Deep Learning or Machine Learning: That is the Question

This is a follow-up to my previous article: **Comparison of Regression Analysis Algorithms**. I applied Deep Learning and compared the results with the performances of the Machine Learning algorithms in the above-mentioned article.

In the previous article, I got quite satisfactory results using various machine learning regression algorithms in estimating the compressive strength values of concrete using 8 different parameters. Regression analysis may be defined as a type of predictive modelling technique which investigates the relationship between a dependent (target) and independent variable(s) (predictor). This technique is used for forecasting, time series modelling and finding the cause-effect relationship between the variables.

Neural networks are defined as a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. Neural networks adapt to changing input; so the network generates the best possible result without a need to redesign the output criteria.

Deep learning is defined as an artificial intelligence (AI) function that imitates the workings of the human brain in processing data and generating patterns for use in decision making. Deep learning is a subset of machine learning in artificial intelligence that has networks capable of learning unsupervised from data that is unstructured or unlabeled. Universal Approximation Theorem suggests that, neural networks can approximate any convex continuous function within a specific range. This may be one of the reasons why deep learning is gaining so much popularity in the recent years.

I used R^2 for evaluating the model performance (error metric) and it can be defined as "The coefficient of determination, R^2, is used to analyze how differences in one variable can be explained by a difference in a second variable. More specifically, R-squared gives you the percentage variation in y explained by x-variables."

A little domain knowledge; more tons of concrete are produced and used than any other technical material. This is due to its low cost and widely available raw materials (water, slag, cement...). Also, cement is the bonding material in the concrete. Concrete strength (target value) is affected by many factors, such as quality of raw materials, water/cement ratio, coarse/fine aggregate ratio, age of concrete etc.

This article involves the application of deep learning, with the aim of comparing its performance for regression analysis with 5 different machine learning algorithms.

I tried to concentrate on the following issues:

- Machine Learning or Deep Learning. Which one provided the best results?
- Effects of vvvvv parameters on the performance of the algorithm.

I used the Kaggle dataset. As explained on the website, the first seven parameters are the addition to the concrete (units in kg in a m^3 mixture and "Age" in days (1 to 365)). There are 8 quantitative input variables, and 1 quantitative output (Compressive Strength (in MPa) of the concrete); 1030 instances with no missing data.

Data Cleaning

The first step is to import and clean the data (if needed) using pandas before starting the analysis.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import train_test_split, GridSearchCV
import warnings
warnings.filterwarnings('ignore')
```

```
df = pd.read_csv('Concrete_Data.csv')
```

```
df.head(3)
```

	cement	slag	flyash	water	superplasticizer	coarseaggregate	fineaggregate	age	csMPa
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

```
df.shape
```

```
(1030, 9)
```

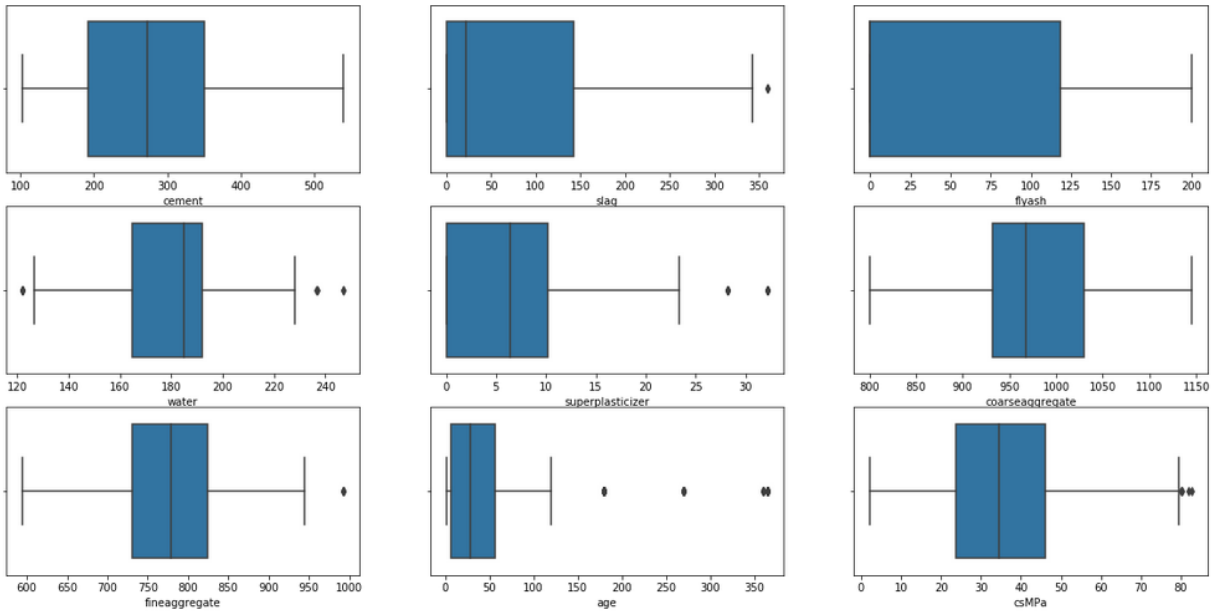
According to the dataset, there were 1030 different prepared concretes and no missing values.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1030 entries, 0 to 1029
Data columns (total 9 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   cement            1030 non-null   float64
 1   slag              1030 non-null   float64
 2   flyash            1030 non-null   float64
 3   water             1030 non-null   float64
 4   superplasticizer  1030 non-null   float64
 5   coarseaggregate   1030 non-null   float64
 6   fineaggregate     1030 non-null   float64
 7   age               1030 non-null   int64  
 8   csMPa             1030 non-null   float64
dtypes: float64(8), int64(1)
memory usage: 72.5 KB
```

