Capstone Project

Osafa Karim

Machine Learning Engineer Nanodegree

29th March 19

Definition

Project Overview - In the era of global warming, the energy consumption is a major concern. Various attempts have been made both academically and by individuals to create solutions to predict the same with better accuracy. This attempt is to explore the power of Deep Learning to better the performance achieved by the original author of the dataset.

Reference to earlier attempts:

Academic: http://dx.doi.org/10.1016/j.enbuild.2017.01.083 https://github.com/LuisM78/Appliances-energy-prediction-data

Individual(Udacity's former student) - https://github.com/div3125/

Note : When I refer **Author** in this project report , it is **Luis Candanedo**

Problem Statement – This can be stated in in below steps:

- problem at hand is to predict the numbers as accurately as possible.
- Since the target labels are known and are numerical values it is a case of supervised regression.
- The author of the data used GBM to explain 57% variance in the test set.
- The attempt here is to use **Deep Learning** to tip the 57% variance.

Metrics - The metric used here is the root_mean_squared_error for model training, evaluation. Then the final performance is judged using the **r_squared** metric, which is common metric for regression analysis. The r_squared metric a measure for the relative fit of the regression line/curve to the data. Also known as "coefficient of determination".

RootMeanSquareError = square_root_of(mean_square_error)

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(P_i - O_i)^2}{n}}$$

R2 = 1 – (Sum_of_squares_of residual/ sum_of_squares_for_an_avg_line

$$R^2 = 1 - \frac{SS_{Regression}}{SS_{Total}}$$

Sum Squared Regression Error

Sum Squared Total Error

Analysis

Data Exploration - The below are the details of the data used for training and testing.
Data source is from the author's git , it's as follows:

Training data - https://github.com/LuisM78/Appliances-energy-prediction- data/blob/master/training.csv

Testing data - https://github.com/LuisM78/Appliances-energy-prediction- data/blob/master/testing.csv

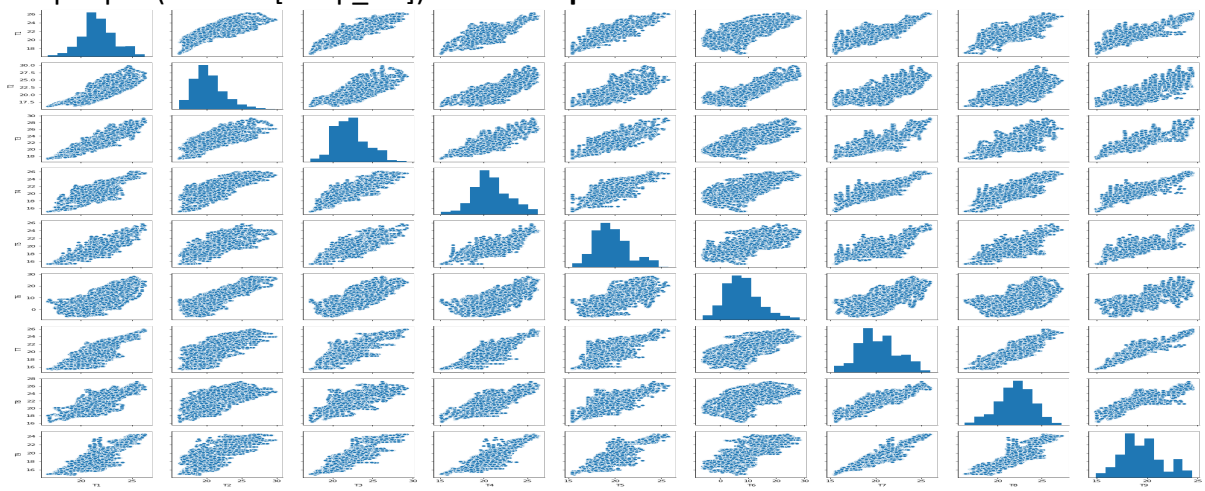
Training set –
- Feature space size – 24 features
- Target space – 1 variable
- Number of data points – 14801

Testing set –
- Feature space size – 24 features
- Target space – 1 variable
- Number of data points – 4934

