CS6005 DEEP LEARNING TECHNIQUES

ASSIGNMENT - 1

IMPLEMENTATION OF ARTIFICIAL NEURAL NETWORK IN PYTHON

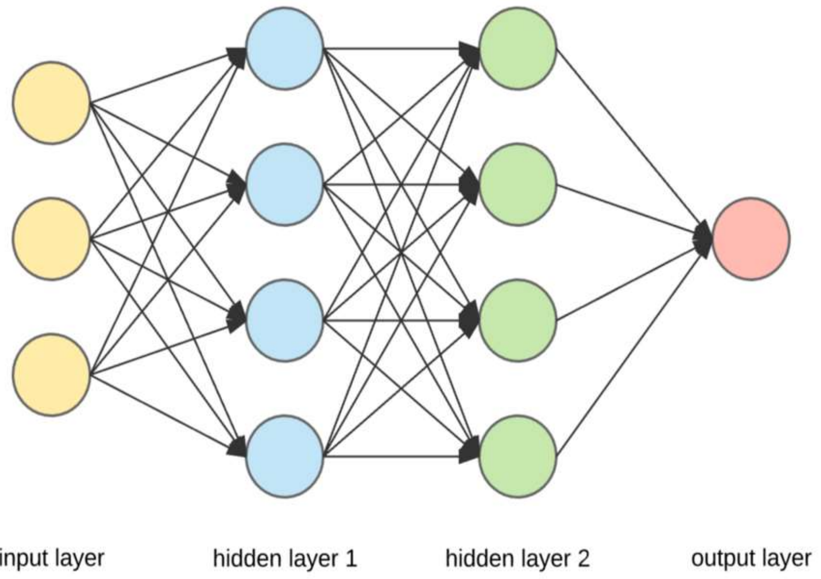
DONE BY

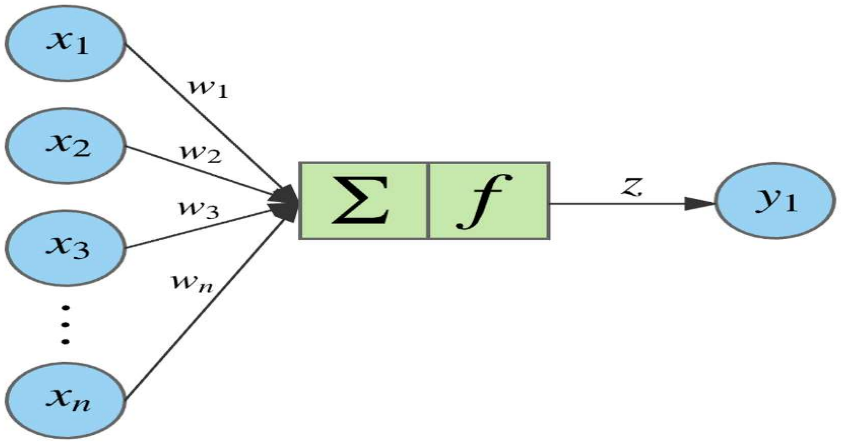
SABHARI P

2018103582

ARTIFICIAL NEURAL NETWORK:

Artificial Neural Networks (ANN) are multi-layer fully-connected neural networks. They consist of an input layer, multiple hidden layers, and an output layer. Every node in one layer is connected to every other node in the next layer. We make the network deeper by increasing the number of hidden layers.





Input Layer:

As the name suggests, it accepts inputs in several different formats provided by the programmer.

Hidden Layer:

