ASHRAE ENERGY PREDICTOR USING NEURAL NETWORKS

-Hemanta Ingle

-Siddhesh Gaiki

INDEX-

I. INTRODUCTION-

Problem statement-Increasing power consumption means increase in the cost and loss of money, when it comes to the power consumption in a residential or a commercial building use of electricity is inevitable but there are means of reducing the usage of power by intelligent technologies. Power saving achieved can only be tested when you actually implement [3]the new power saving techniques but is there any way that we can estimate the power savings before the power saving technique has already been implemented? Or is there a way we can predict the power consumption without the power saving technique being implemented and compare it with the model when power saving is implemented, the answer is yes, we can use algorithms from like Machine learning and Artificial intelligence by which we can predict the power. This is our aim in this project.

Motivation/Challenge- the 1st motivation comes from the fact that it is going to save power and hence save the environment. The second is the when better estimates of power saving are predicted then investors will come forth in investing in this area.

Current Methods- there are current methods of estimating power but they are fragmented means that they are not on a uniform scale so a fair comparison is not possible

Possible Solution-we have now successfully identified the problem and also have come up with an idea to use Neural networks for power prediction. [4]Algorithms can be developed that make accurate predictions for consumed power by training the model on basis of data available from more than 1000 buildings collected over a span of 3 years and also considering the weather.

Evaluation metric –
Evaluation metric determines the performance of a particular model and allows us to choose a perfect model with the best performance so for this competition the metric is RMSLE (Root mean square logarithmic error) that is a simple log extension of a RMS (Root Mean Square) – as the data is too large an so are the differences in order that a proper scale is maintained the metric is evaluated on a Logarithmic scale.

$$\epsilon = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(p_i+1) - \log(a_i+1))^2}$$

Where:

ϵ is the RMSLE value (score)
n is the total number of observations in the (public/private) data set,
p_i is your prediction of target, and
a_i is the actual target for i.
$\log(x)$ is the natural logarithm of x

Note that not all rows will necessarily be scored.

II. DATASET DESCRIPTION-

A model that can predict the energy consumption without using the energy efficient techniques is difficult but not without the abundance of data[3] (over 1000 buildings and over 3 years of time) inputs available to us for training our Neural network Model. So, the data description is as follows, there are 6 excel files available to us namely.

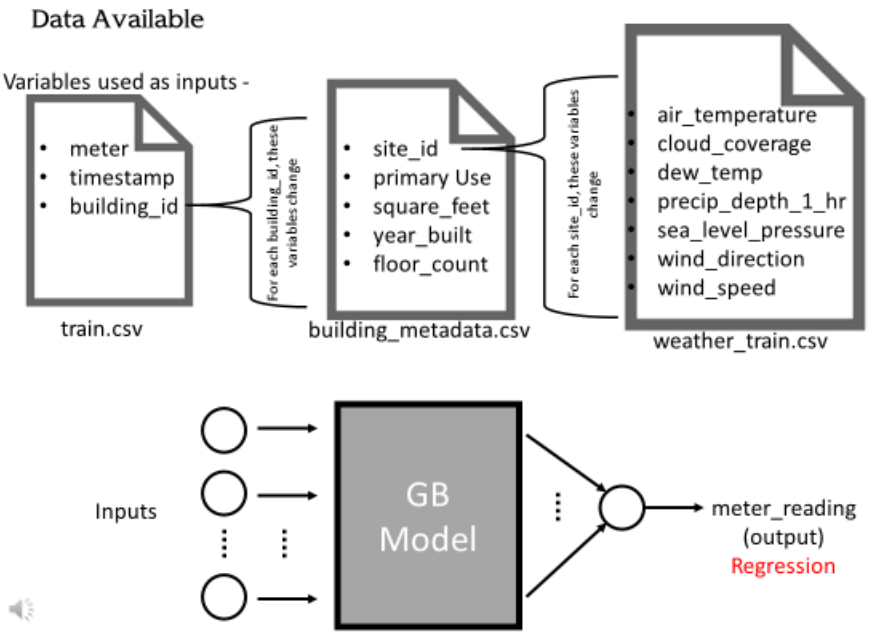

Fig 2.1- The Dataset

1) Building_metadata.csv –
   a) Site_id
   b) Building_id
   c) Primary_use
   d) Square_feet
   e) Year_built
   f) Floor_count

