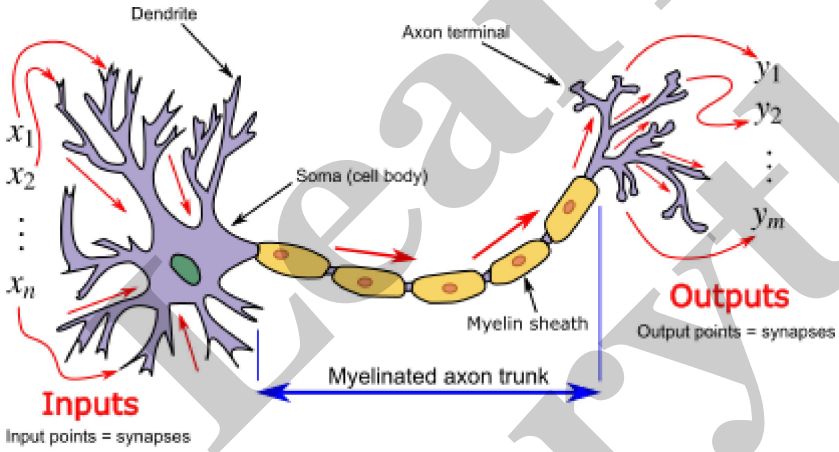
What is Neural Network?

Neural Network (also Artificial Neural Network) is a popular machine learning technique based on the structure and functioning of the human brain. It is a structure that provides the connection between input and output data.

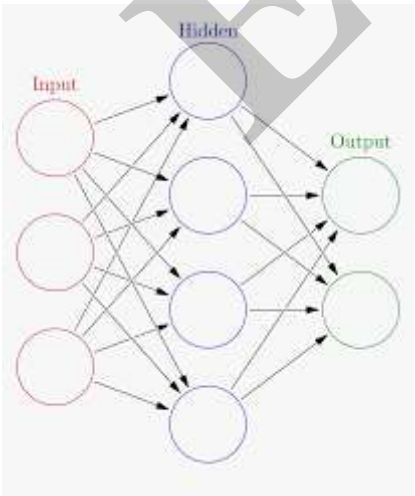
Biological reference of Neural Network

Realising that the brain can solve many complexes in a much easier way than the computer such as image recognition, speech recognition etc. Natural neuron receives a signal from "Synapses" present on the membrane of the neuron. When the signal is strong enough, the neuron is activated and emits a signal through the axon.



Comparing the ANN with natural NN we get the following information:

1. ANN consist of input like synapses
2. which are multiplied by weights like the strength of the signals
3. using some mathematics for the activation of the neurons



Application of Neural Network (or ANN)

Three major application of ANN lies in the areas of:

1. Classification
2. Time series forecasting
3. Function approximation

Architecture of Artificial Neural Network

It consists of nodes and directed graphs (i.e., Weights) that are connected between neuron inputs and neuron outputs. It consists of the following parts:

- **Input Layer**, consisting of 'n' numbers of neurons
- **Hidden Layer**, consist of one or more hidden layer of 'm' numbers of neurons
- **Output Layer**, consists of 'p' neurons (one for each output)