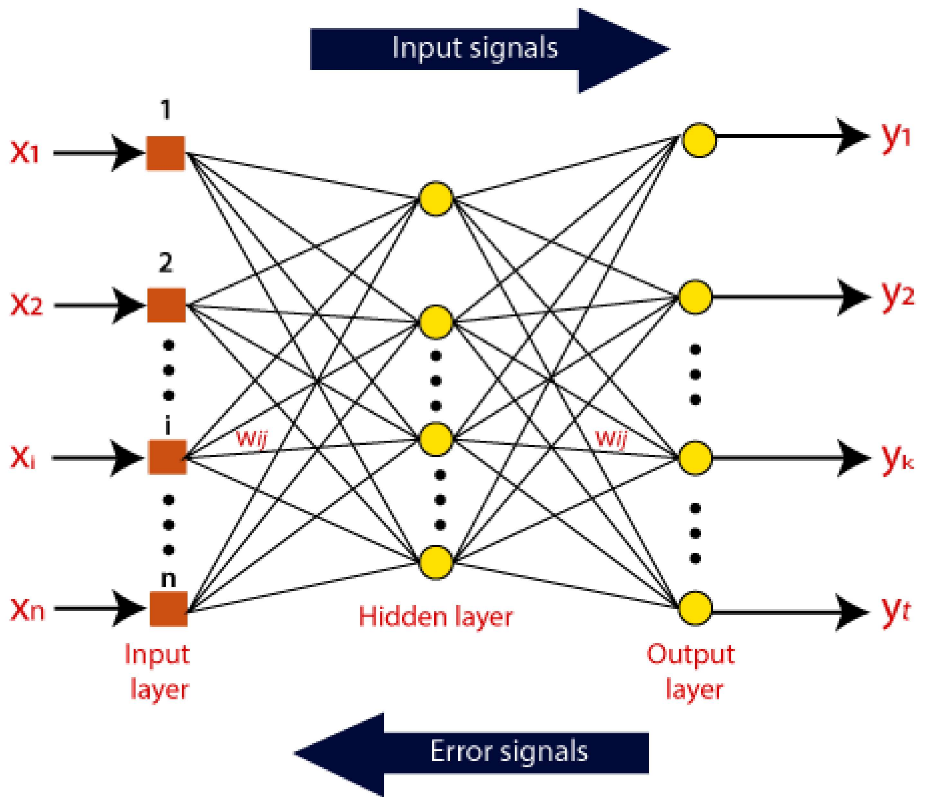
The hidden layer presents in-between input and output layers. It performs all the calculations to find hidden features and patterns.

Output Layer:

The input goes through a series of transformations using the hidden layer, which finally results in output that is conveyed using this layer.

WORKING OF ANN:

Artificial Neural Network can be best represented as a weighted directed graph, where the artificial neurons form the nodes. The association between the neurons outputs and neuron inputs can be viewed as the directed edges with weights. The Artificial Neural Network receives the input signal from the external source in the form of a pattern and image in the form of a vector. These inputs are then mathematically assigned by the notations x(n) for every n number of inputs.



Afterward, each of the input is multiplied by its corresponding weights (these weights are the details utilized by the artificial neural networks to solve a specific problem). In general terms, these weights normally represent the strength of the interconnection between neurons inside the artificial neural network. All the weighted inputs are summarized inside the computing unit.

If the weighted sum is equal to zero, then bias is added to make the output non-zero or something else to scale up to the system's response. Bias has the same input, and weight equals to 1. Here the total of weighted inputs can be in the range of 0 to positive infinity. Here, to keep the response in the limits of the desired value, a certain maximum value is benchmarked, and the total of weighted inputs is passed through the activation function.

The activation function refers to the set of transfer functions used to achieve the desired output. There is a different kind of the activation function, but primarily either linear or non-linear sets of functions. Some of the commonly used sets of activation functions are the Binary, linear, and Tan hyperbolic sigmoidal activation functions.

IMPLEMENTATION OF ANN:

Dataset information:

For our implementation we had used the **breast cancer data** available in UCI machine learning repository.

Attribute Information:

1) ID number 2) Diagnosis (M = malignant, B = benign) 3-32)

