PERFORMANCE STUDY:

Let us try to change some nature of our network and observe the performance.

1. Number of hidden layers
2. Number of epochs
3. Different activation function.

CHANGING NUMBER OF HIDDEN LAYERS:

Adding extra 2 hidden layers:

We just add 2 extra hidden layers with same configuration,

```
In [31]: classifier = models.Sequential()

In [32]: classifier.add(layers.Dense(16, activation='relu' ,input_dim=30,use_bias=True,name = 'input'))
         classifier.add(layers.Dropout(rate=0.1))

In [33]: classifier.add(layers.Dense(16, activation='relu',use_bias=True,name = 'hidden1'))
         classifier.add(layers.Dropout(rate=0.1))

In [35]: classifier.add(layers.Dense(16, activation='relu',use_bias=True,name = 'hidden2'))
         classifier.add(layers.Dropout(rate=0.1))

In [36]: classifier.add(layers.Dense(16, activation='relu',use_bias=True,name = 'hidden3'))
         classifier.add(layers.Dropout(rate=0.1))

In [37]: classifier.add(layers.Dense(1,  activation='sigmoid',use_bias=True,name = 'output'))
```

We name it as the hidden2 and hidden3 and the summary would be,

```
In [38]: classifier.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input (Dense)                (None, 16)                496
_________________________________________________________________
dropout_2 (Dropout)          (None, 16)                0
_________________________________________________________________
hidden1 (Dense)              (None, 16)                272
_________________________________________________________________
dropout_3 (Dropout)          (None, 16)                0
_________________________________________________________________
hidden2 (Dense)              (None, 16)                272
_________________________________________________________________
dropout_4 (Dropout)          (None, 16)                0
_________________________________________________________________
hidden3 (Dense)              (None, 16)                272
_________________________________________________________________
dropout_5 (Dropout)          (None, 16)                0
_________________________________________________________________
output (Dense)               (None, 1)                 17
=================================================================
Total params: 1,329
Trainable params: 1,329
Non-trainable params: 0
_________________________________________________________________
```

We now run our new model for same test and train data for same no of epochs and same activation function to observe training accuracy,

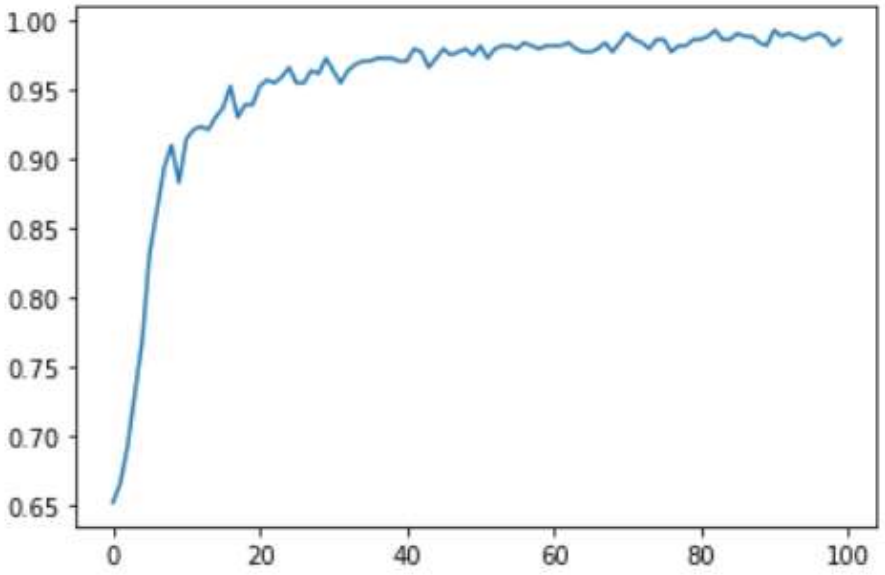
```
In [41]: history = classifier.fit(X_train, y_train, batch_size=100, epochs=100,validation_split = 0.02)
```

```
0
Epoch 95/100
5/5 [==============================] - 0s 7ms/step - loss: 0.0368 - accuracy: 0.9865 - val_loss: 0.0206 - val_accuracy: 1.000
0
Epoch 96/100
5/5 [==============================] - 0s 7ms/step - loss: 0.0416 - accuracy: 0.9888 - val_loss: 0.0208 - val_accuracy: 1.000
0
Epoch 97/100
5/5 [==============================] - 0s 6ms/step - loss: 0.0271 - accuracy: 0.9910 - val_loss: 0.0207 - val_accuracy: 1.000
0
Epoch 98/100
5/5 [==============================] - 0s 7ms/step - loss: 0.0356 - accuracy: 0.9888 - val_loss: 0.0208 - val_accuracy: 1.000
0
Epoch 99/100
5/5 [==============================] - 0s 7ms/step - loss: 0.0488 - accuracy: 0.9820 - val_loss: 0.0218 - val_accuracy: 1.000
0
Epoch 100/100
5/5 [==============================] - 0s 7ms/step - loss: 0.0606 - accuracy: 0.9865 - val_loss: 0.0225 - val_accuracy: 1.000
0
```

