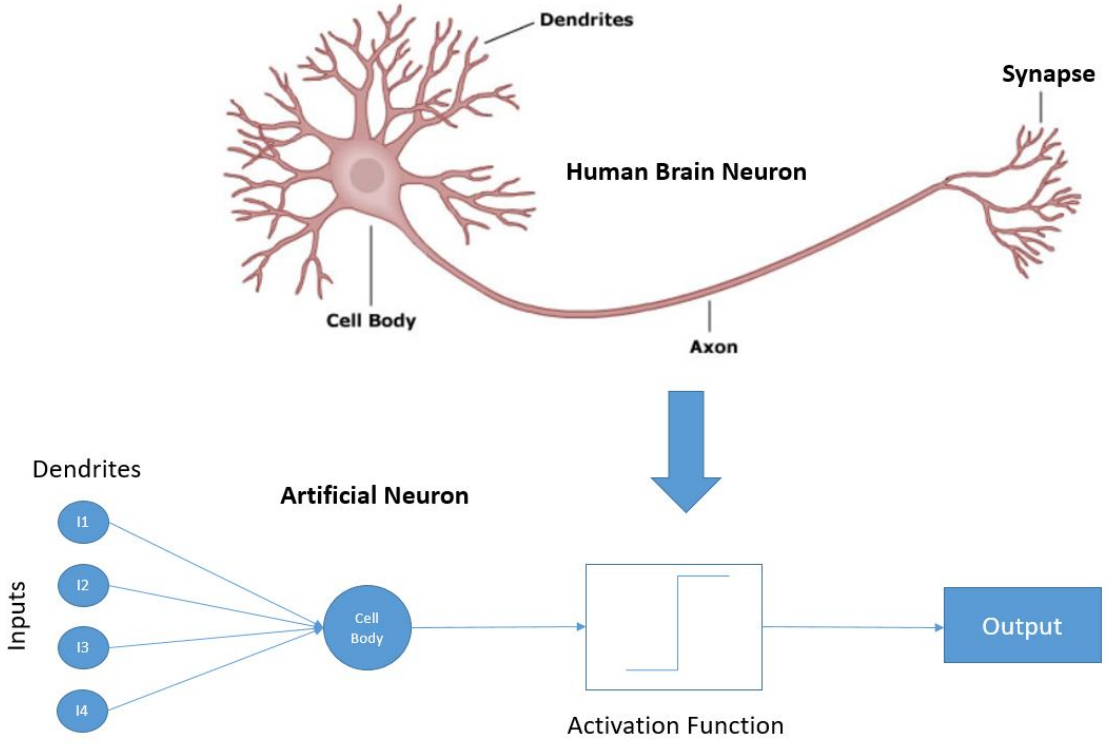
The "Deep" in Deep Leaning

The model of a neuron

The **human brain has input connections** from **other neurons (synapses) that receive stimuli** in the **form of electric charges**, and **then has a nucleus that depends on how the input stimulates** the **neuron that can trigger the neuron's activation.**

At the end of the neuron, the output signal is propagated to other neurons through dendrites, thus forming a network of neurons.



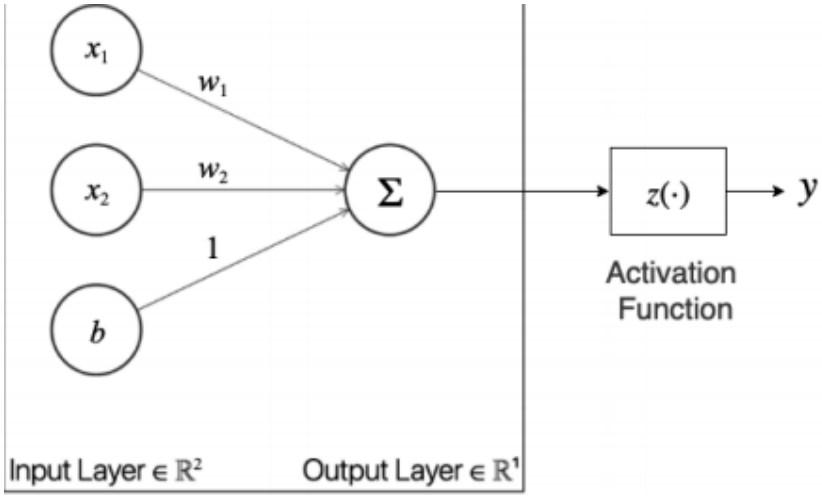
The analogy of the human neuron is depicted in the below Figure ,

where the input is **represented with the vector x**,

the **activation of the neuron is given by some function z(.)**,

and the **output** is **y**.