2) Test.csv
   a) Row_id
   b) Building _id
   c) Meter (0,1,2,3)
   d) Timestamp

3) Train.csv
   a) Building_id
   b) Meter(0,1,2,3)
   c) Timestamp
   d) Meter_reading

4) Weather_test.csv
   a) Site_id
   b) Timestamp
   c) Air_temperature
   d) Cloud_coverage
   e) Dew_temperature
   f) Precip_depth-1-hr
   g) Sea_leavel_pressure
   h) Wind_direction
   i) Wind_pressure

5) Weather_train.csv
   a) Site_id
   b) Timestamp
   c) Air_temperature
   d) Cloud_coverage
   e) Dew_temperature
   f) Precip_depth-1-hr
   g) Sea_leavel_pressure
   h) Wind_direction
   i) Wind_pressure

Figure 2.2 shows the data sets available, there are 6 excel files available out of which the 5 are of main concern namely

1) Building_metadata.csv- Contains the information about the building

Stats	Inputs					
	Site_id	Building_id	primary_use	Square_feet	year_built	Floor_count
Total data Points	1449	1449	1449	1449	675	1449
Min.	0	0	NA	283k	1900	355
Max.	15	1448	NA	875k	2007	1
Mean	6.95	724	NA	92.1k	1970	26
Missing data	0	0	0	0	774	1094

Table 2.1- Building_metadata.csv

2) Test.csv - Contains row ids which are used to check the results and compare for accuracy and errors

Stats	row_id	building_id	meter	timestamp
Total data Points	1048576	1449	4	41.7m
Min.	0	0	0	31 Dec 16
Max.	1048576	1448	4	31 Dec 18
Mean		724	NA	31 Dec 17
Missing data	0	0	0	0

Table 2.2-Test.csv

3) Weather_test.csv-[4] contains the data for the weather of the particular location of the site which is given by the key site_id from the building_metadata.csv and the site id is obtained from the building_id from test.csv

Stats	Inputs								
	Site_id	Timestamp	Air_temperature	Cloud_coverage	Due_temperature	Prcip_depth-1-hr	Sea_level_pressure	Wind_direction	Wind_pressure
Total data Points	277k	277k	277k	137k	277k	182k	256k	265k	277k
Min.	0	31Dec16	-28	0	-31.6	-1	972	0	0
Max.	15	31Dec18	48.3	9	26.7	597	1.05k	360	24.2
Mean	7.48	31Dec17	14.3	2.19	7.49	1.1	1.02k	180	3.55
Missing data	0	0	0	140k	0	95.6k	21.3k	12.4k	0

Table 2.3-Weather_test.csv

We can observe that there is a link between the datasets, the test/ train data set gives us building id that makes us enable to get the data from a particular building like its primary_use , square_feet, year_built ,floor_count . the Building_metadata contains site id which then takes in formation from the weather_test/train.csv giving us the weather data

Test.csv /Train.csv ➡ Building_metadata.csv ➡ Weather_test/train.csv

Serial no.	ID	Energy types to predict
1	0	Electricity
2	1	Chilled Water
3	2	Steam
4	3	Hot water

Table-2.4- types of meters

The above figure shows us the types of meters and their associated ids , note that all buildings do not have all kinds of meters and this also contributes as a challenge for us to build a model that predicts a output of a particular type of meter.

III. PREPROCESSING-

A data preprocessing is a very essential step in any Neural network or machine Learning problem because the data available is in all formats and unfortunately[5] there aren't any such models built that can handle all types of data sets for example some of the data can be binary in case of a electronic sensor like a binary encoder. The data may require preprocessing if there are words instead of a numerical value as the models can only process or rather train on the numerical values as inputs may it be an image or any timeseries data, audio, video etc.

Our data set was described in the previous section, its overall structure has also been mentioned above. We will go one by one over the datasets available and mention the required data pre- processing

a) **Building _metadata.csv-** this file has 6 columns out of which the column containing the "primary_use" information is non numerical so that has to be converted into numerical format.
Primary-use is a categorical value and thus it has 16 different categories, we plot the primary use and then simply map the values as they are. A zip() function is used to bind the values of the keys associated with the use value by sorting the data and then creating a dictionary "use_map".
There are 774 values of year-built input that are missing and 1094 values of floor count

b) **Train.csv-** this file contains time stamp in the format month-date-year format so this data must be converted into a proper integer format. The meter reading values are too large and are scaled by taking log and using the following code .
"train_data['meter_reading'] = np.log1p(train_data['meter_reading'])"
The time stamp is converted into new features that are kind of augmented and those are month, day , year and hour. We also created a new feature of holidays that is called weekend comprising of Saturday and Sunday.

