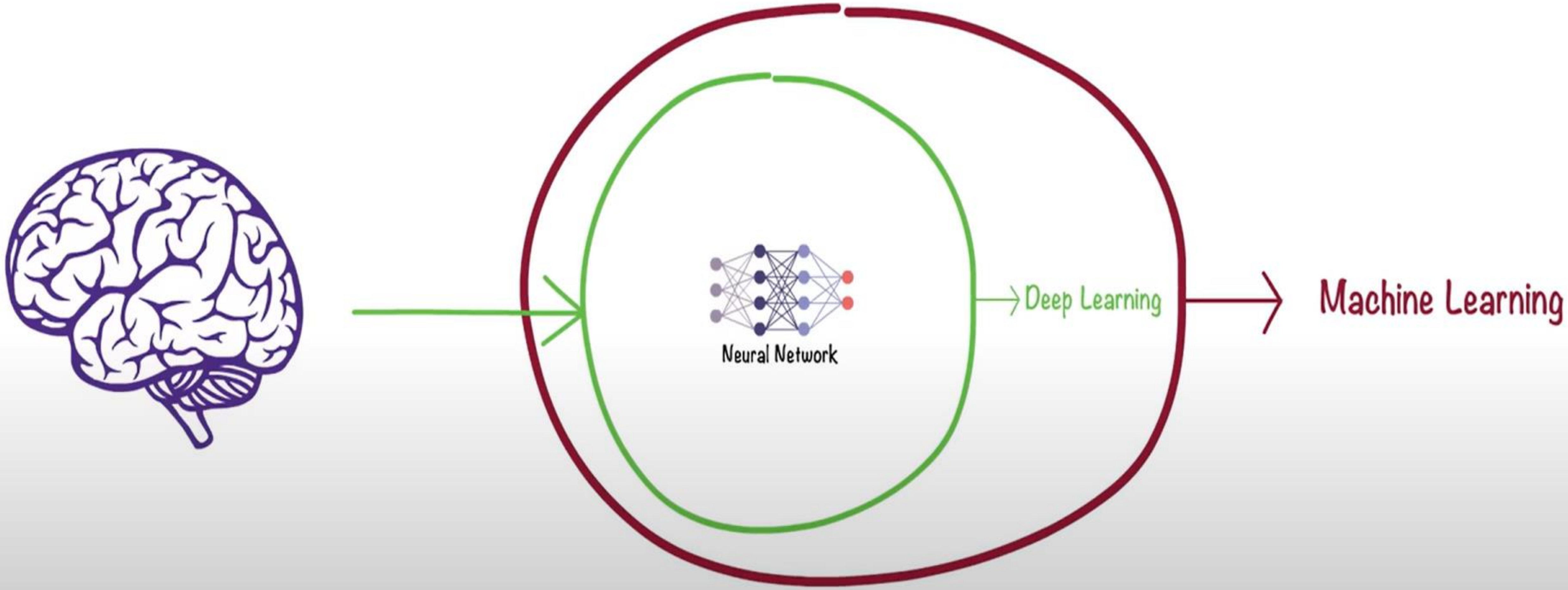
Computational Neuroscience

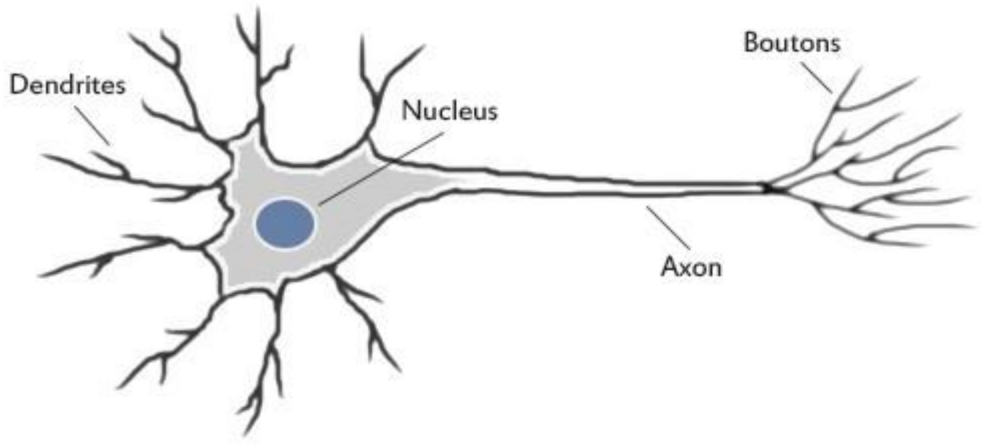
Edited by

A. Prof. Noha El-Attar

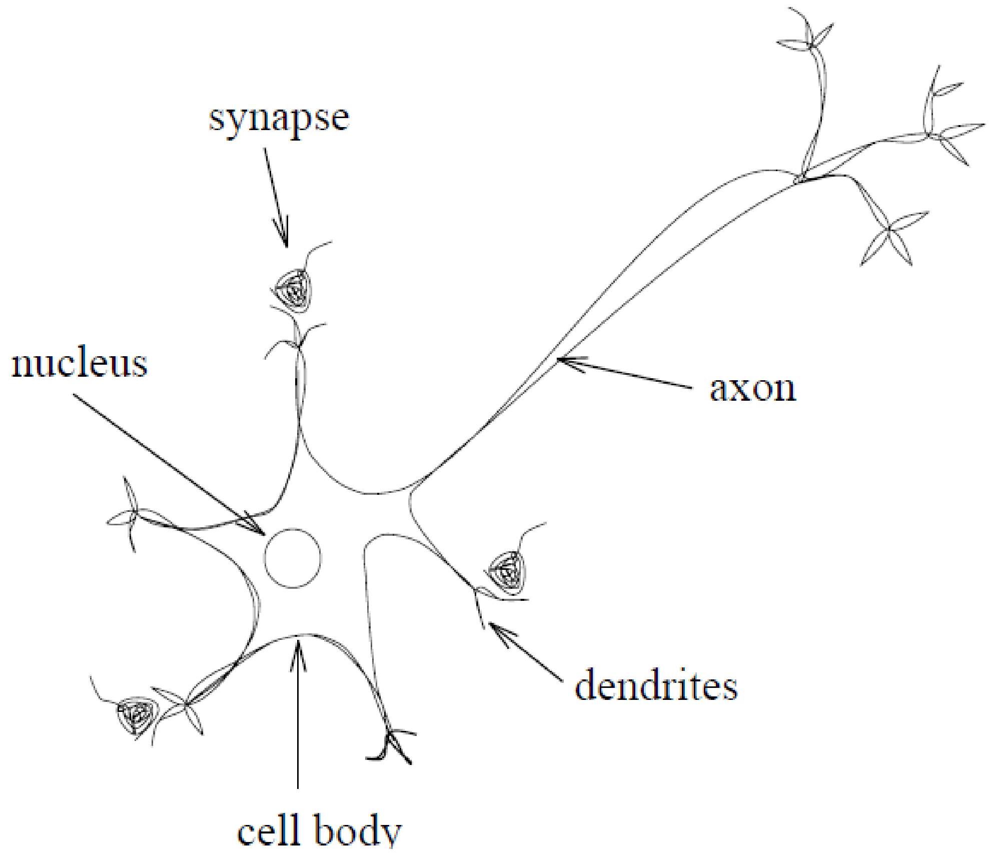

Neural Network
Deep Learning
Machine Learning

MOTIVATION

- Our brain uses the extremely large interconnected network of **neurons** for information processing and to model the world around us. Simply put, a neuron collects inputs from other neurons using ***dendrites***. The neuron **sums all the inputs** and if the resulting value is **greater than a threshold**, it fires. The fired signal is then sent to other connected neurons through the **axon**.



Biological Networks



1. The majority of ***neurons*** encode their outputs or activations as a series of brief electrical pulses (i.e. spikes or action potentials).

2. ***Dendrites*** are the receptive zones that receive activation from other neurons.

3. The ***cell body (soma)*** of the neuron's processes the incoming activations and converts them into output activations.

4. ***Axons*** are transmission lines that send activation to other neurons.

5. ***Synapses*** allow weighted transmission of signals (using ***neurotransmitters***) between axons and dendrites to build up large neural networks.

