1. Importing Libraries

In [1]:

```
import pandas as pd
import numpy as np
from IPython.display import clear_output

import keras
from keras.models import Sequential
from keras.layers import Dense

from sklearn.model_selection import train_test_split as tts
```

2. Loading Data

In [2]:

```
concrete_data = pd.read_csv('https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/DL0101EN/labs/data/concrete_data.csv')
concrete_data.head()
```

Out[2]:

	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 [3]:

```
concrete_data.shape   #Number of rows and columns
```

Out[3]:

```
(1030, 9)
```

3. Target definition and Feature Selection

- Target: 'Strength'
- Features: All columns but 'Strength' and 'Age'

In [4]:

```
features = [col for col in concrete_data.columns if col not in ['Strength', 'Age']]   # All features but 'Strength' and 'Age'.
n_cols = len(features)

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

In [5]:

```
predictors.head()
```

Out[5]:

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

In [6]:

```
target.head()
```

Out[6]:

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

4. Defining Model Building and Scoring Functions

4.1 Build a Regression Neural Network Model Given the Model Settings

In [7]:

```
# build regression model
def regression_model(n_cols, hidden_layers=1, nodes=[10],
                     activations=['relu'], optimizer='adam', loss='mean_squared_error'
):

    # create model
    model = Sequential()
    model.add(Dense(nodes[0], activation=activations[0], input_shape=(n_cols,))) # Addi
ng first node and activation function

    if len(nodes)>1: # If number of nodes is greater than one, add nodes and activation
functions iteratively
        for i in range(1, len(nodes)):
            model.add(Dense(nodes[i], activation=activations[i]))

    model.add(Dense(1))

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

4.2 Build and Evaluate a Regression Neural Network (RNN) Multiple Times (Using different random train/test splits).

In [8]:

```
# build and score regression NN models multiple times using different random train/test
splits each time

def build_score_RNN(n_times, predictors, target, test_size, # Features and target data,
and train and test split size
                    n_cols, hidden_layers, nodes, activations, optimizer, loss, # Param
eters of "regression_model" function defined in the cell above
                    validation_split, epochs # Parameters for the compile of the model
                   ):

    # Loop to store multiple model evaluation results
    results = []
    for i in range(n_times):

        # clear last result output and print current loop stage
        clear_output(wait=True); print(f'Loop State: {i+1}/{n_times}')

        # random split sample into training and testing datasets (holding 30% for testi
ng)
        x_train, x_test, y_train, y_test = tts(predictors, target, test_size=test_size,
random_state=0)

        # build the model
        model = regression_model(n_cols=n_cols, hidden_layers=hidden_layers,
                                 nodes=nodes, activations=activations,
                                 optimizer=optimizer, loss=loss)
        # fit the model
        model.fit(x_train, y_train, validation_split=validation_split, epochs=epochs, v
erbose=0)

        # compute model evaluation
        mean_squared_error = model.evaluate(x_test, y_test, verbose=0)

        # store result in a list
        results.append(mean_squared_error)

    return results
```

4.3 Report Average and Standard Deviation of Multiple Scores (as a list)

In [9]:

```
def report_scoring(result):

    report = pd.DataFrame([np.mean(result), np.std(result)],
                      index=['Average', 'Standard Deviation'],
                      columns=['Mean Squared Error'])
    report.index.name='50 times'

    return report.T
```

5. EXERCISE SOLUTION

PART A. Build a Baseline Model (50 times)

- One hidden layer of 10 nodes, and a ReLU activation function
- Use the adam optimizer and the mean squared error as the loss function.

Model Building and Evaluation (50 times)

In [13]:

```
# sampling settings  (same settings as PART A.)
test_size = 0.3

# hidden layers settings
hidden_layers = 1
nodes = [10]
activations = ['relu']

# compile settings
optimizer = 'adam'
loss = 'mean_squared_error'
epochs = 50 # NEW NUMBER OF EPOCHS
validation_split = 0.3
```

In [14]:

```
n_times = 50
baseline_result = build_score_RNN(
                                    n_times=n_times, predictors=predictors, target=targ
et, test_size=test_size,
                                    n_cols=n_cols, hidden_layers=hidden_layers, nodes=n
odes, activations=activations, optimizer=optimizer, loss=loss,
                                    validation_split=validation_split, epochs=epochs
                                  )
```

```
Loop State: 50/50
```

Mean and the standard deviation of the mean squared errors:

In [15]:

```
report_scoring(baseline_result)
```

Out[15]:

50 times	Average	Standard Deviation
Mean Squared Error	582.322179	658.693103

PART B. Normalize the data

- Repeat Part A but use a normalized version of the data. Recall that one way to normalize the data is by subtracting the mean from the individual predictors and dividing by the standard deviation.