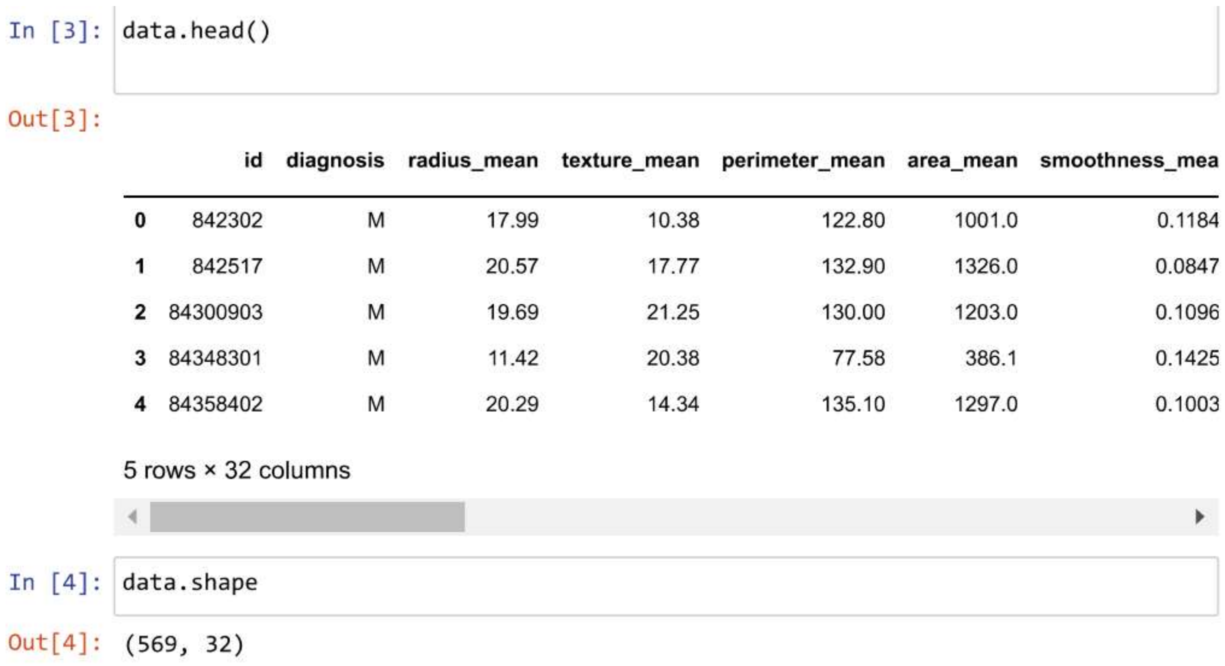
Ten real-valued features are computed for each cell nucleus:a) radius (mean of distances from center to points on the perimeter) b) texture (standard deviation of gray-scale values) c) perimeter d) area e) smoothness (local variation in radius lengths) f) compactness (perimeter^2 / area - 1.0) g) concavity (severity of concave portions of the contour) h) concave points (number of concave portions of the contour) i) symmetry j) fractal dimension ("coastline approximation" - 1)All feature values are recoded with four significant digits.

Missing attribute values: none

Class distribution: 357 benign, 212 malignant

IMPLEMENTATION:

1. At first we import the needed libraries.
2. We load the data set available in our local device. And lets view its dimensions.



```
In [3]: data.head()

Out[3]:
```

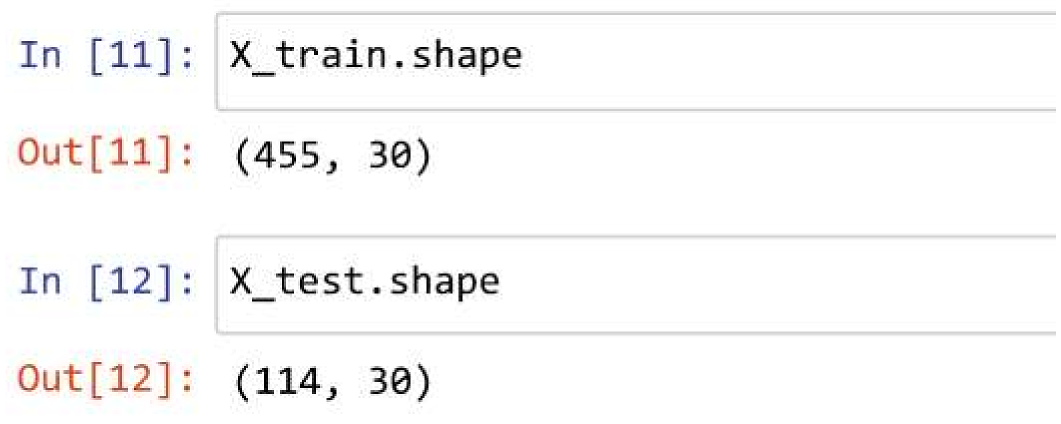
	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mea
0	842302	M	17.99	10.38	122.80	1001.0	0.1184
1	842517	M	20.57	17.77	132.90	1326.0	0.0847
2	84300903	M	19.69	21.25	130.00	1203.0	0.1096
3	84348301	M	11.42	20.38	77.58	386.1	0.1425
4	84358402	M	20.29	14.34	135.10	1297.0	0.1003

5 rows × 32 columns

```
In [4]: data.shape
Out[4]: (569, 32)
```

It has 569 information of a person with each containing 32 attributes.