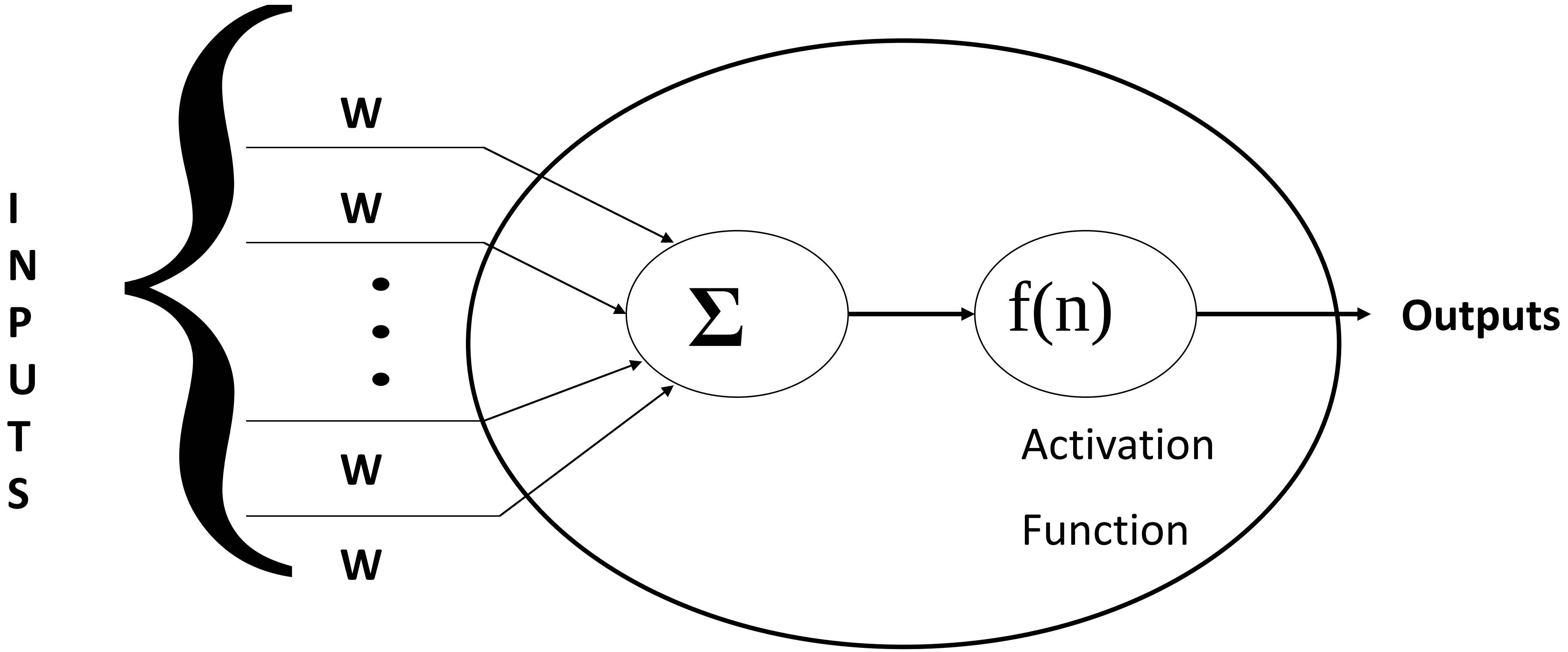
Artificial Neural Networks (ANN)

- **Computational models inspired by the human brain:**
  - Algorithms that try to mimic the brain.
  - Massively parallel, distributed system, made up of simple processing units (**neurons**).
  - Synaptic connection strengths among neurons are used to store the acquired knowledge.
  - Knowledge is acquired by the network from its environment through a learning process

Design of an ANN

- **The design of an ANN involves determining the following elements:**
  - **Neurons**: Nodes of the network
  - **Activation function (Transfer Fn):** a function of input that the neuron receives to convert the input signal on the node of ANN to an output signal based on the application of the network.
  - **Connections and arrangement of neurons:** **topology** (network architecture).
  - **Synaptic weights** [-1,1] or [-0.5,0.5]: real values that are attached with each input/feature to determine the importance of that corresponding feature in predicting the final output.
  - **Bias** (-1,1): is required to shift the activation function across the plane either towards the left or the right.

Artificial Neural Networks



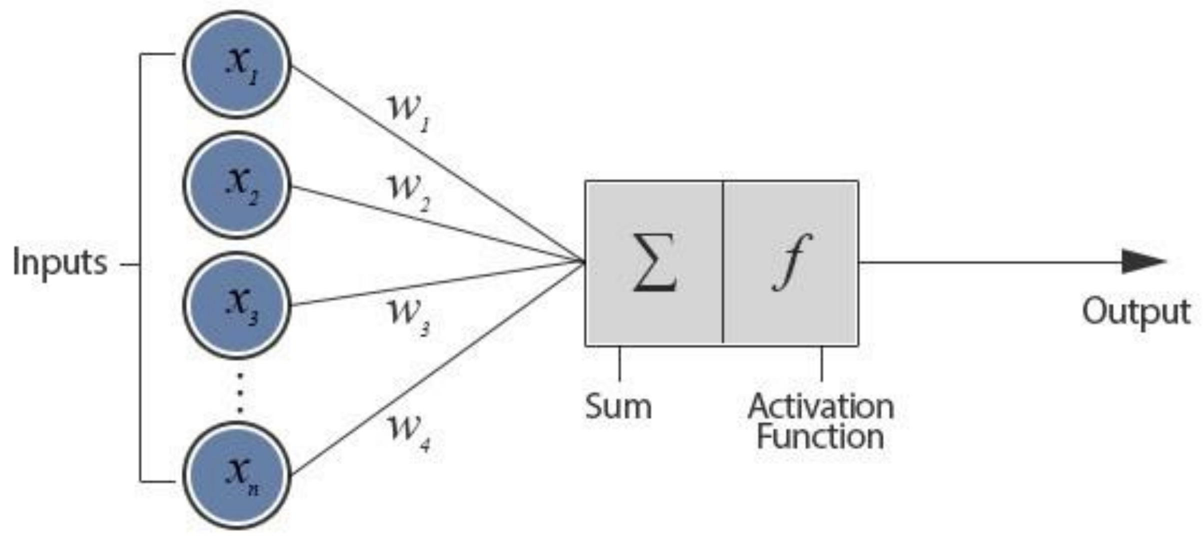
Neuron (Perceptron)

MOTIVATION

- Neural networks loosely mimic the way our brains solve the problem: by taking in inputs, processing them and generating an output. Like us, they *learn* to recognize patterns, but they do this by *training* on **labelled datasets**. Before we get to the learning part, let’s take a look at the most basic of artificial neurons: the **perceptron**, and how it processes inputs and produces an output.

THE PERCEPTRON

- **A neuron** connected with *n* other neurons and thus receives *n* inputs (x_1, x_2, ….. x_n) to produce one **output**.

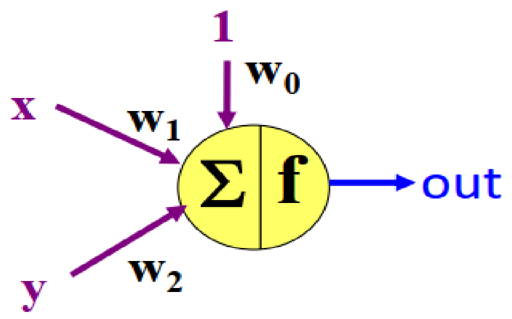


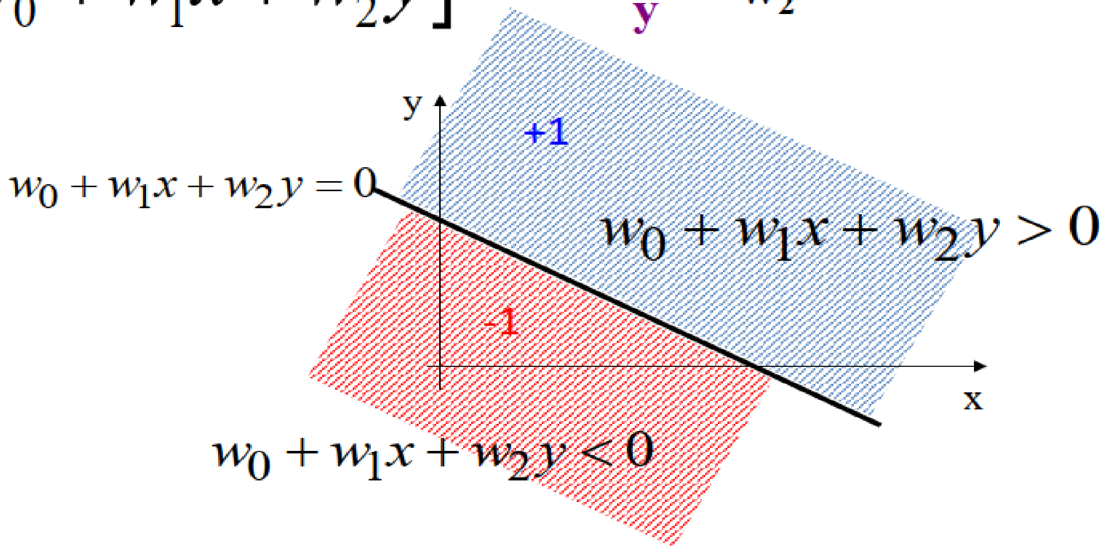
- As you can see, the network of nodes sends signals in one direction. This is called **a feed-forward network**.

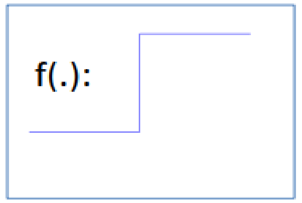
WHAT IS THE PERCEPTRON ACTUALLY DOING?

