Build a Regression Model in Keras _ Part B

Result

predictors.mean()

Item	Mean
Cement	2.432224e-15
Blast Furnace Slag	-8.513686e-16
Fly Ash	3.837815e-16
Water	1.846743e-15
Superplasticizer	-9.641155e-16
Coarse Aggregate	6.818710e-15
Fine Aggregate	1.232571e-14
Age	3.640022e-16

dtype: float64

Table of Contents

Download and Clean Dataset

Let's start by importing the *pandas* and the Numpy libraries.

In [2]:

```
import pandas as pd
import numpy as np
```

We will be playing around with the same dataset that we used in the videos.

The dataset is about the compressive strength of different samples of concrete based on the volumes of the different ingredients that were used to make them. Ingredients include:

1. Cement

2. Blast Furnace Slag

3. Fly Ash

4. Water

5. Superplasticizer

6. Coarse Aggregate

7. Fine Aggregate

Let's download the data and read it into a *pandas* dataframe.

In [3]:

```
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[3]:

	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

So the first concrete sample has 540 cubic meter of cement, 0 cubic meter of blast furnace slag, 0 cubic meter of fly ash, 162 cubic meter of water, 2.5 cubic meter of superplaticizer, 1040 cubic meter of coarse aggregate, 676 cubic meter of fine aggregate. Such a concrete mix which is 28 days old, has a compressive strength of 79.99 MPa.

Split data into predictors and target

The target variable in this problem is the concrete sample strength. Therefore, our predictors will be all the other columns.

In [4]:

```
concrete_data_columns = concrete_data.columns

predictors = concrete_data[concrete_data_columns[concrete_data_columns != 'Strengt
h']] # all columns except Strength
target = concrete_data['Strength'] # Strength column
```

Let's do a quick sanity check of the predictors and the target dataframes.

Finally, the last step is to normalize the data by substracting the mean and dividing by the standard deviation.

In [5]:

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

Out[5]:

	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age
0	2.476712	-0.856472	-0.846733	-0.916319	-0.620147	0.862735	-1.217079	-0.279597
1	2.476712	-0.856472	-0.846733	-0.916319	-0.620147	1.055651	-1.217079	-0.279597
2	0.491187	0.795140	-0.846733	2.174405	-1.038638	-0.526262	-2.239829	3.551340
3	0.491187	0.795140	-0.846733	2.174405	-1.038638	-0.526262	-2.239829	5.055221
4	-0.790075	0.678079	-0.846733	0.488555	-1.038638	0.070492	0.647569	4.976069

Let's save the number of predictors to *n_cols* since we will need this number when building our network.

In [6]:

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

Import Keras

Recall from the videos that Keras normally runs on top of a low-level library such as TensorFlow. This means that to be able to use the Keras library, you will have to install TensorFlow first and when you import the Keras library, it will be explicitly displayed what backend was used to install the Keras library. In CC Labs, we used TensorFlow as the backend to install Keras, so it should clearly print that when we import Keras.

Let's go ahead and import the Keras library

In [7]:

```
import keras
```

```
Using TensorFlow backend.
/home/jupyterlab/conda/envs/python/lib/python3.6/site-packages/tensorfl
ow/python/framework/dtypes.py:519: FutureWarning: Passing (type, 1) or
'1type' as a synonym of type is deprecated; in a future version of nump
y, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
/home/jupyterlab/conda/envs/python/lib/python3.6/site-packages/tensorfl
ow/python/framework/dtypes.py:520: FutureWarning: Passing (type, 1) or
'1type' as a synonym of type is deprecated; in a future version of nump
y, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/home/jupyterlab/conda/envs/python/lib/python3.6/site-packages/tensorfl
ow/python/framework/dtypes.py:521: FutureWarning: Passing (type, 1) or
'1type' as a synonym of type is deprecated; in a future version of nump
y, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
/home/jupyterlab/conda/envs/python/lib/python3.6/site-packages/tensorfl
ow/python/framework/dtypes.py:522: FutureWarning: Passing (type, 1) or
'1type' as a synonym of type is deprecated; in a future version of nump
y, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
/home/jupyterlab/conda/envs/python/lib/python3.6/site-packages/tensorfl
ow/python/framework/dtypes.py:523: FutureWarning: Passing (type, 1) or
'1type' as a synonym of type is deprecated; in a future version of nump
y, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
/home/jupyterlab/conda/envs/python/lib/python3.6/site-packages/tensorfl
ow/python/framework/dtypes.py:528: FutureWarning: Passing (type, 1) or
'1type' as a synonym of type is deprecated; in a future version of nump
y, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

As you can see, the TensorFlow backend was used to install the Keras library.