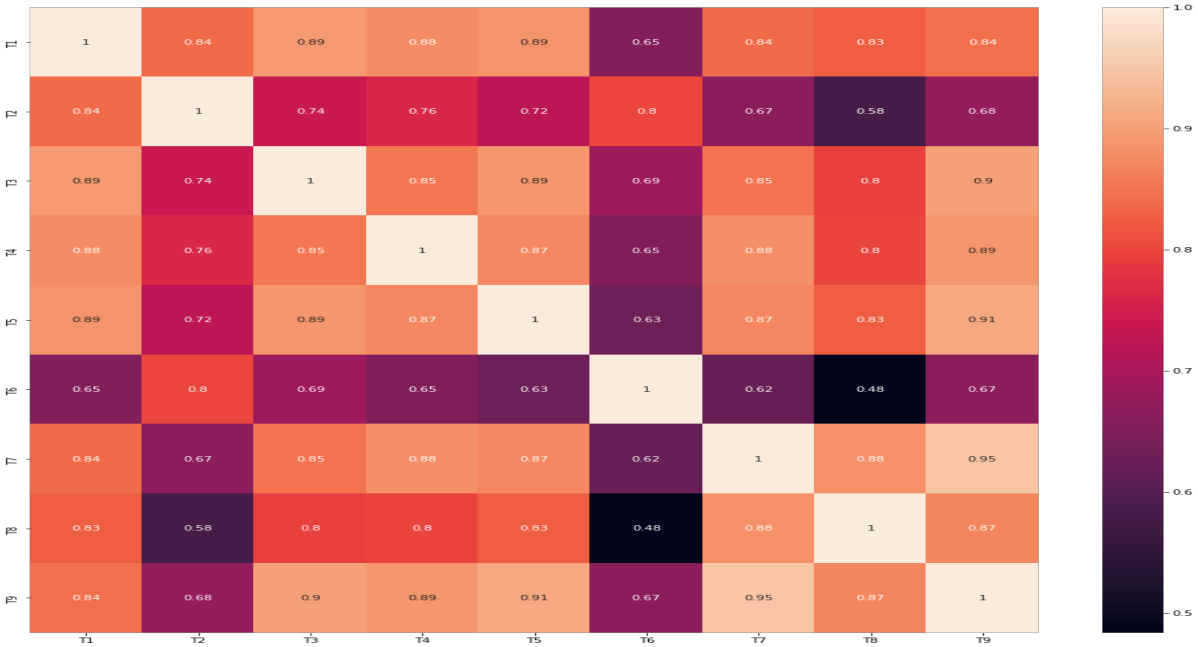
The feature space is segregated based on their categories like temperature columns, humidity columns and other weather features (wind speed, dew points, visibility etc.). Apart from the basic statics exploratory (.describe method) , I emphasized upon the correlations between various predictors which I feel is worthy of depicting here.

Exploratory visualization – Based on the above segregation pair-plots and heat-maps was generated to analyze the level of correlation among the features.

