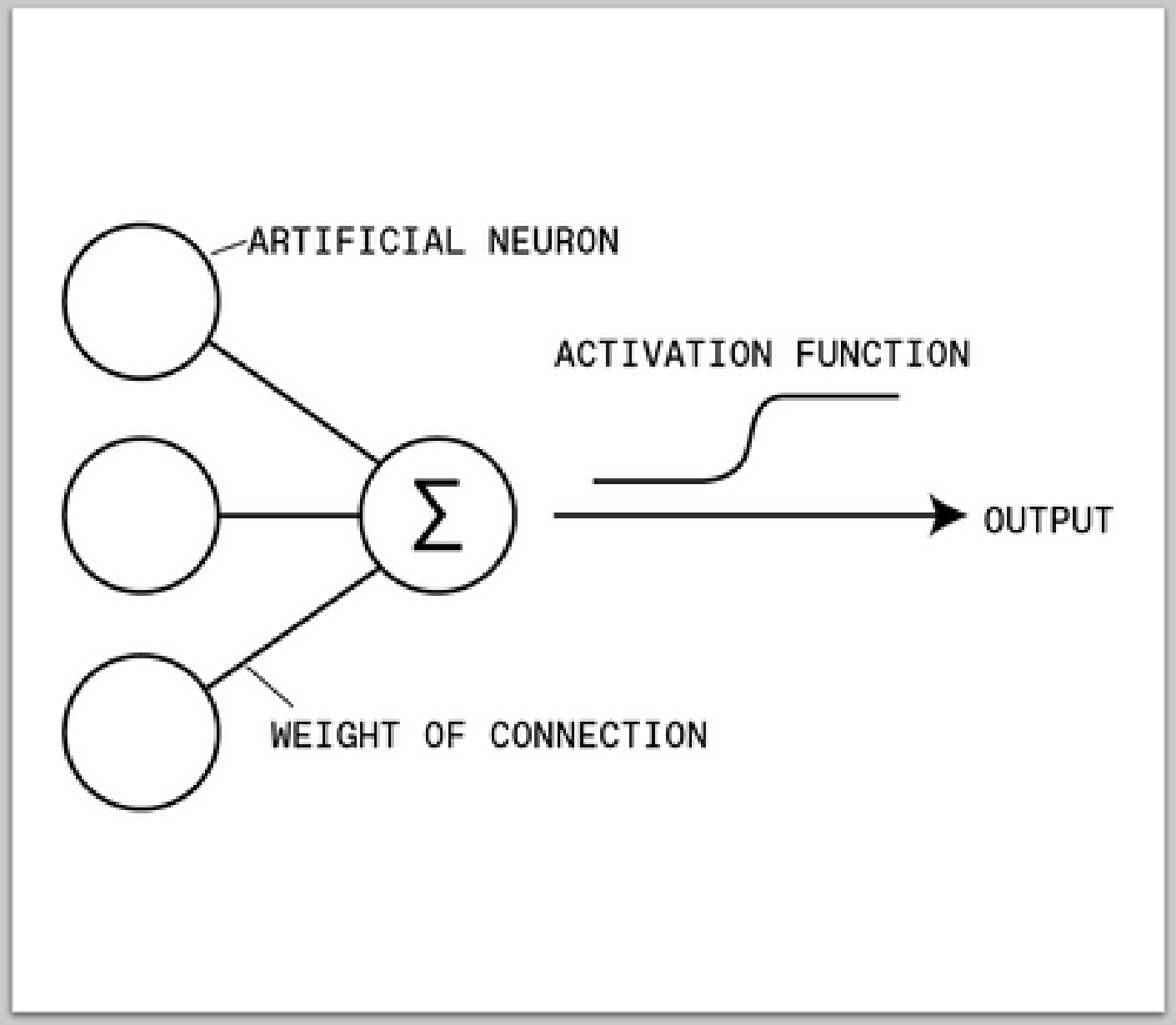
Artificial Neuron



W=Weight

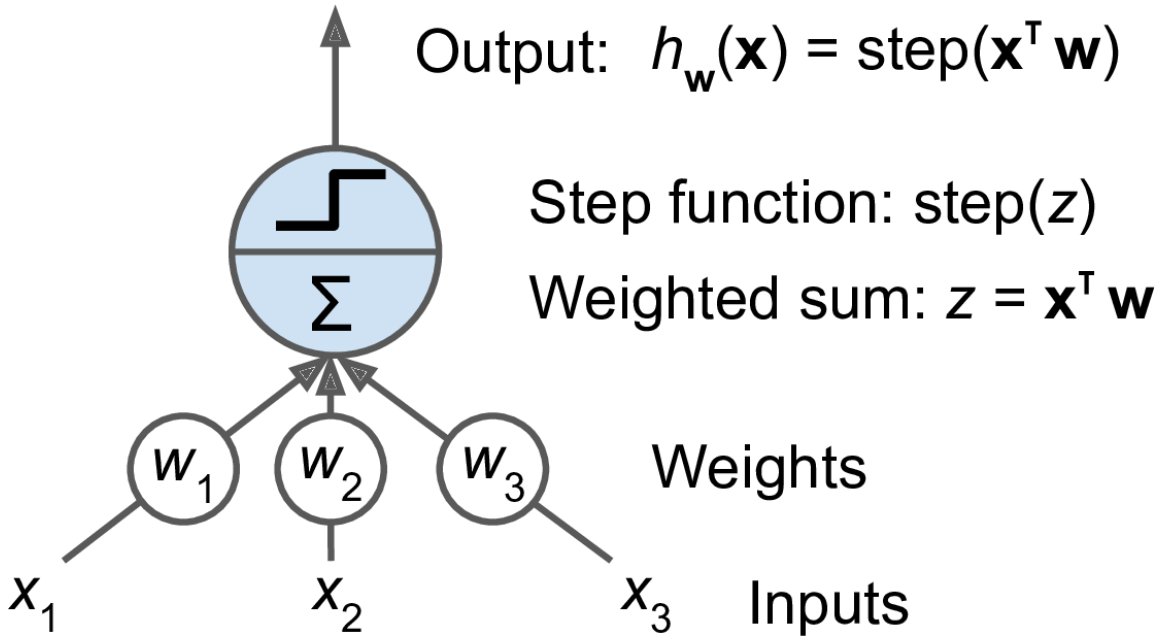
Architecture

- Each neuron in an artificial neural network sums its inputs and applies an activation function to determine its output.
- This architecture was inspired by what goes on in the brain, where neurons transmit signals between one another via synapses.

