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 [1]:

```
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 [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

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 [3]:

```
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 [4]:

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

Out[4]:

	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 [5]:

```
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 [6]:

```
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 [7]:

```
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 [8]:

```
# 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
```

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 [9]:

```
# 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 [10]:

```
# 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
 - 1s - loss: 1697.7120 - val_loss: 1224.8229
Epoch 2/50
 - 1s - loss: 1680.1567 - val_loss: 1212.7504
Epoch 3/50
 - 0s - loss: 1654.9304 - val_loss: 1192.4209
Epoch 4/50
 - 0s - loss: 1614.7245 - val_loss: 1158.8317
Epoch 5/50
 - 0s - loss: 1553.2393 - val_loss: 1108.7484
Epoch 6/50
 - 0s - loss: 1464.8055 - val_loss: 1036.9128
Epoch 7/50
 - 0s - loss: 1340.2559 - val_loss: 933.5972
Epoch 8/50
 - 0s - loss: 1164.7225 - val_loss: 791.4932
Epoch 9/50
 - 0s - loss: 938.6757 - val_loss: 628.6110
Epoch 10/50
 - 0s - loss: 693.5522 - val_loss: 478.8902
Epoch 11/50
 - 0s - loss: 480.2239 - val_loss: 378.8241
Epoch 12/50
 - 0s - loss: 345.8603 - val_loss: 321.3862
Epoch 13/50
 - 0s - loss: 276.8219 - val_loss: 286.5745
Epoch 14/50
 - 0s - loss: 244.0334 - val_loss: 264.5930
Epoch 15/50
 - 0s - loss: 228.8953 - val_loss: 250.1560
Epoch 16/50
 - 0s - loss: 219.5803 - val_loss: 237.6332
Epoch 17/50
 - 0s - loss: 212.8663 - val_loss: 231.3798
Epoch 18/50
 - 0s - loss: 206.6765 - val_loss: 225.3033
Epoch 19/50
 - 0s - loss: 202.5399 - val_loss: 220.5405
Epoch 20/50
 - 0s - loss: 197.9348 - val_loss: 214.3778
Epoch 21/50
 - 0s - loss: 194.1849 - val_loss: 210.1460
Epoch 22/50
 - 0s - loss: 190.7083 - val_loss: 204.2309
Epoch 23/50
 - 0s - loss: 187.4776 - val_loss: 198.0577
Epoch 24/50
 - 0s - loss: 184.2677 - val_loss: 194.4762
Epoch 25/50
 - 0s - loss: 181.0198 - val_loss: 191.0467
Epoch 26/50
 - 0s - loss: 178.1203 - val_loss: 184.9691
Epoch 27/50
 - 0s - loss: 175.3959 - val_loss: 181.3125
Epoch 28/50
 - 1s - loss: 172.0204 - val_loss: 177.6041
```

```
Epoch 29/50
 - 0s - loss: 169.4722 - val_loss: 173.5333
Epoch 30/50
 - 0s - loss: 166.6460 - val_loss: 170.8332
Epoch 31/50
 - 0s - loss: 164.1814 - val_loss: 167.6179
Epoch 32/50
 - 0s - loss: 161.4800 - val_loss: 162.4757
Epoch 33/50
 - 0s - loss: 158.7780 - val_loss: 159.4576
Epoch 34/50
 - 0s - loss: 155.9600 - val_loss: 154.1154
Epoch 35/50
 - 0s - loss: 153.4932 - val_loss: 150.6499
Epoch 36/50
 - 0s - loss: 151.3864 - val_loss: 147.8129
Epoch 37/50
 - 0s - loss: 148.0773 - val_loss: 145.9286
Epoch 38/50
 - 0s - loss: 145.7805 - val_loss: 142.1397
Epoch 39/50
 - 0s - loss: 143.3021 - val_loss: 137.9944
Epoch 40/50
 - 0s - loss: 140.0806 - val_loss: 138.1581
Epoch 41/50
 - 0s - loss: 137.5236 - val_loss: 135.3461
Epoch 42/50
 - 0s - loss: 134.8125 - val_loss: 133.8386
Epoch 43/50
 - 0s - loss: 132.1174 - val_loss: 130.8983
Epoch 44/50
 - 0s - loss: 129.3308 - val_loss: 127.3620
Epoch 45/50
 - 0s - loss: 126.1189 - val_loss: 126.8274
Epoch 46/50
 - 0s - loss: 123.1578 - val_loss: 125.4609
Epoch 47/50
 - 0s - loss: 120.1846 - val_loss: 123.1138
Epoch 48/50
 - 0s - loss: 117.5718 - val_loss: 122.3964
Epoch 49/50
 - 0s - loss: 114.9057 - val_loss: 118.5039
Epoch 50/50
 - 0s - loss: 111.8043 - val_loss: 118.2649
```

Out[10]:

```
<keras.callbacks.History at 0x7f2451ab4828>
```

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 [11]:

```
model = regression_model()
```

In [12]:

```
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: 1698.2179 - val_loss: 1215.7188
Epoch 2/50
 - 0s - loss: 1663.4352 - val_loss: 1190.7206
Epoch 3/50
 - 0s - loss: 1615.9694 - val_loss: 1150.7228
Epoch 4/50
 - 0s - loss: 1540.8572 - val_loss: 1083.7393
Epoch 5/50
 - 0s - loss: 1421.3596 - val_loss: 983.2373
Epoch 6/50
 - 0s - loss: 1248.8737 - val_loss: 846.3096
Epoch 7/50
 - 0s - loss: 1018.6126 - val_loss: 678.3351
Epoch 8/50
 - 0s - loss: 755.0307 - val_loss: 496.8488
Epoch 9/50
 - 0s - loss: 509.7421 - val_loss: 342.3384
Epoch 10/50
 - 0s - loss: 346.9056 - val_loss: 244.8836
Epoch 11/50
 - 0s - loss: 272.3920 - val_loss: 204.4806
Epoch 12/50
 - 0s - loss: 249.0687 - val_loss: 188.1479
Epoch 13/50
 - 0s - loss: 236.2585 - val_loss: 182.8122
Epoch 14/50
 - 0s - loss: 226.3362 - val_loss: 177.2853
Epoch 15/50
 - 0s - loss: 217.6977 - val_loss: 175.5469
Epoch 16/50
 - 0s - loss: 210.9169 - val_loss: 171.9349
Epoch 17/50
 - 0s - loss: 204.5435 - val_loss: 168.7015
Epoch 18/50
 - 0s - loss: 199.2226 - val_loss: 166.5442
Epoch 19/50
 - 0s - loss: 194.1455 - val_loss: 166.6936
Epoch 20/50
 - 0s - loss: 190.3973 - val_loss: 164.6761
Epoch 21/50
 - 0s - loss: 186.4633 - val_loss: 165.6336
Epoch 22/50
 - 0s - loss: 183.4405 - val_loss: 164.7956
Epoch 23/50
 - 0s - loss: 180.0897 - val_loss: 163.7727
Epoch 24/50
 - 0s - loss: 177.5580 - val_loss: 163.8389
Epoch 25/50
 - 0s - loss: 174.8561 - val_loss: 163.9029
Epoch 26/50
 - 0s - loss: 172.6467 - val_loss: 162.2881
Epoch 27/50
 - 0s - loss: 170.4997 - val_loss: 163.4355
Epoch 28/50
 - 0s - loss: 168.3390 - val_loss: 163.0648
```

```
Epoch 29/50
 - 0s - loss: 166.4030 - val_loss: 161.0971
Epoch 30/50
 - 0s - loss: 165.2311 - val_loss: 163.0435
Epoch 31/50
 - 0s - loss: 163.0253 - val_loss: 162.8007
Epoch 32/50
 - 0s - loss: 161.4569 - val_loss: 163.9248
Epoch 33/50
 - 0s - loss: 160.0337 - val_loss: 163.0733
Epoch 34/50
 - 0s - loss: 158.2910 - val_loss: 164.2659
Epoch 35/50
 - 0s - loss: 156.9108 - val_loss: 164.2933
Epoch 36/50
 - 0s - loss: 155.6978 - val_loss: 164.6828
Epoch 37/50
 - 0s - loss: 154.7859 - val_loss: 165.0226
Epoch 38/50
 - 0s - loss: 153.0035 - val_loss: 166.9518
Epoch 39/50
 - 0s - loss: 151.9325 - val_loss: 166.4964
Epoch 40/50
 - 0s - loss: 151.0522 - val_loss: 167.9943
Epoch 41/50
 - 0s - loss: 149.5892 - val_loss: 166.5662
Epoch 42/50
 - 0s - loss: 148.5128 - val_loss: 168.2066
Epoch 43/50
 - 0s - loss: 147.6377 - val_loss: 169.7690
Epoch 44/50
 - 0s - loss: 146.5488 - val_loss: 170.3760
Epoch 45/50
 - 0s - loss: 145.7479 - val_loss: 168.8901
Epoch 46/50
 - 0s - loss: 145.3422 - val_loss: 171.4515
Epoch 47/50
 - 0s - loss: 144.0320 - val_loss: 171.3076
Epoch 48/50
 - 0s - loss: 142.9652 - val_loss: 173.2709
Epoch 49/50
 - 0s - loss: 142.1486 - val_loss: 173.2225
Epoch 50/50
 - 0s - loss: 141.5812 - val_loss: 173.2362
```

Out[12]:

```
<keras.callbacks.History at 0x7f245037ed30>
```

In [13]:

```
model = regression_model()
```

In [14]:

```
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: 1704.5568 - val_loss: 1231.4729
Epoch 2/50
 - 0s - loss: 1687.8204 - val_loss: 1221.6282
Epoch 3/50
 - 0s - loss: 1668.1951 - val_loss: 1208.6375
Epoch 4/50
 - 0s - loss: 1640.5399 - val_loss: 1189.0441
Epoch 5/50
 - 0s - loss: 1599.9645 - val_loss: 1159.2489
Epoch 6/50
 - 0s - loss: 1537.8771 - val_loss: 1113.5315
Epoch 7/50
 - 0s - loss: 1449.6832 - val_loss: 1050.2998
Epoch 8/50
 - 0s - loss: 1330.4825 - val_loss: 965.3668
Epoch 9/50
 - 0s - loss: 1172.7266 - val_loss: 858.7152
Epoch 10/50
 - 0s - loss: 982.5863 - val_loss: 725.2443
Epoch 11/50
 - 0s - loss: 765.9098 - val_loss: 576.4156
Epoch 12/50
 - 0s - loss: 553.9582 - val_loss: 426.7120
Epoch 13/50
 - 0s - loss: 382.7408 - val_loss: 308.7035
Epoch 14/50
 - 0s - loss: 282.0862 - val_loss: 237.1588
Epoch 15/50
 - 0s - loss: 243.7353 - val_loss: 202.8895
Epoch 16/50
 - 0s - loss: 223.9785 - val_loss: 193.2828
Epoch 17/50
 - 0s - loss: 212.7122 - val_loss: 184.7323
Epoch 18/50
 - 0s - loss: 203.3505 - val_loss: 176.4084
Epoch 19/50
 - 0s - loss: 197.0796 - val_loss: 171.1934
Epoch 20/50
 - 0s - loss: 191.5619 - val_loss: 165.3285
Epoch 21/50
 - 0s - loss: 187.4990 - val_loss: 157.9938
Epoch 22/50
 - 0s - loss: 182.6198 - val_loss: 157.2669
Epoch 23/50
 - 0s - loss: 179.4335 - val_loss: 154.8402
Epoch 24/50
 - 0s - loss: 176.6428 - val_loss: 151.2046
Epoch 25/50
 - 0s - loss: 174.2190 - val_loss: 146.9907
Epoch 26/50
 - 0s - loss: 172.0937 - val_loss: 145.6086
Epoch 27/50
 - 0s - loss: 169.8503 - val_loss: 143.5039
Epoch 28/50
 - 0s - loss: 167.7833 - val_loss: 142.0920
```

```
Epoch 29/50
 - 0s - loss: 165.9921 - val_loss: 141.5505
Epoch 30/50
 - 0s - loss: 164.0796 - val_loss: 140.6397
Epoch 31/50
 - 0s - loss: 162.3230 - val_loss: 138.0121
Epoch 32/50
 - 0s - loss: 160.7015 - val_loss: 137.8299
Epoch 33/50
 - 0s - loss: 158.6024 - val_loss: 136.5406
Epoch 34/50
 - 0s - loss: 156.2944 - val_loss: 136.4598
Epoch 35/50
 - 0s - loss: 154.2929 - val_loss: 135.3828
Epoch 36/50
 - 0s - loss: 152.3761 - val_loss: 136.0370
Epoch 37/50
 - 0s - loss: 150.0692 - val_loss: 137.7441
Epoch 38/50
 - 0s - loss: 148.0991 - val_loss: 137.8791
Epoch 39/50
 - 0s - loss: 145.7735 - val_loss: 137.3048
Epoch 40/50
 - 0s - loss: 143.6673 - val_loss: 138.0521
Epoch 41/50
 - 0s - loss: 141.8274 - val_loss: 138.6647
Epoch 42/50
 - 0s - loss: 139.7745 - val_loss: 139.2951
Epoch 43/50
 - 0s - loss: 137.7955 - val_loss: 140.7642
Epoch 44/50
 - 0s - loss: 136.0377 - val_loss: 141.7609
Epoch 45/50
 - 1s - loss: 134.4538 - val_loss: 140.9436
Epoch 46/50
 - 0s - loss: 132.6085 - val_loss: 144.4127
Epoch 47/50
 - 0s - loss: 131.1760 - val_loss: 143.7522
Epoch 48/50
 - 0s - loss: 129.4891 - val_loss: 147.4993
Epoch 49/50
 - 0s - loss: 127.9810 - val_loss: 146.4812
Epoch 50/50
 - 0s - loss: 126.4801 - val_loss: 148.7281
```

Out[14]:

```
<keras.callbacks.History at 0x7f24486a26d8>
```

In [15]:

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

```
1030/1030 [==============================] - 0s 118us/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 [16]:

```
target.mean()
```

Out[16]:

```
35.817961165048544
```

In [17]:

```
target.std()
```

Out[17]:

```
16.705741961912512
```

In [19]:

```
predictors_norm.mean()
```

Out[19]:

```
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 [20]:

```
predictors_norm.std()
```

Out[20]:

```
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 [ ]:

```

```