c) **Weather_train.csv**- this file also contains time stamp that has to be changes to a proper format.
Additionally, 140k values of cloud coverage, 95.6k of percip_depth_1_hr ,21.3k of sea_leavel_pressure and 12.4k of wind direction are missing. Using "imp_most.fit_transform"
will fill in the missing data with the the most frequently occurring data. the wind direction is a angle of the wind blowing so we convert it into a proper format by multiplying it by 2*pi and dividing it by 360.
The weather data is having four different input parameters namely "air_temperature", "cloud_coverage", "sea_level_pressure", "wind_speed" we normalize the data by taking the meaa and dividing it by the standard deviation.

We also merge the data the train data and the building data are merged on basis of the timestamp, whereas the time stamp is sequentially sorted similarly the train data and the train weather are also joined together in accordance to a standard timeframe

Test -Train Splitting- training size =202161 test size=202161.

Splitting of data into training and test is an essential part of building any Neural network model , initially the training size was chosen to be 0.01 of the whole data and rest for the test but as we tried to train the model it tool about 6 hours for training over one epoch and the value of RMSLE was found to be 4.45 which was not that good considering whole data for training and test . so we tried to take a small portion of the data and

IV. BASELINE MODEL-

Please note

-Due to limiting computing resources and Kaggle kernel switching off frequently, I am choosing a small portion of the training data and trying to tweak the model performance.

-Update- later I switched on Jupyter notebook and ran the whole LSTM models, results are in conclusion

-A full data training is done also on Light GBM, and Neural network model using Embeddings whose results were mentioned in the conclusion

- The baseline model now I have chosen is a LSTM Recurrent Neural network model.

-Model Chosen- LSTM (Long Short-Term Memory) Recurrent Neural Network.

-Reason- LSTMs are well known for the characteristics of performing excellent of the time series data and our data is over a time series.

-Identified Problem type- Regression, as the output is a possibility of integer values.
Information storage was never a thing until LSTM (Long Short Term Memory) Neural Network models came in existence- a combination of a Fully Connected Feedforward Neural Network and a memory storing device gives us LSTM .it is faster and more efficient than the conventional Recurrent neural Networks (RNN) the LSTM is a gradient based learning algorithm that learns the weights through back propagation . constant error carousels in the special units reduce the gradient where its possible in turn reducing the lag in the discrete time steps by constant error flow. Exploding and vanishing problems are eliminated by this type of algorithm.

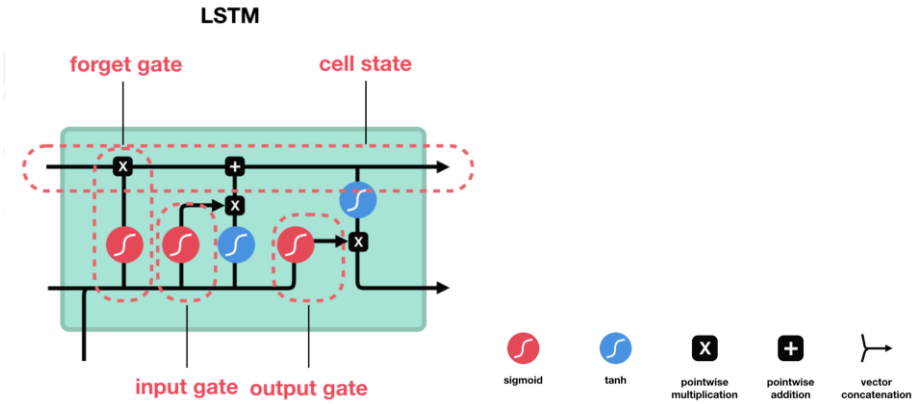

Fig. 4.1 inside a LSTM cell – (Pic courtesy- https://towardsdatascience.com/)

The LSTM cell is shown in above fig has self-feedback loops (inner recurrence) within itself that provide a time delay (Note- they also have outer recurrence loops), remember the weights that are essential up to a certain duration. this cell also has a forget block inside that erases the trivial weights that are no longer required. in short, each cell has same parameters, input and output as in an RNN, but has some parameters and gating units that control the path.