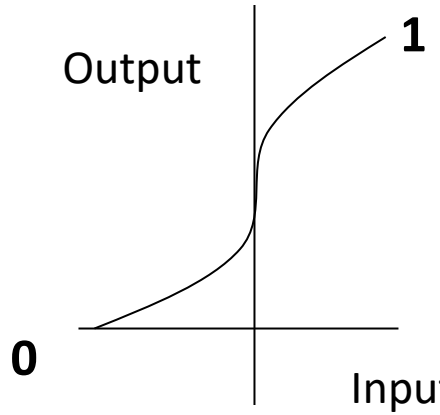


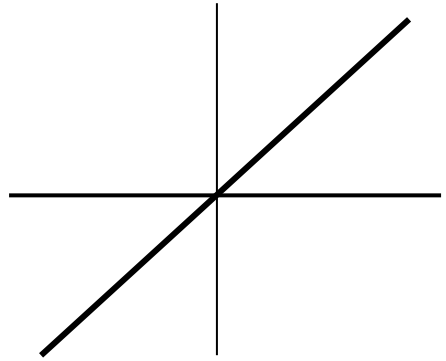
The Perceptron

- The Perceptron is one of the simplest ANN architectures, invented in 1957 by Frank Rosenblatt.
- It is based on a slightly different artificial neuron called a threshold logic unit (TLU), or sometimes a linear threshold unit (LTU).
- he inputs and output are numbers (instead of binary on/off values), and each input connection is associated with a weight.
- The TLU computes a weighted sum of its inputs ($z = w1\ x1 + w2\ x2 + \cdots + wn\ xn = x^T\ w$), then applies a step function to that sum and outputs the result:



Transfer Functions





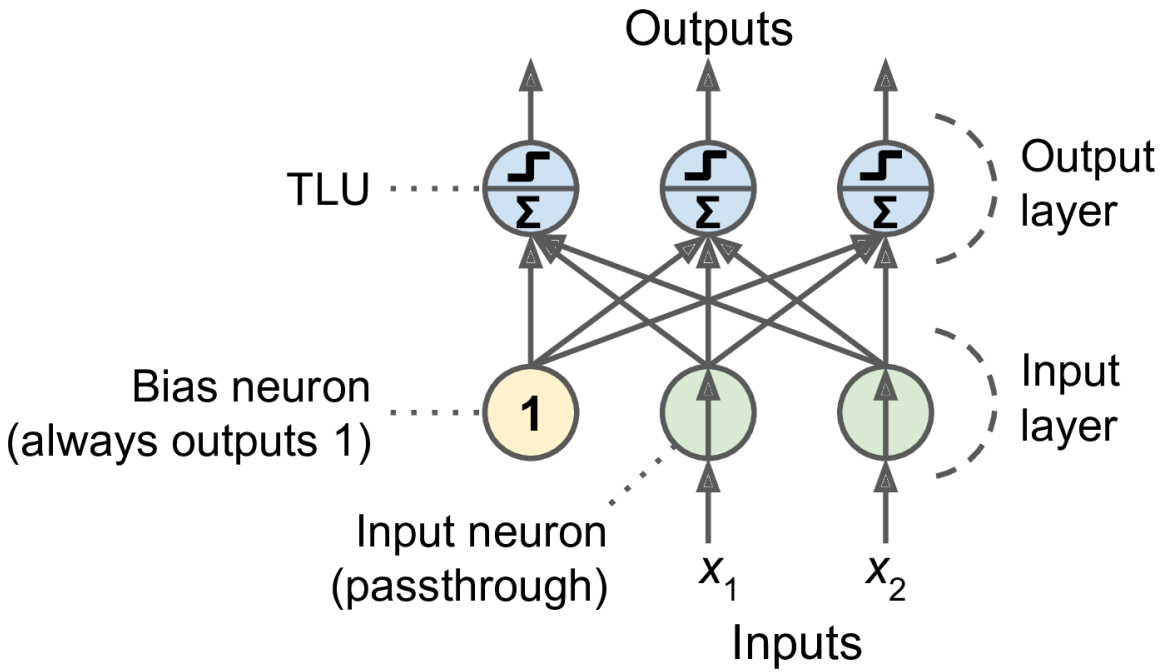
$$SIGMOID: f(n) = \frac{1}{1 + e^{-n}}$$

$$LINEAR: f(n) = n$$

A single Threshold Logic Unit (TLU) can be used for simple linear binary classification. It computes a linear combination of the inputs, and if the result exceeds a threshold, it outputs the positive class. Otherwise, it outputs the negative class (just like a Logistic Regression or linear SVM classifier).

The Perceptron

- A Perceptron is simply composed of a single layer of TLUs, with each TLU connected to all the inputs.
- When all the neurons in a layer are connected to every neuron in the previous layer (i.e., its input neurons), the layer is called a fully connected layer, or a dense layer.
- The inputs of the Perceptron are fed to special passthrough neurons called input neurons: they output whatever input they are fed. All the input neurons form the input layer
- An extra bias feature is generally added (x0 = 1): it is typically represented using a special type of neuron called a bias neuron, which outputs 1 all the time.



The Perceptron - Equation

- As always, X represents the matrix of input features. It has one row per instance and one column per feature.
- The weight matrix W contains all the connection weights except for the ones from the bias neuron. It has one row per input neuron and one column per artificial neuron in the layer.
- The bias vector b contains all the connection weights between the bias neuron and the artificial neurons. It has one bias term per artificial neuron.
- The function φ is called the activation function: when the artificial neurons are TLUs, it is a step function (but we will discuss other activation functions shortly).

$$h_{\mathbf{W},\mathbf{b}}(\mathbf{X}) = \phi(\mathbf{X}\mathbf{W} + \mathbf{b})$$

Perceptron learning rule (weight update)