Let's import the rest of the packages from the Keras library that we will need to build our regressoin model.

In [8]:

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

Build a Neural Network

Let's define a function that defines our regression model for us so that we can conveniently call it to create our model.

In [9]:

```
# define regression model
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
```

The above function create a model that has two hidden layers, each of 10 hidden units.

Train and Test the Network

Let's call the function now to create our model.

In [10]:

```
# build the model
model = regression_model()
```

Next, we will train and test the model at the same time using the *fit* method. We will leave out 30% of the data for validation and we will train the model for 50 epochs.

In [11]:

```
# fit the model
model.fit(predictors_norm, target, validation_split=0.3, epochs=50, verbose=2)
```

```
Train on 721 samples, validate on 309 samples
Epoch 1/50
 - 2s - loss: 1711.4989 - val_loss: 1221.8312
Epoch 2/50
 - 0s - loss: 1692.4718 - val_loss: 1209.9213
Epoch 3/50
 - 0s - loss: 1673.0860 - val_loss: 1197.9194
Epoch 4/50
 - 1s - loss: 1653.3507 - val_loss: 1185.7700
Epoch 5/50
 - 1s - loss: 1633.0977 - val_loss: 1173.5639
Epoch 6/50
 - 1s - loss: 1611.7509 - val_loss: 1160.8569
Epoch 7/50
 - 1s - loss: 1589.7819 - val_loss: 1147.9310
Epoch 8/50
 - 1s - loss: 1566.4176 - val_loss: 1134.5165
Epoch 9/50
 - 1s - loss: 1541.9591 - val_loss: 1120.4375
Epoch 10/50
 - 1s - loss: 1516.4314 - val_loss: 1105.9220
Epoch 11/50
 - 1s - loss: 1489.3917 - val_loss: 1091.0506
Epoch 12/50
 - 0s - loss: 1461.6187 - val_loss: 1075.4174
Epoch 13/50
 - 1s - loss: 1432.4808 - val_loss: 1059.5400
Epoch 14/50
 - 0s - loss: 1402.4288 - val_loss: 1042.8070
Epoch 15/50
 - 1s - loss: 1370.8199 - val_loss: 1025.9860
Epoch 16/50
 - 1s - loss: 1339.1600 - val_loss: 1008.4214
Epoch 17/50
 - 1s - loss: 1305.4521 - val_loss: 990.5801
Epoch 18/50
 - 1s - loss: 1271.0235 - val_loss: 972.4215
Epoch 19/50
 - 1s - loss: 1235.8714 - val_loss: 953.8644
Epoch 20/50
 - 1s - loss: 1200.4883 - val_loss: 934.6821
Epoch 21/50
 - 0s - loss: 1164.6588 - val_loss: 915.2722
Epoch 22/50
 - 1s - loss: 1128.4606 - val_loss: 895.5838
Epoch 23/50
 - 0s - loss: 1091.9728 - val_loss: 875.9351
Epoch 24/50
 - 1s - loss: 1055.4538 - val_loss: 855.7119
Epoch 25/50
 - 1s - loss: 1019.5089 - val_loss: 835.4262
Epoch 26/50
 - 1s - loss: 983.3954 - val_loss: 814.9198
Epoch 27/50
 - 0s - loss: 947.7911 - val_loss: 794.9194
Epoch 28/50
 - 0s - loss: 912.3611 - val_loss: 774.8496
```

```
Epoch 29/50
 - 0s - loss: 877.6398 - val_loss: 755.0444
Epoch 30/50
 - 0s - loss: 843.9990 - val_loss: 734.8582
Epoch 31/50
 - 0s - loss: 811.3524 - val_loss: 715.1335
Epoch 32/50
 - 0s - loss: 779.5729 - val_loss: 695.0538
Epoch 33/50
 - 0s - loss: 748.3396 - val_loss: 675.8676
Epoch 34/50
 - 0s - loss: 719.0644 - val_loss: 656.0712
Epoch 35/50
 - 0s - loss: 689.8357 - val_loss: 637.6149
Epoch 36/50
 - 0s - loss: 662.3096 - val_loss: 619.1739
Epoch 37/50
 - 0s - loss: 636.0754 - val_loss: 600.4880
Epoch 38/50
 - 0s - loss: 610.2736 - val_loss: 582.4201
Epoch 39/50
 - 0s - loss: 586.4063 - val_loss: 564.4967
Epoch 40/50
 - 0s - loss: 563.2232 - val_loss: 547.2692
Epoch 41/50
 - 0s - loss: 541.2252 - val_loss: 530.7381
Epoch 42/50
 - 0s - loss: 520.6786 - val_loss: 514.3229
Epoch 43/50
 - 0s - loss: 500.7559 - val_loss: 497.9842
Epoch 44/50
 - 0s - loss: 482.0781 - val_loss: 482.3282
Epoch 45/50
 - 0s - loss: 464.5343 - val_loss: 466.4495
Epoch 46/50
 - 0s - loss: 447.7795 - val_loss: 451.4834
Epoch 47/50
 - 0s - loss: 431.8689 - val_loss: 437.0190
Epoch 48/50
 - 0s - loss: 417.0691 - val_loss: 423.2871
Epoch 49/50
 - 0s - loss: 403.0833 - val_loss: 409.1368
Epoch 50/50
 - 0s - loss: 389.5511 - val_loss: 395.7581
```