The **parameters** of the **neuron are w and b**:



The **trainable parameters** of a neuron are **w and b**, and **they are unknown**.

Thus, we can use **training data D** to **determine these parameters using some learning strategy**.

From the picture,

x1 multiplies w1

, **then x2 multiplies w2**

, **and b is multiplied by 1**;

all these products

are added, which can be simplified as follows:

$$x_1 w_1 + x_2 w_2 + b = \mathbf{w}^T \mathbf{x} + b$$

The **activation function operates** as a way to ensure the **output is within the desired output range.**

Let's say that we want a simple linear activation, then **the function z(.) is nonexisting or can be bypassed, as follows:**

$$z(\mathbf{w}^T \mathbf{x} + b) = \mathbf{w}^T \mathbf{x} + b$$

This is **usually the case when we want to solve a regression problem** and the output data can have a range from

-∞ to +∞.

However, **we may want to train the neuron to determine** whether a **vector x belongs** to **one of two classes, say -1 and +1**.

Then we would be better suited using a **function called a sign activation**:

$$z(\mathbf{w}^T\mathbf{x}+b) = sign(\mathbf{w}^T\mathbf{x}+b)$$

Where the sign(.) function is denoted as follows:

$$sign(\mathbf{w}^T\mathbf{x}+b) = \begin{cases} +1 & if\ \mathbf{w}^T\mathbf{x}+b \geq 0 \\ -1 & otherwise \end{cases}$$

There are **many other activation functions**, but we **will introduce those later on**.

For now, **we will briefly show one of the simplest learning algorithms**, **the perceptron learning algorithm (PLA).**

The perceptron learning algorithm

The PLA begins from the assumption that you want to classify data, X, into two different groups, the positive group (+) and the
negative group (-).

It will find some **w and b** by training to predict the **corresponding correct labels y**.

The **PLA uses the sign(.)** function as **the activation. Here are the steps that the PLA follows**:

1. **Initialize w to zeros**, and **iteration counter t = 0**
2. **While there are any incorrectly classified examples**:

```
Pick an **incorrectly classified example**, call it x*, whose true label is y*

Update w as follows: wt+1 = wt+ y*x*

Increase iteration counter t++ and repeat
```

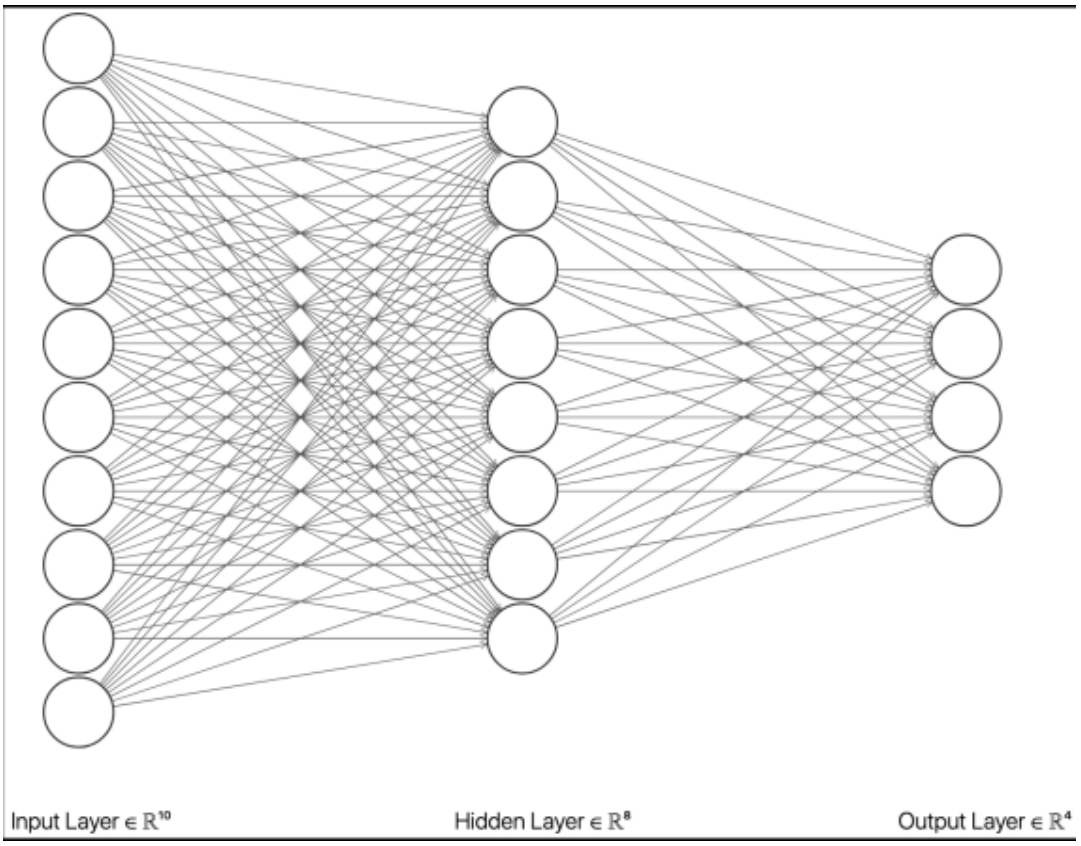
Notice that, for **the PLA to work as we want, we have to make an adjustment.**

