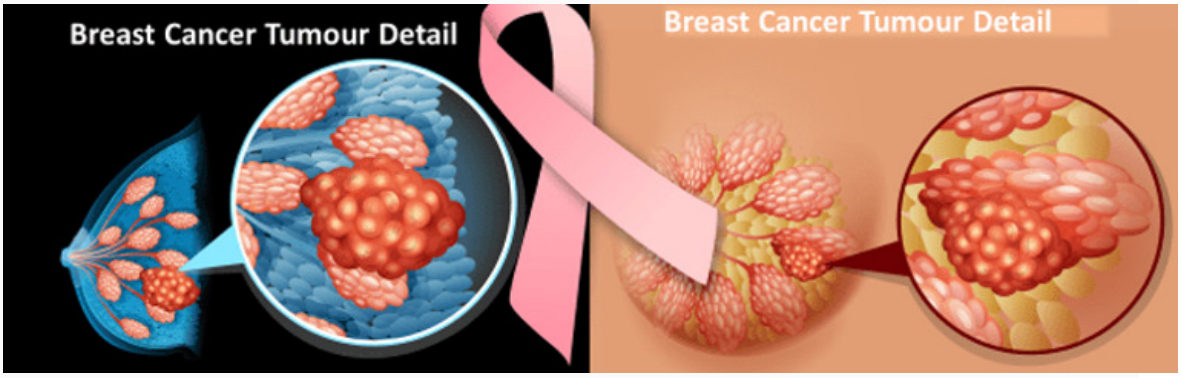
Project: Breast cancer prediction using deep learning

Breast cancer is a dangerous disease for women. If it does not identify in the early-stage then the result will be the death of the patient. It is a common cancer in women worldwide. Worldwide near about 12% of women affected by breast cancer and the number is still increasing.

The doctors do not identify each and every breast cancer patient. That's the reason Machine Learning Engineer / Data Scientist comes into the picture because they have knowledge of maths and computational power.

Goal of the project

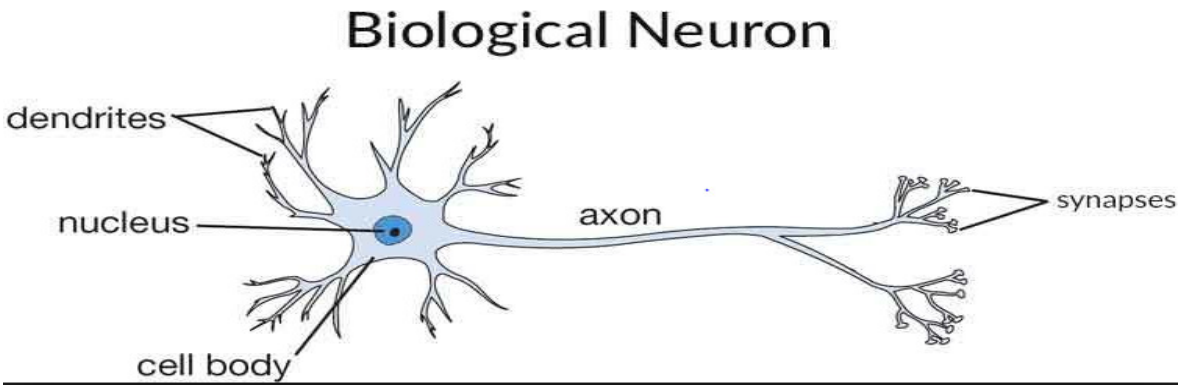
We have extracted features of breast cancer patient cells and normal person cells. As a Data Scientist has to create an ML model to classify **malignant** and **benign** tumor. To complete this project we are using the artificial neural network classifier algorithm.



Biological Neural Networks:

- The incoming signals come in through the dendrites which is a biochemical process and enters a neural circuit.
- A neural circuit is a population of neurons interconnected by synapses to carry out a specific function when activated. Neural circuits interconnect to one another to form large scale brain networks.