sns.pairplot(features[temp_col]) # for **temp columns**

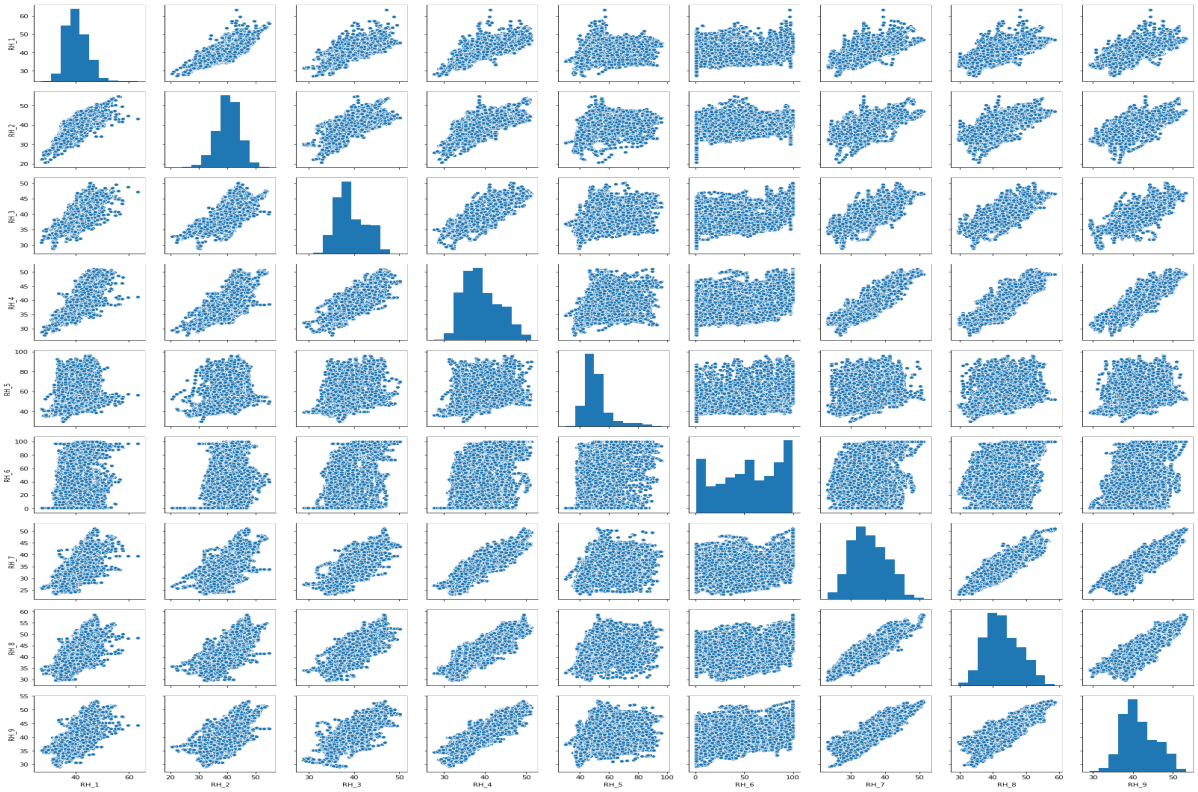


plt.figure(figsize=(17,17))
sns.heatmap(features[temp_col].corr(),annot=True)

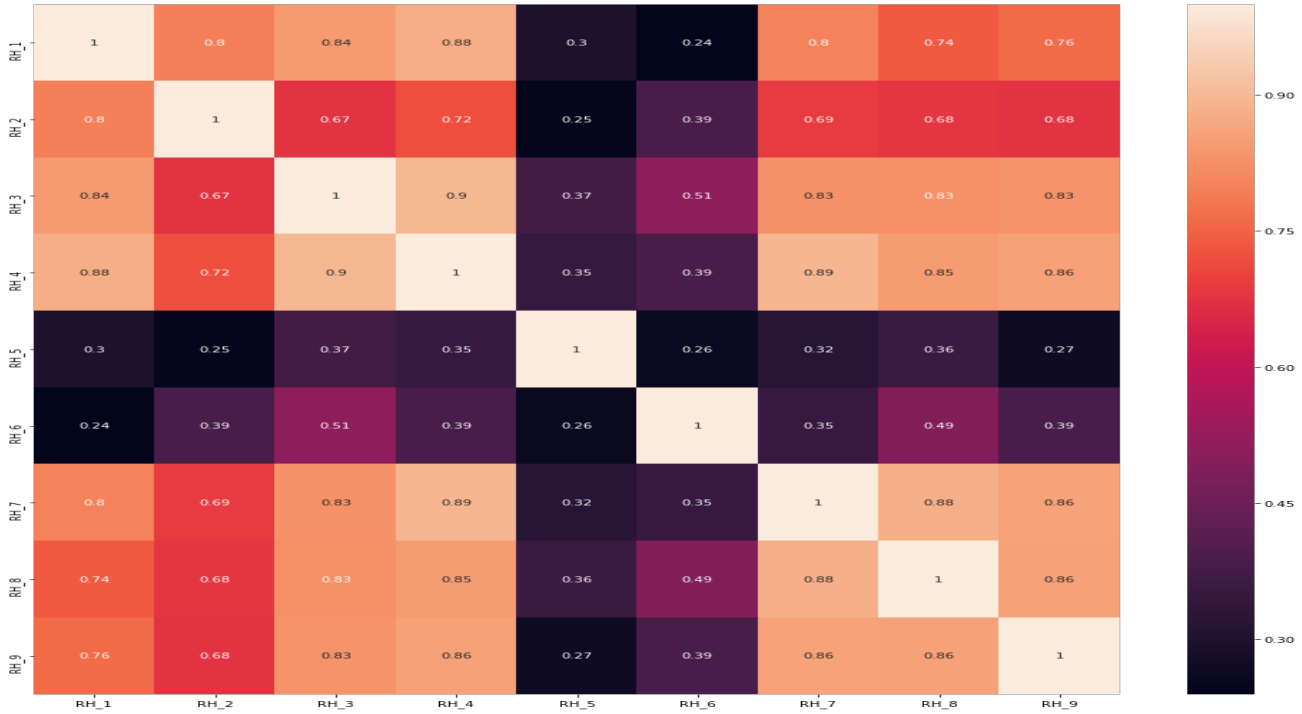


From the above pairplots and heatmap one can see that T9&T7(.95) and T9&T5(.91) have high degree of correlation and may be potential candidate for dropping while fitting the model.