Out[11]:

```
<keras.callbacks.History at 0x7f0166367588>
```

You can refer to this [link](https://keras.io/models/sequential/) to learn about other functions that you can use for prediction or evaluation.

In [12]:

```
model = regression_model()
```

In [13]:

```
model.fit(predictors_norm, target, validation_split=0.3, epochs=50, verbose=2)
```

```
Train on 721 samples, validate on 309 samples
Epoch 1/50
 - 1s - loss: 1696.1677 - val_loss: 1205.1732
Epoch 2/50
 - 0s - loss: 1674.4248 - val_loss: 1192.9089
Epoch 3/50
 - 0s - loss: 1652.1294 - val_loss: 1180.3829
Epoch 4/50
 - 0s - loss: 1629.5993 - val_loss: 1167.5001
Epoch 5/50
 - 0s - loss: 1606.4634 - val_loss: 1154.0415
Epoch 6/50
 - 0s - loss: 1582.5934 - val_loss: 1139.9782
Epoch 7/50
 - 1s - loss: 1557.4204 - val_loss: 1125.3566
Epoch 8/50
 - 0s - loss: 1531.4932 - val_loss: 1110.1046
Epoch 9/50
 - 0s - loss: 1504.3119 - val_loss: 1094.3082
Epoch 10/50
 - 0s - loss: 1475.7358 - val_loss: 1078.1465
Epoch 11/50
 - 0s - loss: 1446.4053 - val_loss: 1061.3482
Epoch 12/50
 - 1s - loss: 1415.9390 - val_loss: 1043.5264
Epoch 13/50
 - 1s - loss: 1383.8973 - val_loss: 1025.4712
Epoch 14/50
 - 1s - loss: 1351.6654 - val_loss: 1006.6807
Epoch 15/50
 - 1s - loss: 1318.1523 - val_loss: 987.8780
Epoch 16/50
 - 0s - loss: 1284.1083 - val_loss: 968.4411
Epoch 17/50
 - 0s - loss: 1249.1506 - val_loss: 948.5207
Epoch 18/50
 - 1s - loss: 1213.5545 - val_loss: 928.2050
Epoch 19/50
 - 1s - loss: 1177.6338 - val_loss: 908.0533
Epoch 20/50
 - 1s - loss: 1140.9328 - val_loss: 887.1632
Epoch 21/50
 - 1s - loss: 1104.5342 - val_loss: 866.1762
Epoch 22/50
 - 1s - loss: 1067.3289 - val_loss: 844.7585
Epoch 23/50
 - 1s - loss: 1030.3784 - val_loss: 823.5787
Epoch 24/50
 - 1s - loss: 993.4599 - val_loss: 802.8675
Epoch 25/50
 - 0s - loss: 957.4142 - val_loss: 781.5620
Epoch 26/50
 - 1s - loss: 921.0617 - val_loss: 760.7687
Epoch 27/50
 - 1s - loss: 885.9281 - val_loss: 739.4633
Epoch 28/50
 - 0s - loss: 850.5479 - val_loss: 718.9233
```

```
Epoch 29/50
 - 1s - loss: 816.8905 - val_loss: 698.0400
Epoch 30/50
 - 1s - loss: 783.7427 - val_loss: 677.2268
Epoch 31/50
 - 1s - loss: 751.1527 - val_loss: 657.4656
Epoch 32/50
 - 0s - loss: 720.2465 - val_loss: 637.8150
Epoch 33/50
 - 0s - loss: 690.1726 - val_loss: 618.5611
Epoch 34/50
 - 1s - loss: 660.7655 - val_loss: 600.2462
Epoch 35/50
 - 1s - loss: 633.2927 - val_loss: 581.3520
Epoch 36/50
 - 1s - loss: 605.8629 - val_loss: 563.6073
Epoch 37/50
 - 1s - loss: 580.4696 - val_loss: 545.6735
Epoch 38/50
 - 0s - loss: 555.7655 - val_loss: 528.6938
Epoch 39/50
 - 1s - loss: 532.5409 - val_loss: 512.2938
Epoch 40/50
 - 0s - loss: 510.4634 - val_loss: 496.6151
Epoch 41/50
 - 1s - loss: 489.7261 - val_loss: 481.0181
Epoch 42/50
 - 0s - loss: 470.1424 - val_loss: 466.7966
Epoch 43/50
 - 0s - loss: 451.8457 - val_loss: 452.2396
Epoch 44/50
 - 0s - loss: 434.3136 - val_loss: 439.2589
Epoch 45/50
 - 1s - loss: 418.2094 - val_loss: 426.0502
Epoch 46/50
 - 1s - loss: 403.1780 - val_loss: 413.8536
Epoch 47/50
 - 1s - loss: 388.9723 - val_loss: 401.7463
Epoch 48/50
 - 1s - loss: 375.5718 - val_loss: 391.1045
Epoch 49/50
 - 1s - loss: 363.5033 - val_loss: 380.5180
Epoch 50/50
 - 1s - loss: 352.1709 - val_loss: 370.6975
```