Gates in LSTM- Input gate, forget gate (what to throw away) , output gate , the whole idea behind LSTM is to create self-loops to produce paths so that gradient can flow a long duration.

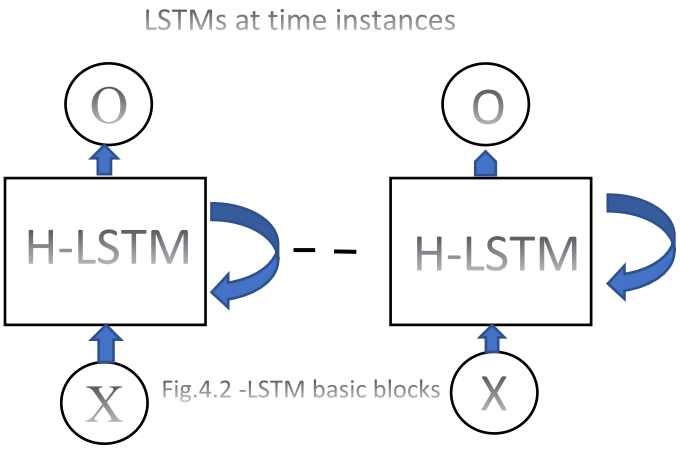

Fig.4.2 -LSTM basic blocks

In the above fig. 4.2 we can see that there are three basic building blocks in a LSTM neural network model, here

X- Inputs at a particular time instance, H-Hidden layer(LSTM block) and O- output layer, the three layers process data as it comes in an time series manner , the input layers receives input the hidden layer processes that data and an output is generated , in our case the inputs are the rows generated out of the datasets

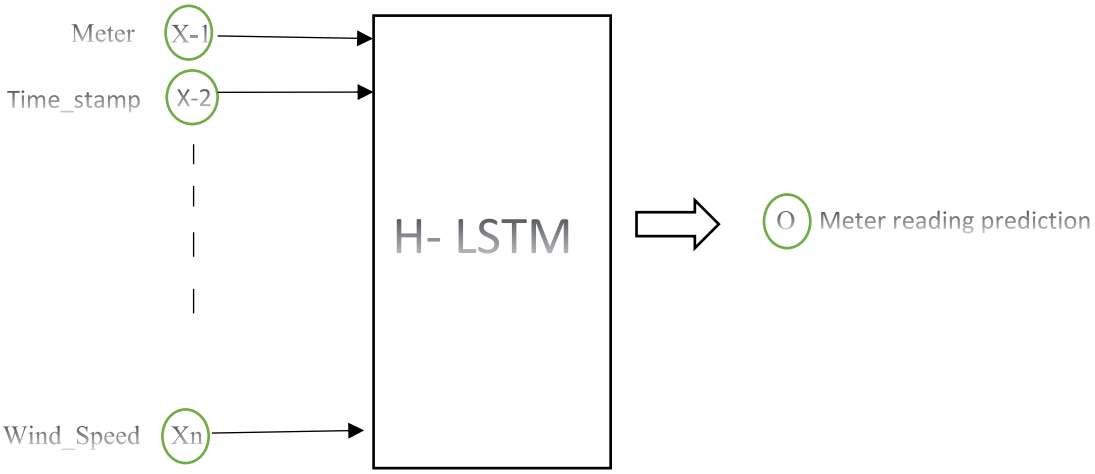

Fig.4.3- LSTM at a particular time instance, having inputs at a time instance.

Problem of huge data training- the training data is 202216100 data points, and that's a huge number, so a data optimizing technique is to be utilized for the same. generating datasets on multiple cores and providing them to our model can be done by a data generator that is based on keras.

Use of RNN- we create a [3] very basic LSTM RNN for our baseline model to see the performance and th en we will increase the performance by switching to a larger and more efficient neural network model. A s equential model is defined model = Sequential () which signifies a fully connected feed forward neural net work but now we add model. add (LSTM(units=8, activation = 'relu', input_shape=(1, input_dim))) which signifies a LSTM model with 8 units and the activation function as RELU , we also add a dense layer with a activation function as RELU.

Performance metric- RMSLE is added as a performance metric that provides an RMS error value but in our case the error values tend to be too large making the predictions unstable and so they are scaled by converting into a log scale so we have considered RMSLE (Root Mean Square Logarithmic Error) over the conventional RMSE.

