Build a Regression Model in Keras _ Part A

Result

predictors.mean()

Item	Mean
Cement	281.167864
Blast Furnace Slag	73.895825
Fly Ash	54.188350
Water	181.567282
Superplasticizer	6.204660
Coarse Aggregate	972.918932
Fine Aggregate	773.580485
Age	45.662136

dtype: float64

predictors.std()

Item	Mean
Cement	104.506364
Blast Furnace Slag	86.279342
Fly Ash	63.997004
Water	21.354219
Superplasticizer	5.973841
Coarse Aggregate	77.753954
Fine Aggregate	80.175980
Age	63.169912

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-c
ourses-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.

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

In [4]:

```
n_cols = predictors.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 [5]:

```
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 [6]:

```
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 [7]:

```
# 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 [8]:

```
# 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 [9]:

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

```
Train on 721 samples, validate on 309 samples
Epoch 1/50
 - 1s - loss: 40562.6075 - val_loss: 19354.4713
Epoch 2/50
 - 0s - loss: 8997.8915 - val_loss: 5446.0898
Epoch 3/50
 - 0s - loss: 3408.7361 - val_loss: 3742.4317
Epoch 4/50
 - 0s - loss: 2916.0421 - val_loss: 3533.7680
Epoch 5/50
 - 0s - loss: 2695.0247 - val_loss: 3413.3553
Epoch 6/50
 - 0s - loss: 2482.8500 - val_loss: 3269.1465
Epoch 7/50
 - 0s - loss: 2289.8921 - val_loss: 3106.4426
Epoch 8/50
 - 0s - loss: 2102.1340 - val_loss: 2951.4280
Epoch 9/50
 - 0s - loss: 1925.2430 - val_loss: 2804.2327
Epoch 10/50
 - 0s - loss: 1769.0078 - val_loss: 2677.3645
Epoch 11/50
 - 0s - loss: 1627.3966 - val_loss: 2559.1774
Epoch 12/50
 - 0s - loss: 1504.8699 - val_loss: 2443.4871
Epoch 13/50
 - 0s - loss: 1383.9740 - val_loss: 2312.0766
Epoch 14/50
 - 0s - loss: 1285.6321 - val_loss: 2224.8974
Epoch 15/50
 - 0s - loss: 1197.4286 - val_loss: 2145.6935
Epoch 16/50
 - 0s - loss: 1117.1521 - val_loss: 2033.2754
Epoch 17/50
 - 0s - loss: 1048.5382 - val_loss: 1965.0139
Epoch 18/50
 - 0s - loss: 988.5790 - val_loss: 1878.0319
Epoch 19/50
 - 0s - loss: 929.8005 - val_loss: 1807.1395
Epoch 20/50
 - 0s - loss: 882.2844 - val_loss: 1732.6019
Epoch 21/50
 - 0s - loss: 839.3709 - val_loss: 1662.2680
Epoch 22/50
 - 0s - loss: 799.5471 - val_loss: 1604.6183
Epoch 23/50
 - 0s - loss: 765.7856 - val_loss: 1550.7620
Epoch 24/50
 - 0s - loss: 729.9290 - val_loss: 1491.4123
Epoch 25/50
 - 0s - loss: 702.2763 - val_loss: 1437.7337
Epoch 26/50
 - 0s - loss: 675.4987 - val_loss: 1381.2722
Epoch 27/50
 - 0s - loss: 649.3208 - val_loss: 1336.0414
Epoch 28/50
 - 0s - loss: 627.0339 - val_loss: 1284.3306
```

```
Epoch 29/50
 - 0s - loss: 605.3076 - val_loss: 1240.0054
Epoch 30/50
 - 0s - loss: 584.6712 - val_loss: 1193.6695
Epoch 31/50
 - 0s - loss: 565.0752 - val_loss: 1155.4967
Epoch 32/50
 - 0s - loss: 547.0044 - val_loss: 1108.0508
Epoch 33/50
 - 0s - loss: 536.5699 - val_loss: 1071.7931
Epoch 34/50
 - 0s - loss: 511.5580 - val_loss: 1024.1752
Epoch 35/50
 - 0s - loss: 495.3470 - val_loss: 978.8584
Epoch 36/50
 - 0s - loss: 479.5066 - val_loss: 948.9949
Epoch 37/50
 - 0s - loss: 465.4431 - val_loss: 900.5298
Epoch 38/50
 - 0s - loss: 450.0316 - val_loss: 875.6420
Epoch 39/50
 - 0s - loss: 436.2808 - val_loss: 839.7511
Epoch 40/50
 - 0s - loss: 423.4676 - val_loss: 800.7503
Epoch 41/50
 - 0s - loss: 411.6116 - val_loss: 780.8262
Epoch 42/50
 - 0s - loss: 398.5978 - val_loss: 731.8497
Epoch 43/50
 - 0s - loss: 386.3814 - val_loss: 714.4407
Epoch 44/50
 - 0s - loss: 375.2968 - val_loss: 660.4929
Epoch 45/50
 - 0s - loss: 361.4536 - val_loss: 646.0115
Epoch 46/50
 - 0s - loss: 353.6602 - val_loss: 605.1034
Epoch 47/50
 - 0s - loss: 342.0523 - val_loss: 578.3189
Epoch 48/50
 - 0s - loss: 334.0425 - val_loss: 554.6108
Epoch 49/50
 - 0s - loss: 322.5900 - val_loss: 534.8040
Epoch 50/50
 - 0s - loss: 313.2981 - val_loss: 509.0243
```

Out[9]:

```
<keras.callbacks.History at 0x7faa1ef06588>
```

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 [10]:

```
model = regression_model()
```

In [11]:

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

```
Train on 721 samples, validate on 309 samples
Epoch 1/50
 - 1s - loss: 8416.2898 - val_loss: 2930.2769
Epoch 2/50
 - 0s - loss: 3852.0342 - val_loss: 2420.4482
Epoch 3/50
 - 0s - loss: 3313.5255 - val_loss: 2155.0027
Epoch 4/50
 - 0s - loss: 2942.2414 - val_loss: 1841.9167
Epoch 5/50
 - 0s - loss: 2596.5308 - val_loss: 1629.2634
Epoch 6/50
 - 0s - loss: 2280.7058 - val_loss: 1460.3179
Epoch 7/50
 - 0s - loss: 1995.0305 - val_loss: 1299.6314
Epoch 8/50
 - 0s - loss: 1748.1031 - val_loss: 1162.1754
Epoch 9/50
 - 0s - loss: 1528.8826 - val_loss: 1055.3529
Epoch 10/50
 - 0s - loss: 1344.5965 - val_loss: 981.1630
Epoch 11/50
 - 0s - loss: 1172.9462 - val_loss: 894.0042
Epoch 12/50
 - 0s - loss: 1031.0918 - val_loss: 838.3811
Epoch 13/50
 - 0s - loss: 911.7652 - val_loss: 786.0792
Epoch 14/50
 - 0s - loss: 812.3485 - val_loss: 762.5485
Epoch 15/50
 - 0s - loss: 722.2308 - val_loss: 714.5188
Epoch 16/50
 - 0s - loss: 645.6830 - val_loss: 694.8396
Epoch 17/50
 - 0s - loss: 584.3922 - val_loss: 664.5404
Epoch 18/50
 - 0s - loss: 530.5072 - val_loss: 649.6963
Epoch 19/50
 - 0s - loss: 492.0117 - val_loss: 618.6195
Epoch 20/50
 - 0s - loss: 446.1357 - val_loss: 598.7580
Epoch 21/50
 - 0s - loss: 411.2005 - val_loss: 586.2897
Epoch 22/50
 - 0s - loss: 385.8526 - val_loss: 557.4580
Epoch 23/50
 - 0s - loss: 358.8177 - val_loss: 536.8256
Epoch 24/50
 - 0s - loss: 338.1012 - val_loss: 514.2364
Epoch 25/50
 - 0s - loss: 318.0001 - val_loss: 496.0567
Epoch 26/50
 - 0s - loss: 301.4869 - val_loss: 470.1007
Epoch 27/50
 - 0s - loss: 288.5329 - val_loss: 444.3394
Epoch 28/50
 - 0s - loss: 272.6623 - val_loss: 422.8754
```

```
Epoch 29/50
 - 0s - loss: 260.8902 - val_loss: 402.3423
Epoch 30/50
 - 0s - loss: 246.8919 - val_loss: 379.0095
Epoch 31/50
 - 0s - loss: 236.4207 - val_loss: 358.1976
Epoch 32/50
 - 0s - loss: 226.6401 - val_loss: 351.0278
Epoch 33/50
 - 0s - loss: 219.0350 - val_loss: 318.1313
Epoch 34/50
 - 0s - loss: 209.6482 - val_loss: 306.6366
Epoch 35/50
 - 0s - loss: 201.5324 - val_loss: 283.7971
Epoch 36/50
 - 0s - loss: 193.8761 - val_loss: 273.6053
Epoch 37/50
 - 0s - loss: 187.4440 - val_loss: 251.6677
Epoch 38/50
 - 0s - loss: 181.8609 - val_loss: 238.5323
Epoch 39/50
 - 0s - loss: 176.9942 - val_loss: 225.8732
Epoch 40/50
 - 0s - loss: 171.5999 - val_loss: 216.0015
Epoch 41/50
 - 0s - loss: 166.1838 - val_loss: 220.8213
Epoch 42/50
 - 0s - loss: 163.1542 - val_loss: 196.0382
Epoch 43/50
 - 0s - loss: 159.5071 - val_loss: 187.7536
Epoch 44/50
 - 0s - loss: 155.7434 - val_loss: 179.4457
Epoch 45/50
 - 0s - loss: 152.6399 - val_loss: 171.4061
Epoch 46/50
 - 0s - loss: 150.1296 - val_loss: 166.2962
Epoch 47/50
 - 0s - loss: 147.0033 - val_loss: 161.4621
Epoch 48/50
 - 0s - loss: 146.7481 - val_loss: 155.1636
Epoch 49/50
 - 0s - loss: 142.6591 - val_loss: 149.1615
Epoch 50/50
 - 0s - loss: 143.0025 - val_loss: 145.4346
```

Out[11]:

```
<keras.callbacks.History at 0x7faa1c1e28d0>
```

In [12]:

```
model = regression_model()
```

In [13]:

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

```
Train on 721 samples, validate on 309 samples
Epoch 1/50
 - 1s - loss: 795.2996 - val_loss: 554.0186
Epoch 2/50
 - 0s - loss: 523.5439 - val_loss: 470.9915
Epoch 3/50
 - 0s - loss: 416.7169 - val_loss: 307.7284
Epoch 4/50
 - 0s - loss: 346.4876 - val_loss: 276.3472
Epoch 5/50
 - 0s - loss: 318.4829 - val_loss: 226.1263
Epoch 6/50
 - 0s - loss: 307.7949 - val_loss: 213.1514
Epoch 7/50
 - 0s - loss: 298.6301 - val_loss: 191.7274
Epoch 8/50
 - 0s - loss: 292.3293 - val_loss: 188.5618
Epoch 9/50
 - 0s - loss: 287.0091 - val_loss: 175.1596
Epoch 10/50
 - 0s - loss: 281.6690 - val_loss: 165.7769
Epoch 11/50
 - 0s - loss: 277.3691 - val_loss: 161.5906
Epoch 12/50
 - 0s - loss: 272.4128 - val_loss: 159.5739
Epoch 13/50
 - 0s - loss: 268.0451 - val_loss: 154.3427
Epoch 14/50
 - 0s - loss: 264.3309 - val_loss: 150.1205
Epoch 15/50
 - 0s - loss: 262.4972 - val_loss: 147.3556
Epoch 16/50
 - 0s - loss: 258.0879 - val_loss: 145.7099
Epoch 17/50
 - 0s - loss: 254.7169 - val_loss: 145.9030
Epoch 18/50
 - 0s - loss: 251.9108 - val_loss: 142.5226
Epoch 19/50
 - 0s - loss: 249.2911 - val_loss: 142.6395
Epoch 20/50
 - 0s - loss: 246.3667 - val_loss: 140.1178
Epoch 21/50
 - 0s - loss: 243.4286 - val_loss: 140.2505
Epoch 22/50
 - 0s - loss: 240.7583 - val_loss: 138.3024
Epoch 23/50
 - 0s - loss: 238.0386 - val_loss: 137.6034
Epoch 24/50
 - 0s - loss: 235.8715 - val_loss: 133.9560
Epoch 25/50
 - 0s - loss: 232.5535 - val_loss: 133.9363
Epoch 26/50
 - 0s - loss: 229.8911 - val_loss: 131.4385
Epoch 27/50
 - 0s - loss: 227.2344 - val_loss: 129.2517
Epoch 28/50
 - 0s - loss: 224.6342 - val_loss: 128.4986
```

```
Epoch 29/50
 - 0s - loss: 221.5578 - val_loss: 126.2728
Epoch 30/50
 - 0s - loss: 219.0286 - val_loss: 124.1832
Epoch 31/50
 - 0s - loss: 217.2474 - val_loss: 124.0830
Epoch 32/50
 - 0s - loss: 214.0404 - val_loss: 121.9254
Epoch 33/50
 - 0s - loss: 210.8602 - val_loss: 121.2150
Epoch 34/50
 - 0s - loss: 208.6816 - val_loss: 118.5277
Epoch 35/50
 - 0s - loss: 205.4320 - val_loss: 117.8958
Epoch 36/50
 - 0s - loss: 203.4639 - val_loss: 113.3904
Epoch 37/50
 - 0s - loss: 200.3719 - val_loss: 117.9152
Epoch 38/50
 - 0s - loss: 197.6920 - val_loss: 109.3236
Epoch 39/50
 - 0s - loss: 195.3348 - val_loss: 111.8046
Epoch 40/50
 - 0s - loss: 193.0749 - val_loss: 108.9683
Epoch 41/50
 - 0s - loss: 190.3472 - val_loss: 105.7171
Epoch 42/50
 - 0s - loss: 188.1152 - val_loss: 105.9286
Epoch 43/50
 - 1s - loss: 185.5927 - val_loss: 104.4759
Epoch 44/50
 - 0s - loss: 183.3283 - val_loss: 103.1977
Epoch 45/50
 - 0s - loss: 181.3317 - val_loss: 101.8796
Epoch 46/50
 - 0s - loss: 178.7418 - val_loss: 100.0174
Epoch 47/50
 - 0s - loss: 177.4148 - val_loss: 103.7187
Epoch 48/50
 - 0s - loss: 174.8999 - val_loss: 97.1213
Epoch 49/50
 - 0s - loss: 172.8278 - val_loss: 96.2261
Epoch 50/50
 - 0s - loss: 170.6372 - val_loss: 98.3315
```

Out[13]:

```
<keras.callbacks.History at 0x7faa081c5240>
```

In [14]:

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

```
1030/1030 [==============================] - 0s 99us/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 [15]:

```
target.mean()
```

Out[15]:

```
35.817961165048544
```

In [16]:

```
target.std()
```

Out[16]:

```
16.705741961912512
```

In [17]:

```
predictors.mean()
```

Out[17]:

```
Cement                 281.167864
Blast Furnace Slag      73.895825
Fly Ash                 54.188350
Water                  181.567282
Superplasticizer         6.204660
Coarse Aggregate       972.918932
Fine Aggregate         773.580485
Age                     45.662136
dtype: float64
```

In [18]:

```
predictors.std()
```

Out[18]:

```
Cement                 104.506364
Blast Furnace Slag      86.279342
Fly Ash                 63.997004
Water                   21.354219
Superplasticizer         5.973841
Coarse Aggregate        77.753954
Fine Aggregate          80.175980
Age                     63.169912
dtype: float64
```

In [19]:

```
model.mean()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-19-fd5124642809> in <module>
----> 1 model.mean()

AttributeError: 'Sequential' object has no attribute 'mean'
```

In [ ]:

```
```