Part D

Increase the number of hidden layers and Repeat part B but use a neural network with the following instead:

- Three hidden layers, each of 10 nodes and ReLU activation function

In [2]:
```
import keras
import numpy as np
import pandas as pd
from sklearn.metrics import mean_squared_error
```

In [3]:
```
# Read the data
concrete_data = pd.read_csv("concrete_data.csv")
concrete_data
```

Out[3]:

	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age	Str
0	540.0	0.0	0.0	162.0	2.5	1040.0	676.0	28	
1	540.0	0.0	0.0	162.0	2.5	1055.0	676.0	28	
2	332.5	142.5	0.0	228.0	0.0	932.0	594.0	270	
3	332.5	142.5	0.0	228.0	0.0	932.0	594.0	365	
4	198.6	132.4	0.0	192.0	0.0	978.4	825.5	360	
...	...	...	...	...	...	...	...	...	
1025	276.4	116.0	90.3	179.6	8.9	870.1	768.3	28	
1026	322.2	0.0	115.6	196.0	10.4	817.9	813.4	28	
1027	148.5	139.4	108.6	192.7	6.1	892.4	780.0	28	
1028	159.1	186.7	0.0	175.6	11.3	989.6	788.9	28	
1029	260.9	100.5	78.3	200.6	8.6	864.5	761.5	28	

1030 rows × 9 columns

In [4]:
```
concrete_data.head()
```

Out[4]:

	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	79.9
1	540.0	0.0	0.0	162.0	2.5	1055.0	676.0	28	61.8
2	332.5	142.5	0.0	228.0	0.0	932.0	594.0	270	40.2
3	332.5	142.5	0.0	228.0	0.0	932.0	594.0	365	41.0
4	198.6	132.4	0.0	192.0	0.0	978.4	825.5	360	44.3

In [5]:
```python
# Size of the data
concrete_data.shape
```

Out[5]: (1030, 9)

In [6]:
```python
concrete_data.describe()
```

Out[6]:

	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coars Aggregat
count	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.00000
mean	281.167864	73.895825	54.188350	181.567282	6.204660	972.91893
std	104.506364	86.279342	63.997004	21.354219	5.973841	77.75395
min	102.000000	0.000000	0.000000	121.800000	0.000000	801.00000
25%	192.375000	0.000000	0.000000	164.900000	0.000000	932.00000
50%	272.900000	22.000000	0.000000	185.000000	6.400000	968.00000
75%	350.000000	142.950000	118.300000	192.000000	10.200000	1029.40000
max	540.000000	359.400000	200.100000	247.000000	32.200000	1145.00000

In [7]:
```python
concrete_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1030 entries, 0 to 1029
Data columns (total 9 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Cement              1030 non-null   float64
 1   Blast Furnace Slag  1030 non-null   float64
 2   Fly Ash             1030 non-null   float64
 3   Water               1030 non-null   float64
 4   Superplasticizer    1030 non-null   float64
 5   Coarse Aggregate    1030 non-null   float64
 6   Fine Aggregate      1030 non-null   float64
 7   Age                 1030 non-null   int64  
 8   Strength            1030 non-null   float64
dtypes: float64(8), int64(1)
memory usage: 72.5 KB
```

In [8]:
```python
# Sum of the null values
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
```

Split data into predictors and target

In [11]:
```
concrete_data_columns = concrete_data.columns

# all columns except Strength
predictors = concrete_data[concrete_data_columns[concrete_data_columns

# Strength column
target = concrete_data['Strength']
```

In [12]:
```
predictors.head()
```

Out[12]:

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

In [13]:
```
target.head()
```

Out[13]:
```
0    79.99
1    61.89
2    40.27
3    41.05
4    44.30
Name: Strength, dtype: float64
```

In [14]:
```
predictors_norm = (predictors - predictors.mean()) / predictors.std()
predictors_norm.head()
```

Out[14]:

	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	
0	2.476712	-0.856472	-0.846733	-0.916319	-0.620147	0.862735	-1.217079	-0
1	2.476712	-0.856472	-0.846733	-0.916319	-0.620147	1.055651	-1.217079	-0
2	0.491187	0.795140	-0.846733	2.174405	-1.038638	-0.526262	-2.239829	3
3	0.491187	0.795140	-0.846733	2.174405	-1.038638	-0.526262	-2.239829	5
4	-0.790075	0.678079	-0.846733	0.488555	-1.038638	0.070492	0.647569	4

