Table of Contents :

In [1]:
```
import keras
```

```
Using TensorFlow backend.
```

In [2]:
```
import pandas as pd
import numpy as np
```

In [3]:
```
from sklearn.model_selection import train_test_split
```

In [41]:
```
file = 'concrete_data.csv'
```

Part A - Build the baseline

In [42]:
```
concrete_data = pd.read_csv(file)
concrete_data.head()
```

Out[42]:

	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age	Strength
0	540.0	0.0	0.0	162.0	2.5	1040.0	676.0	28	79.99
1	540.0	0.0	0.0	162.0	2.5	1055.0	676.0	28	61.89
2	332.5	142.5	0.0	228.0	0.0	932.0	594.0	270	40.27
3	332.5	142.5	0.0	228.0	0.0	932.0	594.0	365	41.05
4	198.6	132.4	0.0	192.0	0.0	978.4	825.5	360	44.30

In [6]:
```
concrete_data.shape
```

Out[6]:
```
(1030, 9)
```

In [7]:
```
concrete_data.describe()
```

Out[7]:

	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Aggı
count	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.0
mean	281.167864	73.895825	54.188350	181.567282	6.204660	972.918932	773.5
std	104.506364	86.279342	63.997004	21.354219	5.973841	77.753954	80.1
min	102.000000	0.000000	0.000000	121.800000	0.000000	801.000000	594.0
25%	192.375000	0.000000	0.000000	164.900000	0.000000	932.000000	730.9
50%	272.900000	22.000000	0.000000	185.000000	6.400000	968.000000	779.5
75%	350.000000	142.950000	118.300000	192.000000	10.200000	1029.400000	824.0
max	540.000000	359.400000	200.100000	247.000000	32.200000	1145.000000	992.6

In [8]:
```
concrete_data.isnull().sum()
```

Out[8]:
```
Cement                0
Blast Furnace Slag    0
Fly Ash               0
Water                 0
Superplasticizer      0
Coarse Aggregate      0
Fine Aggregate        0
Age                   0
Strength              0
dtype: int64
```

Splitting the data

In [9]:
```
concrete_data_columns = concrete_data.columns

X = concrete_data[concrete_data_columns[concrete_data_columns != 'Streng
th']] # all columns except Strength
y = concrete_data['Strength'] # Strength column
n_cols=X.shape[1]
```

In [10]:
```
#Split the data into training dataset and testing dataset with 30% test
 dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

Building the model

In [11]:
```
from keras.models import Sequential
from keras.layers import Dense
```

In [12]:
```
# One hidden layer with 10 nodes and relu function
# adam optimizer and mean_squared_error as loss function
def regression_model():
    # create model
    model = Sequential()
    model.add(Dense(10, activation='relu', input_shape=(n_cols,)))
    model.add(Dense(1))

    # compile model
    model.compile(optimizer='adam', loss='mean_squared_error')
    return model
```

Training

In [18]:
```
# build the model
model = regression_model()
```

In [19]:
```
# fitting data to the model with 50 epoch
model.fit(X_train, y_train, epochs=50)
```

```
Epoch 1/50
721/721 [==============================] - 0s 249us/step - loss: 63630.
6960
Epoch 2/50
721/721 [==============================] - 0s 60us/step - loss: 14181.1
870
Epoch 3/50
721/721 [==============================] - 0s 53us/step - loss: 4618.68
05
Epoch 4/50
721/721 [==============================] - 0s 46us/step - loss: 4128.49
29
Epoch 5/50
721/721 [==============================] - 0s 51us/step - loss: 3852.22
32
Epoch 6/50
721/721 [==============================] - 0s 57us/step - loss: 3603.91
48
Epoch 7/50
721/721 [==============================] - 0s 53us/step - loss: 3376.93
53
Epoch 8/50
721/721 [==============================] - 0s 49us/step - loss: 3141.30
85
Epoch 9/50
721/721 [==============================] - 0s 68us/step - loss: 2931.36
04
Epoch 10/50
721/721 [==============================] - 0s 64us/step - loss: 2728.23
32
Epoch 11/50
721/721 [==============================] - 0s 60us/step - loss: 2545.43
43
Epoch 12/50
721/721 [==============================] - 0s 68us/step - loss: 2366.19
39
Epoch 13/50
721/721 [==============================] - 0s 46us/step - loss: 2198.94
79
Epoch 14/50
721/721 [==============================] - 0s 67us/step - loss: 2051.12
86
Epoch 15/50
721/721 [==============================] - 0s 64us/step - loss: 1918.20
48
Epoch 16/50
721/721 [==============================] - 0s 54us/step - loss: 1798.00
97
Epoch 17/50
721/721 [==============================] - 0s 55us/step - loss: 1664.71
88
Epoch 18/50
721/721 [==============================] - 0s 62us/step - loss: 1560.35
49
Epoch 19/50
721/721 [==============================] - 0s 50us/step - loss: 1459.95
78
```