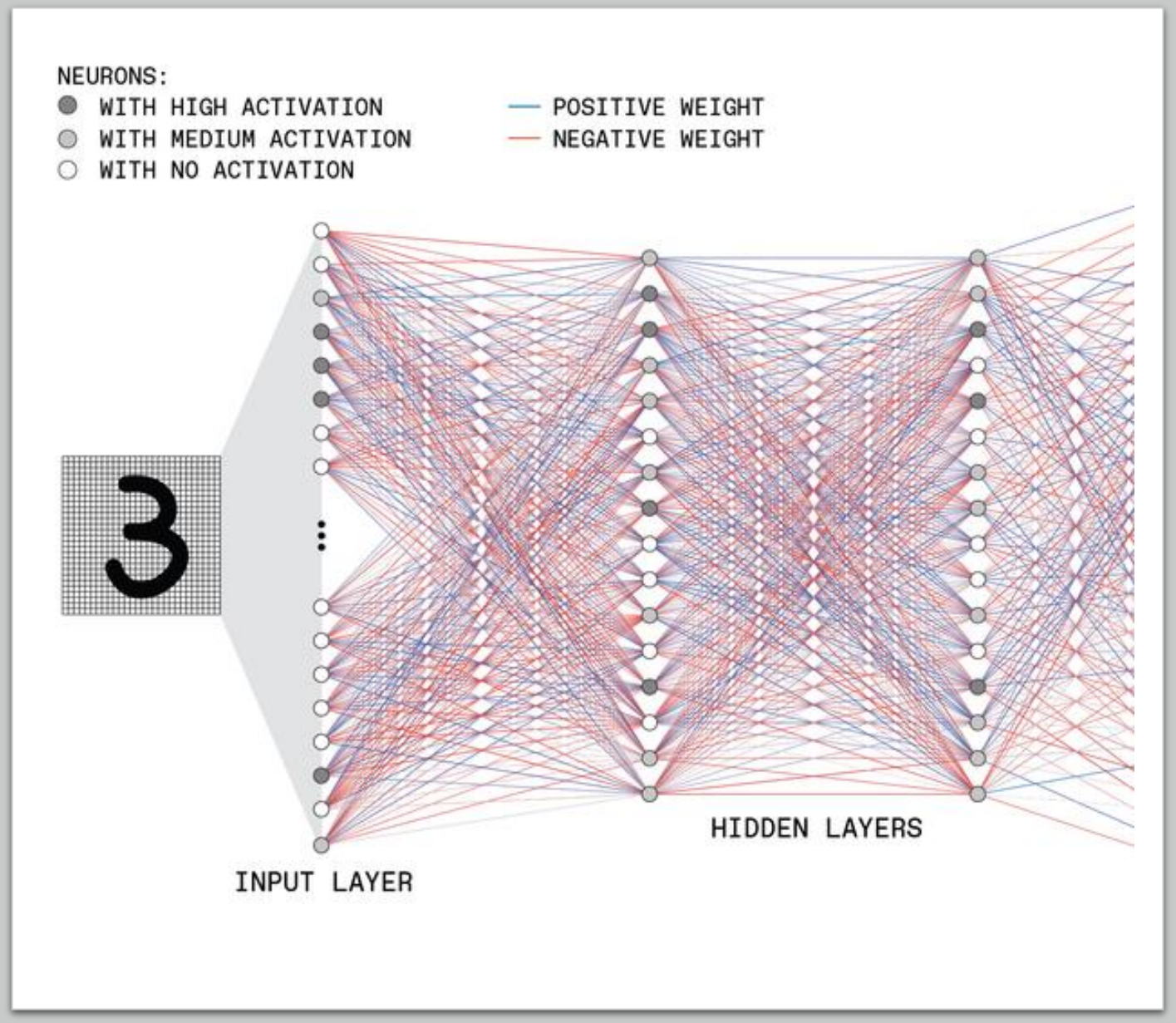
$$w_{i,j}^{(\text{next step})} = w_{i,j} + \eta\left(y_j - \hat{y}_j\right)x_i$$

In this equation:

- $w_{i,j}$ is the connection weight between the i^{th} input neuron and the j^{th} output neuron.
- x_i is the i^{th} input value of the current training instance.
- $\hat{y}_j$ is the output of the j^{th} output neuron for the current training instance.
- y_j is the target output of the j^{th} output neuron for the current training instance.
- η is the learning rate.