Biological neural networks have inspired the design of artificial neural networks, but artificial neural networks are usually not strict copies of their biological counterparts

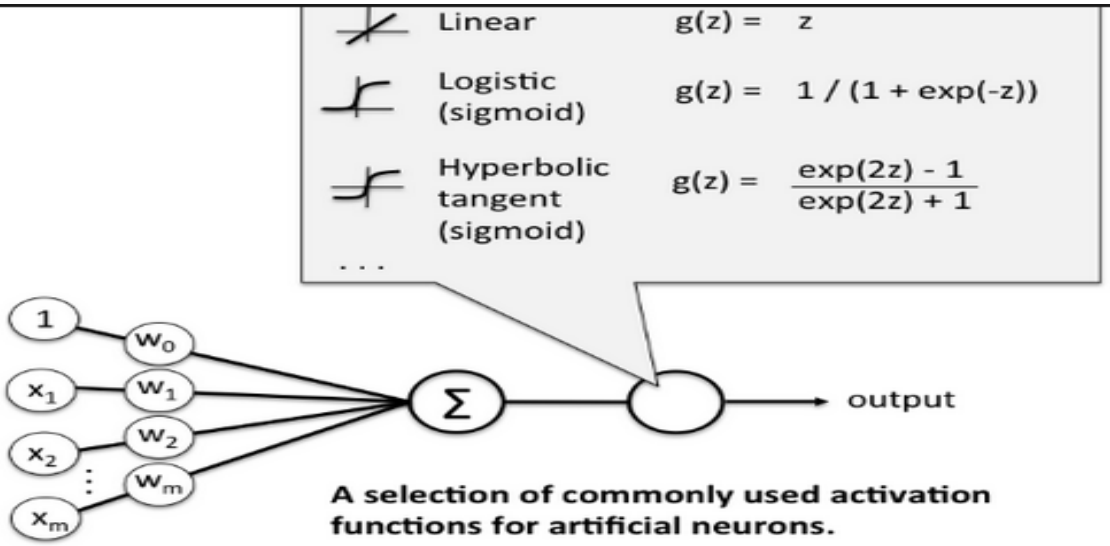


- A cockroach has around a million neurons.
- A mouse has around a 75 million neurons.
- A cat has around a billion neurons.
- A human has around a 85 billion neurons.
- The scientists until now have achieved to create only few hundred artificial neurons, so artificial brain can't be achieved in the near future.

Artificial Neural Networks:

ANN models the relationship between a set of input signals and output signals using a model derived from our understanding of how a biological brain responds to stimuli from sensory inputs.

Similarly, like how a brain uses a network of interconnected cells called neurons to create a massive parallel processor, ANN uses a network of artificial nodes(similar to neurons) to solve learning problems.



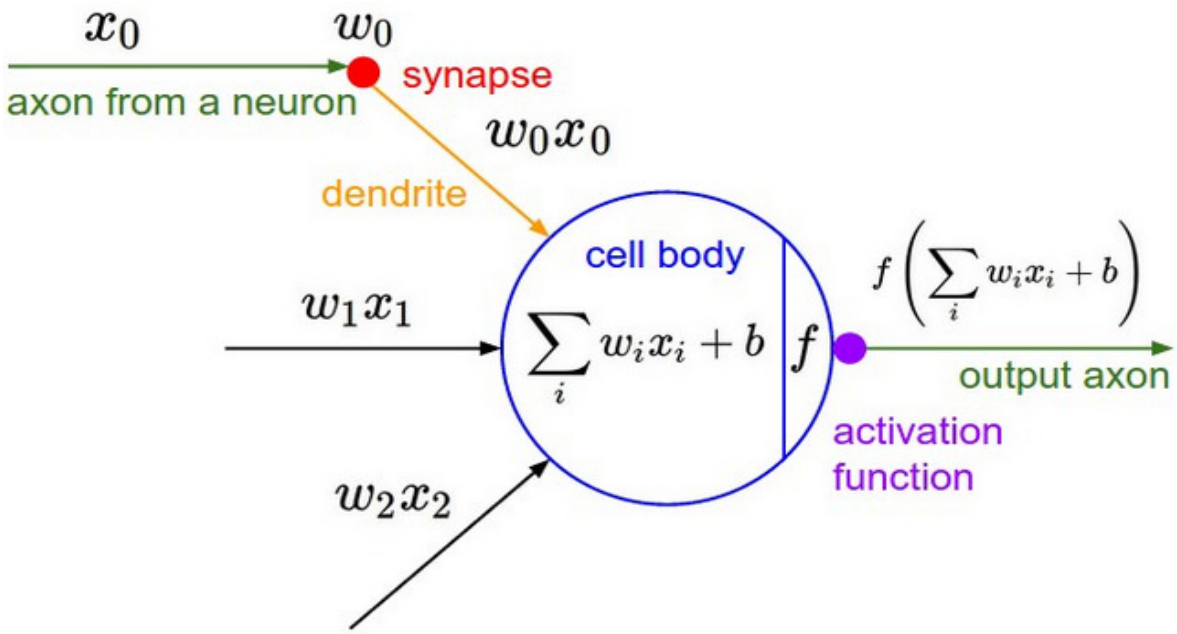
A selection of commonly used activation functions for artificial neurons.