Model 1 -Sequential			
	Layer Type	**Output shape**	**# Parameters**
1	LSTM	(None, 8)	1056
2	DENSE	(None,1)	9
Total			1065

Table.4.1- The overall summary of the network.

The table 4.1 shows the summary of the Model , The layer one is a LSTM layer with 1056 parameters and all are trainable .the second layer is a dense neural network layer with 9 parameters . a total of 1065 parameters are trainable.

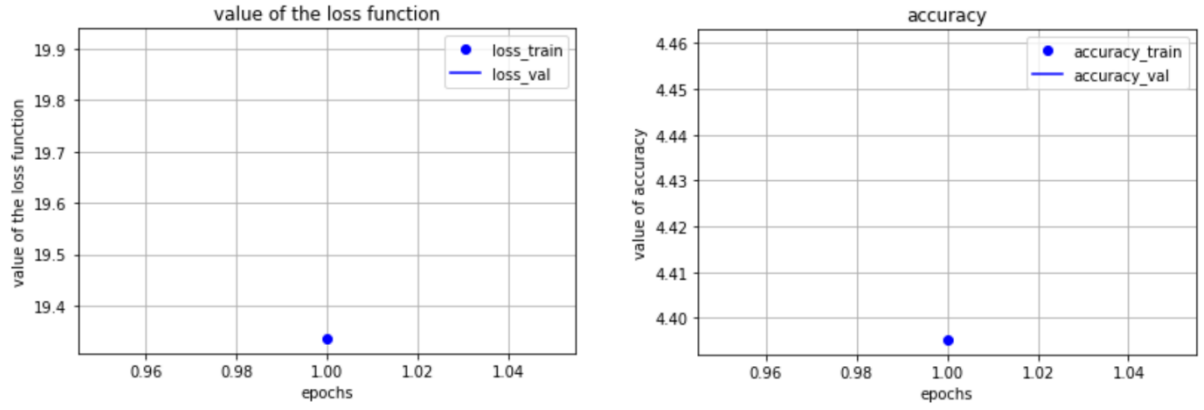

Fig. 4.4- Accuracy and loss plotted against the number of epochs

```
Epoch 1/1
197/197 [==============================] - 9s 44ms/step - loss: 19.3362 - rmse: 4.3953 - val_loss: 19.9107 - val_rmse: 4.459
7
```

the results for the baseline model. Training RMSLE= 4.39 and Test RMSLE of 4.459, by taking 202161 data points for training and similar for test as well out of a total of 20206100 training dataset available

V. MODEL IMPROVEMENTS-

Identified problem- 1) from the results in the baseline model we can see that there is a slight over fitting where the training error is lower than the test error
2) Overall performance is not good it should be ideally RMSLE< 1, but we have it high
3) Computation takes a lot of time
4) Kaggle accepts the submission files having rows=41697600

Possible Solutions-

1) **Increasing input data size-**with the problems successfully identified, we came up with some solutions. to solve the bad performance problem, we increase the training and test dataset to a full dataset of 20216100 data points. increasing the data size also solves the problem of submission to the Kaggle website for evaluation.

For optimizing the data, we use Adam optimizer that

2) **Using an optimizer in the model-** Adam optimizer is used to make the deep learning Neural network work faster by finding out best learning rates for each parameter, it performs good on sparse gradients and not so good on non-convex optimizations of neural networks with Adagrad. we try using RMSProp that solves the problems associated with the Adagrad. Adam is a coalesced version of both RMSprop and the Stochastic gradient descent.

3) **Changing the number of Epochs-**While this seems to be a good solution but we have observed that a single epoch takes about 4 hours for training on the full dataset, so we are skipping this technique for now.

4) **Removing the outliers –** observing the irrelevant data, and making significant data available to the training.

5) **Preprocessing -**Handling the missing data of building and weather data, by appending a average value to the missing value. missing data create a disconnected connection in neurons and hence the overall efficiency of the model decreases, so to handle this problem we are preprocessing the data.