Shallow networks

A **neural network consists of multiple networks connected in different layers**.

In contrast, a **perceptron has only one neuron and its architecture consists of an input layer and an output layer**.

In neural networks, there are **additional layers between the input and output layer**.



Neural networks can solve more difficult problems than **without a network, for** example, **with a single neural unit such as the perceptron**.

This must feel intuitive and must be easy to conceive. **A neural network can solve problems including and beyond those that are linearly separable**.

For **linearly separable problems**, **we can use both the perceptron model and a neural network.**

However, **for more complex and non-linearly separable problems**, the perceptron cannot offer a high-quality solution, while a neural network does.

Neural networks can solve more difficult problems than without a network, for example, **with a single neural unit such as the perceptron. This must feel intuitive and must be easy to conceive.**

A **neural network can solve problems including and beyond those that** are **linearly separable**.For **linearly separable problems, we can use both the perceptron model and a neural network**.

However, **for more complex and non-linearly separable** problems, **the perceptron cannot offer a high-quality solution, while a neural network does.**

Non-separable data is such that **there is no line that can separate groups of data (or classes) into two groups**.

Non-linear models, or solutions, **are those that naturally and commonly occur when the best solution to a classification problem** is not a line.

For example, it **can be some curve described by some polynomial of any degree greater than one.**

The input-to-hidden layer

In a neural network, **the input vector x** is **connected to a number of neurons through weights w for each neuron, which can be now thought of as a number of weight vectors forming a matrix W**.

The **matrix W has as many columns as neurons as the layer has**, and as many rows as the number of features (or dimensions) x has.

Thus, **the output of the hidden layer can be thought of as the following** vector:

$$\mathbf{h} = z(\mathbf{w}^T\mathbf{x} + b)$$

Where b is a vector of biases, whose elements correspond to one neural unit, and the size of h is proportional to the number of hidden units.

The hidden-to-hidden layer

In a **neural network, we could have more than one single hidden layer**, In such case, the **matrix W can be expressed as a three dimensional matrix** that will have as many elements in the **third dimension and as many hidden layers as the network has**.

In the case of the i-th layer, we will refer to that matrix as Wi for convenience.

Therefore, we can refer to the output of the i-th hidden layer as follows:

$$\mathbf{h}_i = z(\mathbf{W}_i^T\mathbf{h}_{i-1} + \mathbf{b}_i)$$

The hidden-to-output layer

The overall output of the network is the output at the last layer

$$\mathbf{h}_k = z(\mathbf{W}_k^T\mathbf{h}_{k-1} + \mathbf{b}_k)$$

Here, the **last activation function is usually different from the hidden layer activations**.

The **activation function in the last layer (output) traditionally depends on the type of problem** we are trying to solve.