From the above diagram, the input signals(x) are received by nodes then the weights(w) of the signals are given depending on the strength then the signal, then the summation of the signals is passed through the activation function f. Then we get the output(y).

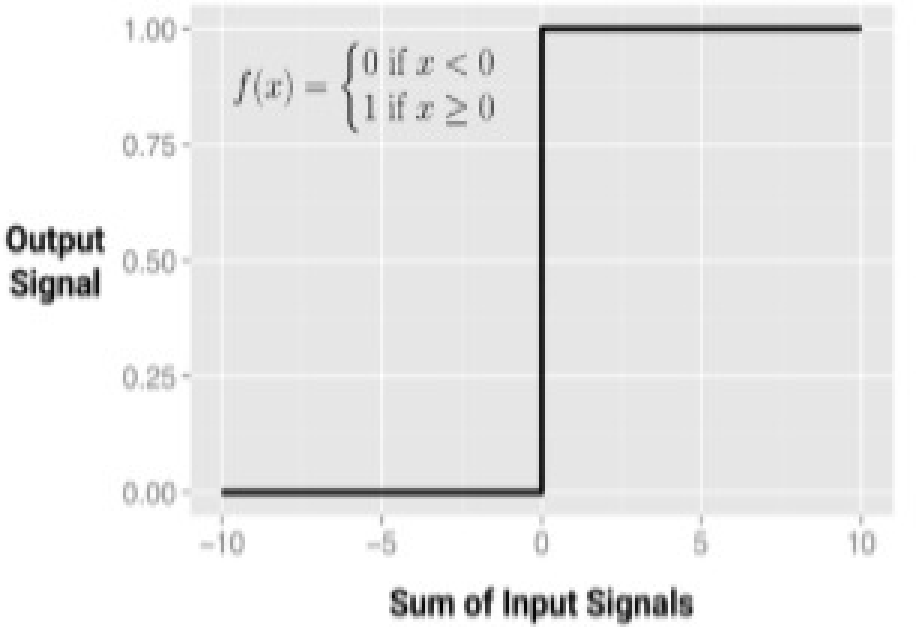
- The equation can be simplified as:

$$y(x) = f\left(\sum_{i=1}^{n} w_i x_i\right)$$

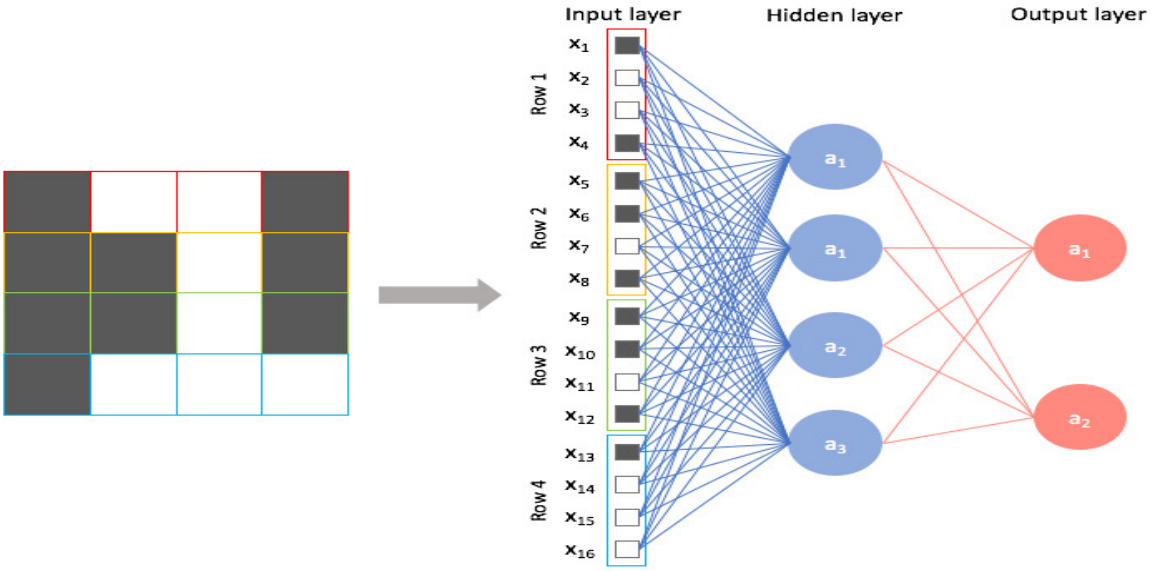
- Where, f= activation function
- n = number of inputs
- w = weights assigned to the input signal