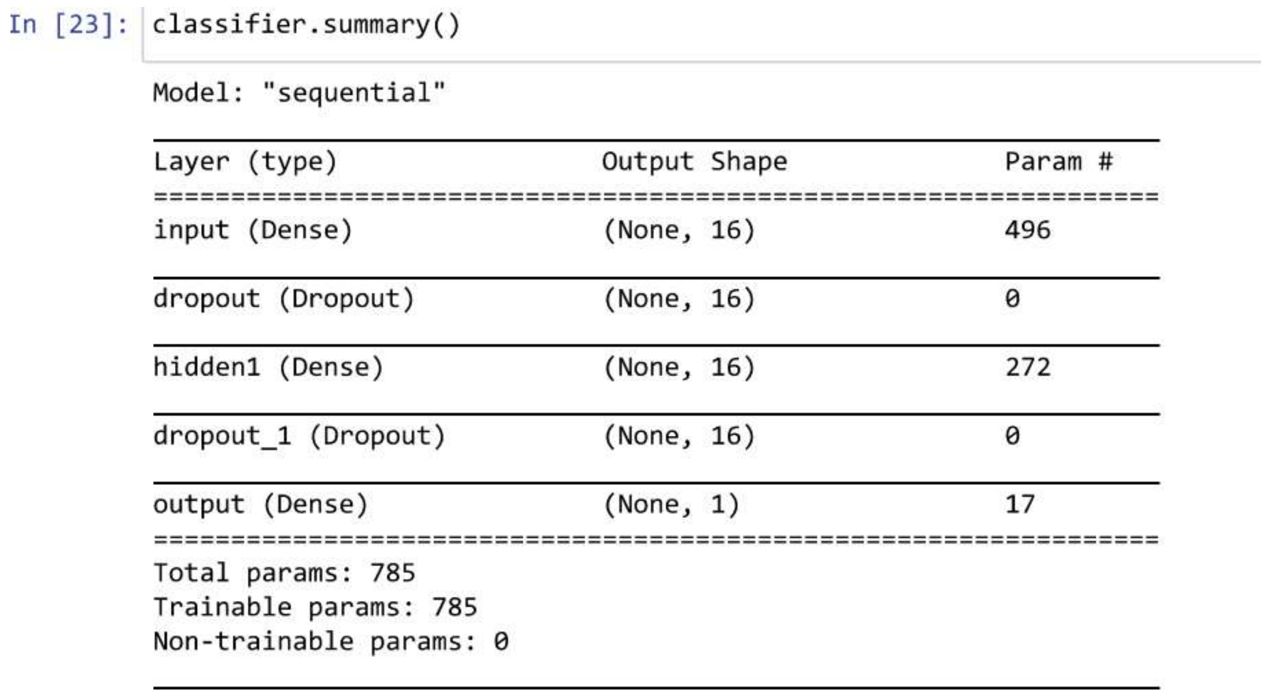
3. We seperate the data and the label seperately
4. Label consists of two classes. Thus we encode it as 0 or 1, which will be used for binary classification.
5. Then we allocate 80% data for training and the remaining 20% for testing. Lets check their dimensions.



```
In [11]: X_train.shape

Out[11]: (455, 30)

In [12]: X_test.shape

Out[12]: (114, 30)
```