The training accuracy goes to 98.65%

Lets observe how the accuracy changes as 2 new layers are added,

```
In [42]: plt.plot(history.history['accuracy'])
         plt.show()
```



It varies same as before. Now lets test our data to see whether there is a performance change,

```
In [43]: y_pred = classifier.predict(X_test)
         y_pred = (y_pred > 0.5)
```

```
In [44]: from sklearn.metrics import confusion_matrix
         cm = confusion_matrix(y_test, y_pred)
         cm
```

```
Out[44]: array([[63,  4],
                [ 1, 46]], dtype=int64)
```

```
In [45]: print("Our accuracy is {}%".format(((cm[0][0] + cm[1][1])/114)*100))
```

```
Our accuracy is 95.6140350877193%
```

Our test accuracy is 95.61%

Adding extra 4 hidden layers:

We just add 4 extra hidden layers with same configuration,

```
In [46]: import keras
         from keras.models import Sequential
         from keras.layers import Dense, Dropout
         from keras import models
         from keras import layers

In [47]: classifier = models.Sequential()

In [48]: classifier.add(layers.Dense(16, activation='relu' ,input_dim=30,use_bias=True,name = 'input'))
         classifier.add(layers.Dropout(rate=0.1))

In [49]: classifier.add(layers.Dense(16, activation='relu',use_bias=True,name = 'hidden1'))
         classifier.add(layers.Dropout(rate=0.1))

In [50]: classifier.add(layers.Dense(16, activation='relu',use_bias=True,name = 'hidden2'))
         classifier.add(layers.Dropout(rate=0.1))

In [51]: classifier.add(layers.Dense(16, activation='relu',use_bias=True,name = 'hidden3'))
         classifier.add(layers.Dropout(rate=0.1))

In [52]: classifier.add(layers.Dense(16, activation='relu',use_bias=True,name = 'hidden4'))
         classifier.add(layers.Dropout(rate=0.1))

In [53]: classifier.add(layers.Dense(16, activation='relu',use_bias=True,name = 'hidden5'))
         classifier.add(layers.Dropout(rate=0.1))

In [54]: classifier.add(layers.Dense(1,  activation='sigmoid',use_bias=True,name = 'output'))
```