sns.pairplot(features[hum_col]) **# humidity columns**



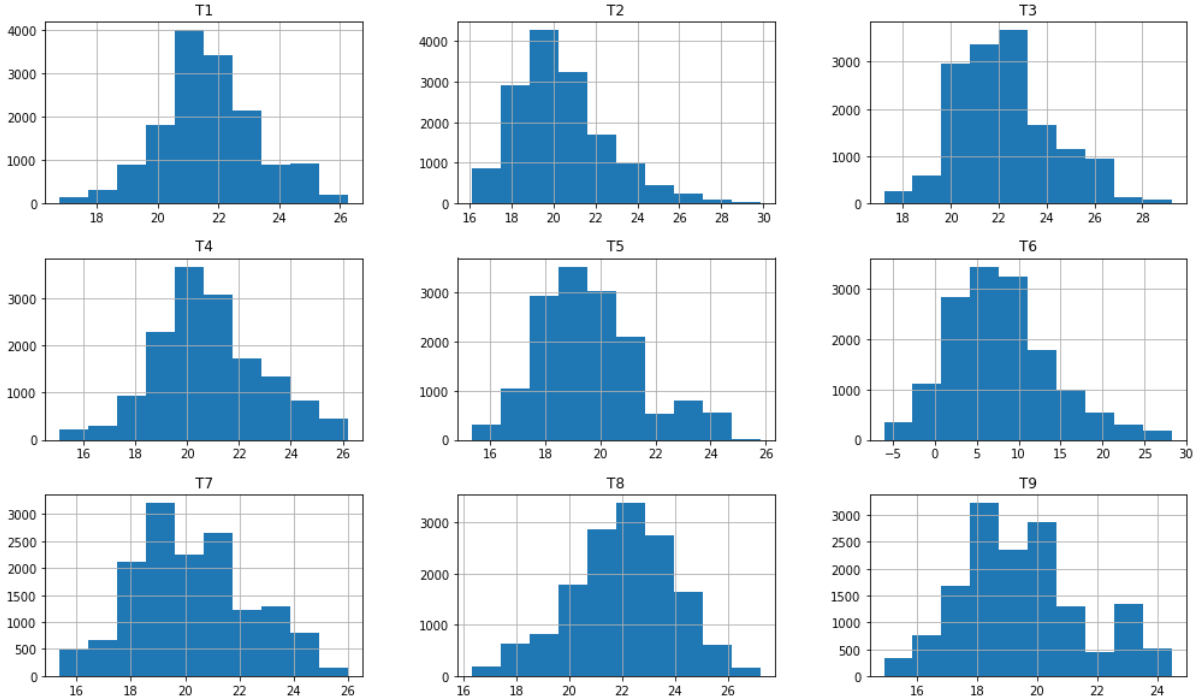
plt.figure(figsize=(17,17))
sns.heatmap(features[hum_col].corr(), annot=True)



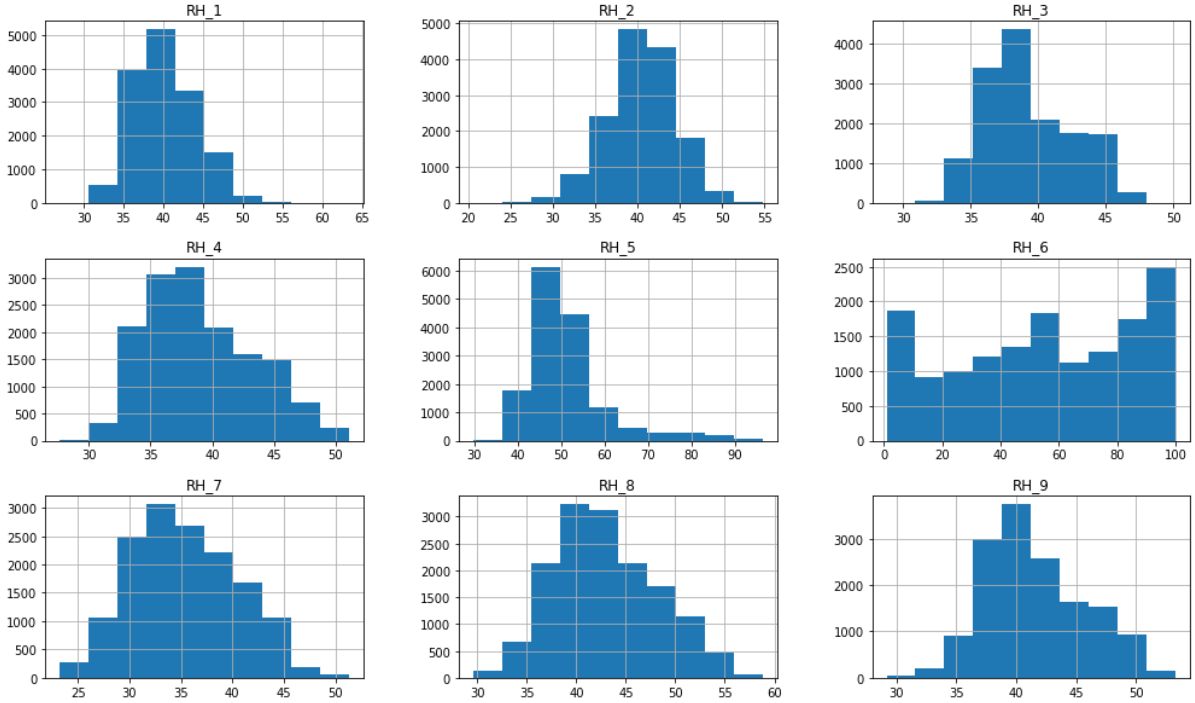
RH_3 & RH_4 (.9) show high degree of correlation among themselves. Although most of the features are correlated but not as high as .9.