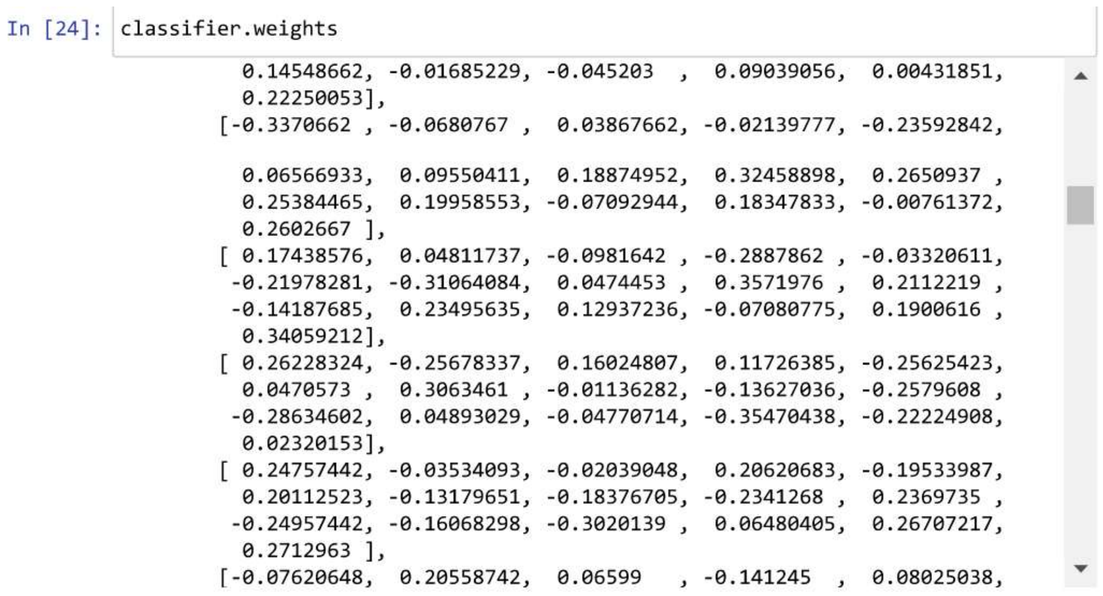
6. We perform preprocessing on data.
7. We choose sequential model.
8. We add the input layer, hidden layer and the output layer. Here we add a single input,hidden and output layer. Here we used the relu activation function for input and hidden layer and sigmoid for output as it is a binary classification. We name each layer. We also use bias at each layer. We also use the dropout regularization with rate 0.1 in order to prevent overfit and underfit.input_dim - number of columns of the dataset. output_dim - number of outputs to be fed to the next layer, if any.activation - activation function which is ReLU in this case.The **ReLU** function is f(x)=max(0,x).Sigmoid function is used when dealing with

classification problems with 2 types of results. Thus our model summary would look like,



```
In [23]: classifier.summary()

Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input (Dense)                (None, 16)                496
_________________________________________________________________
dropout (Dropout)            (None, 16)                0
_________________________________________________________________
hidden1 (Dense)              (None, 16)                272
_________________________________________________________________
dropout_1 (Dropout)          (None, 16)                0
_________________________________________________________________
output (Dense)               (None, 1)                 17