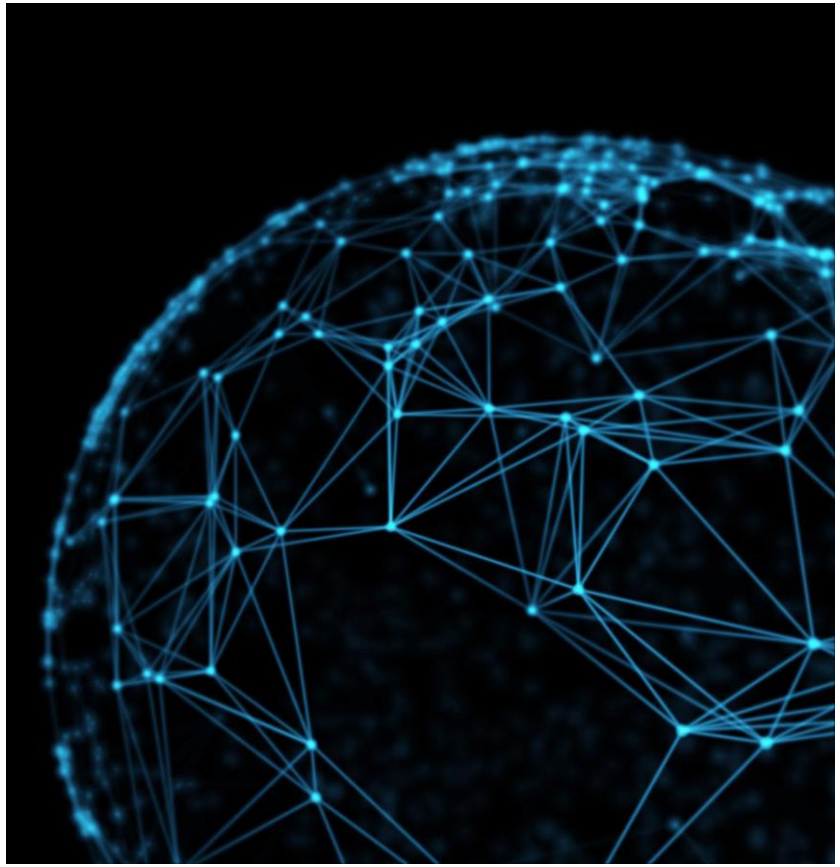
Architecture Example

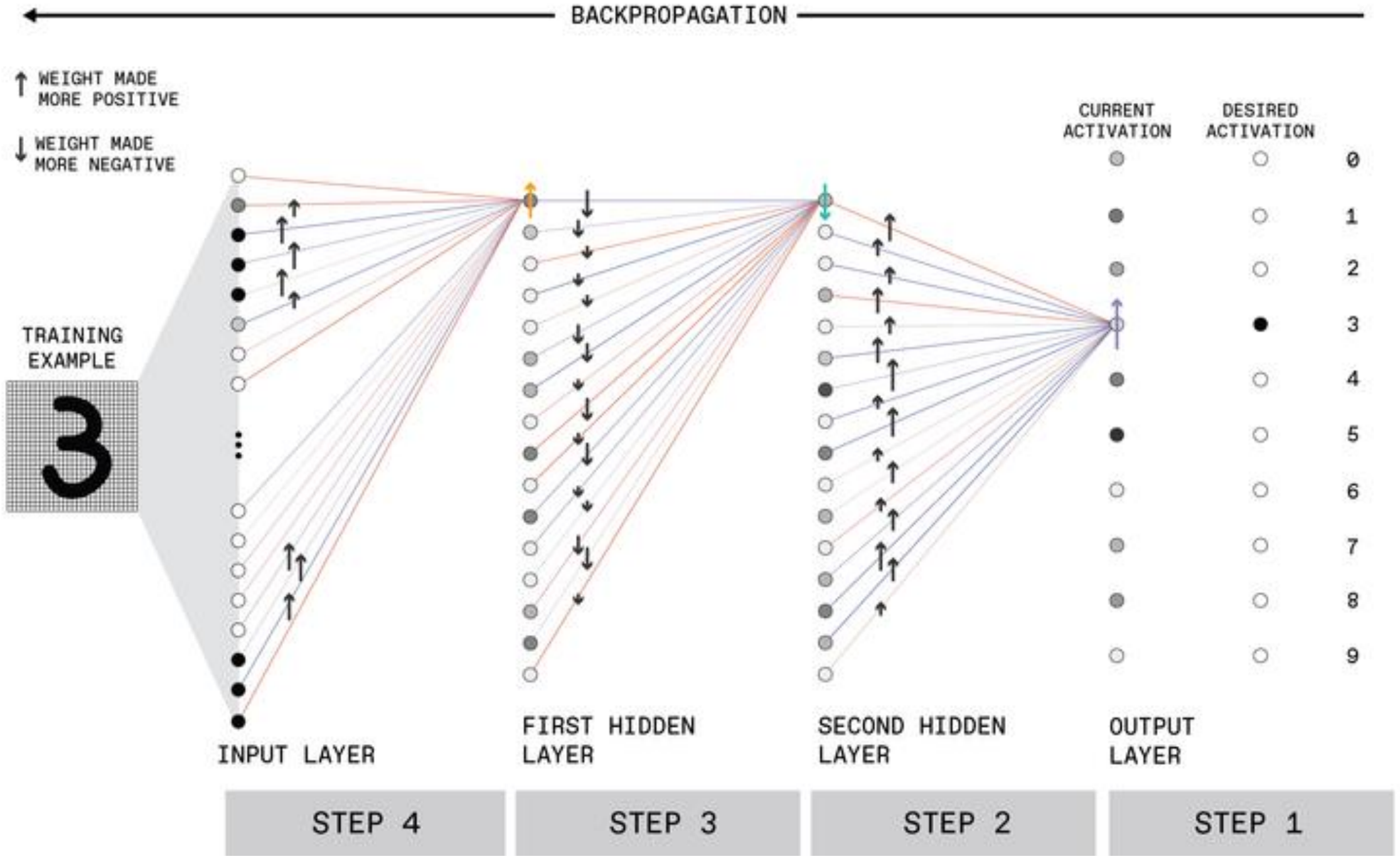
- Here's the structure of a hypothetical feed-forward deep neural network ("deep" because it contains multiple hidden layers).
- This example shows a network that interprets images of hand-written digits and classifies them as one of the 10 possible numerals.