Explore the **skewness** in the features as it may affect the training speed and sometimes may not reach global minima while performing the gradient descent to minimize the cost function.

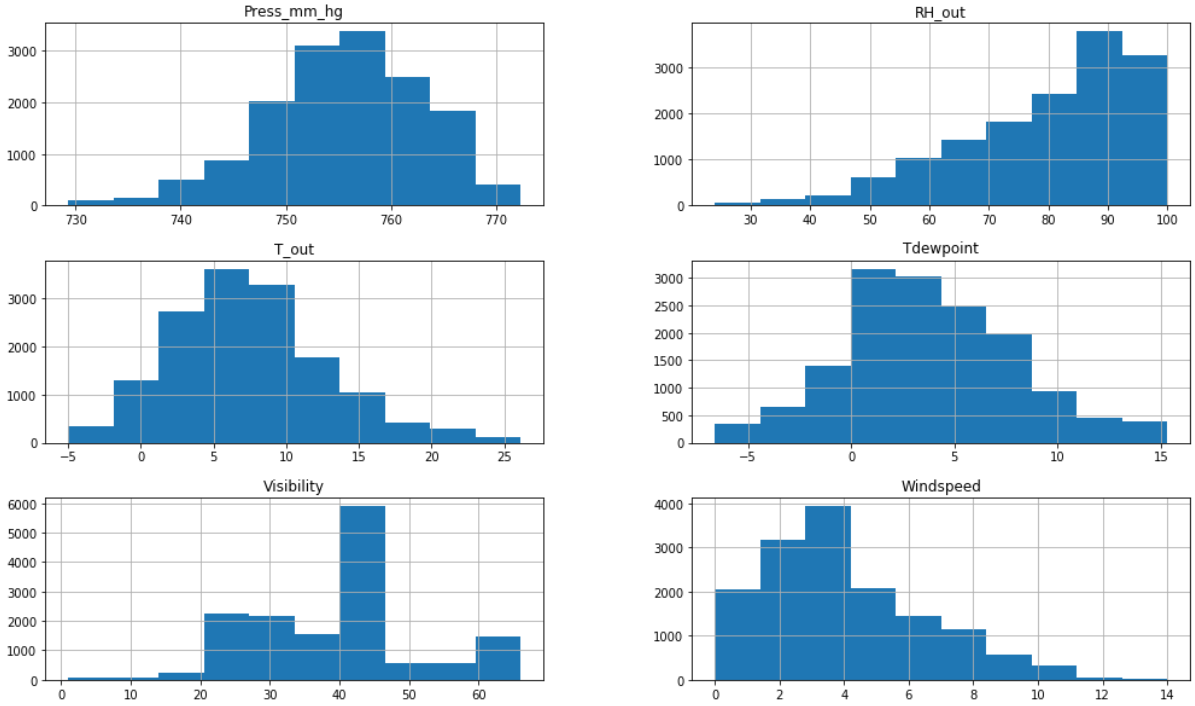
features[temp_col].hist(figsize=(17,10)) **# Temperature columns**



features[hum_col].hist(figsize=(17,10)) # **Humidity columns**



features[weather_col].hist(figsize=(17,10)) **# Weather columns**



From the above histograms most of the features are normally distributed barring **RH_out and winspeed and visibility**. In addition to this, skewness is not so high so I expect that by feature scaling skewness would be weaned out.

Algorithms and Techniques - As mentioned at the onset of this project, I used the deep neural net with multiple layers and multiple nodes in each hidden layers. Since, deep learnings are generally capable of learning many hidden patterns which classical models may not be able to discern hence explored that in this case.