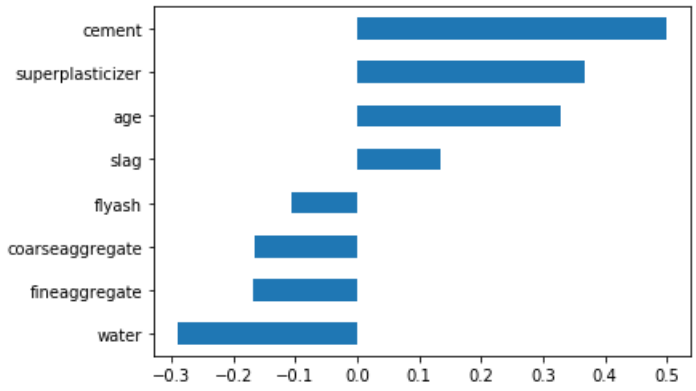
I also checked the outliers, which may cause negative effects on the performance of the analysis. The boxplots below show that there were not many outliers, so I moved to the distribution of the values.

```
fig = plt.figure(figsize=(20,20))
for col in range(len(df.columns)) :
    fig.add_subplot(6,3,col+1)
    sns.boxplot(x=df.iloc[ : , col])
plt.show()
```



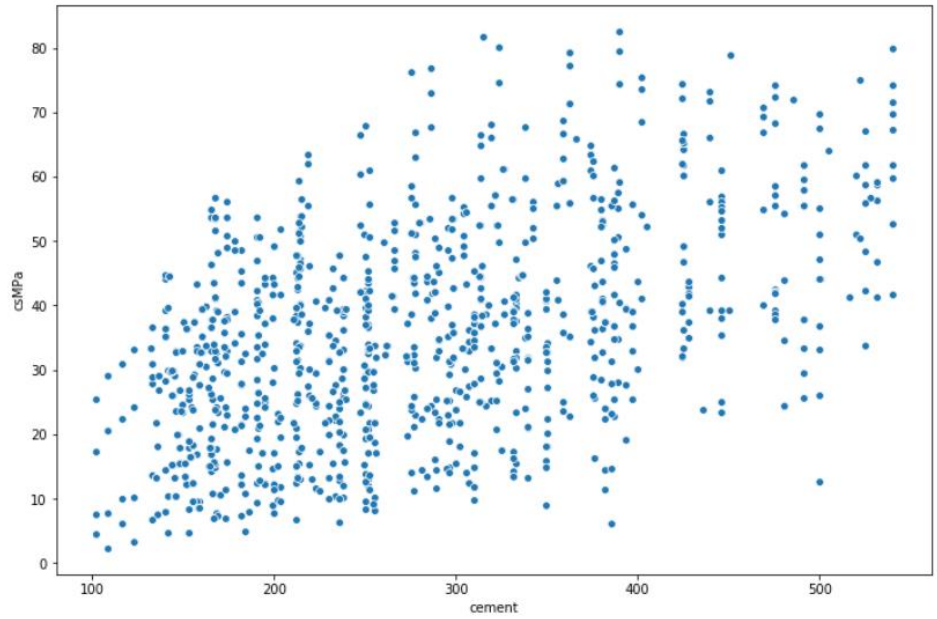
The dataset is clean (there are no NaNs, Dtypes are correct), so we will directly start by checking the correlation of the parameters with the target value – csMPa (Compressive Strength of concrete). The figure below shows that cement and superplasticizer have the highest positive correlation coefficient values and water has the highest negative correlation coefficient value.

```
df.corr()['csMPa'].sort_values().head(11)[:-1].plot.barh();
```



