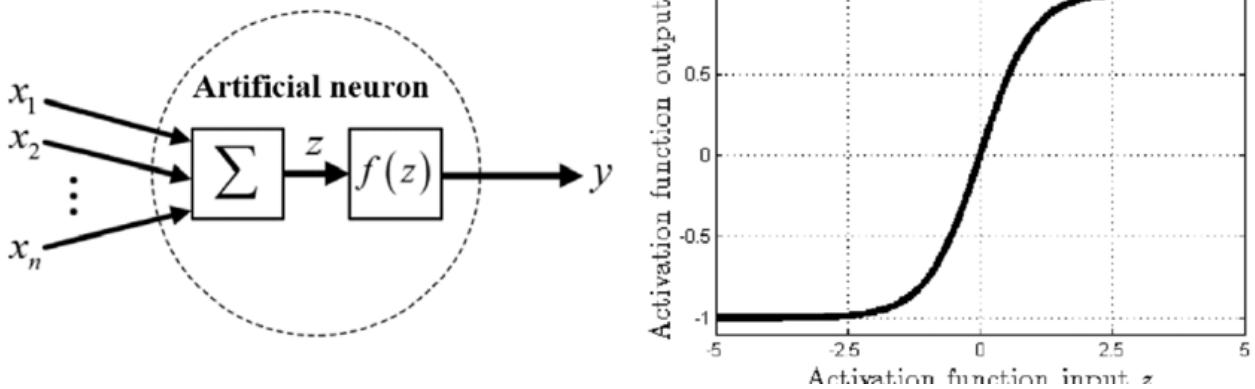
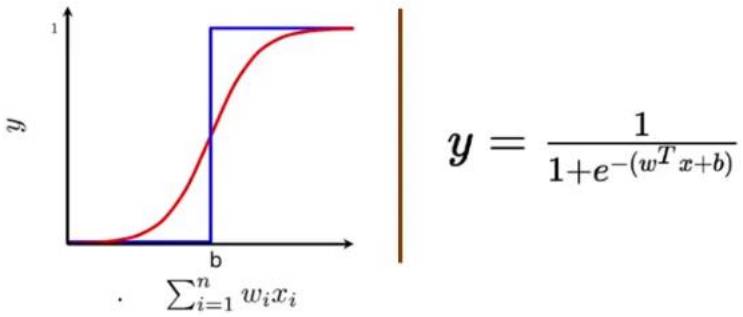
Sigmoid Neurons

Sigmoid functions are the fundamental building block of the deep neural network. Sigmoid functions are similar to perceptrons and MP Neuron Model, but the significant difference is that sigmoid neurons are smoother at the boundary than perceptrons and MP Neuron Model.

In a sigmoid neuron, for every input xi, it weights wi associated with it. The weights depict the importance of the input in the decision-making process. The output from sigmoid ranges between zero to one, which we can interpret as a probability rather than zero or one like in the perceptron model. One of the most commonly used sigmoid functions is the logistic function, characteristic of an "S" shaped curve.





Sigmoid Neuron Representation (logistic function)

Introducing sigmoid neurons where the output function is much smoother than the step function. In the sigmoid neuron, a small change in the input only causes a small change in the output as opposed to the stepped output. There are many functions with the characteristic of an "S" shaped curve known as sigmoid functions.

Learning Algorithm:

An algorithm for learning the parameters w and b of the sigmoid neuron model use the gradient descent algorithm.

Find w and b such that:

$$\underset{w,b}{\text{minimize}}\ \mathscr{L}(w,b) = \sum_{i=1}^{N} (y_i - f(x_i))^2$$

The objective of the learning algorithm is to determine the best possible values for the parameters, such that the overall loss (squared error loss) of the model is minimized as much as possible. Here goes the learning algorithm:

Initialise w, b

Iterate over data:

compute $\hat{y}$

compute $\mathscr{L}(w,b)$

$w_{t+1} = w_t - \eta \Delta w_t$

$b_{t+1} = b_t - \eta \Delta b_t$

till satisfied

Initialize w and b randomly. We then iterate over all the observations in the data, for each observation find the corresponding predicted outcome using the sigmoid function and compute the squared error loss. Based on the loss value, we will update the weights such that the overall loss of the model at the new parameters will be less than the current loss of the model.

Keep doing the update operation until,

- The overall loss of the model becomes zero.
- The overall loss of the model becomes a very small value closer to zero.
- Iterating for a fixed number of passes based on computational capacity.

Why Sigmoid Neuron?

Let's take an example, we have a person's salary in thousands and based on that, and we are trying to decide whether the person can buy a car or not. Our Perceptron model has a threshold of 50k. So model says that a person with a 50.1k salary can likely buy a car, and the person with a 49.9k wage can not. This decision made by Perceptron is very harsh in real-time, whereas we generally make a smooth decision. If we think practically, isn't it a bit odd that a person with 50.1K can buy a car, but someone with 49.9K can not buy a car? The slight change in the input to a perceptron can sometimes cause the output to flip, say from zero to one ultimately. This behavior showed by the perceptron model is not a characteristic of the specific problem we choose or the particular weight or the threshold we define. This is the behavior of the perceptron neuron itself, which behaves like a step function. We can overcome this problem by introducing a new artificial neuron called a sigmoid neuron.