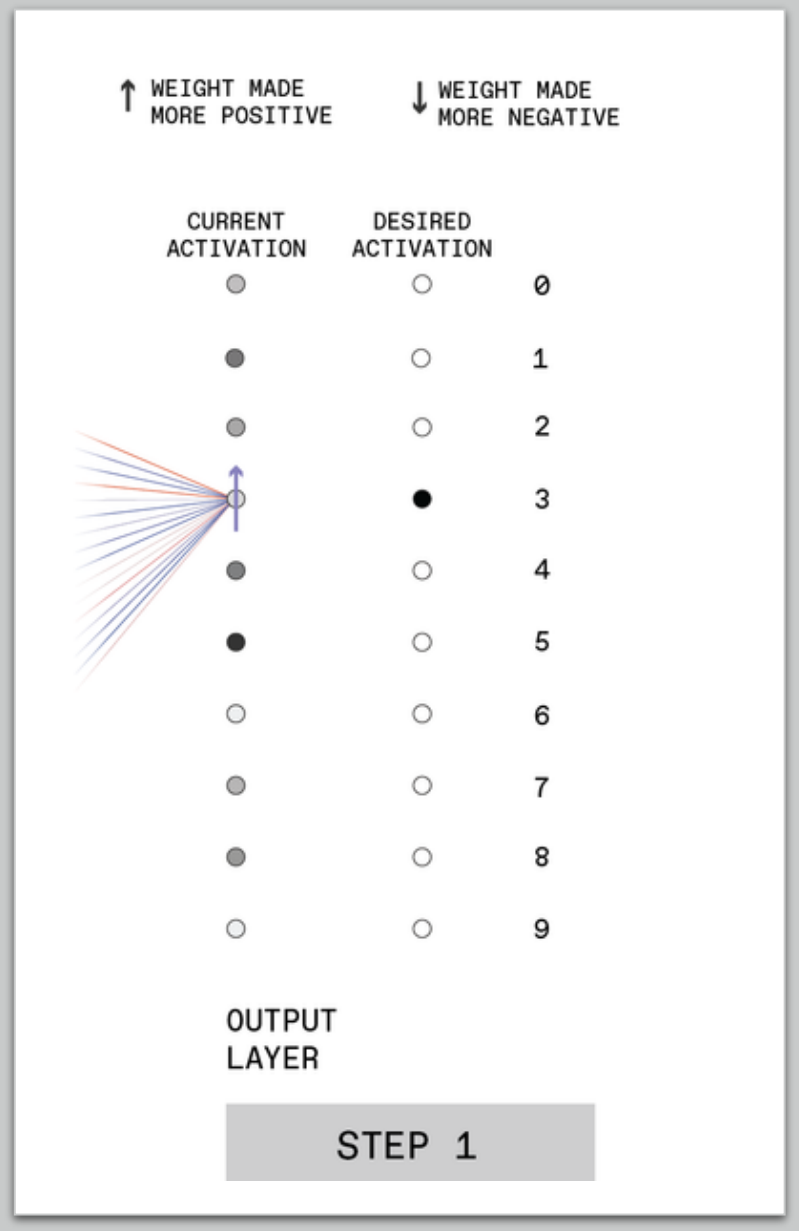


The backpropagation process

- This kind of neural network is trained by calculating the difference between the actual output and the desired output.
- The mathematical optimization problem here has as many dimensions as there are adjustable parameters in the network—primarily the weights of the connections between neurons, which can be positive or negative.
- Training the network is essentially finding a minimum of this multidimensional "loss" or "cost" function.
- In practice, that entails making many small adjustments to the network's weights based on the outputs that are computed for a random set of input examples, each time starting with the weights that control the output layer and moving backward through the network.
- This backpropagation process is repeated over many random sets of training examples until the loss function is minimized, and the network then provides the best results it can for any new input.

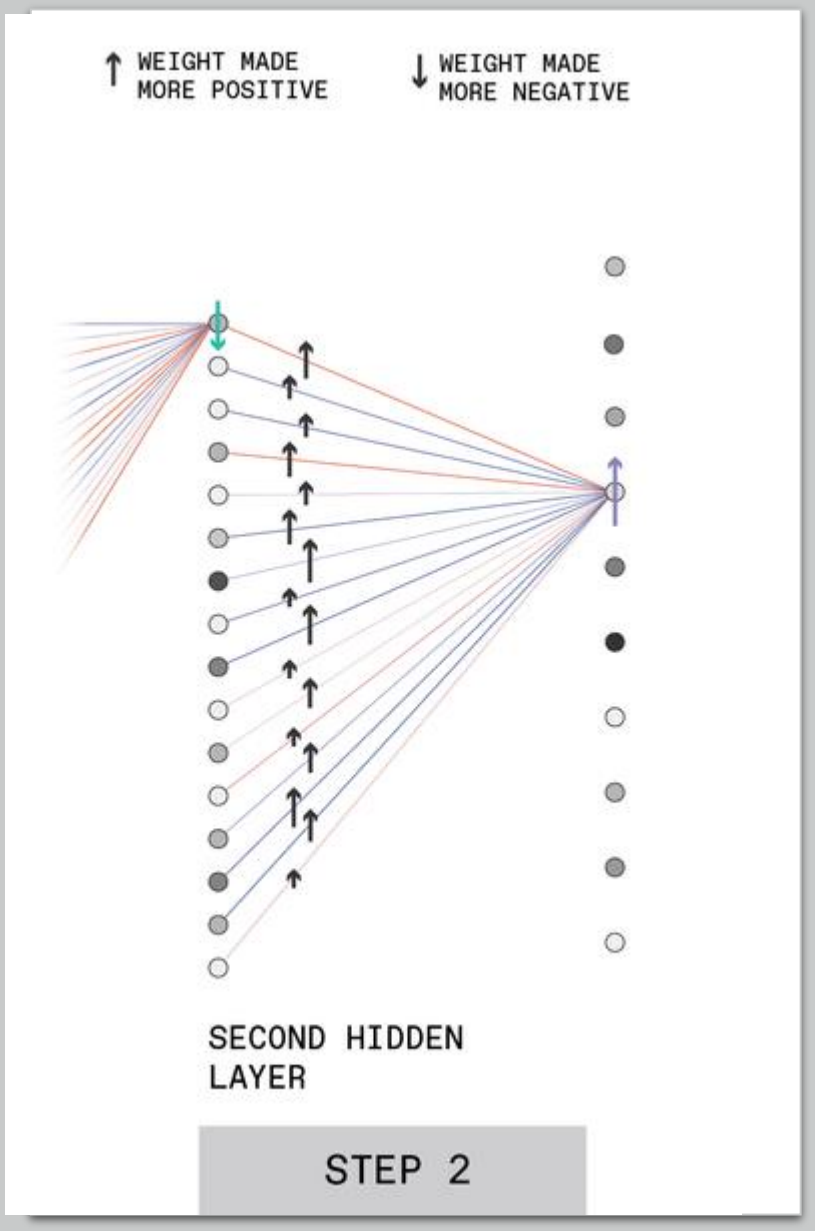



BACKPROPAGATION
↑ WEIGHT MADE MORE POSITIVE
↓ WEIGHT MADE MORE NEGATIVE
TRAINING EXAMPLE
3
CURRENT ACTIVATION
DESIRED ACTIVATION
0
1
2
3
4
5
6
7
8
9
INPUT LAYER
FIRST HIDDEN LAYER
SECOND HIDDEN LAYER
OUTPUT LAYER
STEP 4
STEP 3
STEP 2
STEP 1



The backpropagation process – Step 1

- When presented with a handwritten "3" at the input, the output neurons of an untrained network will have random activations.
- The desire is for the output neuron associated with 3 to have high activation [dark shading] and other output neurons to have low activations [light shading].
- So the activation of the neuron associated with 3, for example, must be increased [purple arrow].