Here is the scatter plot of the highest correlated parameter (cement) and compressive strength (csMPa). The correlation is 0.5 and according to the graph, for cement values below kg in a m^3 mixture, the results are better.

```
plt.figure(figsize=(12,8))
sns.scatterplot(x='cement',y='csMPa',data=df);
```



Deep Learning

The first step is Train-Test-Split and then scaling the dataset.

```
from sklearn.preprocessing import MinMaxScaler
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=42)
```

```
scaler = MinMaxScaler()
```

```
X_train = scaler.fit_transform(X_train)
```

```
X_test = scaler.transform(X_test)
```

```
X_train.shape
```

```
(824, 8)
```

After that, I used Sequential model in order to build the deep learning model. Sequential model in Keras makes it easy to build the model by defining:

- Number of layers,
- Number of neurons,
- Activation functions,
- Optimizer,

- Loss function,
- Batch size and
- Number of epochs.

I tried different combinations by changing all the above parameters but got the best results with the below settings.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
from tensorflow.keras.optimizers import Adam
```

```
model = Sequential()

model.add(Dense(22, activation = 'relu'))
model.add(Dense(22, activation = 'relu'))
model.add(Dense(22, activation = 'relu'))
model.add(Dense(22, activation = 'relu'))
model.add(Dense(1))

model.compile(optimizer = 'adam', loss = 'mse')
```

```
model.fit(x = X_train, y = y_train,validation_data = (X_test, y_test), batch_size = 32, epochs = 600)
```

The dataframe shows the loss (Mean Square Error -MSE) and validation loss values for each epoch. The expectancy is to start high and then decrease to a very small value, which is a proof that the model "learned".

```
losses = pd.DataFrame(model.history.history)
```

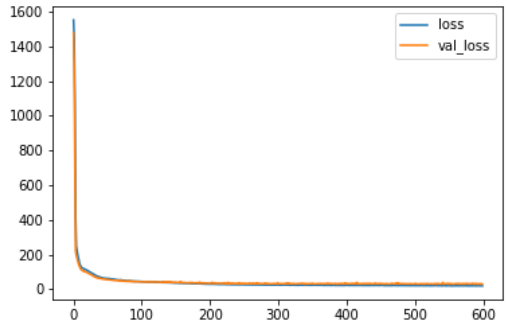
```
losses
```

	loss	val_loss
0	1552.832031	1480.111328
1	1446.979614	1286.358643
2	1097.491089	696.681274
3	426.138550	219.691177
4	249.021408	195.879700
...	...	...
595	19.408344	29.036921
596	18.919868	31.120558
597	18.501926	28.391815
598	18.306942	30.223837
599	18.508415	28.746172

600 rows × 2 columns

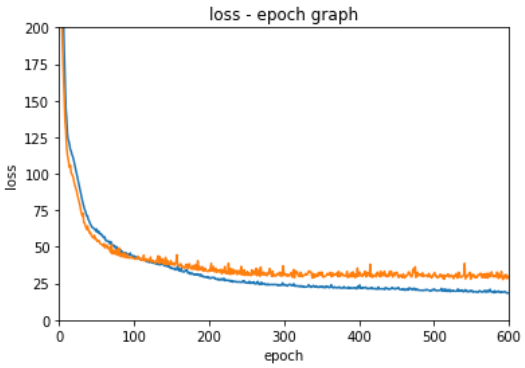
The graphs for the dataframe are shown below. The graph on the left shows that early loss values are around 1400, quickly go down to values around 50 and then stay around low 30s (for validation loss).