Out[13]:

```
<keras.callbacks.History at 0x7f0164551ef0>
```

In [14]:

```
model = regression_model()
```

In [15]:

```
model.fit(predictors_norm, target, validation_split=0.3, epochs=50, verbose=2)
```

```
Train on 721 samples, validate on 309 samples
Epoch 1/50
 - 2s - loss: 1728.5768 - val_loss: 1247.3424
Epoch 2/50
 - 1s - loss: 1708.5013 - val_loss: 1236.0506
Epoch 3/50
 - 0s - loss: 1688.9116 - val_loss: 1224.9578
Epoch 4/50
 - 0s - loss: 1669.2916 - val_loss: 1214.4407
Epoch 5/50
 - 1s - loss: 1649.7405 - val_loss: 1204.0223
Epoch 6/50
 - 0s - loss: 1629.9785 - val_loss: 1193.6450
Epoch 7/50
 - 1s - loss: 1609.9223 - val_loss: 1183.4566
Epoch 8/50
 - 1s - loss: 1589.6233 - val_loss: 1173.2516
Epoch 9/50
 - 1s - loss: 1569.4277 - val_loss: 1162.8928
Epoch 10/50
 - 1s - loss: 1548.1741 - val_loss: 1152.7625
Epoch 11/50
 - 1s - loss: 1526.9603 - val_loss: 1142.1318
Epoch 12/50
 - 1s - loss: 1504.6511 - val_loss: 1131.4254
Epoch 13/50
 - 0s - loss: 1482.2491 - val_loss: 1120.3812
Epoch 14/50
 - 0s - loss: 1458.9476 - val_loss: 1109.1980
Epoch 15/50
 - 1s - loss: 1435.2923 - val_loss: 1097.6263
Epoch 16/50
 - 0s - loss: 1410.9882 - val_loss: 1085.8319
Epoch 17/50
 - 1s - loss: 1386.1910 - val_loss: 1074.1326
Epoch 18/50
 - 0s - loss: 1361.3060 - val_loss: 1061.9616
Epoch 19/50
 - 0s - loss: 1335.8924 - val_loss: 1049.6028
Epoch 20/50
 - 0s - loss: 1310.0226 - val_loss: 1036.8008
Epoch 21/50
 - 1s - loss: 1283.8813 - val_loss: 1024.1250
Epoch 22/50
 - 1s - loss: 1257.4994 - val_loss: 1010.8475
Epoch 23/50
 - 1s - loss: 1230.6285 - val_loss: 997.4467
Epoch 24/50
 - 1s - loss: 1203.8660 - val_loss: 983.8288
Epoch 25/50
 - 1s - loss: 1177.0770 - val_loss: 969.5986
Epoch 26/50
 - 1s - loss: 1149.7590 - val_loss: 954.7330
Epoch 27/50
 - 0s - loss: 1122.0375 - val_loss: 939.8554
Epoch 28/50
 - 1s - loss: 1094.1217 - val_loss: 924.8475
```

```
Epoch 29/50
 - 1s - loss: 1065.8663 - val_loss: 908.1475
Epoch 30/50
 - 0s - loss: 1036.7879 - val_loss: 890.9676
Epoch 31/50
 - 1s - loss: 1007.5199 - val_loss: 873.1516
Epoch 32/50
 - 1s - loss: 977.6479 - val_loss: 854.8861
Epoch 33/50
 - 1s - loss: 946.8845 - val_loss: 836.1368