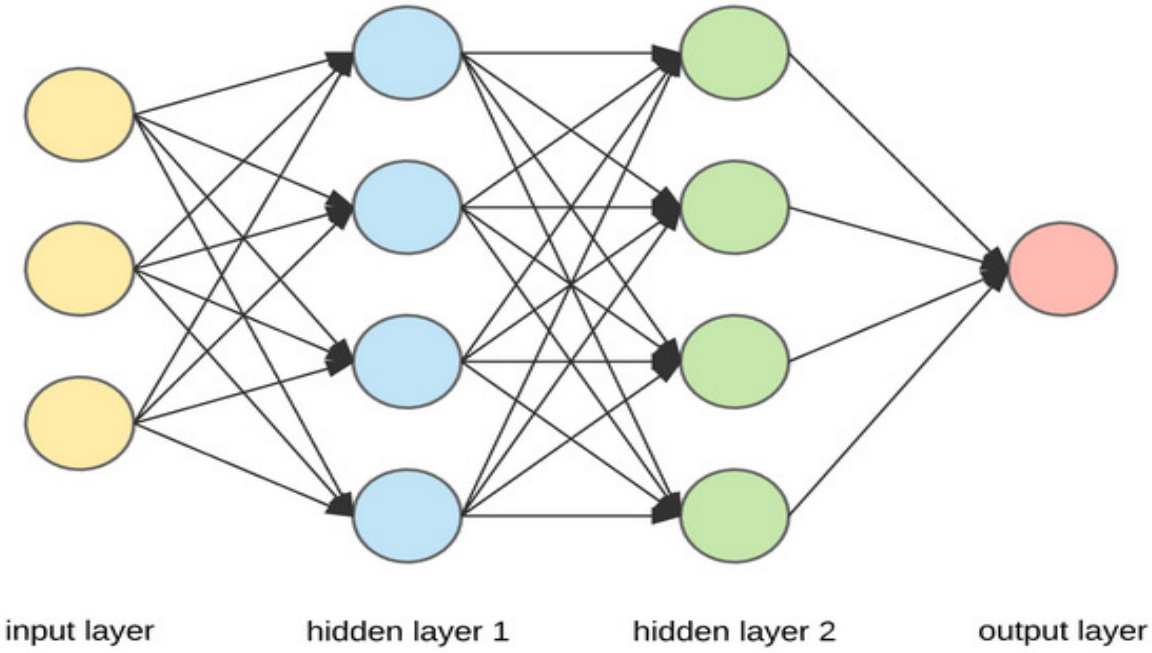


- In biological sense, the activation function can be imagined as a process that involves whether the inputs meets the firing threshold to perform a task or take a decision. If so, the neuron passes on the signal; otherwise, it does nothing.
- In ANN terms, this is known as a threshold activation function, as it results in an output signal only once a specified input threshold has been attained
- The following figure depicts a typical threshold function
- in this case, the neuron fires when the sum of input signals is at least zero. Because its shape resembles a stair, it is sometimes called a unit step activation function



- The ability of a neural network to learn is rooted in its topology
- Neural Networks Topology: patterns and structures of interconnected neurons





Python code:

Importing libraries:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Reading the dataset:

```
wbcd = pd.read_csv("C:/Users/GOPINADH/Desktop/datasets/wbcd.csv")
```

Display the first five records of the dataset:

```
wbcd.head()
```

us_worst	texture_worst	perimeter_worst	area_worst	smoothness_worst	compactness_worst	concavity_worst	points_worst	symmetry_worst	dimension_worst
13.50	15.64	86.97	549.1	0.1385	0.1266	0.12420	0.09391	0.2827	0.06771
11.88	22.94	78.28	424.8	0.1213	0.2515	0.19160	0.07926	0.2940	0.07587
12.41	26.44	79.93	471.4	0.1369	0.1482	0.10670	0.07431	0.2998	0.07881
11.92	15.77	76.53	434.0	0.1367	0.1822	0.08669	0.08611	0.2102	0.06784
16.20	15.73	104.50	819.1	0.1126	0.1737	0.13620	0.08178	0.2487	0.06766