```
losses.plot();
```




```
fig, ax= plt.subplots()
ax.plot(losses)
ax.set_xlabel("epoch")
ax.set_ylabel("loss")
ax.set_title("loss - epoch graph")
ax.set_xlim([0,600]) # focusing on the given x values
ax.set_ylim([0,200]) # focusing on the given y values
```

```
(0, 200)
```



I used the function below in order to evaluate the error metrics and r2-score is 0.884. Explained Variance Score is quite close to that value, which was expected.

```
def eval_metrics(actual, pred):
    rmse = np.sqrt(mean_squared_error(actual, pred))
    mae = mean_absolute_error(actual, pred)
    mse = mean_squared_error(actual, pred)
    score = r2_score(actual, pred)
    return print("r2_score:", score, "\n","mae:", mae, "\n","mse:",mse, "\n","rmse:",rmse)
```

```
y_pred = model.predict(X_test)
```

```
eval_metrics(y_test, y_pred)
```

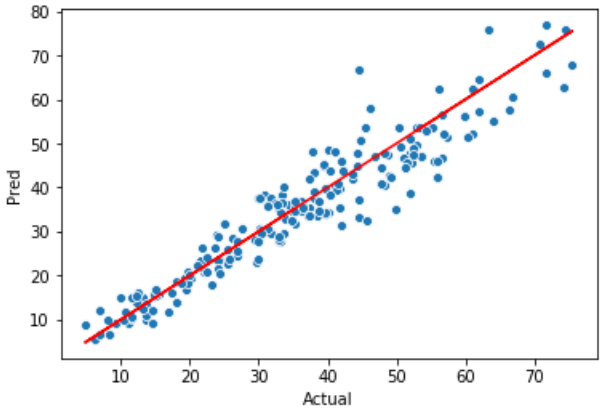
```
r2_score: 0.8884410191778757
 mae: 3.917554241661887
 mse: 28.746171716577322
 rmse: 5.361545646227151
```