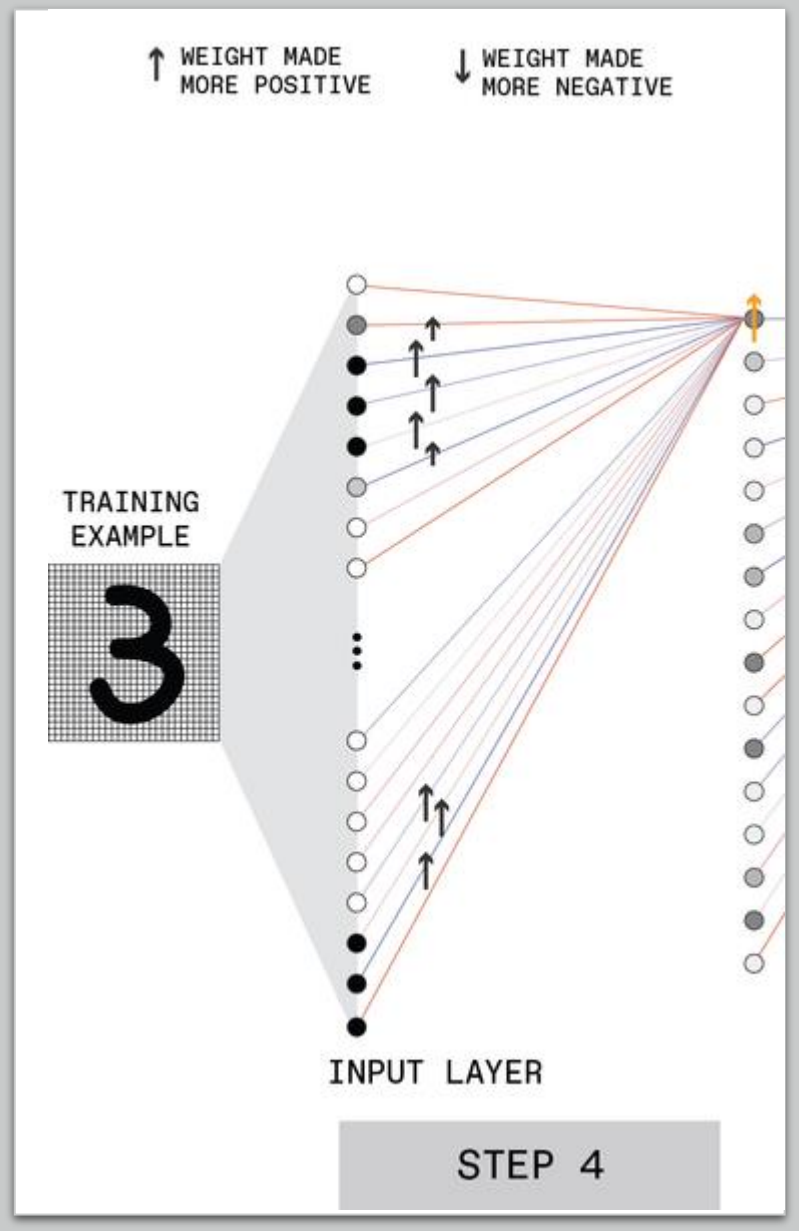


The backpropagation process – Step 2

- To do that, the weights of the connections from the neurons in the second hidden layer to the output neuron for the digit "3" should be made more positive [black arrows], with the size of the change being proportional to the activation of the connected hidden neuron.

The backpropagation process – Step 3

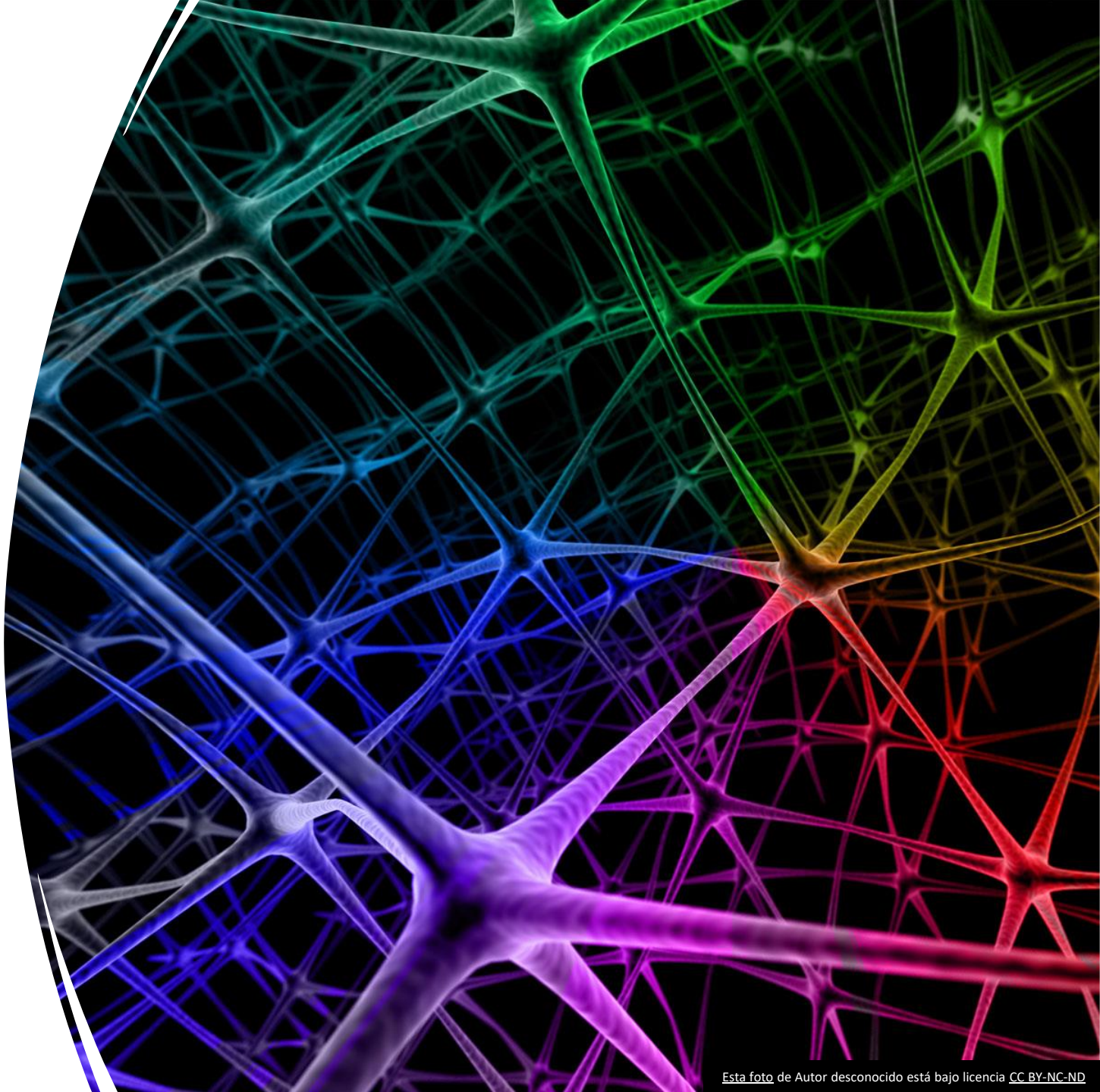
- A similar process is then performed for the neurons in the second hidden layer.
- For example, to make the network more accurate, the top neuron in this layer may need to have its activation reduced [green arrow].
- The network can be pushed in that direction by adjusting the weights of its connections with the first hidden layer [black arrows].