Drop the unnecessary columns from the dataset:

```
wbcd.drop(["id"],axis=1,inplace=True)
```

Find the columns of the dataset:

```
wbcd.columns
```

```
Index(['diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'points_mean', 'symmetry_mean', 'dimension_mean', 'radius_se',
       'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'points_se', 'symmetry_se',
       'dimension_se', 'radius_worst', 'texture_worst', 'perimeter_worst',
       'area_worst', 'smoothness_worst', 'compactness_worst',
       'concavity_worst', 'points_worst', 'symmetry_worst', 'dimension_worst'],
      dtype='object')
```

Find the shape of the dataset:

```
wbcd.shape
```

```
(569, 31)
```

Find the missing values of the dataset:

```
wbcd.isnull().sum() # No missing values
```

```
diagnosis              0
radius_mean            0
texture_mean           0
perimeter_mean         0
area_mean              0
smoothness_mean        0
compactness_mean       0
concavity_mean         0
points_mean            0
symmetry_mean          0
dimension_mean         0
radius_se              0
texture_se             0
perimeter_se           0
area_se                0
smoothness_se          0
compactness_se         0
concavity_se           0
points_se              0
symmetry_se            0
dimension_se           0
radius_worst           0
texture_worst          0
perimeter_worst        0
area_worst             0
smoothness_worst       0
```

Malignant as 0 and Beningn as 1

```
#  Malignant as 0 and Beningn as 1

wbcd.loc[wbcd.diagnosis=="B","diagnosis"] = 1
wbcd.loc[wbcd.diagnosis=="M","diagnosis"] = 0
```

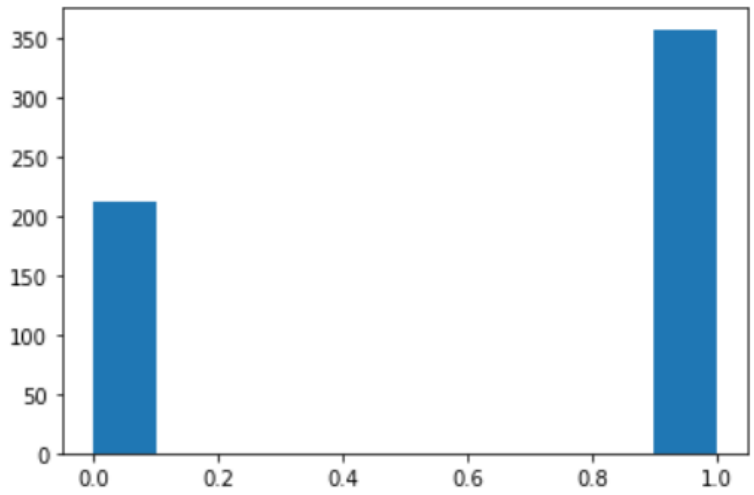
Selecting the x and y variables:

```
X = wbcd.drop(["diagnosis"],axis=1)
Y = wbcd["diagnosis"]
```

Find the distribution of the dataset:

```
plt.hist(Y)
```

```
(array([212.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0., 357.]),
 array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]),
 <a list of 10 Patch objects>)
```



Analyzing the target feature:

```
wbcd.diagnosis.value_counts()
```

```
1    357
0    212
Name: diagnosis, dtype: int64
```

Train and test split:

```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(X,Y,test_size=0.3)
```

Normalization:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(x_train)
```

Model building:

```
from sklearn.neural_network import MLPClassifier

mlp = MLPClassifier(hidden_layer_sizes= (30,30,15),activation='logistic',max_iter=200)
mlp.fit(x_train,y_train)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:566: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
MLPClassifier(activation='logistic', alpha=0.0001, batch_size='auto',
              beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(30, 30, 15), learning_rate='constant',
              learning_rate_init=0.001, max_iter=200, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
```

Prediction:

```
predictions = mlp.predict(x_test)
```

Confusion Metrix and accuracy:

```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test,predictions))
np.mean(y_test==predictions)
```

```
[[ 61   2]
 [  1 107]]

0.9824561403508771
```