We name it as the hidden2, hidden3, hidden4 and hidden5. and the summary would be,

```
In [55]: classifier.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input (Dense)                (None, 16)                496
_________________________________________________________________
dropout_6 (Dropout)          (None, 16)                0
_________________________________________________________________
hidden1 (Dense)              (None, 16)                272
_________________________________________________________________
dropout_7 (Dropout)          (None, 16)                0
_________________________________________________________________
hidden2 (Dense)              (None, 16)                272
_________________________________________________________________
dropout_8 (Dropout)          (None, 16)                0
_________________________________________________________________
hidden3 (Dense)              (None, 16)                272
_________________________________________________________________
dropout_9 (Dropout)          (None, 16)                0
_________________________________________________________________
hidden4 (Dense)              (None, 16)                272
_________________________________________________________________
dropout_10 (Dropout)         (None, 16)                0
_________________________________________________________________
hidden5 (Dense)              (None, 16)                272
_________________________________________________________________
dropout_11 (Dropout)         (None, 16)                0
_________________________________________________________________
output (Dense)               (None, 1)                 17
=================================================================
Total params: 1,873
Trainable params: 1,873
Non-trainable params: 0
```

We now run our new model for same test and train data for same no of epochs and same activation function to observe training accuracy,

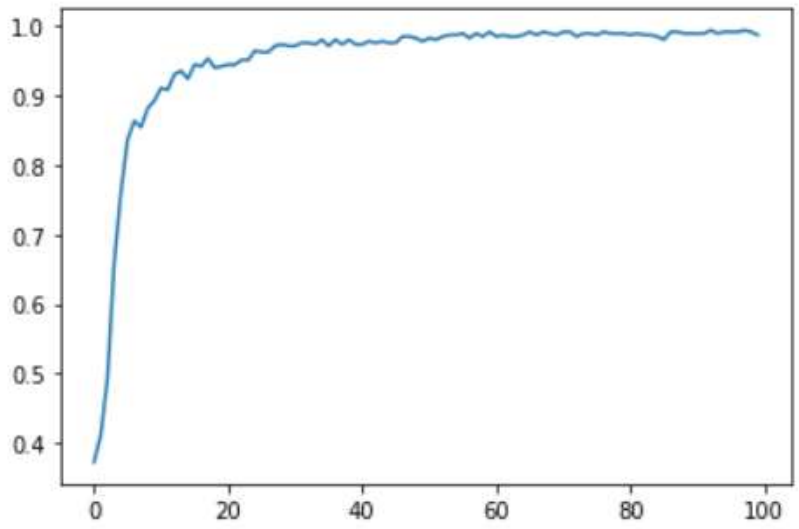
```
In [58]: history = classifier.fit(X_train, y_train, batch_size=100, epochs=100,validation_split = 0.02)
```

```
5/5 [==============================] - 0s 8ms/step - loss: 0.0402 - accuracy: 0.9888 - val_loss: 0.0699 - val_accuracy: 1.000
0
Epoch 95/100
5/5 [==============================] - 0s 7ms/step - loss: 0.0331 - accuracy: 0.9910 - val_loss: 0.0611 - val_accuracy: 1.000
0
Epoch 96/100
5/5 [==============================] - 0s 8ms/step - loss: 0.0386 - accuracy: 0.9910 - val_loss: 0.0593 - val_accuracy: 1.000
0
Epoch 97/100
5/5 [==============================] - 0s 8ms/step - loss: 0.0451 - accuracy: 0.9910 - val_loss: 0.0714 - val_accuracy: 1.000
0
Epoch 98/100
5/5 [==============================] - 0s 7ms/step - loss: 0.0319 - accuracy: 0.9933 - val_loss: 0.0843 - val_accuracy: 0.900
0
Epoch 99/100
5/5 [==============================] - 0s 7ms/step - loss: 0.0355 - accuracy: 0.9910 - val_loss: 0.0967 - val_accuracy: 0.900
0
Epoch 100/100
5/5 [==============================] - 0s 7ms/step - loss: 0.0407 - accuracy: 0.9865 - val_loss: 0.1164 - val_accuracy: 0.900
0
```

The training accuracy goes to 98.65%

Lets observe how the accuracy changes as 4 new layers are added,

```
In [59]: plt.plot(history.history['accuracy'])
         plt.show()
```



It varies same as before. Now lets test our data to see whether there is a performance change,

```
In [60]: y_pred = classifier.predict(X_test)
         y_pred = (y_pred > 0.5)

In [61]: from sklearn.metrics import confusion_matrix
         cm = confusion_matrix(y_test, y_pred)
         cm

Out[61]: array([[65,  2],
                [ 3, 44]], dtype=int64)

In [62]: print("Our accuracy is {}%".format(((cm[0][0] + cm[1][1])/114)*100))

         Our accuracy is 95.6140350877193%
```

Our test accuracy is 95.61%

	1 HIDDEN LAYER	3 HIDDEN LAYERS	5 HIDDEN LAYERS
TRAINING ACCURACY	98.65	98.65	98.65
TESTING ACCURACY	96.49	95.61	95.61

THUS THE INCREASE IN THE DEPTH OF NETWORK DOES NOT AFFECT THE PERFORMANCE TO LARGER EXTENT. ONLY LITTLE VARIATION IS OBSERVED FOR THIS DATA.

NUMBER OF EPOCHS:

We now increase the no of epochs to 300 to observe whether there is a change in the performance.

We work with the same network as before,

In [63]:
```
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras import models
from keras import layers
```

In [64]:
```
classifier = models.Sequential()
```

In [65]:
```
classifier.add(layers.Dense(16, activation='relu' ,input_dim=30,use_bias=True,name = 'input'))
classifier.add(layers.Dropout(rate=0.1))
```

In [66]:
```
classifier.add(layers.Dense(16, activation='relu',use_bias=True,name = 'hidden1'))
classifier.add(layers.Dropout(rate=0.1))
```

In [67]:
```
classifier.add(layers.Dense(1,  activation='sigmoid',use_bias=True,name = 'output'))
```

```
In [68]: classifier.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input (Dense)                (None, 16)                496
_________________________________________________________________
dropout_12 (Dropout)         (None, 16)                0
_________________________________________________________________
hidden1 (Dense)              (None, 16)                272
_________________________________________________________________
dropout_13 (Dropout)         (None, 16)                0
_________________________________________________________________
output (Dense)               (None, 1)                 17
=================================================================
Total params: 785
Trainable params: 785
Non-trainable params: 0
_________________________________________________________________
```

Same model as the single hidden layer network. Now we change the no of epochs to observe the result.

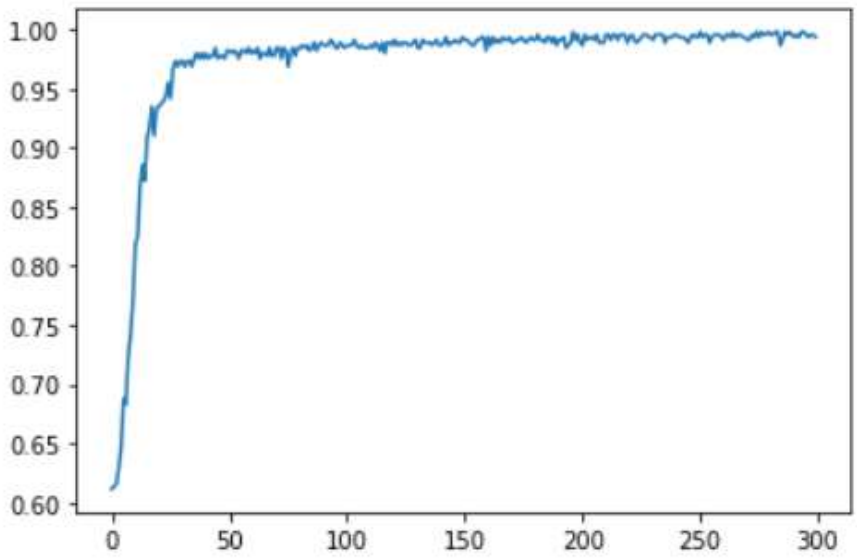
```
In [71]: history = classifier.fit(X_train, y_train, batch_size=100, epochs=300,validation_split = 0.02)
```

```
00
Epoch 295/300
5/5 [==============================] - 0s 12ms/step - loss: 0.0158 - accuracy: 0.9978 - val_loss: 0.0112 - val_accuracy: 1.00
00
Epoch 296/300
5/5 [==============================] - 0s 12ms/step - loss: 0.0168 - accuracy: 0.9955 - val_loss: 0.0112 - val_accuracy: 1.00
00
Epoch 297/300
5/5 [==============================] - 0s 13ms/step - loss: 0.0176 - accuracy: 0.9933 - val_loss: 0.0111 - val_accuracy: 1.00
00
Epoch 298/300
5/5 [==============================] - 0s 12ms/step - loss: 0.0152 - accuracy: 0.9955 - val_loss: 0.0107 - val_accuracy: 1.00
00
Epoch 299/300
5/5 [==============================] - 0s 12ms/step - loss: 0.0157 - accuracy: 0.9955 - val_loss: 0.0107 - val_accuracy: 1.00
00
Epoch 300/300
5/5 [==============================] - 0s 12ms/step - loss: 0.0160 - accuracy: 0.9933 - val_loss: 0.0106 - val_accuracy: 1.00
00
```