Epoch 34/50
 - 1s - loss: 916.3753 - val_loss: 815.6918
Epoch 35/50
 - 6s - loss: 884.4618 - val_loss: 794.5632
Epoch 36/50
 - 0s - loss: 852.3296 - val_loss: 773.0929
Epoch 37/50
 - 1s - loss: 820.0540 - val_loss: 749.8852
Epoch 38/50
 - 1s - loss: 787.0309 - val_loss: 726.6136
Epoch 39/50
 - 1s - loss: 753.7467 - val_loss: 702.8593
Epoch 40/50
 - 0s - loss: 721.0309 - val_loss: 678.6941
Epoch 41/50
 - 0s - loss: 688.7194 - val_loss: 653.6791
Epoch 42/50
 - 1s - loss: 656.5256 - val_loss: 628.8406
Epoch 43/50
 - 0s - loss: 625.3068 - val_loss: 603.5848
Epoch 44/50
 - 1s - loss: 594.5003 - val_loss: 579.5138
Epoch 45/50
 - 1s - loss: 564.8973 - val_loss: 555.7843
Epoch 46/50
 - 1s - loss: 536.8039 - val_loss: 532.1986
Epoch 47/50
 - 1s - loss: 509.7000 - val_loss: 510.3199
Epoch 48/50
 - 1s - loss: 484.4889 - val_loss: 488.9479
Epoch 49/50
 - 1s - loss: 460.3913 - val_loss: 468.0664
Epoch 50/50
 - 0s - loss: 437.8351 - val_loss: 448.4843
```

Out[15]:

```
<keras.callbacks.History at 0x7f015473ba58>
```

In [16]:

```
score = model.evaluate(predictors_norm, target)
```

```
1030/1030 [==============================] - 0s 115us/step
```

Feel free to vary the following and note what impact each change has on the model's performance:

1. Increase or decreate number of neurons in hidden layers
2. Add more hidden layers
3. Increase number of epochs

In [17]:

```
target.mean()
```

Out[17]:

```
35.817961165048544
```

In [18]:

```
target.std()
```

Out[18]:

```
16.705741961912512
```

In [20]:

```
predictors_norm.mean()
```

Out[20]:

```
Cement                2.432224e-15
Blast Furnace Slag   -8.513686e-16
Fly Ash               3.837815e-16
Water                 1.846743e-15
Superplasticizer     -9.641155e-16
Coarse Aggregate      6.818710e-15
Fine Aggregate        1.232571e-14
Age                   3.640022e-16
dtype: float64
```

In [21]:

```
predictors_norm.std()
```

Out[21]:

```
Cement                1.0
Blast Furnace Slag    1.0
Fly Ash               1.0
Water                 1.0
Superplasticizer      1.0
Coarse Aggregate      1.0
Fine Aggregate        1.0
Age                   1.0
dtype: float64
```

In [ ]:

```

```