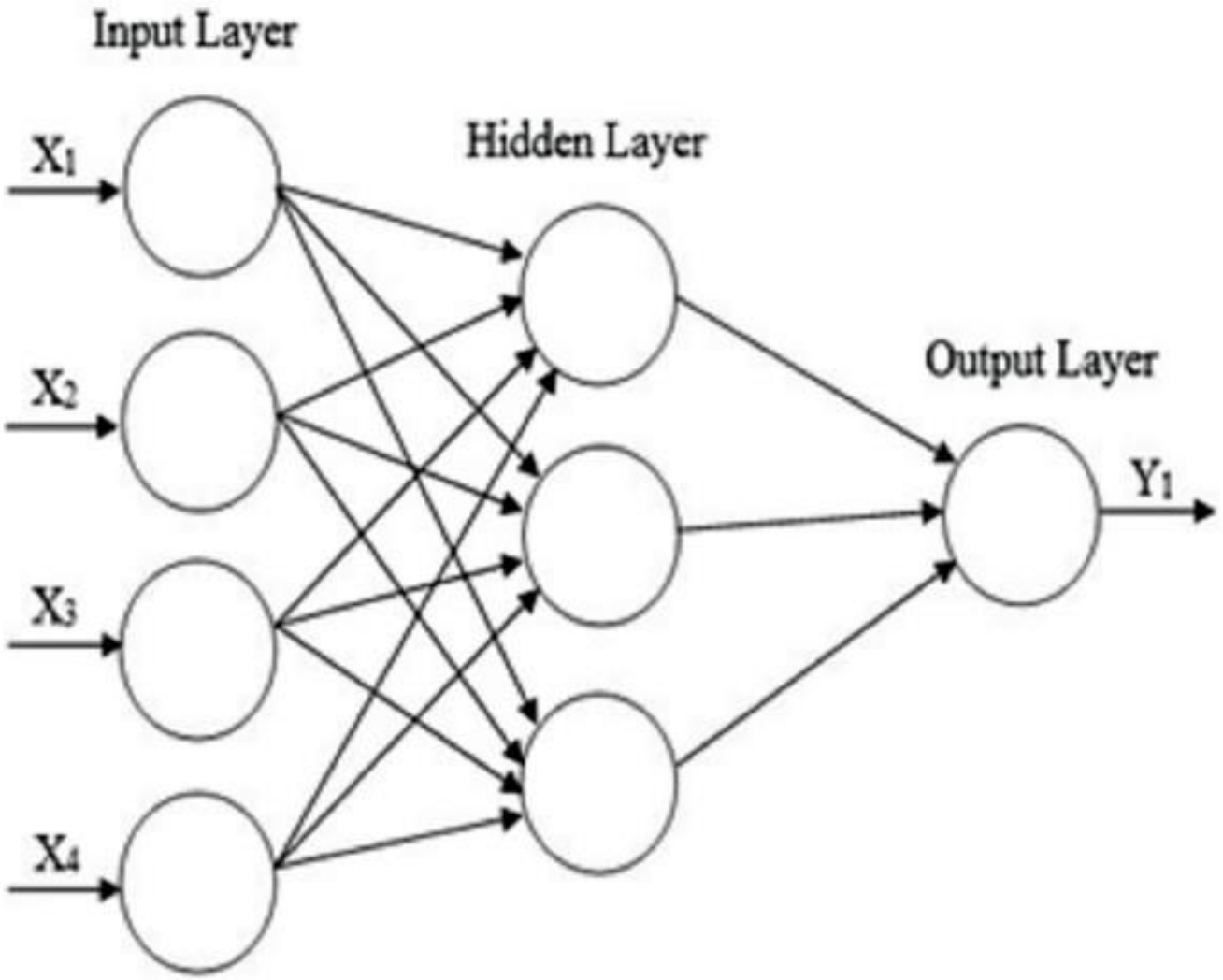
The backpropagation process – Step 4

- The process is then repeated for the first hidden layer.
- For example, the first neuron in this layer may need to have its activation increased [orange arrow].