Normalizing Feature Variables

In [16]:

```
predictors_norm = (predictors - predictors.mean()) / predictors.std()

predictors_norm.head()
```

Out[16]:

	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
1	2.476712	-0.856472	-0.846733	-0.916319	-0.620147	1.055651	-1.217079
2	0.491187	0.795140	-0.846733	2.174405	-1.038638	-0.526262	-2.239829
3	0.491187	0.795140	-0.846733	2.174405	-1.038638	-0.526262	-2.239829
4	-0.790075	0.678079	-0.846733	0.488555	-1.038638	0.070492	0.647569

Model Building and Evaluation (50 times)

Using normalized data but same model settings as the baseline model in part A.

In [17]:

```
# sampling settings           (same settings as baseline model in PART A.)
test_size = 0.3             # (Check next cell comments to see the difference from the
 baseline model)

# hidden layers settings
hidden_layers = 1
nodes = [10]
activations = ['relu']

# compile settings
optimizer = 'adam'
loss = 'mean_squared_error'
epochs = 50
validation_split = 0.3
```

In [18]:

```
n_times = 50
norm_result = build_score_RNN( # "predictors" parameter changed to "predictors_norm" (n
ormalized data)
                               n_times=n_times, predictors=predictors_norm, target=tar
get, test_size=test_size,
                               n_cols=n_cols, hidden_layers=hidden_layers, nodes=nodes
, activations=activations, optimizer=optimizer, loss=loss,
                               validation_split=validation_split, epochs=epochs
                                )
```

```
Loop State: 50/50
```

How does the mean of the mean squared errors compare to that from Step A?

- ...

In [19]:

```
report_scoring(norm_result)
```

Out[19]:

	50 times	Average	Standard Deviation
Mean Squared Error	682.79322		136.464619

PART C. Increase the number of epochs (100 epochs)

- Repeat Part B but use 100 epochs this time for training.

In [20]:

```
# sampling settings
test_size = 0.3

# hidden layers settings
hidden_layers = 1
nodes = [10]
activations = ['relu']

# compile settings
optimizer = 'adam'
loss = 'mean_squared_error'
epochs = 100                          # INCREASED NUMBER OF EPOCHS
validation_split = 0.3
```

In [21]:

```
n_times = 50
epochs_100_result = build_score_RNN( # "predictors" parameter changed to "predictors_norm" (normalized data)
                                    n_times=n_times, predictors=predictors_norm, target=target, test_size=test_size,
                                    n_cols=n_cols, hidden_layers=hidden_layers, nodes=nodes, activations=activations, optimizer=optimizer, loss=loss,
                                    validation_split=validation_split, epochs=epochs
                                   )
```

```
Loop State: 50/50
```

How does the mean of the mean squared errors compare to that from Step B?

- ...

In [22]:

```
report_scoring(epochs_100_result)
```

Out[22]:

	50 times	Average	Standard Deviation
Mean Squared Error	216.388063	28.832575	

PART D. Increase the number of hidden layers

Repeat part B but use a neural network with the following instead:

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

Using three hidden layers of 10 nodes

In [23]:

```
# sampling settings  (same settings as PART A.)
test_size = 0.3

# hidden layers settings
hidden_layers = 3  # NEW NUMBER OF HIDDEN LAYERS
nodes = [10, 10, 10] # 10 NODES IN EACH HIDDEN LAYER
activations = ['relu', 'relu', 'relu'] # "relu" ACTIVATION FUNCTION FOR EACH HIDDEN LAY
ER

# compile settings
optimizer = 'adam'
loss = 'mean_squared_error'
epochs = 50 # SAME NUMBER OF EPOCHS AS PART B. AS REQUESTED.
validation_split = 0.3
```

In [24]:

```
n_times = 50
layers_3_result = build_score_RNN( # "predictors" parameter changed to "predictors_nor
m" (normalized data)
                                   n_times=n_times, predictors=predictors_norm, target=tar
get, test_size=test_size,
                                   n_cols=n_cols, hidden_layers=hidden_layers, nodes=nodes
, activations=activations, optimizer=optimizer, loss=loss,
                                   validation_split=validation_split, epochs=epochs
                                    )
```

```
Loop State: 50/50
```

How does the mean of the mean squared errors compare to that from Step B?

- ...

In [25]:

```
report_scoring(layers_3_result)
```

Out[25]:

50 times	Average	Standard Deviation
Mean Squared Error	168.04655	5.433391