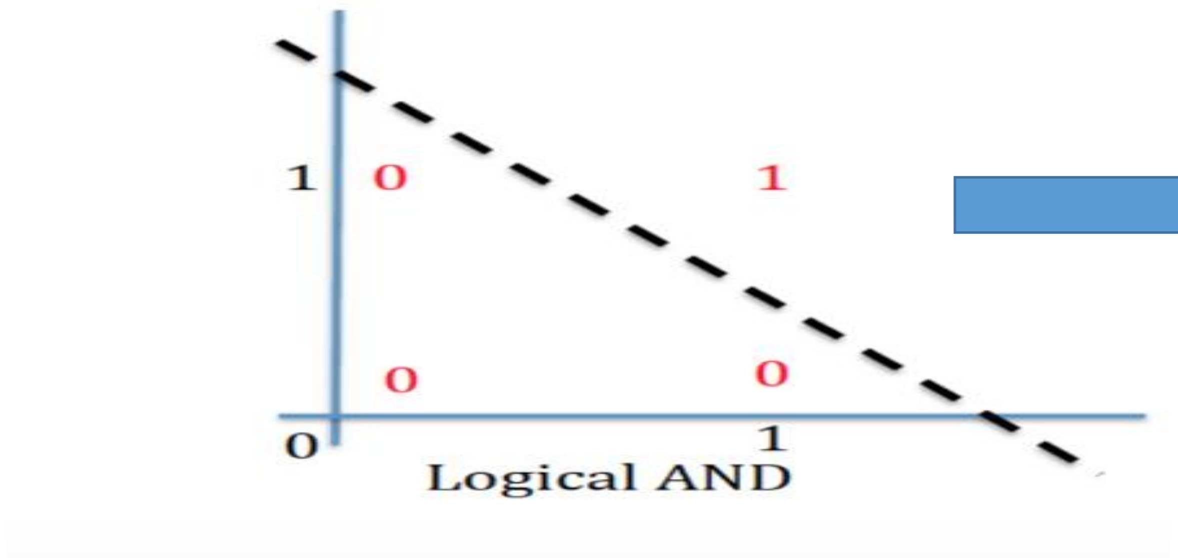
- The perceptron is adding all the inputs and separating them into 2 categories (Yes or No). That is, it is drawing the line:
  - Two inputs: x, y
  - Two weights: w_1, w_2
  - **<u>Bias</u>** input: w_0

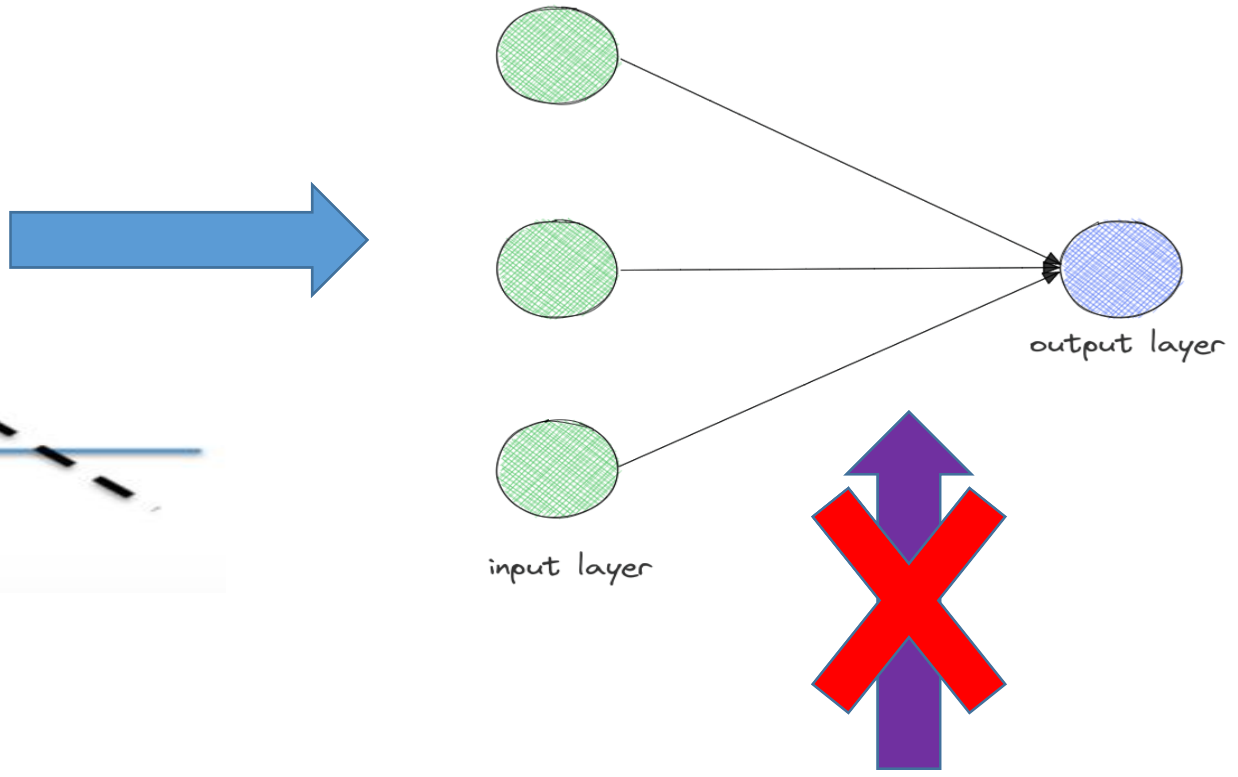
$$out = f\left[w_0 + w_1 x + w_2 y\right]$$



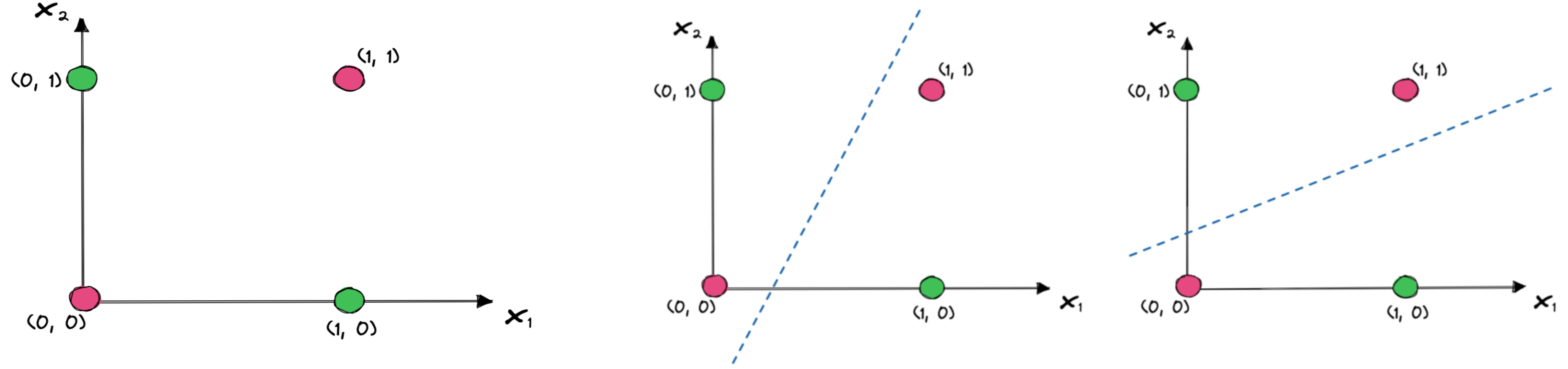

y
+1
$w_0 + w_1 x + w_2 y = 0$
$w_0 + w_1 x + w_2 y > 0$
-1
x
$w_0 + w_1 x + w_2 y < 0$