In [15]:
```
# number of predictors
n_cols = predictors_norm.shape[1]
n_cols
```

Out[15]: 8

Build a Neural Network

In [16]:
```
from keras.models import Sequential
from keras.layers import Dense
from keras import backend as K
```

In [17]:
```
# define regression model
def regression_model():
    # 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
```

Train and Test the network

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

```
C:\Users\nensi\anaconda3\lib\site-packages\keras\src\layers\core\dens
e.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argumen
t to a layer. When using Sequential models, prefer using an `Input(sha
pe)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwarg
s)
```

In [19]:

```python
# fit the model
model.fit(predictors_norm, target, validation_split=0.3, epochs=50, ver
```

```
Epoch 1/50
23/23 - 2s - 66ms/step - loss: 1670.5474 - val_loss: 1177.4442
Epoch 2/50
23/23 - 0s - 9ms/step - loss: 1631.1650 - val_loss: 1136.2482
Epoch 3/50
23/23 - 0s - 6ms/step - loss: 1573.4230 - val_loss: 1076.8727
Epoch 4/50
23/23 - 0s - 4ms/step - loss: 1487.9711 - val_loss: 995.8566
Epoch 5/50
23/23 - 0s - 4ms/step - loss: 1364.9520 - val_loss: 892.0859
Epoch 6/50
23/23 - 0s - 4ms/step - loss: 1200.9633 - val_loss: 764.5522
Epoch 7/50
23/23 - 0s - 4ms/step - loss: 990.3709 - val_loss: 613.4660
Epoch 8/50
23/23 - 0s - 4ms/step - loss: 747.7770 - val_loss: 467.1034
Epoch 9/50
23/23 - 0s - 4ms/step - loss: 530.7358 - val_loss: 366.6128
Epoch 10/50
23/23 - 0s - 4ms/step - loss: 398.6619 - val_loss: 315.5817
Epoch 11/50
23/23 - 0s - 4ms/step - loss: 334.3805 - val_loss: 288.7828
Epoch 12/50
23/23 - 0s - 4ms/step - loss: 300.2288 - val_loss: 262.7346
Epoch 13/50
23/23 - 0s - 6ms/step - loss: 274.8448 - val_loss: 245.6835
Epoch 14/50
23/23 - 0s - 4ms/step - loss: 255.0349 - val_loss: 229.6618
Epoch 15/50
23/23 - 0s - 5ms/step - loss: 239.9467 - val_loss: 219.9129
Epoch 16/50
23/23 - 0s - 4ms/step - loss: 227.6762 - val_loss: 209.9685
Epoch 17/50
23/23 - 0s - 4ms/step - loss: 218.0858 - val_loss: 206.2567
Epoch 18/50
23/23 - 0s - 4ms/step - loss: 209.8603 - val_loss: 198.4656
Epoch 19/50
23/23 - 0s - 4ms/step - loss: 202.7300 - val_loss: 194.4906
Epoch 20/50
23/23 - 0s - 4ms/step - loss: 196.5676 - val_loss: 189.9229
Epoch 21/50
23/23 - 0s - 5ms/step - loss: 191.7756 - val_loss: 186.1128
Epoch 22/50
23/23 - 0s - 5ms/step - loss: 186.7053 - val_loss: 184.7303
Epoch 23/50
23/23 - 0s - 7ms/step - loss: 182.2587 - val_loss: 182.5363
Epoch 24/50
23/23 - 0s - 6ms/step - loss: 178.1536 - val_loss: 179.0039
Epoch 25/50
23/23 - 0s - 7ms/step - loss: 174.4837 - val_loss: 176.4567
Epoch 26/50
23/23 - 0s - 4ms/step - loss: 171.2354 - val_loss: 175.6042
Epoch 27/50
23/23 - 0s - 4ms/step - loss: 168.6610 - val_loss: 173.5421
Epoch 28/50
23/23 - 0s - 4ms/step - loss: 165.4920 - val_loss: 171.2643
Epoch 29/50
23/23 - 0s - 4ms/step - loss: 163.5468 - val_loss: 170.6562
Epoch 30/50
23/23 - 0s - 5ms/step - loss: 160.9094 - val_loss: 169.2920
Epoch 31/50
```

```
23/23 - 0s - 7ms/step - loss: 158.2054 - val_loss: 166.4939
Epoch 32/50
23/23 - 0s - 6ms/step - loss: 155.9313 - val_loss: 166.6151
Epoch 33/50
23/23 - 0s - 5ms/step - loss: 154.0905 - val_loss: 165.2114
Epoch 34/50
23/23 - 0s - 6ms/step - loss: 151.8899 - val_loss: 163.3546
Epoch 35/50
23/23 - 0s - 6ms/step - loss: 151.1924 - val_loss: 161.2376
Epoch 36/50
23/23 - 0s - 5ms/step - loss: 148.7646 - val_loss: 161.7076
Epoch 37/50
23/23 - 0s - 8ms/step - loss: 147.0741 - val_loss: 160.1434
Epoch 38/50
23/23 - 0s - 7ms/step - loss: 145.4942 - val_loss: 157.6632
Epoch 39/50
23/23 - 0s - 6ms/step - loss: 144.0215 - val_loss: 157.2676
Epoch 40/50
23/23 - 0s - 6ms/step - loss: 142.6382 - val_loss: 155.3161
Epoch 41/50
23/23 - 0s - 6ms/step - loss: 141.1045 - val_loss: 153.0733
Epoch 42/50
23/23 - 0s - 4ms/step - loss: 139.4447 - val_loss: 151.6145
Epoch 43/50
23/23 - 0s - 8ms/step - loss: 138.0310 - val_loss: 149.7555
Epoch 44/50
23/23 - 0s - 6ms/step - loss: 136.6174 - val_loss: 146.3487
Epoch 45/50
23/23 - 0s - 4ms/step - loss: 135.1156 - val_loss: 147.5200
Epoch 46/50
23/23 - 0s - 4ms/step - loss: 133.7151 - val_loss: 148.0081
Epoch 47/50
23/23 - 0s - 8ms/step - loss: 132.2387 - val_loss: 146.4902
Epoch 48/50
23/23 - 0s - 7ms/step - loss: 131.0004 - val_loss: 146.2770
Epoch 49/50
23/23 - 0s - 5ms/step - loss: 129.3696 - val_loss: 145.8269
Epoch 50/50
23/23 - 0s - 5ms/step - loss: 128.1855 - val_loss: 144.6627
```