```
explained_variance_score(y_test, y_pred)
```

```
0.8906829462652102
```

MAE and RMSE results are in terms of MPa (unit for the compressive strength) and they tell us that, the difference between the observed and predicted results are ± 3.92 (MAE) and ± 5.36 (RMSE) MPa. R2-score of 0.8884 is a satisfactory value and shows that even noisy, high-variability data can have a significant trend. The trend indicates that the predictor variable still provides information about the response even though data points fall further from the regression line.

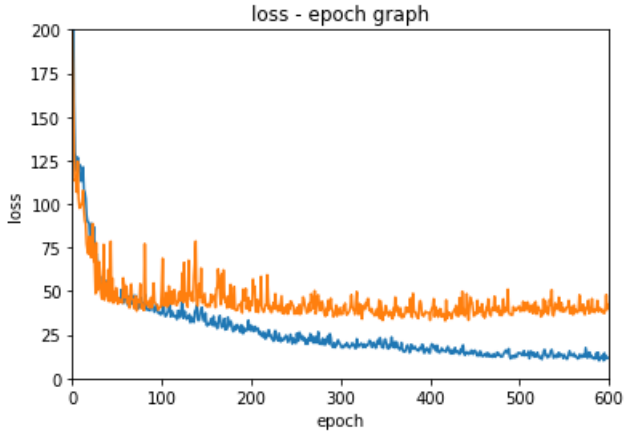
```
sns.scatterplot(x="Actual", y="Pred", data=compare)
plt.plot(y_test,y_test,'r');
```



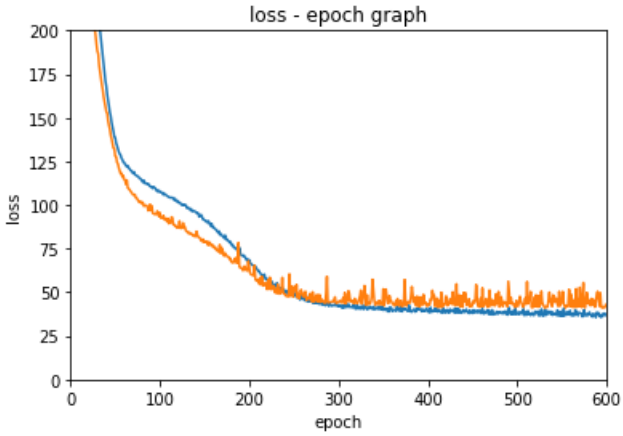
Trial & Error

I tried to change some basic settings in order to decrease the loss values, but the results either slowed down the process (although the dataset was a small one) or caused overfitting.

For example, this graph shows the loss graph when the learning rate was increased from 0.001 to 0.01:



In this graph, the optimizer was changed from “adam” to “rmsprop”.



Also, in order to avoid overfitting, I tried to use Early Stopping, but it was not successful. Validation error did not experience a considerable increase.

Finally, I decreased the batch size to 8 (starting value was 128) and the progress in the r2_score was less than 1 % and the processing time increased considerably.

Conclusion

In this article, I applied deep learning method to the same dataset that I used before for the application of 5 regression algorithms. The analysis provides evidence that:

- r2_score values ranged between 0.83 and 0.89 depending on the combination of parameters.
- I started from 2 layers-4 neurons all the way up to 4 layers and 22 neurons. The common concept is those selections cause overfitting. For the low layer-neurons, r2_score was 0.61, while for the upper values, the score was 0.89.
- Machine learning r2_scores ranged between 0.61 and 0.93, XGBoost giving the best results.
- Finally, the decision between Deep Learning and Machine Learning is analyzed in detail as:

“If deep learning is so powerful, then why don’t we dump all other machine learning algorithms in their favor? One of the biggest caveats of neural networks is the fact that they are black boxes: it is usually impossible to intuitively explain why a neural network has given a certain prediction. Sometimes, the intermediate outputs that the neurons produce can be analyzed and explained, but many times this is not the case. Other algorithms, such as traditional linear models or tree-based models like random forest, can usually be analyzed and explained better.”

You can access to this article and similar ones here.

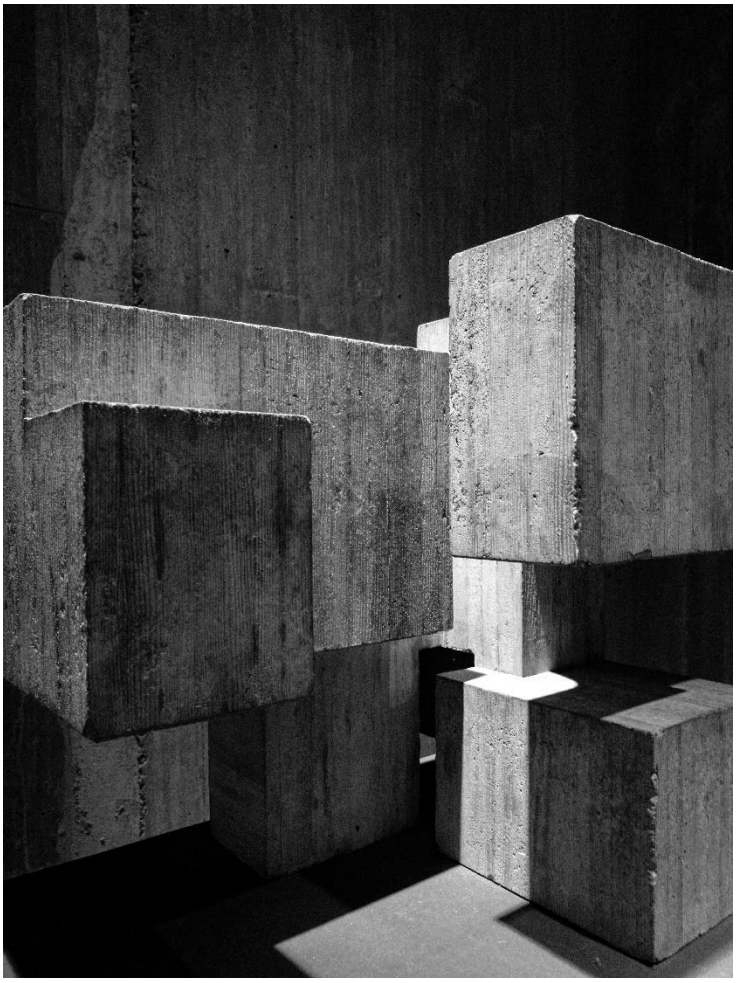


Photo by uve sanchez on Unsplash