Steps:

1. Tried initially with sklearn perceptron(MLP)
2. Used Gridsearch on the MLP to get a beginning point where I can start playing with the hyper parameters to fine tune it.
3. Then build the actual model using the **keras** framework.
4. This is the network shape:

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 512)               11776
_________________________________________________________________
activation_1 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 256)               131328
_________________________________________________________________
activation_2 (Activation)    (None, 256)               0
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_3 (Dense)              (None, 128)               32896
_________________________________________________________________
activation_3 (Activation)    (None, 128)               0
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_4 (Dense)              (None, 64)                8256
_________________________________________________________________
activation_4 (Activation)    (None, 64)                0
_________________________________________________________________
dropout_4 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_5 (Dense)              (None, 32)                2080
_________________________________________________________________
activation_5 (Activation)    (None, 32)                0
_________________________________________________________________
dropout_5 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 33
=================================================================
Total params: 186,369
Trainable params: 186,369
Non-trainable params: 0
_________________________________________________________________
```

5. Saved the model with the best parameters, that is, the weights where there was least validation error.
6. Trained the model with batch size of 10K as lesser size was not yield high accuracy and taking longer time.
7. Trained for 10K epochs.
8. Reduced the **learning rate** so that global minima is not missed, though speed for training time was compromised.
9. Used '**Relu'** activation function to minimize the vanishing gradient descent problem.
10. Used '**Adam'** optimizer for faster updating of the weights as compared to **SGD** or **RMSprop**. I did try them as well but was not better than 'Adam'.

Bench Mark – For this project I had setup 2 bench marks:

- First , The base Gradient Boosted Machine(regressor) without data pre-processing used by the author Luis Candanedo , which is 25 % variance on the test set.
- Second , which Luis Candanedo achieved finally , that is 57% variance on the test set.
  https://www.researchgate.net/publication/313235654_Data_driven_prediction_models_of_energy_use_of_appliances_in_a_low-energy_house

---

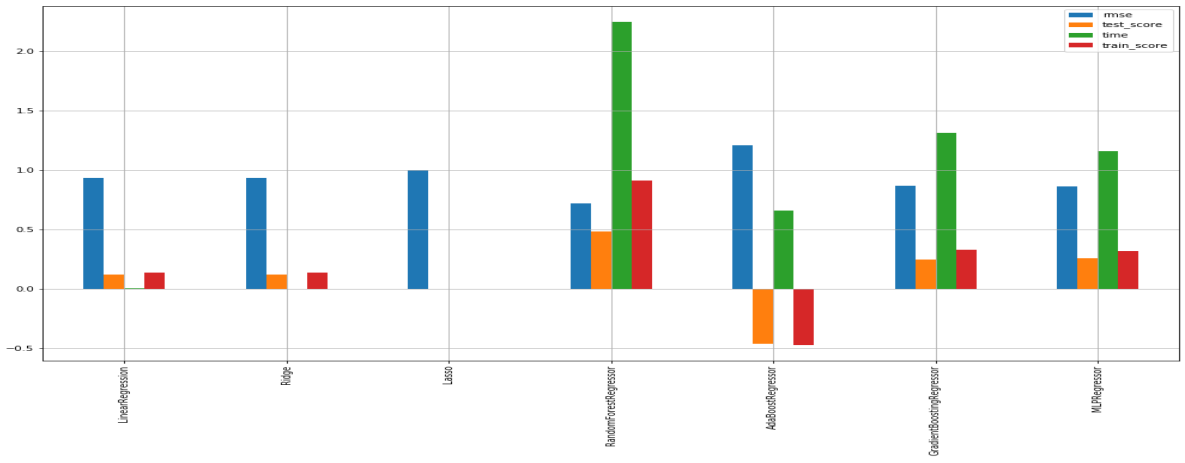
Methodology

Data Pre-processing - The data as visualized in the EDA section is mostly normal except the 3 columns highlighted earlier. However, feature scaling is import for most of the learners, since it aids in faster gradient descent. Besides, being unbiased because of the virtue of having to process a greater magnitude data.
In my case where the focus is on neural-nets it's even more pertinent to scale the features and remove any outliers and skewness. Thus have used the **standard-scaler**, where mean is subtracted and divided by the standard-deviation for each data-point respective to its features. This will make the data centered around 0 as mean. Thus 2 things are achieved:

- The data range is within -1 to + 1, removing any bias because of scale.
- The data will now follow Gaussian distribution which is the basis of most of the hypothesis on which learners are based upon.

Implementation – The followed the below steps :

1. First started with testing out a number of classical regressors like-
   Linear Regression/Ridge Regression / Lasso Regression / RF regression/ AdaBoost Regression / Gradient boosted regression and sklearn's MLP .
2. Performance of the above learners were analyzed.



3. The best learner in terms of rmse is RandomForestRegressor.

   However, RF is worst when it comes to time needed for training.

   So, to find a balance between training_time and performance either GBR or MLP are more suited.

   When GBR and MLP is compared, it's clear that one will pick MLP.

4. Now, I tried to perform some grid search on MLP and RnadomForest regresso rs.

5. With some base line idea of how many hidden layers and number of nodes pe r layer I tried a lot of different combinations.

6. Initially tried with 3 hidden layers and 250 to 10 nodes per layer.

7. Then , stayed around 5 hidden layers and gradually decreased the number of nodes from 512,256,128,64,32 and 1

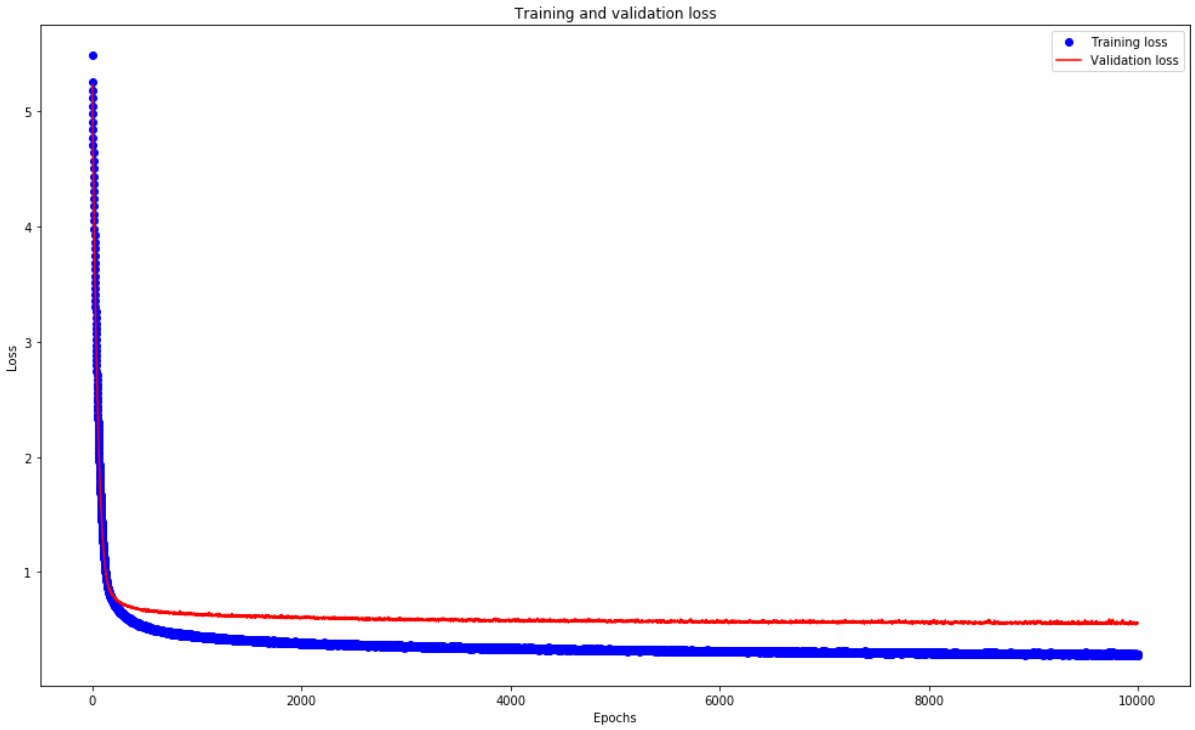
8. Used dropouts in each layers with varying proportions.

9. Used regularization techniques, firstly l1 the l2 and then a combination of bot h l1_l2 , this worked best.

10. Tried all the optimizer, but 'Adam' really worked well form my case.

Training a DL is not an easy task. It requires a lot of patience and understanding of the real problem. The challenge I encountered was right number of layers an d number of neurons. I googled a lot but, all what I discovered is there no perfec t thumb rule. However, there are some guide lines which I discovered while rea ding on medium, quora and stackexchange .

Refinement - Since, I used the gridsearch for initial search of the parameters on MLP , I didn't employ the **kerasregressor** from **keras.wrappers.scikit_learn.** Running with the kerasregressor using a kfold and cross_val was taking a lot of time with the params I was setting and the structure as I defined earlier hence I did one at a time and noted the results separately (rather manually). Used the below graph to monitor the training and validation scores for over or under-fitting. Besides, I used the test data as validation (although this not the best approach) , due to lack of more data and limit of my hardware resources I think this was the best I can do.



Results

Model Evaluation and Validation – The model is built using all the best practices for DL.

- Choosing the '**Relu'** activation instead '**sigmoid'** or '**tanh'**
- Using **Dropout** layers to reduce overfitting
- Using '**Adam'** optimizer which uses stochastic gradient descent method but in an improved manner.
- Using the **regularization** techniques, both **l1** and ∫ to reduce overfit.
- Used **callback** utility to get the best weights for the model which performed best on the train set.

Thus, while comparing against the two bench marks the new DL model has performed what it intended to.

R2 score on train data - **91.69**

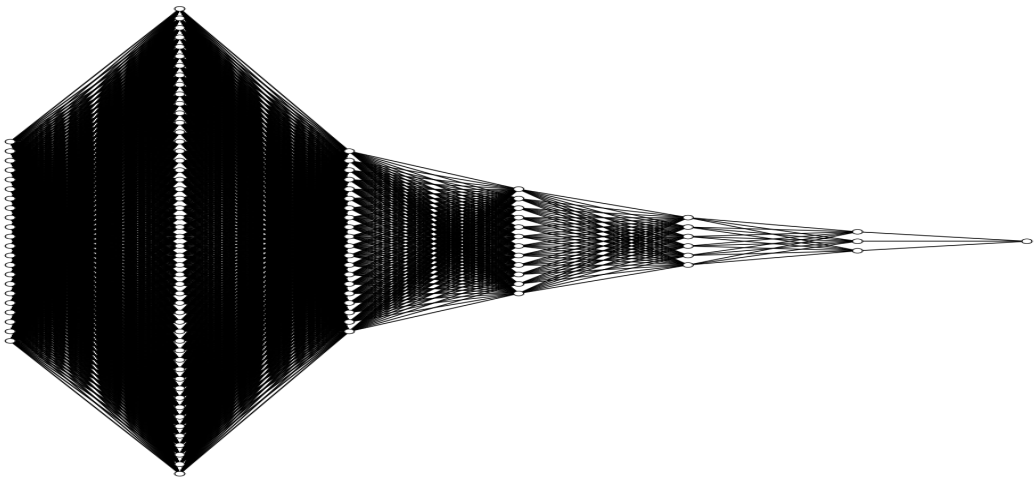
R2 score on test data – **58.67** an **increase of 1.67%** in explaining the variance as compared to bench mark of **57%** set by the author of this data set.