```
Epoch 20/50
721/721 [==============================] - 0s 47us/step - loss: 1369.87
47
Epoch 21/50
721/721 [==============================] - 0s 53us/step - loss: 1292.13
50
Epoch 22/50
721/721 [==============================] - 0s 55us/step - loss: 1206.78
28
Epoch 23/50
721/721 [==============================] - 0s 53us/step - loss: 1136.36
56
Epoch 24/50
721/721 [==============================] - 0s 50us/step - loss: 1078.03
16
Epoch 25/50
721/721 [==============================] - 0s 49us/step - loss: 1009.70
66
Epoch 26/50
721/721 [==============================] - 0s 46us/step - loss: 950.216
4
Epoch 27/50
721/721 [==============================] - 0s 57us/step - loss: 902.265
0
Epoch 28/50
721/721 [==============================] - 0s 37us/step - loss: 846.454
5
Epoch 29/50
721/721 [==============================] - 0s 40us/step - loss: 800.056
6
Epoch 30/50
721/721 [==============================] - 0s 46us/step - loss: 755.732
0
Epoch 31/50
721/721 [==============================] - 0s 40us/step - loss: 713.881
9
Epoch 32/50
721/721 [==============================] - 0s 42us/step - loss: 675.440
0
Epoch 33/50
721/721 [==============================] - 0s 44us/step - loss: 638.058
1
Epoch 34/50
721/721 [==============================] - 0s 49us/step - loss: 603.847
2
Epoch 35/50
721/721 [==============================] - 0s 49us/step - loss: 571.376
4
Epoch 36/50
721/721 [==============================] - 0s 47us/step - loss: 538.516
8
Epoch 37/50
721/721 [==============================] - 0s 53us/step - loss: 510.680
6
Epoch 38/50
721/721 [==============================] - 0s 51us/step - loss: 485.416
0
```

```
Epoch 39/50
721/721 [==============================] - 0s 47us/step - loss: 458.670
7
Epoch 40/50
721/721 [==============================] - 0s 49us/step - loss: 431.618
5
Epoch 41/50
721/721 [==============================] - 0s 42us/step - loss: 410.852
9
Epoch 42/50
721/721 [==============================] - 0s 40us/step - loss: 389.491
4
Epoch 43/50
721/721 [==============================] - 0s 43us/step - loss: 368.903
3
Epoch 44/50
721/721 [==============================] - 0s 51us/step - loss: 350.552
5
Epoch 45/50
721/721 [==============================] - 0s 44us/step - loss: 331.984
2
Epoch 46/50
721/721 [==============================] - 0s 44us/step - loss: 315.445
0
Epoch 47/50
721/721 [==============================] - 0s 43us/step - loss: 299.845
6
Epoch 48/50
721/721 [==============================] - 0s 40us/step - loss: 285.796
9
Epoch 49/50
721/721 [==============================] - 0s 42us/step - loss: 271.733
5
Epoch 50/50
721/721 [==============================] - 0s 39us/step - loss: 261.959
5
```

Out[19]:
```
<keras.callbacks.callbacks.History at 0x11f46bb6148>
```

In [23]:
```python
# evaluate the model
evaluated_score = model.evaluate(X_test, y_test, verbose=1)
```

```
309/309 [==============================] - 0s 26us/step
```

In [24]:
```python
y_predict=model.predict(X_test)
```

In [25]:
```python
from sklearn.metrics import mean_squared_error
```

In [26]:
```python
squared_error_score = mean_squared_error(y_test, y_predict)
```

Repeating 50 times

In [27]:
```
# Repeat 50 times
error_score=[]
for i in range(50):
    model.fit(X_train, y_train, epochs=50, verbose=0)
    y_predict=model.predict(X_test)
    error_score.append(mean_squared_error(y_test,y_predict))
```

In [33]:
```
Mean = np.mean(error_score)
```

In [34]:
```
Std = np.std(error_score)
```

In [40]:
```
print('The mean of mean_squared_error is : {:.3f}\nThe standard deviatio
n of mean_squared_error is : {:.3f}'.format(Mean, Std))
```

```
The mean of mean_squared_error is : 54.829
The standard deviation of mean_squared_error is : 8.249
```

Part B - Using normalized data

In [44]:
```
#Normalization part by mean and standard deviation
X_nor = (X-np.mean(X))/np.std(X)
y_nor = (y-np.mean(y))/np.std(y)
```

In [45]:
```
X_nor.head()
```

Out[45]:

	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age
0	2.477915	-0.856888	-0.847144	-0.916764	-0.620448	0.863154	-1.217670	-0.279733
1	2.477915	-0.856888	-0.847144	-0.916764	-0.620448	1.056164	-1.217670	-0.279733
2	0.491425	0.795526	-0.847144	2.175461	-1.039143	-0.526517	-2.240917	3.553066
3	0.491425	0.795526	-0.847144	2.175461	-1.039143	-0.526517	-2.240917	5.057677
4	-0.790459	0.678408	-0.847144	0.488793	-1.039143	0.070527	0.647884	4.978487

In [46]:
```
y_nor.head()
```

Out[46]:
```
0    2.645408
1    1.561421
2    0.266627
3    0.313340
4    0.507979
Name: Strength, dtype: float64
```

Splitting the data