Out[19]:
```
<keras.src.callbacks.history.History at 0x273107ca9b0>
```

In [20]:

```python
mean = []

for i in range(50):
    def regression_model():

        concrete_data_columns = concrete_data.columns
        # all columns except Strength
        predictors = concrete_data[concrete_data_columns[concrete_data_
        # Strength column
        target = concrete_data['Strength']

        predictors_norm = (predictors - predictors.mean()) / predictors
        predictors_norm.head()

        # number of predictors
        n_cols = predictors_norm.shape[1]

        # 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

    model = regression_model()
    model.fit(predictors_norm, target, validation_split=0.3, epochs=50,

    # Calculate mse
    y_pred = model.predict(predictors_norm)
    mse = mean_squared_error(y_pred, target)
    print("Mean Squared Error is: ",mse)
    mean.append(mse)

print(mean)
```

```
Epoch 1/50

C:\Users\nensi\anaconda3\lib\site-packages\keras\src\layers\core\de
nse.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` ar
gument to a layer. When using Sequential models, prefer using an `I
nput(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwa
rgs)

23/23 - 2s - 76ms/step - loss: 1653.9700 - val_loss: 1191.4674
Epoch 2/50
23/23 - 0s - 4ms/step - loss: 1616.7388 - val_loss: 1161.5823
Epoch 3/50
23/23 - 0s - 4ms/step - loss: 1565.0276 - val_loss: 1119.1724
Epoch 4/50
23/23 - 0s - 5ms/step - loss: 1486.1349 - val_loss: 1055.8942
Epoch 5/50
23/23 - 0s - 6ms/step - loss: 1367.0267 - val_loss: 961.1902
Epoch 6/50
23/23 - 0s - 5ms/step - loss: 1192.1536 - val_loss: 826.3813
```

In [23]:

```
# mean of mse
mean_mse = np.mean(mean)
print("Mean of mean squared error: ",mean_mse)
```

```
Mean of mean squared error:  126.80366797156556
```

In [24]:

```
# Standard deviation of mse
std_mse = np.std(mean)
print("Standard deviation of mean squared error: ",std_mse)
```

```
Standard deviation of mean squared error:  17.238246621544334
```

In [ ]:

```

```