Justifications - The model was able tip over the 57% variance on the test data, without going to 100% on the training data (**91.69** here). This explains the robustness, that issue of high variance has been taken care. Thus also builds the confidence that such a model will perform at par with its current accuracy .

Conclusion

Free-Form Visualization - Below is the representational architecture of the neural network used in the project. The number of nodes in the hidden layers have been scaled down to nearly 10^{th} for ease of visualization.

Input layer – 22 nodes (number features after pre-processing)
First hidden layer – 516 nodes (scaled down to 50)
Second hidden layer – 256 (scaled down to 20)
Third hidden layer – 128 (scaled down to 12)
Fourth hidden layer – 64 (scaled down to 6)
Fifth hidden layer – 32 (scaled down to 3)
Output layer – 1 (without any activation as we are predicting the numerical values)



Many state-of-the-art networks like the Resnet/VGG etc. NN have many more nodes and have been trained for days and weeks. Hence, this isn't that complex as compared to them. But the truth is, to learn hidden patterns one can really go as complex a model can be and

achieve really awesome results as is for some of the pre-trained models. Time taken in my case is approx. **80 minutes**.

Reflection - The process of finding a challenging problem, finding what has been done and what can be done to better it. Combing through the problem and corresponding approaches helped learn different perspectives to solve a similar issue.

1. Searched the issues to be solved.
2. Searched the appropriate data set.
3. Make sure that the problem and data are within the scope of the timeline.
4. The issue is solvable and can produce desired results with the stipulated time.
5. Worked through the approaches already tried, to get the better understanding of the used approach.
6. Excel upon the new approach.
7. I used keras to train the deep learning models.
8. Started with sklearn's mlp regressor to get a starting point.
9. Tried with lots of different combination of hyperparameters .
10. There were 2 categories of hyperparametrs based of 'how easy it is to optimize them on a given dataset'.
    - Easy to optimize - Learning_rate, activation function, dropout ratio
    - Difficult to optimize – epochs, batch_size
11. Ran training and validation plots to visualize the high variance effects due more epochs.
12. Used call-backs utility within keras to get the best weights for the nn. Since the training was noisy it was difficult to ascertain which epoch will give the best weights by just eye-balling. Even though I had the train/validation plot.
13. Load the best weights (where the mean-squared-error is minimum for validation set) to the network and predict the target values for test-feature space.

Tuning the NN was one of the most difficult aspects in the whole process. It required to comb through a lot of variables and try them, which took major chunk of the time spent for the whole project.

Improvement – To improve the results, i believe these are the steps to be followed:

- Get more data, I had only approx. 14K observations.
- I trained on CPU, better hardware is always a best bet for NN. That is when one can fully realize the true potential of the NNs.
- Using the GPU , one could employ the keras wrapper (kerasregressor) for kfold and cross_val , try to find the best values . Doing this on CPU with n_jobs was not possible as I would take days to try different combinations. Hence , proper utilization of the resources like AWS GPU and others should always be considered.