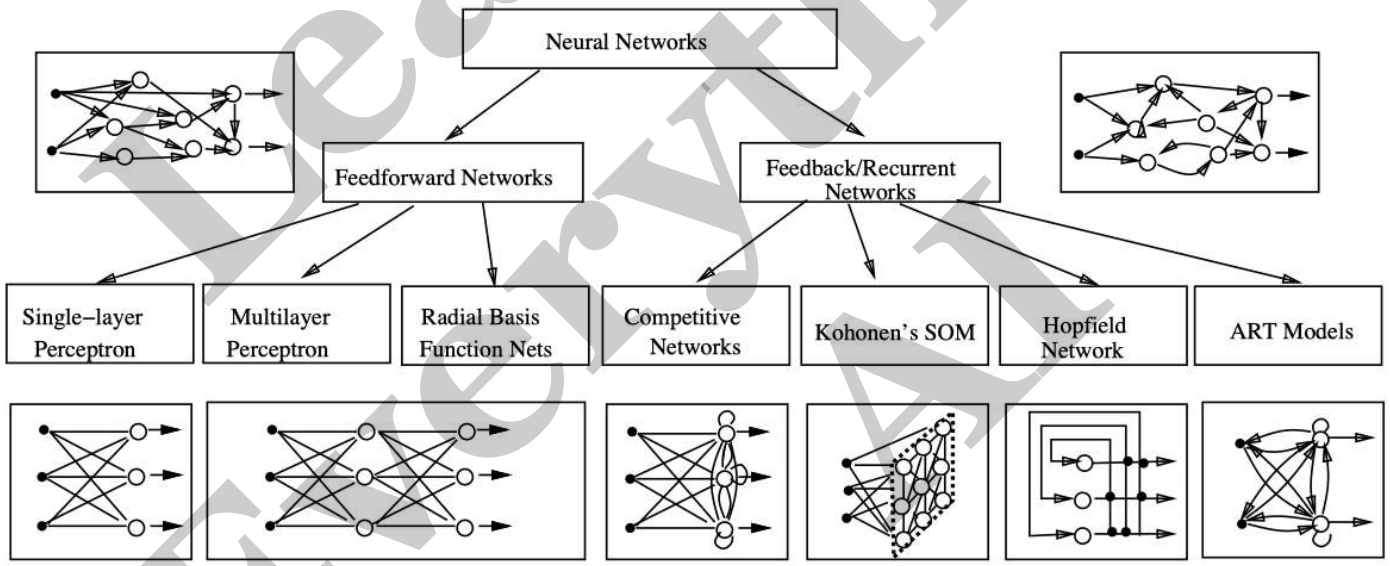


Figure 1: Different network architectures

Based on the connection pattern shown on the above flowchart this can be categorised into two:

1. Feed-forward propagation
2. Feed-back propagation (Recurrent)

Feed-forward Propagation

(Reference: link text)

They produce only a single set of output values rather than a sequence of values from input. In feed-forward, their response to an input does not depend upon the previous state of the network.