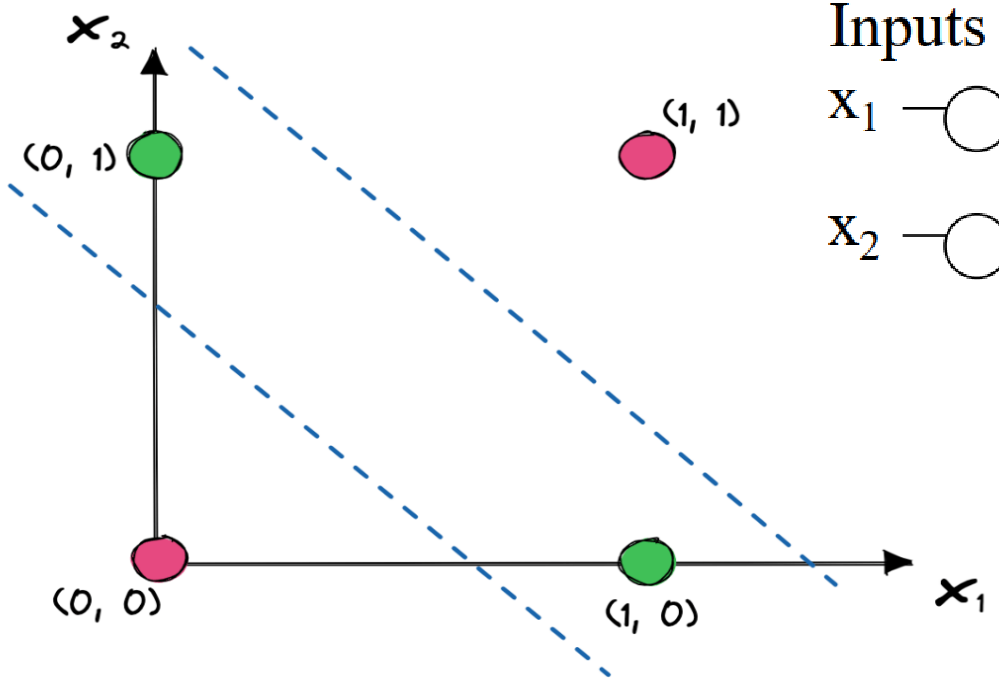

1
0
1
0
0
0
1
Logical AND

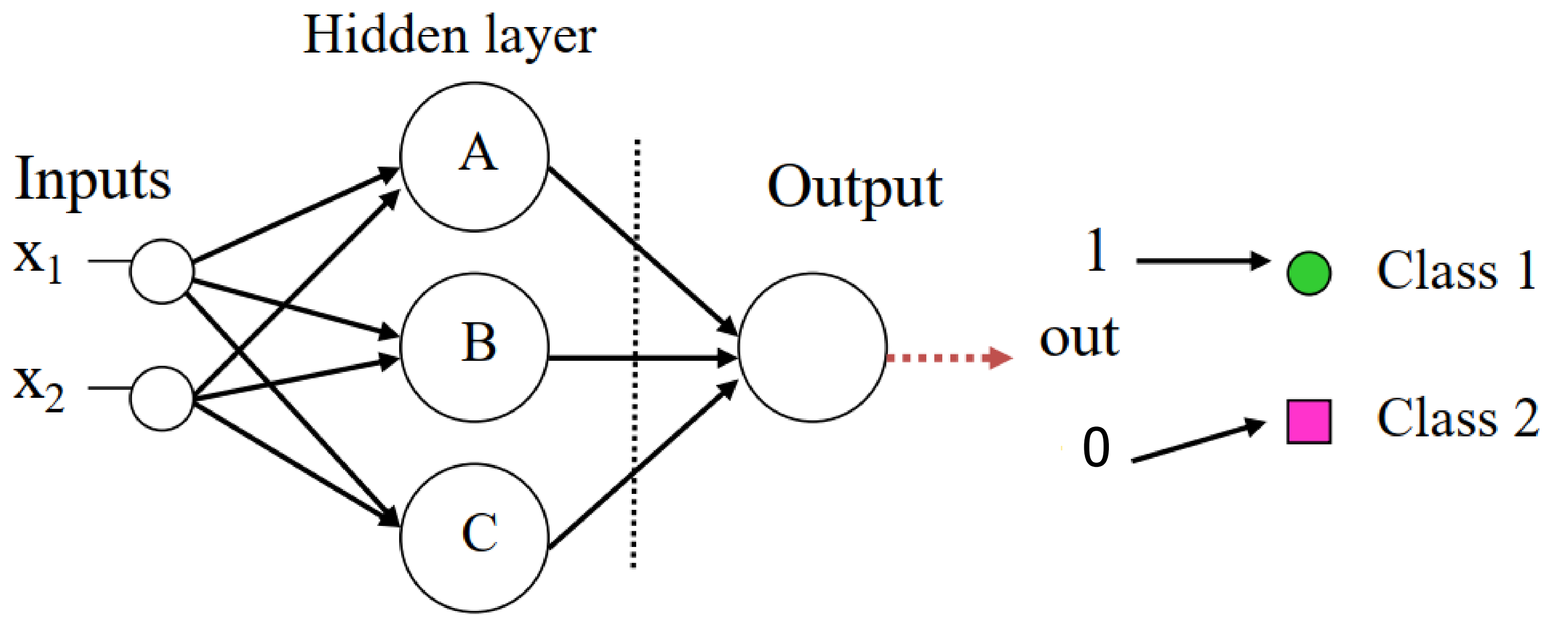

input layer
output layer

XOR problem


x_2
(0, 1)
(1, 1)
(0, 0)
(1, 0)
x_1
x_2
(0, 1)
(1, 1)
(0, 0)
(1, 0)
x_1
x_2
(0, 1)
(1, 1)
(0, 0)
(1, 0)
x_1

XOR


x_2
x_1
(0, 1)
(1, 1)
(0, 0)
(1, 0)

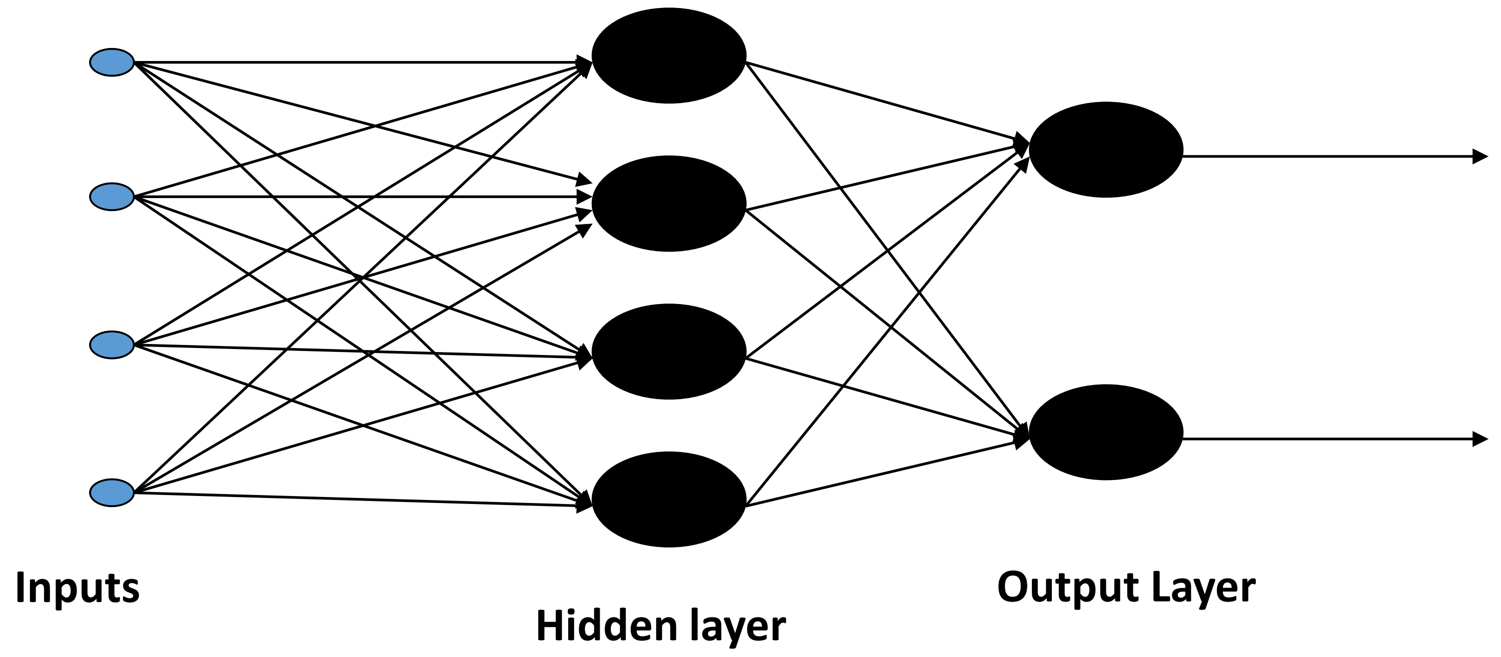

Hidden layer
Inputs
x_1
x_2
A
B
C
Output
out
1
0
Class 1
Class 2

LIMITATION OF PERCEPTRONS

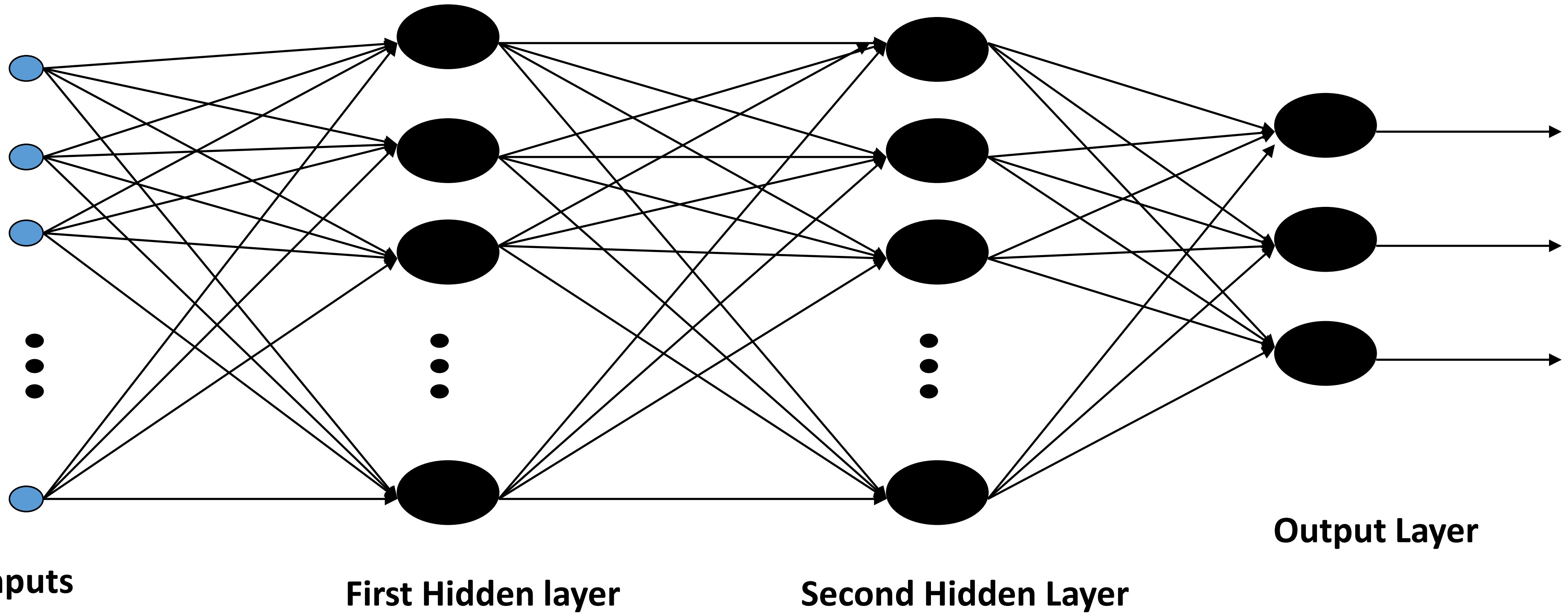
- **Not every set of inputs can be divided by a line** like this. Those that can be are called *linearly separable*. If the vectors are not linearly separable, learning will never reach a point where all vectors are classified properly.

Multilayer Perceptron

- Sets of perceptrons arranged in several layers.
- At least one hidden layer.

