Classification of Fuel type of the car (Petrol or Diesel) using 4 layer Backpropagation

In [29]:

```
import pandas as pd
import numpy as np
import math
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import accuracy_score
import time
from sklearn.metrics import mean_squared_error
import statistics
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
```

In [ ]:

```
```

In [30]:

```
data = pd.read_csv('ToyotaCorolla.csv')#--------------Loading the dataset
```

In [31]:

```

labelencoder = LabelEncoder()#-------------------------------------------using labelencode
data['FuelType'] = labelencoder.fit_transform(data['FuelType'])

```

In [32]:

```
y = data.FuelType#----------------label set
X = data#------------------------without label data
```

In [33]:

```
X= X.drop(columns="FuelType")
```

In [34]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.7)
print(y_train.shape)
print(y_test.shape)
```

```
(430,)
(1006,)
```

In [35]:

```
X_train[1:10]
```

Out[35]:

	Price	Age	KM	CC	Weight
1222	8250	79	84966	1600	1070
378	6500	53	216000	1900	1110
1182	9900	80	92255	1600	1105
227	11690	34	65345	1400	1060
199	11950	39	98823	1600	1119
799	8250	65	74179	1600	1050
1244	6950	70	81663	1600	1050
282	12850	39	45713	1400	1085
502	9900	53	57475	1600	1040

In [36]:

```
y_train[1:10]
#print(y_train.nunique)
```

Out[36]:

```
1222    1
378     2
1182    1
227     1
199     0
799     1
1244    1
282     1
502     1
Name: FuelType, dtype: int32
```

In [37]:

```
X_train=pd.DataFrame(X_train).to_numpy()#--------------converting pandas dataframe to numpy
X_test=pd.DataFrame(X_test).to_numpy()
y_train=pd.DataFrame(y_train).to_numpy()
y_test=pd.DataFrame(y_test).to_numpy()
```

In [38]:

```
def sigmoid(s, deriv=False):
  if(deriv == True):
    return s * (1 - s)
  return 1/(1  + np.exp(-s))
```

In [ ]:

```

```

In [39]:

```
Input_layer = 5
Output_layer = 1
Hidden_layer_1 = 100
Hidden_layer_2 = 100
Hidden_layer_3 = 100
Hidden_layer_4 = 100
learning_rate = 0.01
Weight_1 = np.random.normal(size=[Input_layer, Hidden_layer_1])
Weight_2 = np.random.normal(size=[Hidden_layer_1, Hidden_layer_2])
Weight_3 = np.random.normal(size=[Hidden_layer_2, Hidden_layer_3])
Weight_4 = np.random.normal(size=[Hidden_layer_3, Hidden_layer_4])
Weight_5 = np.random.normal(size=[Hidden_layer_4, Output_layer])
```

In [ ]:

```

```

Main Neural network class starts here

In [40]:

```
import numpy as np
from random import seed

learning_rate = 0.0001
class Four_layer_NeuralNetwork(object):#--------------------------------------Main Neura
    def __init__(self):
        #parameters
        self.Input_layer = 5
        self.Output_layer = 1
        self.Hidden_layer_1 = 10#---------------------------------------------------------s
        self.Hidden_layer_2 = 20
        self.Hidden_layer_3 = 30
        self.Hidden_layer_4 = 15
        learning_rate = 0.0001#-------------------------------------------------------------

        #weights
        self.Weight_1 = np.random.normal(size=[self.Input_layer, self.Hidden_layer_1])
        self.Weight_2 = np.random.normal(size=[self.Hidden_layer_1, self.Hidden_layer_2]) #
        self.Weight_3 = np.random.normal(size=[self.Hidden_layer_2, self.Hidden_layer_3])
        self.Weight_4 = np.random.normal(size=[self.Hidden_layer_3, self.Hidden_layer_4])
        self.Weight_5 = np.random.normal(size=[self.Hidden_layer_4, self.Output_layer])

    def feedForward(self, X_train):#-----------------------------------Creating function
        #forward propogation through the network
        self.Forward = np.dot(X_train, self.Weight_1)
        self.Forward2 = self.sigmoid(self.Forward)

        self.Forward3 = np.dot(self.Forward2, self.Weight_2) #-----------------------------
        self.Forward4 = self.sigmoid(self.Forward3) #------------------Activation function

        self.Forward5 = np.dot(self.Forward4, self.Weight_3)
        self.Forward6 = self.sigmoid(self.Forward5)

        self.Forward7 = np.dot(self.Forward6, self.Weight_4)
        self.Forward8 = self.sigmoid(self.Forward7)

        self.Forward9 = np.dot(self.Forward8, self.Weight_5)
        output = self.sigmoid(self.Forward9)
        return output