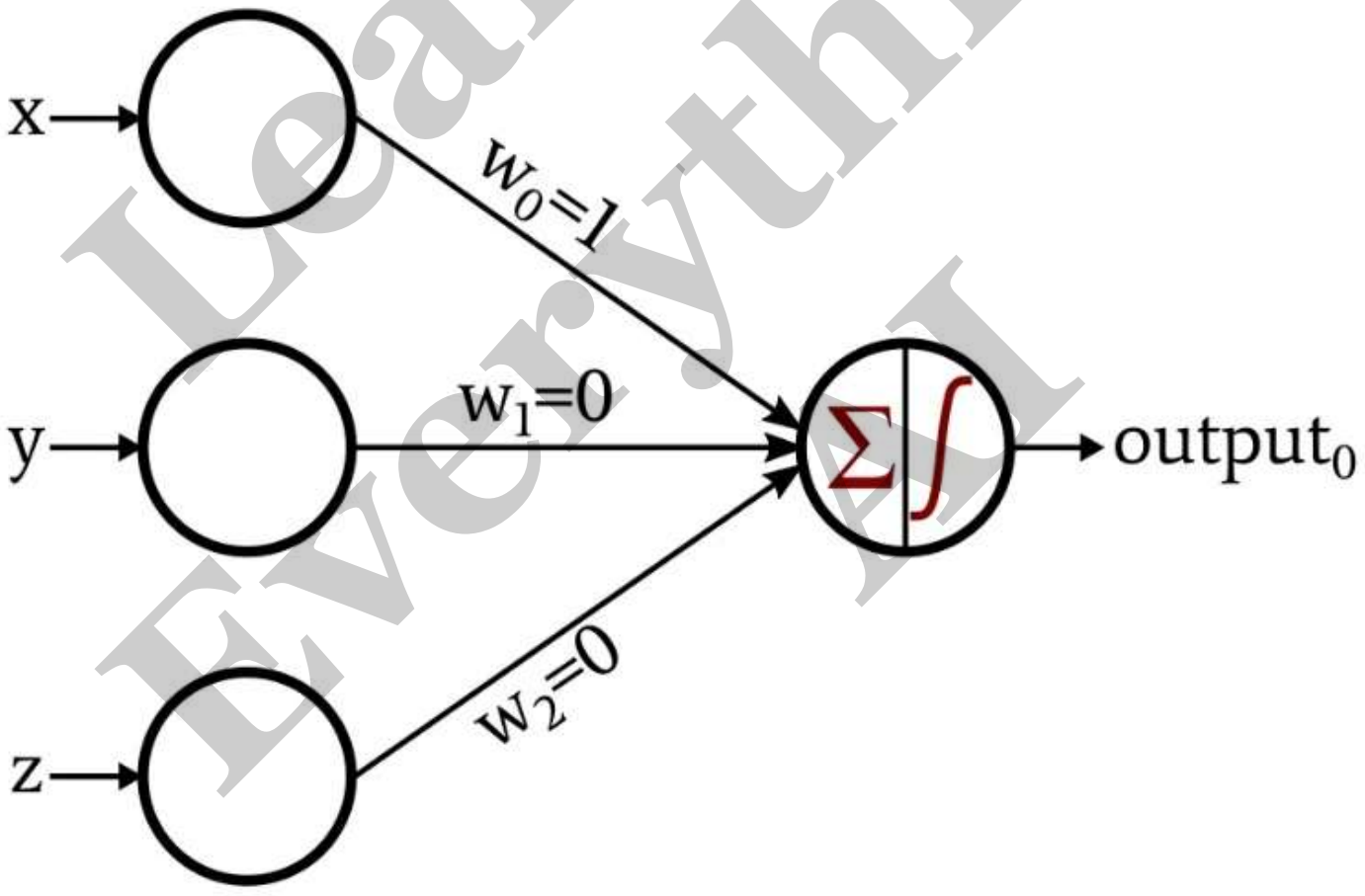
Feed-back Propagation

It is a dynamic architecture. In this, whenever a new input pattern is presented, the neuron output is computed. In general, the output received will propagate back and act as a new input for the architecture

Concept of Perceptron

(Reference: link text)

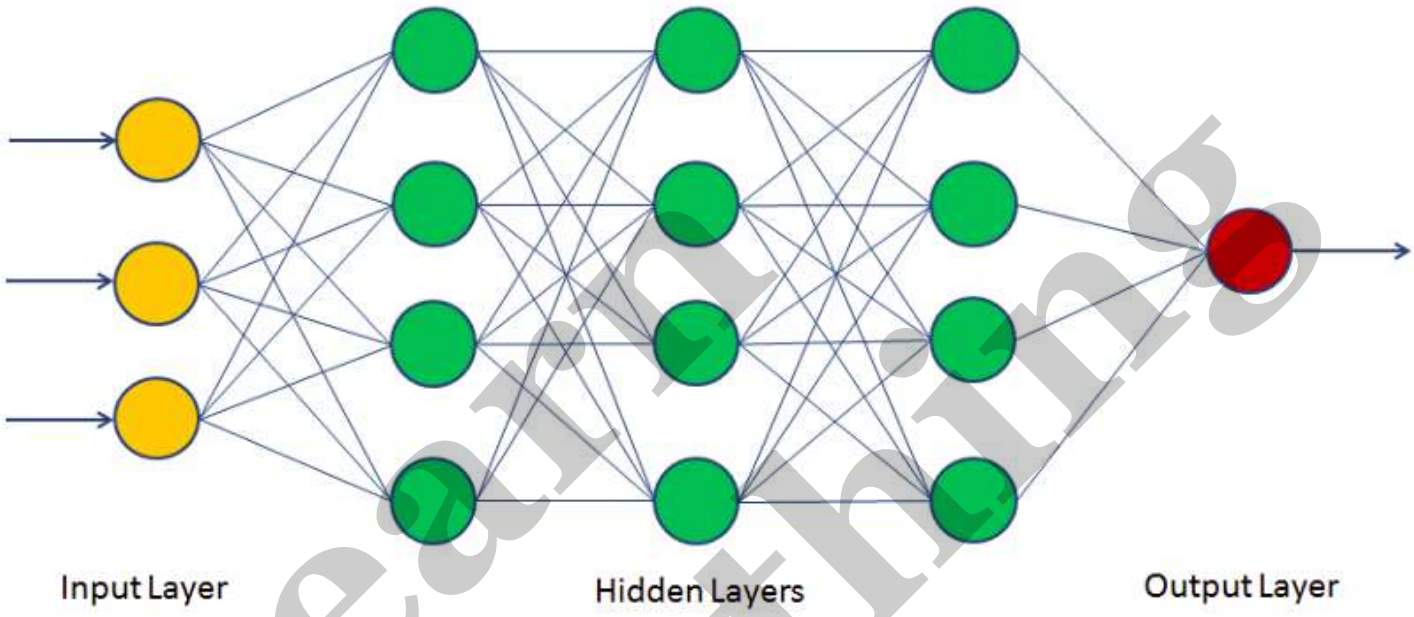
The simplest neural architecture is considered as a perceptron, that is, a neural network that contains a single input and a single output. The architecture of perceptron is shown below:



In the above figure the edge from the input to output contains some weights namely $W_0, W_1 \ldots\ldots W_n$ with which the features are multiplied respectively and added to the output node. Here 'f' represents the activation function. It is used in order to convert the average value into a labelled class as per the required output.

Multi Layer Perceptron

A neural network consisting of an input layer, two or more intermediate layers (also hidden layer) of neuron, and output layer then the architecture is known as MLP. Here, the weight between the layers is calculated starting from some random numeric value and hence making slightly small changes after each output error until the algorithm converges to an acceptable error approximation.



Backpropagation

(Reference: link text)

In order to determine the feature which contributes the most for the classification problem of a particular instance, the concept of backpropagation was introduced. This approach helps in understanding the importance of each feature during the classification. It is also a useful property for the feature selection purpose.

For a better understanding, let consider a test $X = (x_1, x_2 \ldots . x_d)$, for multi-label output $o_1, o_2 \ldots o_n$. Let O_w be the output of winning class from the m output be O_p, $m \varepsilon [1 \ldots . m]$ Our task is, for each x_i, we have to determine the change of the output O_p. In general, we have to identify the features that are most relevant for the classification task.

Solution of this can be carried out by computing the absolute magnitude value of $\frac{\partial O_m}{\partial x_i}$. The features having the greatest absolute value of the partial derivative have the greatest influence on the classification of the winning class. Other classes, derivatives also provide some understanding to the sensitivity, but its importance is very less, more specifically when the number of classes/features is more in quantity.

Recognizing Handwritten Digits by using ANN

The task will use the MNIST dataset of handwritten digits from 0-9. This is a dataset of 60,000 28x28 grayscale images of the 10 digits, along with a test set of 10,000 images. Following steps will be taken during implementation:

1. Importing Dataset
2. Reshaping the dataset
3. Splitting the dataset into test and train
4. Defining Model
5. Training model
6. Predicting accuracy