```
In [47]: #Split the data into training dataset and testing dataset with 30% test
          dataset
         X_train_nor, X_test_nor, y_train_nor, y_test_nor = train_test_split(X_no
         r, y_nor, test_size=0.3)
```

```
In [48]: #Train and evaluate the model for 50 times.
         error_score_nor=[]
         for i in range(50):
             model.fit(X_train_nor, y_train_nor, epochs=50, verbose=0)
             y_predict_nor=model.predict(X_test_nor)
             error_score_nor.append(mean_squared_error(y_test_nor,y_predict_nor))
```

```
In [49]: Mean_nor = np.mean(error_score_nor)
         Std_nor = np.std(error_score_nor)
```

```
In [50]: print('The mean of mean_squared_error is : {:.3f}, while in normalized d
         ata is {:.3f}\nThe standard deviation of mean_squared_error is : {:.3f},
         while in normalized data is {:.3f}'.format(Mean, Mean_nor, Std, Std_nor
         ))
```

```
The mean of mean_squared_error is : 54.829, while in normalized data is
0.137
The standard deviation of mean_squared_error is : 8.249, while in norma
lized data is 0.021
```

How does the mean squared of error compared to Part A ?
As we can see from the results, the difference of number is huge in part A, the range of value is more than 1 while in part B as a consequence of normalization, the loss will have value between 0 and 1. For the practical reason, the result from part B is more convenient.

Part C - Using 100 epochs

```
In [51]: #Train and evaluate the model for 50 times using 100 epochs.
         error_score_nor_100=[]
         for i in range(50):
             model.fit(X_train_nor, y_train_nor, epochs=100, verbose=0)
             y_predict_nor=model.predict(X_test_nor)
             error_score_nor_100.append(mean_squared_error(y_test_nor,y_predict_n
         or))
```

```
In [52]: Mean_nor_100 = np.mean(error_score_nor_100)
         Std_nor_100 = np.std(error_score_nor_100)
```

In [54]:
```
print('The mean of mean_squared_error_nor using 50 epochs is : {:.3f}, w
hile using 100 epochs is {:.3f}\nThe standard deviation of mean_squared_
error_nor using 50 epochs is : {:.3f}, while using 100 epochs is {:.3f}'
.format(Mean_nor, Mean_nor_100, Std_nor, Std_nor_100))
```

```
The mean of mean_squared_error_nor using 50 epochs is : 0.137, while us
ing 100 epochs is 0.124
The standard deviation of mean_squared_error_nor using 50 epochs is :
0.021, while using 100 epochs is 0.001
```

How does the mean squared of error compared to Part B ?
By increasing the epochs the accuracy is increased by applying equation $accuracy = 1 - loss$accuracy=1−loss and also the behaviour is more uniform as we can see from the standard deviation of Part C is less than Part B. So, increasing the number of epochs may increased the accuracy of model and more realible.

Part D - Increase the hidden layers

Building the model

In [56]:
```
# Three hidden layers with 10 nodes and relu function
# adam optimizer and mean_squared_error as loss function
def regression_model_modified():
    # create model
    model = Sequential()
    model.add(Dense(10, activation='relu', input_shape=(n_cols,)))
    model.add(Dense(10, activation='relu'))
    model.add(Dense(10, activation='relu'))
    model.add(Dense(1))

    # compile model
    model.compile(optimizer='adam', loss='mean_squared_error')
    return model
```

In [57]:
```
#build the model and fitting the data with normalized dataset and 50 epo
chs
model_modified = regression_model_modified()
```

In [58]:
```
#Train and evaluate the model for 50 times using 100 epochs.
error_score_nor_3layers=[]
for i in range(50):
    model_modified.fit(X_train_nor, y_train_nor, epochs=50, verbose=0)
    y_predict_nor=model_modified.predict(X_test_nor)
    error_score_nor_3layers.append(mean_squared_error(y_test_nor,y_predi
ct_nor))
```

In [59]:
```
Mean_nor_3layers=np.mean(error_score_nor_3layers)
Std_nor_3layers=np.std(error_score_nor_3layers)
```

In [60]:
```
print('The mean of mean_squared_error_nor using 1 hidden layer is : {:.3f}, while using 3 hidden layers is {:.3f}\nThe standard deviation of mean_squared_error_nor using 1 hidden layer is : {:.3f}, while using 3 hidden layers is {:.3f}'.format(Mean_nor, Mean_nor_3layers, Std_nor, Std_nor_3layers))
```

```
The mean of mean_squared_error_nor using 1 hidden layer is : 0.137, while using 3 hidden layers is 0.111
The standard deviation of mean_squared_error_nor using 1 hidden layer is : 0.021, while using 3 hidden layers is 0.006
```

How does the mean squared of error compared to Part B ?
As we can see, the accuracy is increased and it's more uniform than the part B. So increasing hidden layers may increase the accuracy of model and its realibility.