Our accuracy got increased to 99.33%

Lets observe how the accuracy is varying with increase in the no of epochs. We should remember we had used the dropout generalisation inorder to overcome overfitting.

```
In [72]: plt.plot(history.history['accuracy'])
         plt.show()
```



Varies as same as before. Now lets evaluate our testing set.

```
In [73]: y_pred = classifier.predict(X_test)
         y_pred = (y_pred > 0.5)
```

```
In [74]: from sklearn.metrics import confusion_matrix
         cm = confusion_matrix(y_test, y_pred)
         cm
```

```
Out[74]: array([[65,  2],
                [ 2, 45]], dtype=int64)
```

```
In [75]: print("Our accuracy is {}%".format(((cm[0][0] + cm[1][1])/114)*100))
```

```
Our accuracy is 96.49122807017544%
```

Our accuracy is 96.49%

	100 EPOCHS	300 EPOCHS
TRAINING ACCURACY	98.65	99.33
TESTING ACCURACY	96.49	96.49

WITH INCREASE IN NUMBER OF EPOCHS THE TRAINING ACCURACY INCREASES SLIGHTLY WHILE THE TESTING ACCURACY REMAINS SAME FOR THIS DATA.

DIFFERENT ACTIVATION FUNCTION:

We will use the same above network with different activation function. In previous we had used **relu** activation function, but now we will use the **tanh** activation function,

```
In [76]: import keras
         from keras.models import Sequential
         from keras.layers import Dense, Dropout
         from keras import models
         from keras import layers

In [77]: classifier = models.Sequential()

In [78]: classifier.add(layers.Dense(16, activation='tanh' ,input_dim=30,use_bias=True,name = 'input'))
         classifier.add(layers.Dropout(rate=0.1))

In [79]: classifier.add(layers.Dense(16, activation='tanh',use_bias=True,name = 'hidden1'))
         classifier.add(layers.Dropout(rate=0.1))

In [80]: classifier.add(layers.Dense(1,  activation='sigmoid',use_bias=True,name = 'output'))
```

The output activation function remains the same as it is a binary classification.

```
In [81]: classifier.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input (Dense)                (None, 16)                496
_________________________________________________________________
dropout_14 (Dropout)         (None, 16)                0
_________________________________________________________________
hidden1 (Dense)              (None, 16)                272
_________________________________________________________________
dropout_15 (Dropout)         (None, 16)                0
_________________________________________________________________
output (Dense)               (None, 1)                 17
=================================================================
Total params: 785
Trainable params: 785
Non-trainable params: 0
_________________________________________________________________
```

