## Q) 5) a):

The hyperparameters for my final neural network are:

**No of Hidden Layers = 1, No of hidden nodes = 10, Learning Rate = 0.1, no of iterations/epochs = 1000, solver = 'adam', activation function = 'relu'**

To make the algorithm simple, I used rectangular weight matrix i.e. the number of nodes in all hidden layers is same

I used the following grid for the Grid Search method to select the best hyperparameters:

1. hidden = [1,2,5]
2. nodes = [2,5,10]
3. lr =[0.1,1]
4. af =['tanh', 'relu']
5. solver =['sgd','adam']

I tried every combination of the above (72 combinations) and calculated the cross-validation error. Then, I picked the hyperparameters which gave the least cross validation error.

I applied the best model on the test dataset and got **77** percent accuracy

**While doing the search, I got a few convergence warnings which say that the no of iterations = 1000 is not enough for the model to converge. This may be due to the 0.1 learning rate**

## Q) 5) b):

The concentration bound obtained at 95% confidence interval using Hoeffding bound is **0.096**

## Q) 5) c):

I am getting different results (accuracy) every time I run my neural network because of

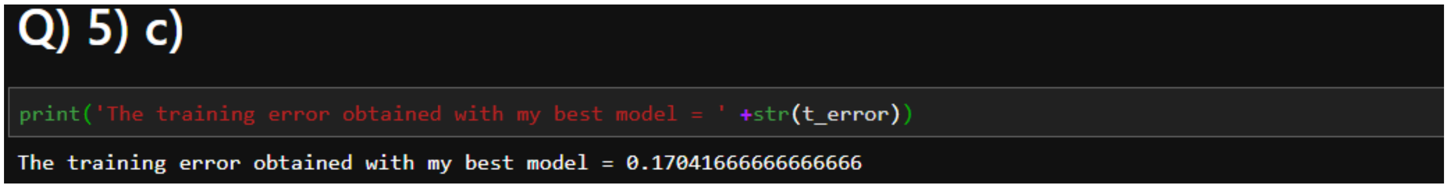
1) Random initialization of weights
2) Randomness in the training data (I set shuffle = True in the MLPClassifier)
3) Due to the adaptive Learning Rate (I set Learning Rate = 'adaptive' in the MLP Classifier)
4) Multiple Local Minima (in case of Non-Convex cost functions)

## Q) 5) d):

We can say whether a model is overfitting or not by checking the Training error.

I saved the training error of all the models I built using the cross_validate function of the Sklearn.model_selection. To get the training error, I set the return_train_score parameter of cross_validate to True.

For the model with the least validation error (the model that I picked as the best) has training error of **0.17.**



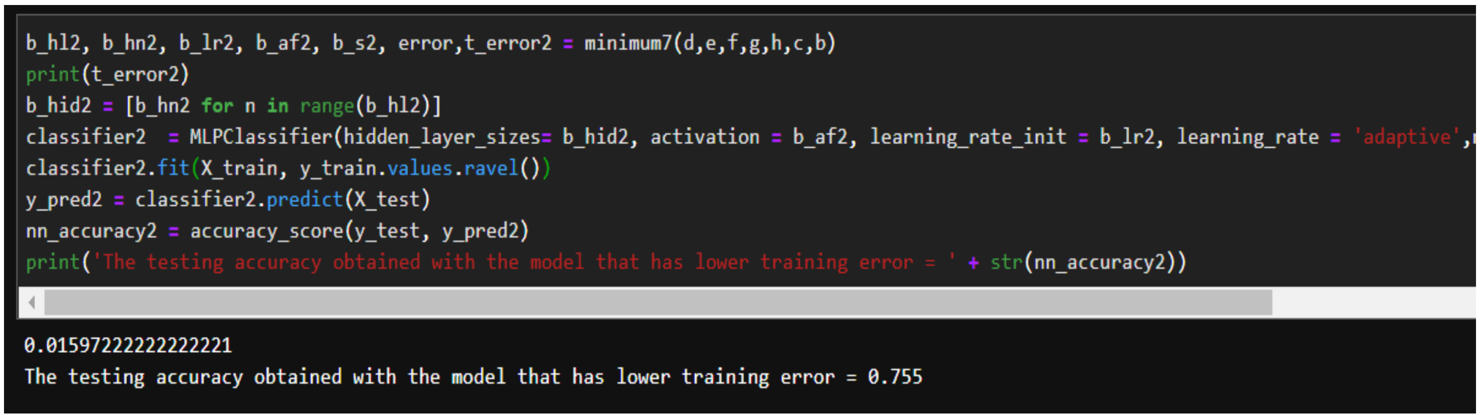
Q) 5) c)

```
print('The training error obtained with my best model = ' +str(t_error))
```

```
The training error obtained with my best model = 0.17041666666666666
```

Since, the training error is not zero, I can say that my model is not overfitting.

I checked the testing accuracy of the model that has low training error:



```
b_hl2, b_hn2, b_lr2, b_af2, b_s2, error,t_error2 = minimum7(d,e,f,g,h,c,b)
print(t_error2)
b_hid2 = [b_hn2 for n in range(b_hl2)]
classifier2  = MLPClassifier(hidden_layer_sizes= b_hid2, activation = b_af2, learning_rate_init = b_lr2, learning_rate = 'adaptive',
classifier2.fit(X_train, y_train.values.ravel())
y_pred2 = classifier2.predict(X_test)
nn_accuracy2 = accuracy_score(y_test, y_pred2)
print('The testing accuracy obtained with the model that has lower training error = ' + str(nn_accuracy2))
```

```
0.015972222222222221
The testing accuracy obtained with the model that has lower training error = 0.755
```

I got lesser testing accuracy of 0.755 with the model that has low training error of 0.016.

**Note: I used random_state =100 in the MLP classifier in the code to check the testing accuracies of my best model and the model that has low training error of 0.016**