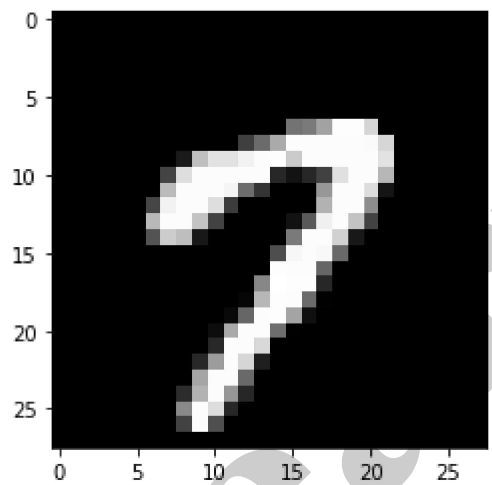

```
#Apply ANN on MNIST Dataset

import tensorflow.keras as keras
from tensorflow.keras.datasets import mnist   ##importing required libraries
import matplotlib.pyplot as plt
(x_train, y_train), (x_valid, y_valid) = mnist.load_data()  ##loading data set
print(x_train.shape)  #exploring the dataset
print(x_valid.shape)
print(x_train.dtype)
```

```
(60000, 28, 28)
(10000, 28, 28)
uint8
```

```
# Checking which image consists of what number
x_train[15]
image = x_train[15]
plt.imshow(image, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7fda46b90e90>
```



```
y_train[15]
```

```
7
```

```
#reshaping the dataset i.e., transposing the matrix
x_train = x_train.reshape(60000, 784)
x_valid = x_valid.reshape(10000, 784)
x_train.shape
x_train[15]
```

```
0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,

0,   0,   0, 115, 121, 162, 253, 253, 213,   0,   0,   0,   0,
0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
0,   0,  63, 107, 170, 251, 252, 252, 252, 252, 250, 214,   0,
0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
```

```
 25, 192, 226, 226, 241, 252, 253, 202, 252, 252, 252, 252, 252,
225,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,  68, 223, 252, 252, 252, 252, 252,  39,  19,  39,  65, 224,
252, 252, 183,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0, 186, 252, 252, 252, 245, 108,  53,   0,   0,   0,
150, 252, 252, 220,  20,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,  70, 242, 252, 252, 222,  59,   0,   0,   0,
  0,   0, 178, 252, 252, 141,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0, 185, 252, 252, 194,  67,   0,   0,
  0,   0,  17,  90, 240, 252, 194,  67,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,  83, 205, 190,  24,   0,
  0,   0,   0,   0, 121, 252, 252, 209,  24,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,  77, 247, 252, 248, 106,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0, 253, 252, 252, 102,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0, 134, 255, 253, 253,  39,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   6, 183, 253, 252,
107,   2,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  10, 102, 252,
253, 163,  16,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  13, 168,
252, 252, 110,   2,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
 41, 252, 252, 217,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,  40, 155, 252, 214,  31,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0, 165, 252, 252, 106,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,  43, 179, 252, 150,  39,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0, 137, 252, 221,  39,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,  67, 252,  79,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0], dtype=uint8)
```

```
#defining train and valid set for x variable
#assigning it to x_train & x_valid
x_train = x_train / 255
x_valid = x_valid / 255
x_train.dtype
```

```
dtype('float64')
```

```
#defining the train and valid data for y variable
#assigning it to y_train & y_valid
num_categories = 10
```

```
y_train = keras.utils.to_categorical(y_train, num_categories)
y_valid = keras.utils.to_categorical(y_valid, num_categories)
y_train[0:9]
```

```
array([[0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.]], dtype=float32)
```

```
#Defining the ANN model
from tensorflow.keras.models import Sequential
model = Sequential()
from tensorflow.keras.layers import Dense #layers - Returns all the layers of the model as li
model.add(Dense(units=512, activation='relu', input_shape=(784,))) #add() is used to add the
model.add(Dense(units = 512, activation='relu')) #relu is an activation function similarly si
model.add(Dense(units = 10, activation='sigmoid'))
model.summary()#Summary is used to get the full information about the model and its layers.
model.compile(loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit( x_train, y_train, epochs=5, verbose=1, validation_data=(x_valid, y_valid
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 512)               401920

 dense_1 (Dense)             (None, 512)               262656

 dense_2 (Dense)             (None, 10)                5130

=================================================================
Total params: 669,706
Trainable params: 669,706
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
1875/1875 [==============================] - 30s 16ms/step - loss: 0.1915 - accuracy: 0
Epoch 2/5
1875/1875 [==============================] - 24s 13ms/step - loss: 0.1009 - accuracy: 0
Epoch 3/5
1875/1875 [==============================] - 25s 13ms/step - loss: 0.0830 - accuracy: 0
Epoch 4/5
1875/1875 [==============================] - 26s 14ms/step - loss: 0.0736 - accuracy: 0
Epoch 5/5
1875/1875 [==============================] - 25s 13ms/step - loss: 0.0633 - accuracy: 0
```