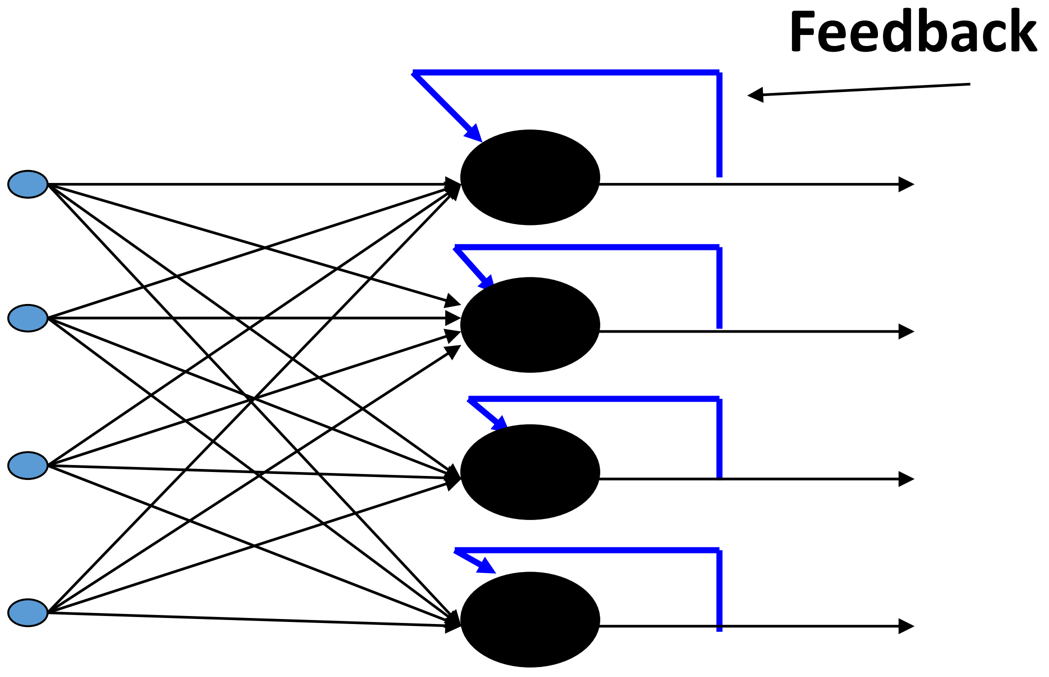


Multilayer Perceptron



ANN Training

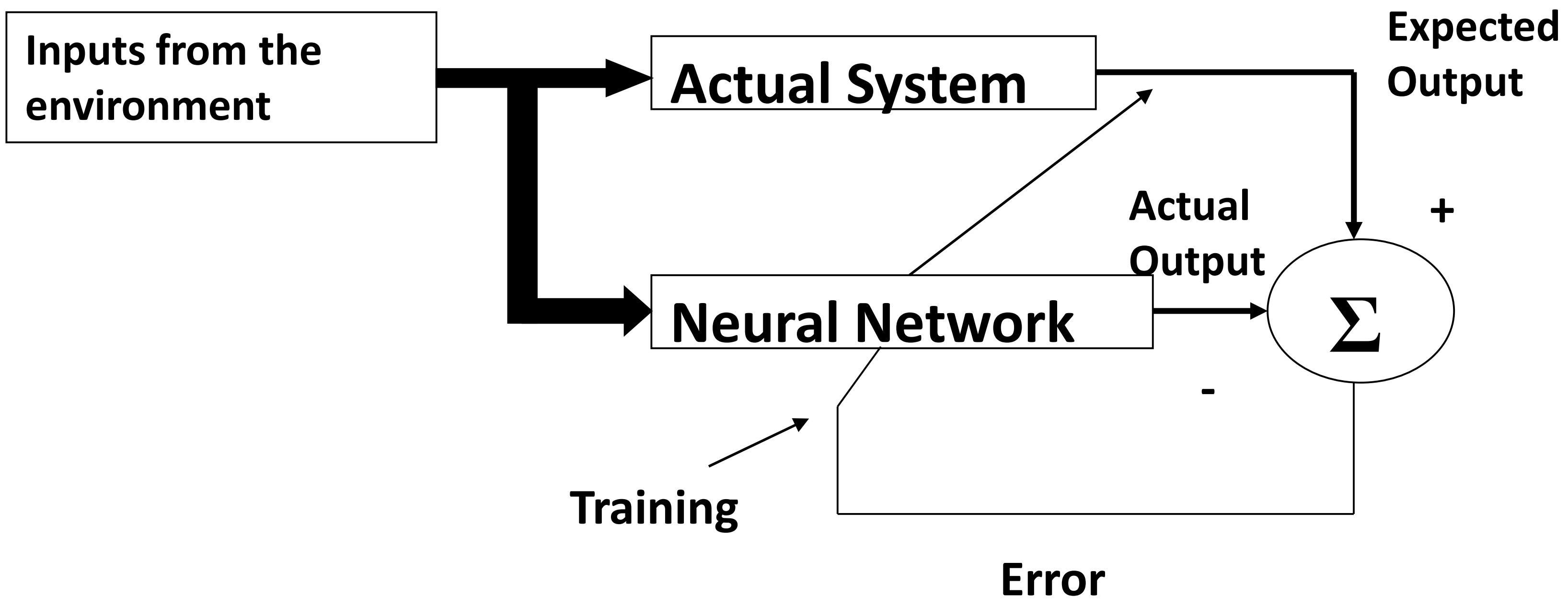
- – Initialize weights for all neurons
- – Present input layer
- – Calculate outputs
- – Compare outputs with actual target
- – Update weights to attempt a match
- – Repeat until all examples presented



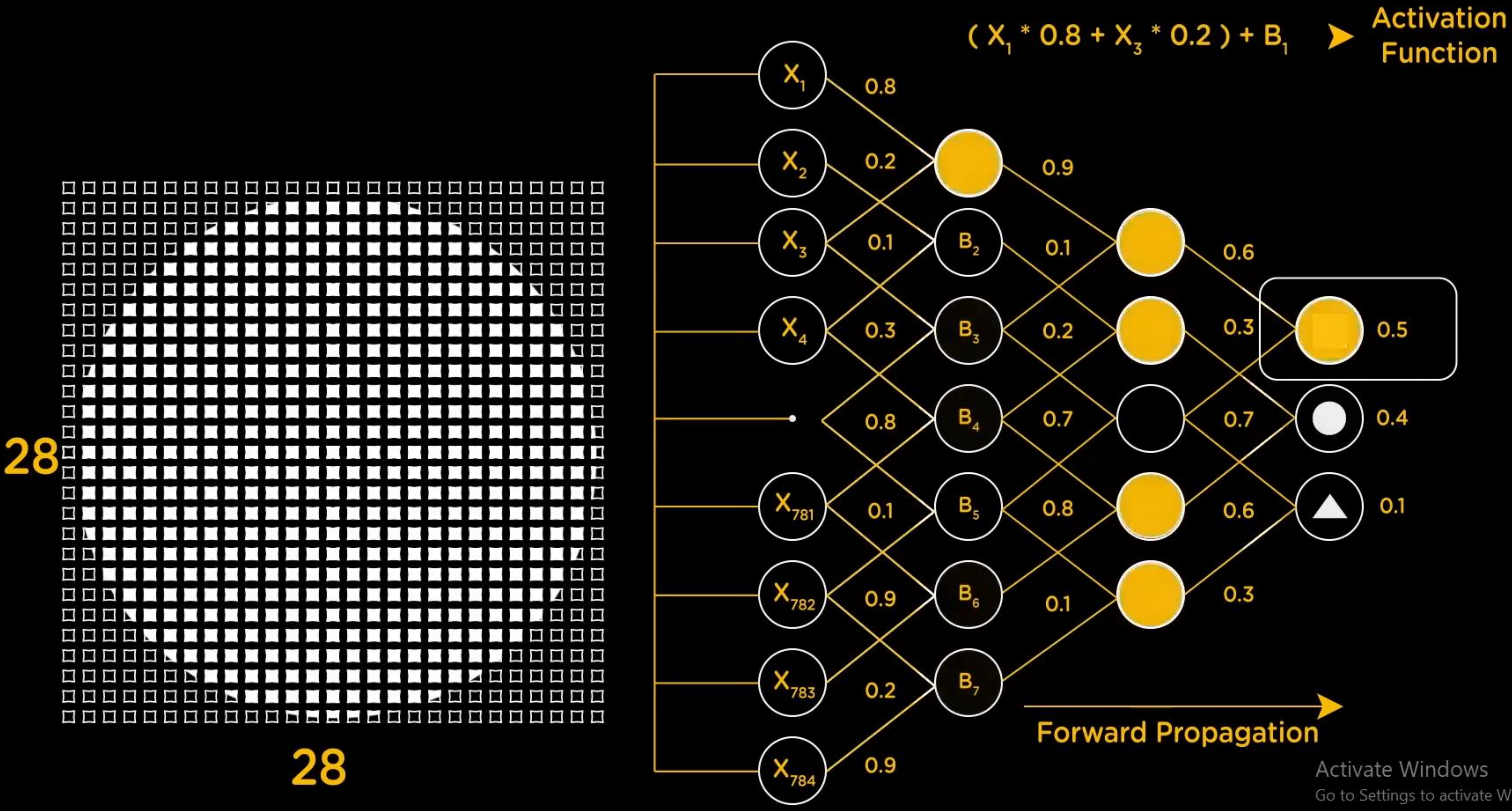
Recurrent Networks

Learning Techniques

- The **'learning rule"** modifies the weights according to the input patterns that it is presented with.