=================================================================
Total params: 785
Trainable params: 785
Non-trainable params: 0
_________________________________________________________________
```

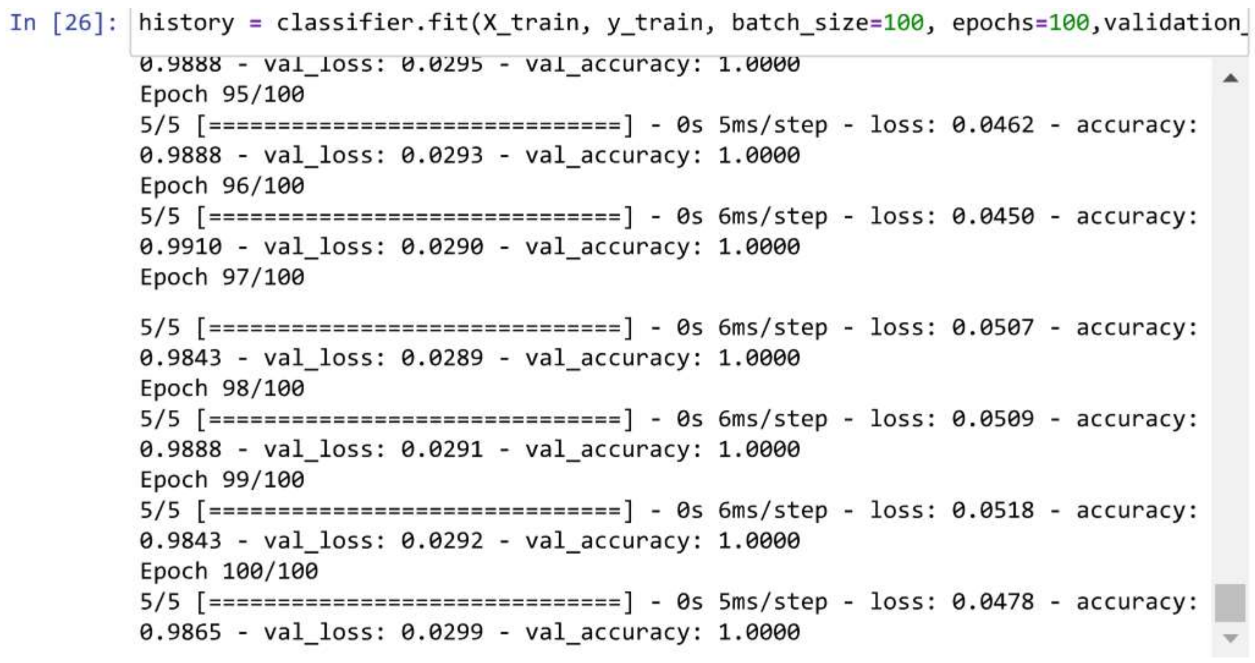
9. We check the initial weights for model.



```
In [24]: classifier.weights
          0.14548662, -0.01685229, -0.045203  ,  0.09039056,  0.00431851,
          0.22250053],
        [-0.3370662 , -0.0680767 ,  0.03867662, -0.02139777, -0.23592842,

          0.06566933,  0.09550411,  0.18874952,  0.32458898,  0.2650937 ,
          0.25384465,  0.19958553, -0.07092944,  0.18347833, -0.00761372,
          0.2602667 ],
        [ 0.17438576,  0.04811737, -0.0981642 , -0.2887862 , -0.03320611,
         -0.21978281, -0.31064084,  0.0474453 ,  0.3571976 ,  0.2112219 ,
         -0.14187685,  0.23495635,  0.12937236, -0.07080775,  0.1900616 ,
          0.34059212],
        [ 0.26228324, -0.25678337,  0.16024807,  0.11726385, -0.25625423,
          0.0470573 ,  0.3063461 , -0.01136282, -0.13627036, -0.2579608 ,
         -0.28634602,  0.04893029, -0.04770714, -0.35470438, -0.22224908,
          0.02320153],
        [ 0.24757442, -0.03534093, -0.02039048,  0.20620683, -0.19533987,
          0.20112523, -0.13179651, -0.18376705, -0.2341268 ,  0.2369735 ,
         -0.24957442, -0.16068298, -0.3020139 ,  0.06480405,  0.26707217,
          0.2712963 ],
        [-0.07620648,  0.20558742,  0.06599   , -0.141245  ,  0.08025038,
```