Breast Cancer Prediction by implementing ANN from scratch

```
import sklearn #importing required library for loading dataset
from sklearn.datasets import load_breast_cancer
import matplotlib as mpl   # for Visualisation
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import tqdm as tqdm # for progress bar
import warnings
warnings.filterwarnings("ignore")  #suppressed warnings
```

```
full_df = pd.read_csv('/content/breastcancer.csv') #preprocessed data
full_df.drop(['Unnamed: 0'], inplace=True, axis=1)
start_index = 126 #@param {type:"slider", min:0, max:564, step:1}
full_df[start_index:start_index+5]
```

start_index: 126

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	con po
126	-0.146918	1.256698	-0.173371	-0.234039	-0.269025	-0.487451	-0.451343	-0.46
127	1.383918	-0.088347	1.294648	1.374017	-1.009858	-0.455991	0.049099	0.18

```
X_train = full_df.drop('target', inplace=False, axis=1) #removing 'target' column from the da
y_train = full_df['target'] #storing target in a separate array

X_train = X_train.reset_index(drop=True) #to rearrange the data we use reset
y_train = y_train.reset_index(drop=True)

X_train = np.array(X_train, dtype=float) #converting numpy arrays with float
y_train = np.array(y_train, dtype=float)

y_train = np.array(y_train).reshape(-1, 1) #reshaping for matrix multiplication possible
```

```
import numpy as np
def binary_crossentropy(y, yhat): #defining the loss function
   if y == 0:
    return -np.log(1.0-yhat)

   if y == 1:
    return -np.log(yhat)

y = 0
yhat = 0.05

print(f'Loss: {binary_crossentropy(y, yhat)}')
```

```
Loss: 0.05129329438755058
```

```
class Perceptron:
    def __init__(self, x, y):

        self.input = np.array(x, dtype=float) #assigning input
        self.label = np.array(y, dtype=float) #reading label data
        self.weights = np.random.rand(x.shape[1], y.shape[1]) #randomly initialising the weig
        self.z = self.input@self.weights
        self.yhat = self.sigmoid(self.z)  #applying activation function


    def sigmoid(self, x):
        return 1.0/(1.0+np.exp(-x))

    def sigmoid_deriv(self, x):
        s = sigmoid(x)
        return s(1-s)
```

```
    def forward_prop(self): #defining the forward propagation layer
        self.yhat = self.sigmoid(self.input @ self.weights) #@ represents matrix multiplicati
        return self.yhat

    def back_prop(self): #defining the backward propagation layer
        gradient = self.input.T @ (-2.0*(self.label - self.yhat)*self.sigmoid(self.yhat))  #s

        self.weights -= gradient #to find the loss
```

```
simple_nn = Perceptron(X_train, y_train)
training_iterations = 1000 #assigning the no. of iteration

history = [] #storing how mean squared error changes after each iteration in this array

def mse(yhat, y): #defining the error term i.e., Mean square error
    sum = 0.0
    for pred, label in zip(yhat, y):
        sum += (pred-label)**2
    return sum/len(yhat)

for i in range(training_iterations): #training model
    simple_nn.forward_prop()
    simple_nn.back_prop()
    yhat = simple_nn.forward_prop()
    history.append(mse(yhat, simple_nn.label))



yhat = simple_nn.forward_prop()




print(f'Final Mean Squared Error: {mse(yhat, simple_nn.label)}') #printing the error occurred
```

```
 Final Mean Squared Error: [0.0123614]
```

```
#plotting the graph using 'plt.plot'
#plot gives the change in the error term with respect to no. of iteration
plt.plot(history)
plt.ylabel('Mean Squared Error')
plt.xlabel('Training Iteration')
```

```
Text(0.5, 0, 'Training Iteration')
```