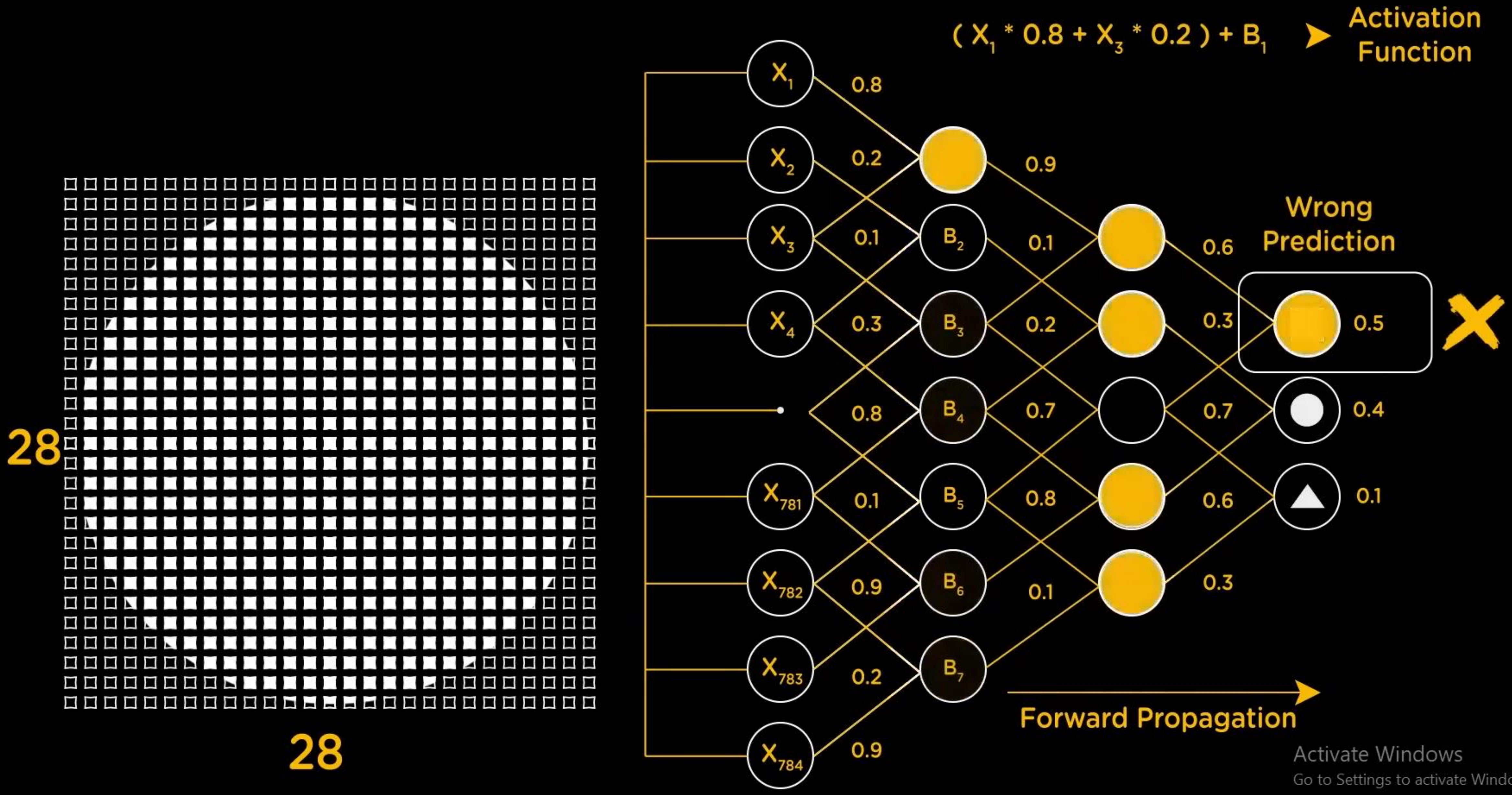


Supervised Learning

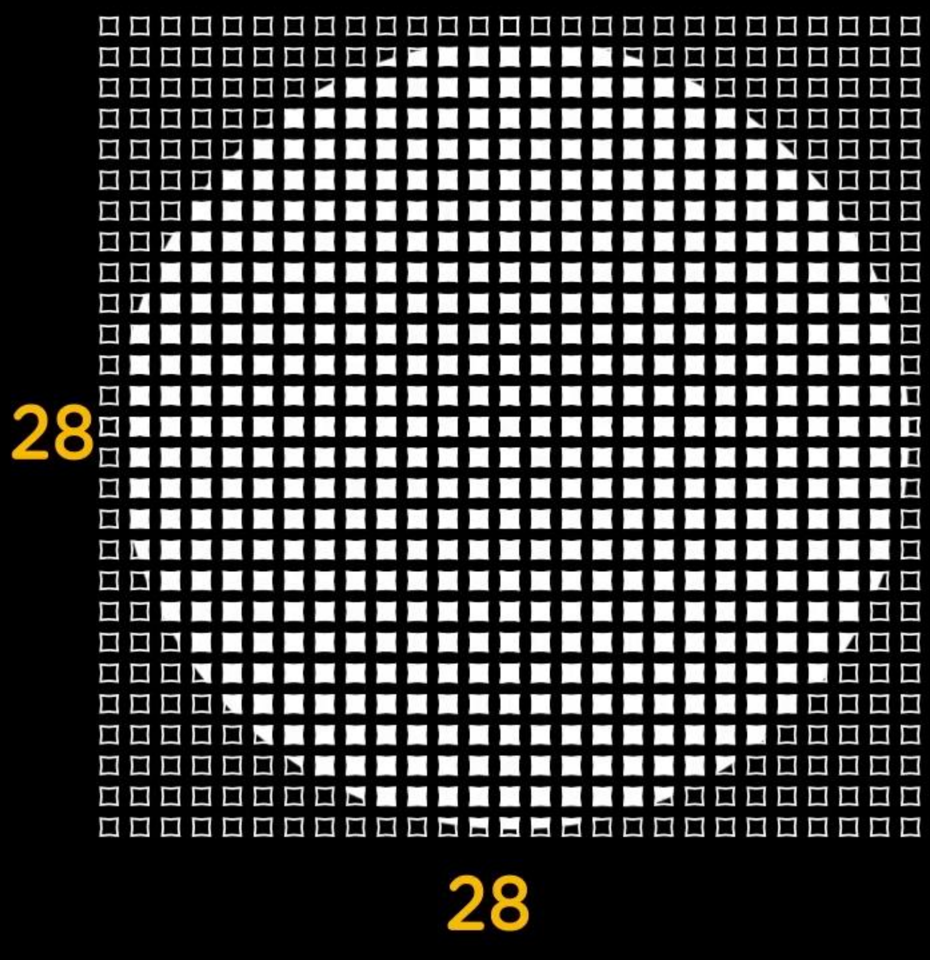


28 x 28 = 784 Pixels

simplilearn

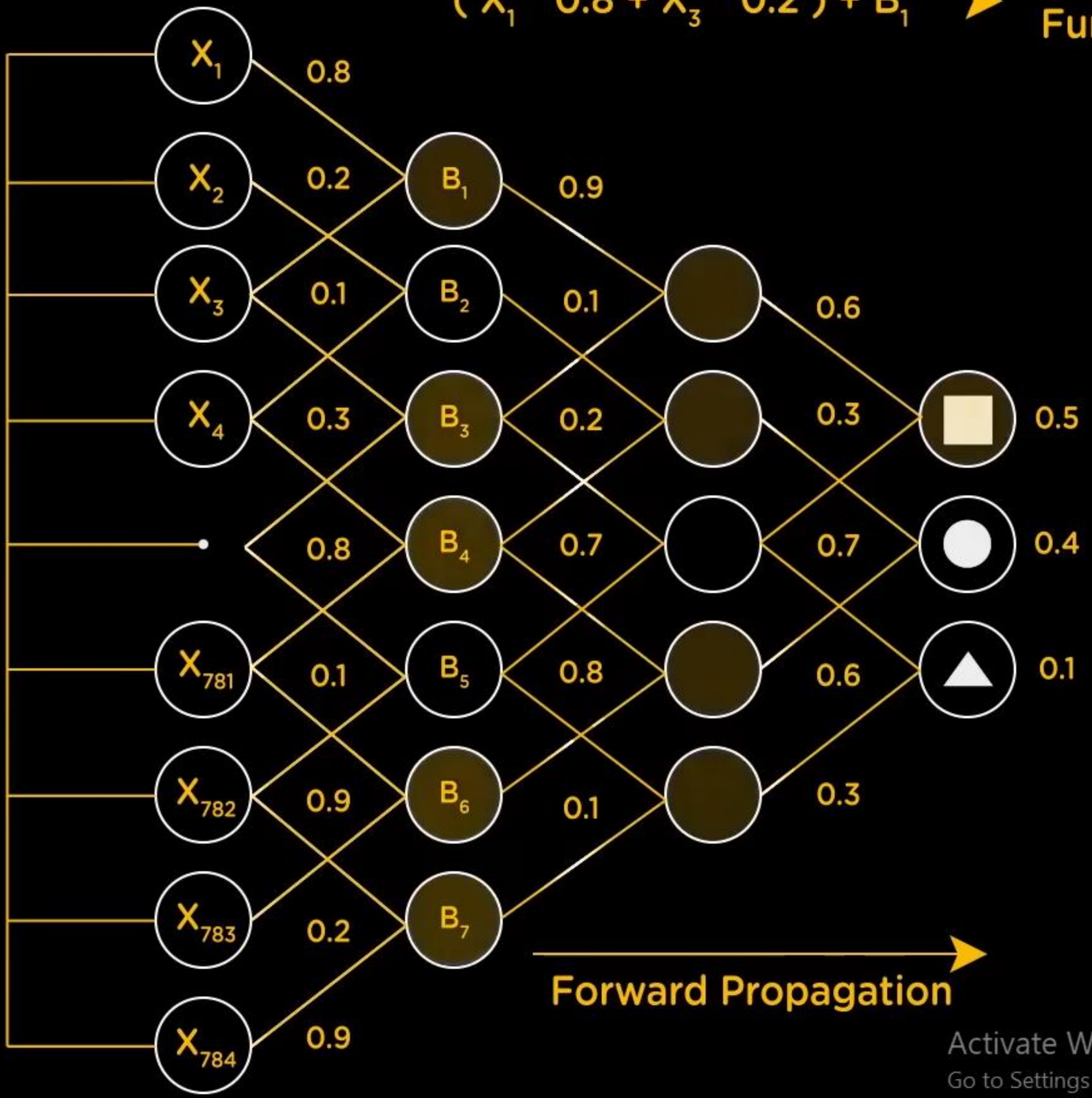


28 x 28 = 784 Pixels



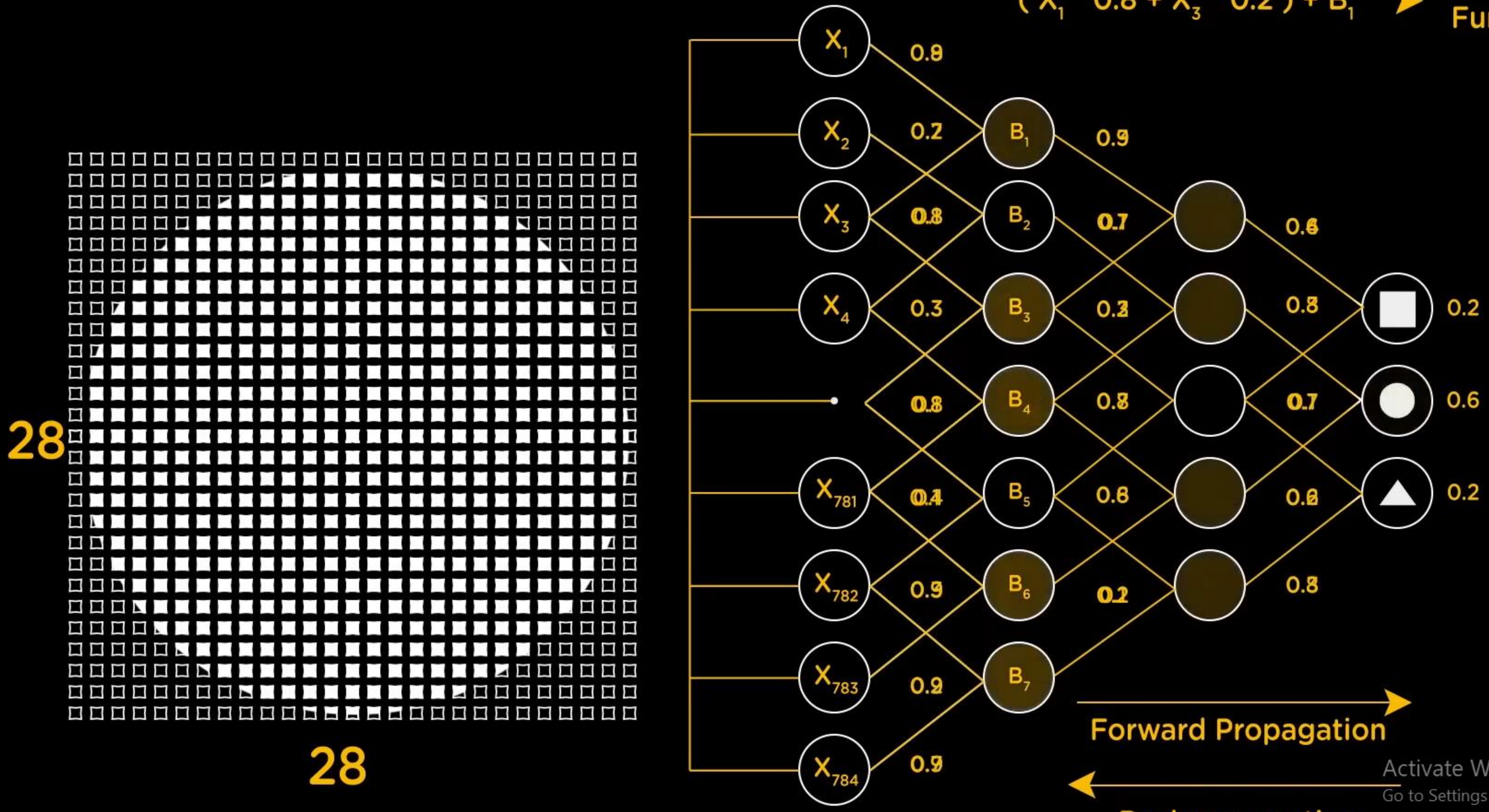
28 x 28 = 784 Pixels

$(X_1 * 0.8 + X_3 * 0.2) + B_1$ ➤ Activation Function



Actual Output	Error
0	-0.5
1	+0.6
0	-0.1

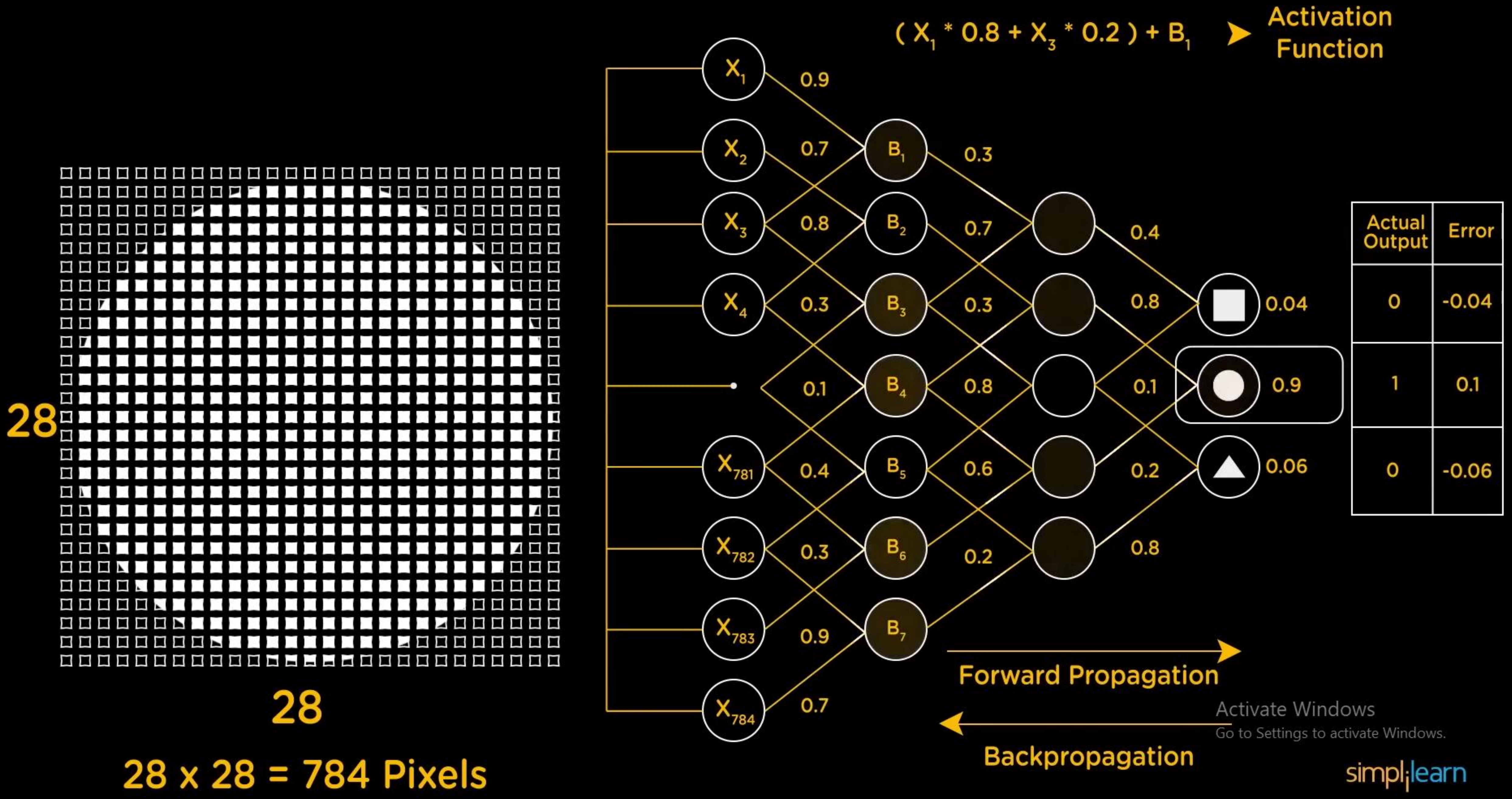
$(X_1 * 0.8 + X_3 * 0.2) + B_1$ ➤ Activation Function



28 x 28 = 784 Pixels

	Actual Output	Error
↓	0	-0.2
↑	1	+0.4
↓	0	-0.2

simplilearn



$$(X_1 * 0.8 + X_3 * 0.2) + B_1 \rightarrow \text{Activation Function}$$

Actual Output	Error
0	-0.04
1	0.1
0	-0.06

ACTIVATION FUNCTION

- A function that transforms the values or states the conditions for the decision of the output neuron is known as an **activation function**.
- What does an artificial neuron do? Simply, it calculates a “weighted sum” of its input, adds a bias and then decides whether it should be “fired” or not.
- So consider a neuron.

$$Y = \sum (weight * input) + bias$$

ACTIVATION FUNCTION

- The value of Y can be anything ranging from -inf to +inf. The neuron really doesn’t know the bounds of the value. So how do we decide whether the neuron should fire or not.
- We decided to add “activation functions” for this purpose. To check the Y value produced by a neuron and decide whether outside connections should consider this neuron as “fired” or not. Or rather let’s say — “activated” or not.