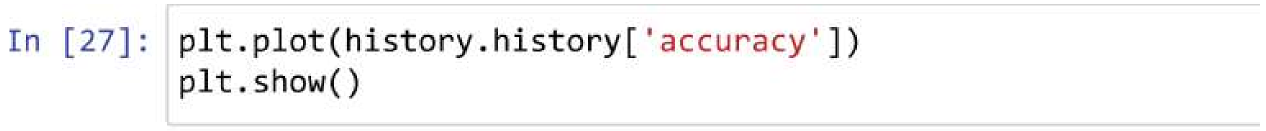
10. Then we compile the model with adam optimizer and binary cross entropy cross function. Cross-entropy loss, or log loss, measures the performance of a classification model whose output is a probability value between 0 and 1. Cross-entropy loss increases as the predicted probability diverges from the actual label. So predicting a probability of .012 when the actual observation label is 1 would be bad and result in a high loss value. A perfect model would have a log loss of 0.

11. Then we train the model along with 20% validation set for 100 epochs with batch size as 100 to observe the training accuracy,

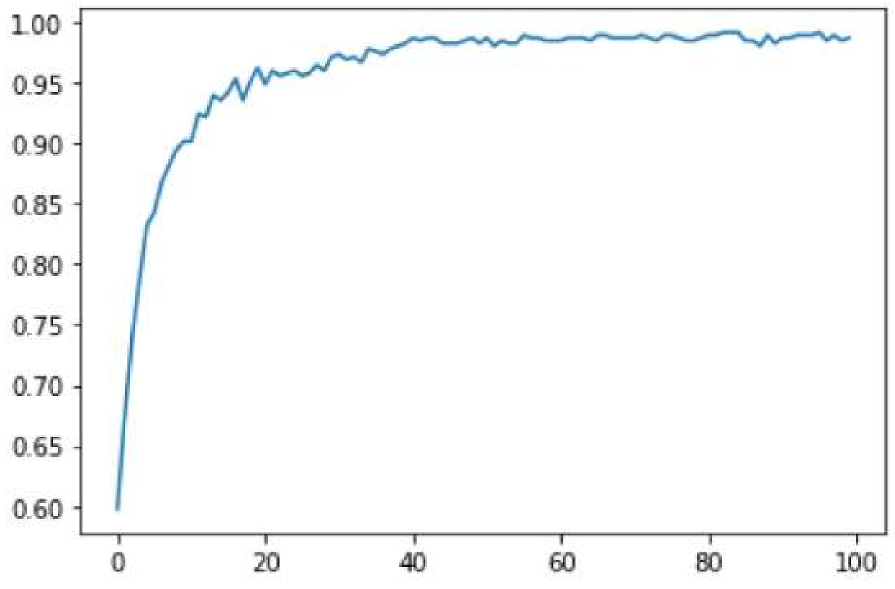


```
In [26]: history = classifier.fit(X_train, y_train, batch_size=100, epochs=100,validation_
         0.9888 - val_loss: 0.0295 - val_accuracy: 1.0000
         Epoch 95/100
         5/5 [==============================] - 0s 5ms/step - loss: 0.0462 - accuracy:
         0.9888 - val_loss: 0.0293 - val_accuracy: 1.0000
         Epoch 96/100
         5/5 [==============================] - 0s 6ms/step - loss: 0.0450 - accuracy:
         0.9910 - val_loss: 0.0290 - val_accuracy: 1.0000
         Epoch 97/100

         5/5 [==============================] - 0s 6ms/step - loss: 0.0507 - accuracy:
         0.9843 - val_loss: 0.0289 - val_accuracy: 1.0000
         Epoch 98/100
         5/5 [==============================] - 0s 6ms/step - loss: 0.0509 - accuracy:
         0.9888 - val_loss: 0.0291 - val_accuracy: 1.0000
         Epoch 99/100
         5/5 [==============================] - 0s 6ms/step - loss: 0.0518 - accuracy:
         0.9843 - val_loss: 0.0292 - val_accuracy: 1.0000
         Epoch 100/100
         5/5 [==============================] - 0s 5ms/step - loss: 0.0478 - accuracy:
         0.9865 - val_loss: 0.0299 - val_accuracy: 1.0000
```

12. We plot a graph to observe how the performance varies with no of epochs,

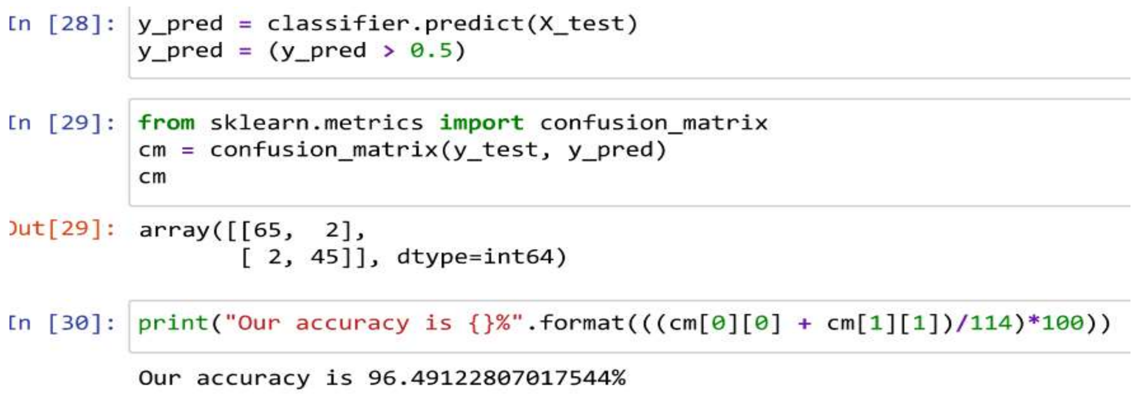


```
In [27]: plt.plot(history.history['accuracy'])
         plt.show()
```



We could observe as the epochs increases, the accuracy increases and after sometime it remains close to previous epoch.

13. Then we test our trained model with our testing data



```
In [28]: y_pred = classifier.predict(X_test)
         y_pred = (y_pred > 0.5)

In [29]: from sklearn.metrics import confusion_matrix
         cm = confusion_matrix(y_test, y_pred)
         cm

Out[29]: array([[65,  2],
                [ 2, 45]], dtype=int64)

In [30]: print("Our accuracy is {}%".format(((cm[0][0] + cm[1][1])/114)*100))

         Our accuracy is 96.49122807017544%
```

We have succesfully implented our sequential model with

TRAINING ACCURACY : 98.65%

TESTING ACCURACY   : 96.49%

BELOW IS THE IMPLEMENTATION ON THE JUPYTER NOTEBOOK.

DATASET IS ALSO ATTACHED WITH THIS DOCUMENT FOR REFERENCE.