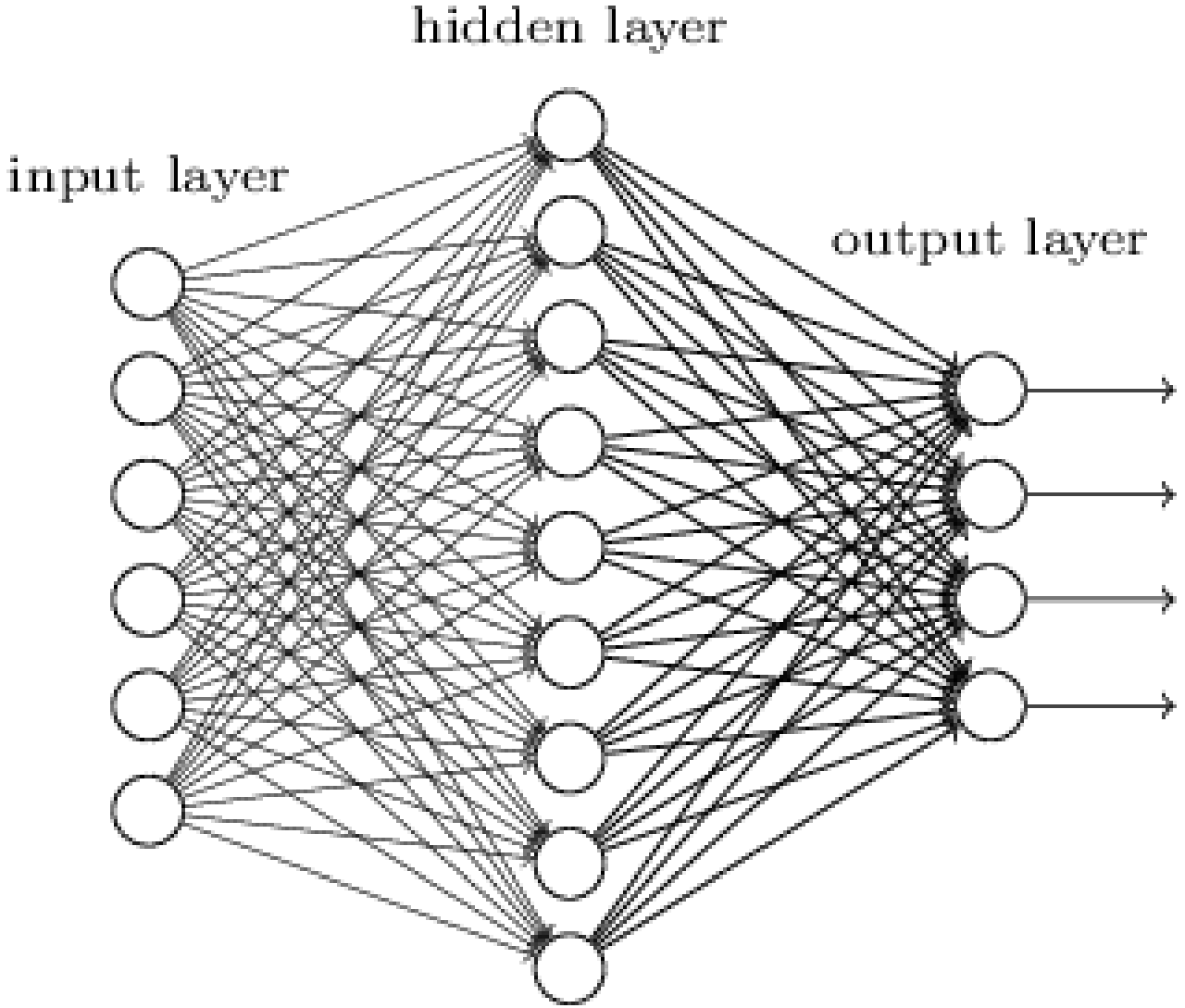
Types of networks



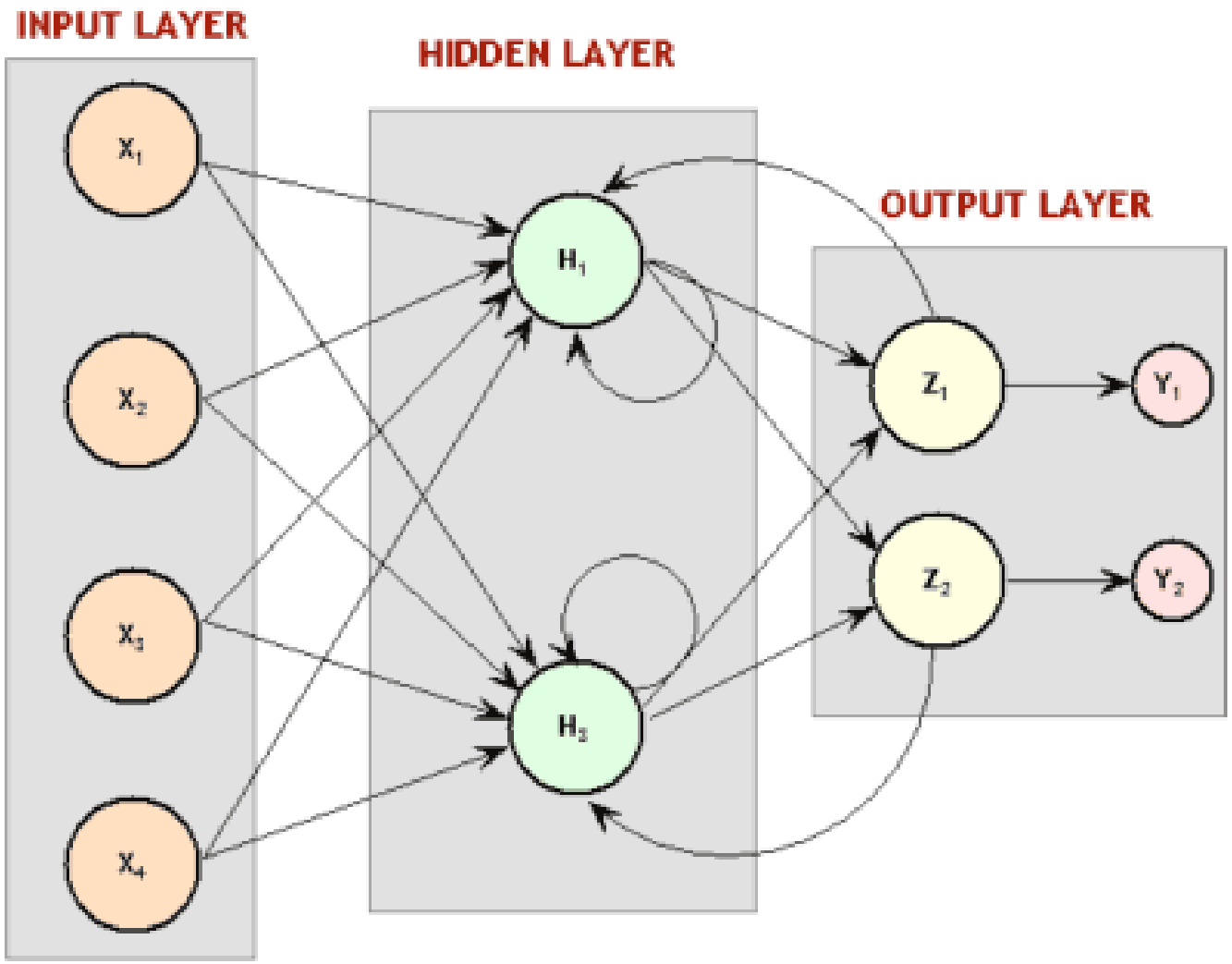
Multiple Inputs and Single Layer



Multiple Inputs and layers



Recurrent Networks



Example

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.linear_model import Perceptron

iris = load_iris()
X = iris.data[:, (2, 3)]  # petal length, petal width
y = (iris.target == 0).astype(np.int)  # Iris setosa?

per_clf = Perceptron()
per_clf.fit(X, y)

y_pred = per_clf.predict([[2, 0.5]])
```

References

- Géron, A. (2019). Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems. O'Reilly Media.
- How Deep Learning Works - IEEE Spectrum. (n.d.). Retrieved October 7, 2021, from https://spectrum.ieee.org/what-is-deep-learning?utm_source=roboticsnews&utm_medium=email&utm_campaign=aialert-10-05-21&mkt_tok=NzU2LUdQSC04OTkAAAF_7_I9SssS36I1WE4y_3Oryaq4gpSGsrzuEmSe4BYbMcTRXY34gfOX_mnQPW_Spv4t-TMst_p0Ft40iPNS_mvrukprNu4ZQB9yk9ha2tlDvKI