ACTIVATION FUNCTION

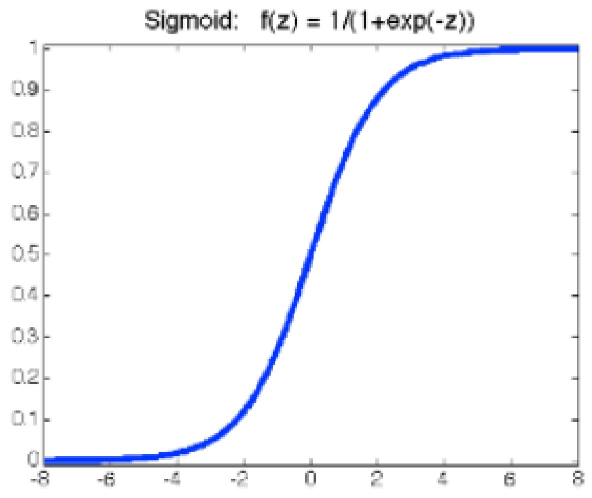
- If we do not apply an Activation function, then the output signal would simply be a simple ***linear function***. A *linear function* is just a polynomial of **one degree.**
- A linear equation is easy to solve but they are limited in their complexity and have less power to learn complex functional mappings from data.
- A Neural Network without Activation function would simply be a **Linear Regression Model,** which has limited power and does not performs good most of the times.
- We want our Neural Network to not just learn and compute a linear function but something more complicated than that.
- Also, without activation function our Neural network would not be able to learn and model other complicated kinds of data such as images, videos , audio , speech etc.

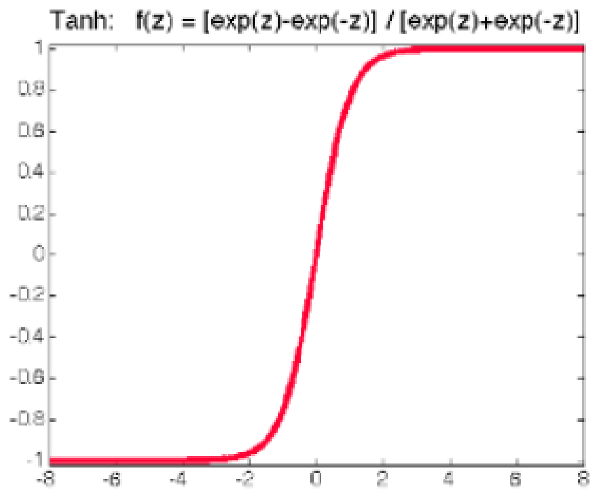
Activation Functions

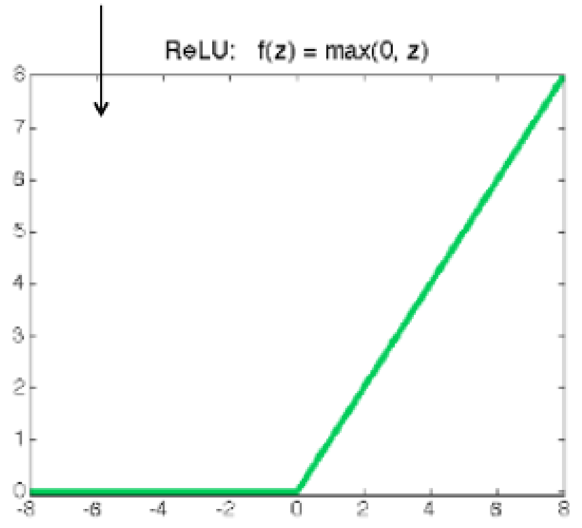
Most commonly used activation functions:

- Sigmoid: $\sigma(z) = \frac{1}{1+\exp(-z)}$
- Tanh: $\tanh(z) = \frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)}$
- ReLU (Rectified Linear Unit): $\mathrm{ReLU}(z) = \max(0, z)$

Most popular recently for deep learning







Numerical Example for a Simple ANN

The Forward Pass:

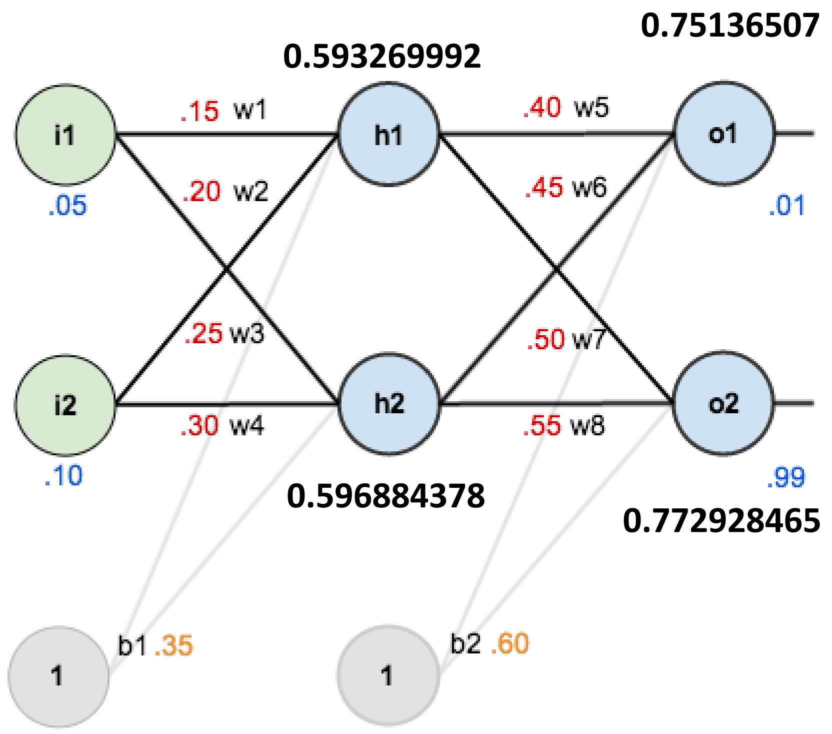
For first layer:

$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$

$net_{h1} = 0.15*0.05+0.2*0.1+0.35*1 = 0.3775$

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}} = \frac{1}{1+e^{-0.3775}} = 0.59326992$$

$out_{h2} = 0.596884378$



For output layer:

Calculating the Total Error: squared error function

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

$$E_{o1} = \sum \frac{1}{2}(0.1 - 0.75136507)^2 = 0.274811083$$

$$E_{o2} = \sum \frac{1}{2}(0.99 - 0.772928465)^2 = 0.023560026$$

$E_{total} = E_{o1} + E_{o2} = 0.298371109$

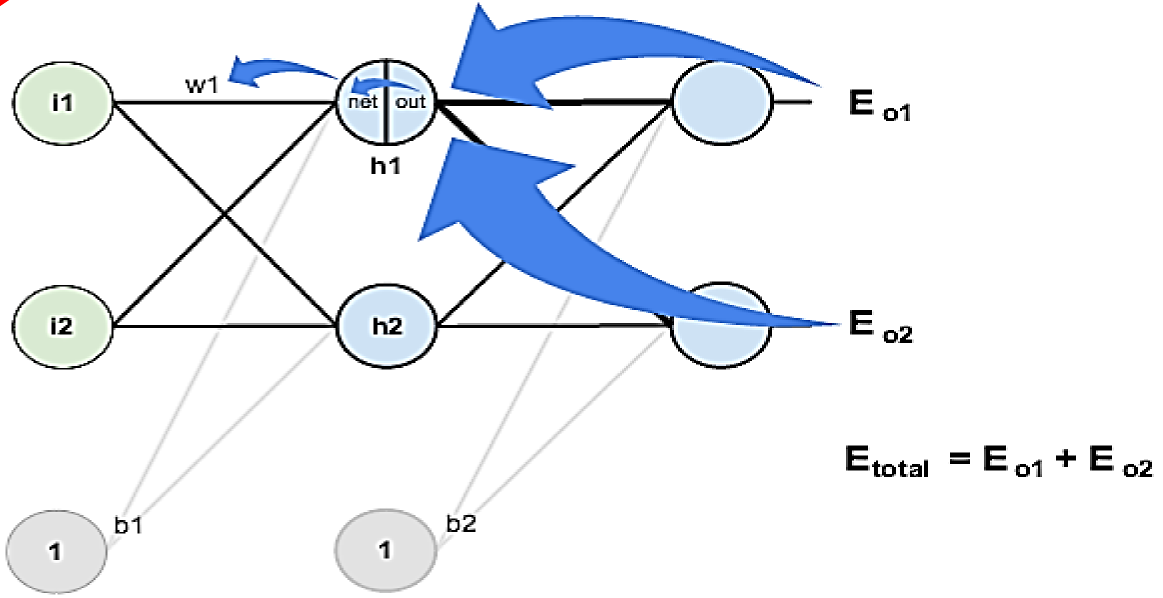
The Backwards Pass

- Our goal with backpropagation is to update each of the weights in the network so that they cause the actual output to be closer the target output, thereby minimizing the error for each output neuron and the network as a whole.

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$

Delta rule



Exercise

- Write the python code for the previous network for the tanh activation function.
- choose the weight random from interval [-0.5, 0.5].
- b1, b2= 0.5, 0.7 respectively.
- Print the output of the network.