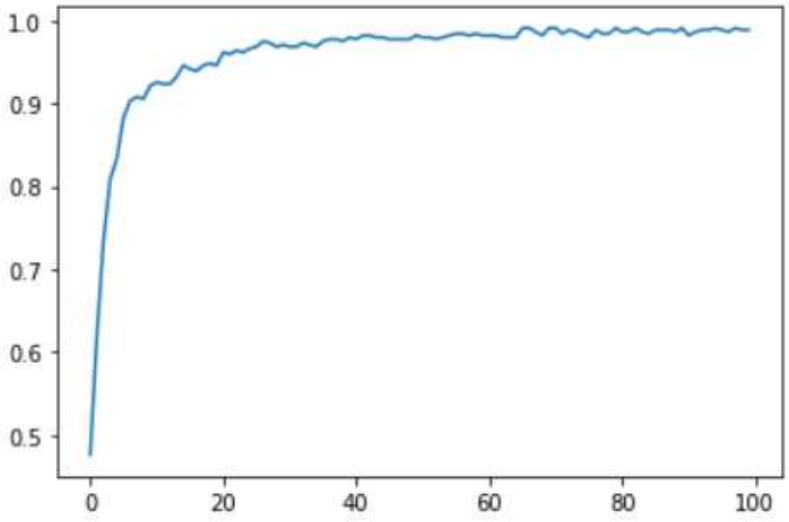
Now we train our data to observe the training accuracy,

```
In [84]: history = classifier.fit(X_train, y_train, batch_size=100, epochs=100,validation_split = 0.02)
```

```
00
Epoch 95/100
5/5 [==============================] - 0s 18ms/step - loss: 0.0489 - accuracy: 0.9910 - val_loss: 0.0471 - val_accuracy: 1.00
00
Epoch 96/100
5/5 [==============================] - 0s 19ms/step - loss: 0.0527 - accuracy: 0.9888 - val_loss: 0.0470 - val_accuracy: 1.00
00
Epoch 97/100
5/5 [==============================] - 0s 20ms/step - loss: 0.0546 - accuracy: 0.9865 - val_loss: 0.0471 - val_accuracy: 1.00
00
Epoch 98/100
5/5 [==============================] - 0s 18ms/step - loss: 0.0550 - accuracy: 0.9910 - val_loss: 0.0458 - val_accuracy: 1.00
00
Epoch 99/100
5/5 [==============================] - 0s 19ms/step - loss: 0.0543 - accuracy: 0.9888 - val_loss: 0.0458 - val_accuracy: 1.00
00
Epoch 100/100
5/5 [==============================] - 0s 20ms/step - loss: 0.0542 - accuracy: 0.9888 - val_loss: 0.0452 - val_accuracy: 1.00
00
```

The training accuracy is 98.88%. Lets observe how the accuracy increases with epochs,

```
In [85]: plt.plot(history.history['accuracy'])
         plt.show()
```



It varies the same as before. Now lets test the model with our testing set,

```
In [86]: y_pred = classifier.predict(X_test)
         y_pred = (y_pred > 0.5)

In [87]: from sklearn.metrics import confusion_matrix
         cm = confusion_matrix(y_test, y_pred)
         cm

Out[87]: array([[65,  2],
                [ 3, 44]], dtype=int64)

In [88]: print("Our accuracy is {}%".format(((cm[0][0] + cm[1][1])/114)*100))

         Our accuracy is 95.6140350877193%
```

The accuracy is 95.61%

	RELU	TANH
TRAINING ACCURACY	98.65	98.88
TESTING ACCURACY	96.49	95.61

WITH VARYING ACTIVATION FUNCTION IT PERFORMS NEARLY SAME FOR THIS DATA.

THUS THE FACTORS AFFECTING PERFORMANCE ARE VIEWED EXPRIMENTALLY.

BELOW IS THE IMPLEMENTATION OF ANN FOR A LARGE DATA SET.