    def sigmoid(self, s, deriv=False):#-------------------------------------Defining Sigma
        if (deriv == True):
            return s * (1 - s)
        return 1/(1 + np.exp(-s))

    def Backpropagate(self, X_train, y_train, output):#---------------------------------------
        #Backpropagate propogate through the network
        self.output_error = y_train - output
        self.output_delta = self.output_error * self.sigmoid(output, deriv=True)#----------

        self.Forward8_error = self.output_delta.dot(self.Weight_5.T)
        self.Forward8_delta = self.Forward8_error * self.sigmoid(self.Forward8, deriv=True)

        self.Forward6_error = self.Forward8_delta.dot(self.Weight_4.T)
        self.Forward6_delta = self.Forward6_error * self.sigmoid(self.Forward6, deriv=True)

        self.Forward4_error = self.Forward6_delta.dot(self.Weight_3.T)
        self.Forward4_delta = self.Forward4_error * self.sigmoid(self.Forward4, deriv=True)
```

```
        self.Forward2_error = self.Forward4_delta.dot(self.Weight_2.T)
        self.Forward2_delta = self.Forward2_error * self.sigmoid(self.Forward2, deriv=True)




#--------------------------Adjusting weight below


        self.Weight_1 += learning_rate*X_train.T.dot(self.Forward2_delta) #----------------
        self.Weight_2 += learning_rate*self.Forward2.T.dot(self.Forward4_delta)
        self.Weight_3 += learning_rate*self.Forward4.T.dot(self.Forward6_delta)
        self.Weight_4 += learning_rate*self.Forward6.T.dot(self.Forward8_delta)
        self.Weight_5 += learning_rate*self.Forward8.T.dot(self.output_delta)

    def train(self, X_train, y_train):
        output = self.feedForward(X_train)
        self.Backpropagate(X_train, y_train, output)
```

Final Training Accuracy

In [53]:

```
NN = Four_layer_NeuralNetwork()

for i in range(30): #-----------------------------NUmber of epochs for neural network and

    if i%1==0:
      print("Training Accuracy: epoch", + i+1, str(accuracy_score(y_train.round(),NN.feedFo
    NN.train(X_train, y_train)


print("Accuracy: " + str(accuracy_score(y_train.round(),NN.feedForward(X_train).round())*10
```

```
Training Accuracy: epoch 1 0.9302325581395349
Training Accuracy: epoch 2 0.9302325581395349
Training Accuracy: epoch 3 0.9302325581395349
Training Accuracy: epoch 4 0.9302325581395349
Training Accuracy: epoch 5 1.8604651162790697
Training Accuracy: epoch 6 1.8604651162790697
Training Accuracy: epoch 7 1.8604651162790697
Training Accuracy: epoch 8 77.90697674418605
Training Accuracy: epoch 9 81.16279069767441
Training Accuracy: epoch 10 86.27906976744187
Training Accuracy: epoch 11 86.27906976744187
Training Accuracy: epoch 12 86.27906976744187
Training Accuracy: epoch 13 87.44186046511628
Training Accuracy: epoch 14 87.44186046511628
Training Accuracy: epoch 15 87.44186046511628
Training Accuracy: epoch 16 87.44186046511628
Training Accuracy: epoch 17 87.44186046511628
Training Accuracy: epoch 18 87.44186046511628
Training Accuracy: epoch 19 87.44186046511628
Training Accuracy: epoch 20 87.44186046511628
Training Accuracy: epoch 21 87.44186046511628
Training Accuracy: epoch 22 87.44186046511628
Training Accuracy: epoch 23 87.44186046511628
Training Accuracy: epoch 24 87.44186046511628
Training Accuracy: epoch 25 87.44186046511628
Training Accuracy: epoch 26 87.44186046511628
Training Accuracy: epoch 27 87.44186046511628
Training Accuracy: epoch 28 87.44186046511628
Training Accuracy: epoch 29 87.44186046511628
Training Accuracy: epoch 30 87.44186046511628
Accuracy: 87.44186046511628

<ipython-input-40-534bfa9dd00c>:44: RuntimeWarning: overflow encountered in
exp
  return 1/(1 + np.exp(-s))
```

In [ ]:

```

```

Final Testing Accuracy

In [57]:

```
#---testing accuracy

NN = Four_layer_NeuralNetwork()


print("Testing Accuracy: " + str(accuracy_score(y_test.round(),NN.feedForward(X_test).round
```

```
Testing Accuracy: 79.72166998011929

<ipython-input-40-534bfa9dd00c>:44: RuntimeWarning: overflow encountered in
exp
  return 1/(1 + np.exp(-s))
```

Type *Markdown* and LaTeX: α^2

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: