Regression Models with Keras

Introduction

As we discussed in the videos, despite the popularity of more powerful libraries such as PyToch and TensorFlow, they are not easy to use and have a steep learning curve. So, for people who are just starting to learn deep learning, there is no better library to use other than the Keras library.

Keras is a high-level API for building deep learning models. It has gained favor for its ease of use and syntactic simplicity facilitating fast development. As you will see in this lab and the other labs in this course, building a very complex deep learning network can be achieved with Keras with only few lines of code. You will appreciate Keras even more, once you learn how to build deep models using PyTorch and TensorFlow in the other courses.

So, in this lab, you will learn how to use the Keras library to build a regression model.

Regression Models with Keras

Objective for this Notebook

1. How to use the Keras library to build a regression model.
2. Download and Clean dataset
3. Build a Neural Network
4. Train and Test the Network.

Table of Contents

Download and Clean Dataset

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

[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.

[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()
```

[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.

Let's check how many data points we have.

[3]:
```
concrete_data.shape
```

[3]:
```
(1030, 9)
```

So, there are approximately 1000 samples to train our model on. Because of the few samples, we have to be careful not to overfit the training data.

Let's check the dataset for any missing values.

[4]:
```
concrete_data.describe()
```

[4]:

	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age	Strength
count	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000
mean	281.167864	73.895825	54.188350	181.567282	6.204660	972.918932	773.580485	45.662136	35.817961
std	104.506364	86.279342	63.997004	21.354219	5.973841	77.753954	80.175980	63.169912	16.705742
min	102.000000	0.000000	0.000000	121.800000	0.000000	801.000000	594.000000	1.000000	2.330000
25%	192.375000	0.000000	0.000000	164.900000	0.000000	932.000000	730.950000	7.000000	23.710000
50%	272.900000	22.000000	0.000000	185.000000	6.400000	968.000000	779.500000	28.000000	34.445000
75%	350.000000	142.950000	118.300000	192.000000	10.200000	1029.400000	824.000000	56.000000	46.135000
max	540.000000	359.400000	200.100000	247.000000	32.200000	1145.000000	992.600000	365.000000	82.600000

[5]:
```
concrete_data.isnull().sum()
```

[5]:
```
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
```

The data looks very clean and is ready to be used to build our model.

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.

[6]:
```
concrete_data_columns = concrete_data.columns

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

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

[7]:
```
predictors.head()
```

[7]:

	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

[8]:
```
target.head()
```

[8]:
```
0    79.99
1    61.89
2    40.27
3    41.05
4    44.30
Name: Strength, dtype: float64
```

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

[9]:
```
predictors_norm = (predictors - predictors.mean()) / predictors.std()
predictors_norm.head()
```

[9]:

	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.

[10]:
```
n_cols = predictors_norm.shape[1] # number of predictors
```

Import Keras

Did you know? IBM Watson Studio lets you build and deploy an AI solution, using the best of open source and IBM software and giving your team a single environment to work in. Learn more here.

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

[11]:
```
import keras
```

```
Using TensorFlow backend.
/home/jupyterlab/conda/envs/python/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:519: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a f
uture version of numpy, 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/tensorflow/python/framework/dtypes.py:520: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a f
uture version of numpy, 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/tensorflow/python/framework/dtypes.py:521: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a f
uture version of numpy, 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/tensorflow/python/framework/dtypes.py:522: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a f
uture version of numpy, 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/tensorflow/python/framework/dtypes.py:523: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a f
uture version of numpy, 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/tensorflow/python/framework/dtypes.py:528: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a f
uture version of numpy, 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.

[12]:
```
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.

[13]:
```
# define regression model
def regression_model():
    # create model
    model = Sequential()
    model.add(Dense(50, activation='relu', input_shape=(n_cols,)))
    model.add(Dense(50, 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 50 hidden units.

Train and Test the Network

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

[14]:
```
# 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 100 epochs.

[15]:
```
# fit the model
model.fit(predictors_norm, target, validation_split=0.3, epochs=100, verbose=2)
```

```
Train on 721 samples, validate on 309 samples
Epoch 1/100
 - 1s - loss: 1676.4478 - val_loss: 1179.5998
Epoch 2/100
 - 0s - loss: 1572.7880 - val_loss: 1081.9592
Epoch 3/100
 - 0s - loss: 1403.6108 - val_loss: 924.0878
Epoch 4/100
 - 1s - loss: 1127.2756 - val_loss: 700.3807
Epoch 5/100
 - 0s - loss: 767.4412 - val_loss: 457.3704
Epoch 6/100
 - 0s - loss: 447.6497 - val_loss: 278.2729
Epoch 7/100
 - 0s - loss: 278.4068 - val_loss: 198.1580
Epoch 8/100
 - 0s - loss: 234.1713 - val_loss: 177.5138
Epoch 9/100
 - 0s - loss: 218.3102 - val_loss: 173.5507
Epoch 10/100
 - 1s - loss: 208.7461 - val_loss: 168.6374
Epoch 11/100
 - 0s - loss: 198.5447 - val_loss: 170.0157
Epoch 12/100
 - 0s - loss: 191.1957 - val_loss: 165.5570
Epoch 13/100
 - 0s - loss: 185.7995 - val_loss: 163.1052
Epoch 14/100
 - 1s - loss: 180.2976 - val_loss: 163.6594
Epoch 15/100
 - 0s - loss: 176.0688 - val_loss: 159.8120
Epoch 16/100
 - 0s - loss: 171.4702 - val_loss: 156.9342
Epoch 17/100
 - 0s - loss: 167.0945 - val_loss: 155.9684
Epoch 18/100
 - 0s - loss: 163.8739 - val_loss: 155.5362
Epoch 19/100
 - 0s - loss: 160.1036 - val_loss: 150.6741
Epoch 20/100
 - 0s - loss: 157.1156 - val_loss: 149.6056
Epoch 21/100
 - 0s - loss: 154.0548 - val_loss: 147.7299
Epoch 22/100
 - 0s - loss: 151.6549 - val_loss: 146.3484
Epoch 23/100
 - 0s - loss: 149.2237 - val_loss: 146.1775
Epoch 24/100
 - 0s - loss: 147.3748 - val_loss: 146.4858
Epoch 25/100
 - 0s - loss: 144.7505 - val_loss: 143.0964
Epoch 26/100
 - 0s - loss: 142.4548 - val_loss: 143.8951
Epoch 27/100
 - 1s - loss: 140.2020 - val_loss: 141.9965
Epoch 28/100
 - 1s - loss: 138.2355 - val_loss: 142.6388
Epoch 29/100
 - 1s - loss: 136.7809 - val_loss: 140.5156
Epoch 30/100
 - 1s - loss: 134.7158 - val_loss: 140.0406
Epoch 31/100
 - 1s - loss: 132.7721 - val_loss: 143.0536
Epoch 32/100
```

```
 - 0s - loss: 130.6649 - val_loss: 139.0087
Epoch 33/100
 - 1s - loss: 128.6540 - val_loss: 140.6569
Epoch 34/100
 - 0s - loss: 126.7214 - val_loss: 138.7366
Epoch 35/100
 - 0s - loss: 125.0624 - val_loss: 139.3592
Epoch 36/100
 - 0s - loss: 123.1202 - val_loss: 136.7178
Epoch 37/100
 - 0s - loss: 121.6531 - val_loss: 140.1370
Epoch 38/100
 - 0s - loss: 119.2358 - val_loss: 134.3423
Epoch 39/100
 - 0s - loss: 117.1158 - val_loss: 135.7271
Epoch 40/100
 - 0s - loss: 115.8326 - val_loss: 133.2634
Epoch 41/100
 - 1s - loss: 112.8638 - val_loss: 129.3699
Epoch 42/100
 - 0s - loss: 110.6232 - val_loss: 129.6133
Epoch 43/100
 - 1s - loss: 108.0162 - val_loss: 129.3136
Epoch 44/100
 - 0s - loss: 105.6112 - val_loss: 127.3529
Epoch 45/100
 - 0s - loss: 102.8202 - val_loss: 127.5341
Epoch 46/100
 - 0s - loss: 100.4816 - val_loss: 127.1125
Epoch 47/100
 - 0s - loss: 98.2477 - val_loss: 123.4459
Epoch 48/100
 - 0s - loss: 95.4223 - val_loss: 124.6799
Epoch 49/100
 - 0s - loss: 93.1187 - val_loss: 121.6297
Epoch 50/100
 - 0s - loss: 90.5471 - val_loss: 122.3835
Epoch 51/100
 - 0s - loss: 87.3615 - val_loss: 120.5861
Epoch 52/100
 - 0s - loss: 84.6065 - val_loss: 119.3349
Epoch 53/100
 - 0s - loss: 81.3659 - val_loss: 118.8020
Epoch 54/100
 - 0s - loss: 79.0007 - val_loss: 117.7861
Epoch 55/100
 - 0s - loss: 75.9740 - val_loss: 115.6616
Epoch 56/100
 - 0s - loss: 73.8440 - val_loss: 112.9307
Epoch 57/100
 - 0s - loss: 70.9222 - val_loss: 115.6594
Epoch 58/100
 - 1s - loss: 70.3754 - val_loss: 111.7155
Epoch 59/100
 - 0s - loss: 67.2706 - val_loss: 120.7676
Epoch 60/100
 - 0s - loss: 65.2941 - val_loss: 114.9698
Epoch 61/100
 - 0s - loss: 62.9653 - val_loss: 111.5674
Epoch 62/100
 - 0s - loss: 60.8487 - val_loss: 113.5159
Epoch 63/100
 - 0s - loss: 59.0952 - val_loss: 113.1233
Epoch 64/100
 - 0s - loss: 57.6093 - val_loss: 110.2556
Epoch 65/100
 - 0s - loss: 56.1343 - val_loss: 116.0106
Epoch 66/100
 - 0s - loss: 54.6249 - val_loss: 107.2110
Epoch 67/100
 - 0s - loss: 53.9483 - val_loss: 113.1710
Epoch 68/100
 - 0s - loss: 51.9638 - val_loss: 109.1095
Epoch 69/100
 - 0s - loss: 50.9763 - val_loss: 111.7982
Epoch 70/100
 - 0s - loss: 49.8232 - val_loss: 111.7802
Epoch 71/100
 - 0s - loss: 48.5932 - val_loss: 111.0534
Epoch 72/100
 - 0s - loss: 47.5720 - val_loss: 108.7852
Epoch 73/100
 - 0s - loss: 46.5750 - val_loss: 111.1066
Epoch 74/100
 - 0s - loss: 45.7226 - val_loss: 114.6249
Epoch 75/100
 - 0s - loss: 44.6763 - val_loss: 107.0380
Epoch 76/100
 - 0s - loss: 44.2080 - val_loss: 114.5911
Epoch 77/100
 - 0s - loss: 43.3514 - val_loss: 109.2560
Epoch 78/100
 - 0s - loss: 42.7813 - val_loss: 115.7189
Epoch 79/100
 - 0s - loss: 43.3092 - val_loss: 107.3113
Epoch 80/100
 - 1s - loss: 41.5666 - val_loss: 118.3811
Epoch 81/100
 - 0s - loss: 41.1802 - val_loss: 109.7825
Epoch 82/100
 - 0s - loss: 40.7421 - val_loss: 114.9110
Epoch 83/100
 - 0s - loss: 39.9703 - val_loss: 112.9931
Epoch 84/100
 - 0s - loss: 39.4204 - val_loss: 108.0415
Epoch 85/100
 - 0s - loss: 38.7700 - val_loss: 113.8556
Epoch 86/100
 - 0s - loss: 37.9761 - val_loss: 109.1890
Epoch 87/100
 - 0s - loss: 37.6354 - val_loss: 118.5705
Epoch 88/100
 - 0s - loss: 37.6422 - val_loss: 106.1828
Epoch 89/100
 - 0s - loss: 36.7572 - val_loss: 108.9176
Epoch 90/100
 - 0s - loss: 36.2819 - val_loss: 114.6672
Epoch 91/100
 - 0s - loss: 35.8975 - val_loss: 104.2054
Epoch 92/100
 - 0s - loss: 35.3923 - val_loss: 114.1984
Epoch 93/100
 - 0s - loss: 35.0272 - val_loss: 111.6822
Epoch 94/100
 - 0s - loss: 35.9905 - val_loss: 115.8440
Epoch 95/100
 - 0s - loss: 35.2249 - val_loss: 118.9255
Epoch 96/100
 - 0s - loss: 33.9796 - val_loss: 104.2387
Epoch 97/100
 - 0s - loss: 34.4499 - val_loss: 115.6164
Epoch 98/100
 - 0s - loss: 33.2753 - val_loss: 109.0092
```

```
Epoch 99/100
 - 0s - loss: 33.1627 - val_loss: 112.8771
Epoch 100/100
 - 0s - loss: 32.7665 - val_loss: 108.5197
```

[15]:
```
<keras.callbacks.History at 0x7f4913369710>
```

You can refer to this link to learn about other functions that you can use for prediction or evaluation.

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

Thank you for completing this lab!

This notebook was created by Alex Aklson. I hope you found this lab interesting and educational. Feel free to contact me if you have any questions!

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2020-09-21	2.0	Srishti	Migrated Lab to Markdown and added to course repo in GitLab

---

This notebook is part of a course on **Coursera** called *Introduction to Deep Learning & Neural Networks with Keras*. If you accessed this notebook outside the course, you can take this course online by clicking here.