6) **Creating new features-** new features like month, day, year and hour are created, creation of these features gives the model more pre-information so that it takes less time to perform feature extraction and be more accurate

7) **Future development-**our model doesn't perform excellent so, we decided to choose many such working models and then creating a ensemble that could perform very well. (note- this is a part of future development)

VI. DISCUSSIONS AND CONCLUSIONS –

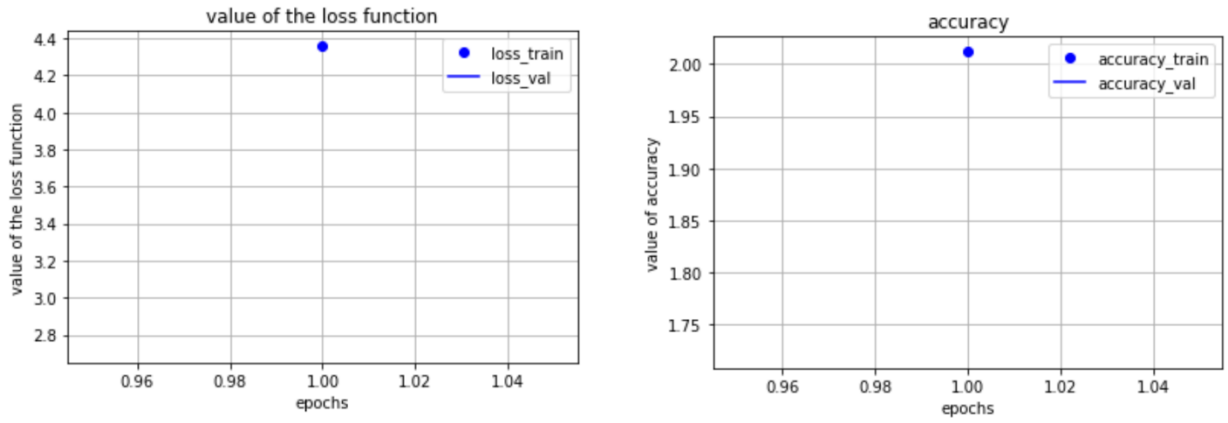

Fig 6.1-A short analysis of the train results.

the results for the improved model. Training RMSLE= 2.012 and Test RMSLE of 2.73, by taking 15162075 data points for training and 5054025 for test as well out of a total of 20206100 training dataset available.

```
Epoch 1/1
14806/14806 [==============================] - 24118s 2s/step - loss: 4.3577 - rmse: 2.0120 - v
al_loss: 2.7315 - val_rmse: 1.7224
```

Fig 6.2- Results with full data of 20216100 data points.

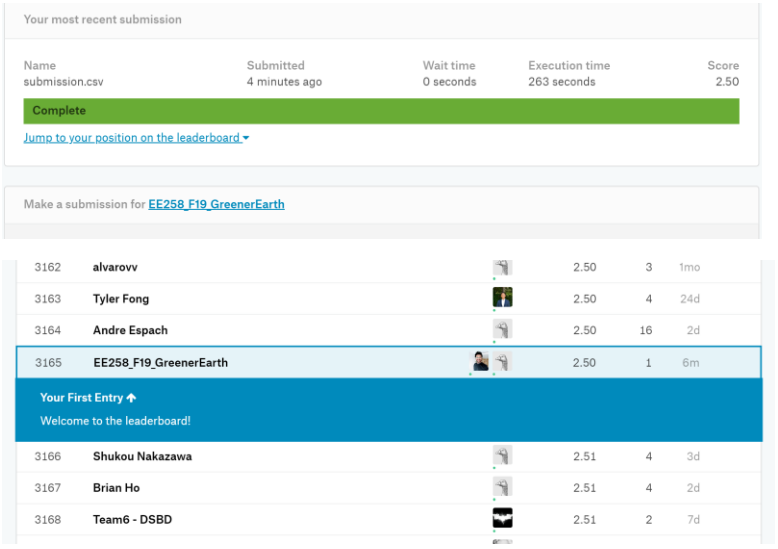

Fig6.3 -Results for LSTM method (Project report is based on this method

NOTE- With a aim to create an ensemble in future, We also tried other methods which are not mentioned in this report in detail, we took existing models as reference and built a model by tweaking some of the parameters the results of that models are shown below

Results for Embedding method-

```
Epoch 00008: val_root_mean_squared_error did not improve from 0.97769
Epoch 9/25
14833419/14833419 [==============================] - 151s 10us/step - loss: 1.1048 - root_mean_squared_error: 1.0
495 - val_loss: 1.0162 - val_root_mean_squared_error: 0.9861

Epoch 00009: val_root_mean_squared_error did not improve from 0.97769
Epoch 00009: early stopping
**************************************************
```

Fig. 6.4 – Embeddings method- RMSE- Training=1.04, Validation=0.9861

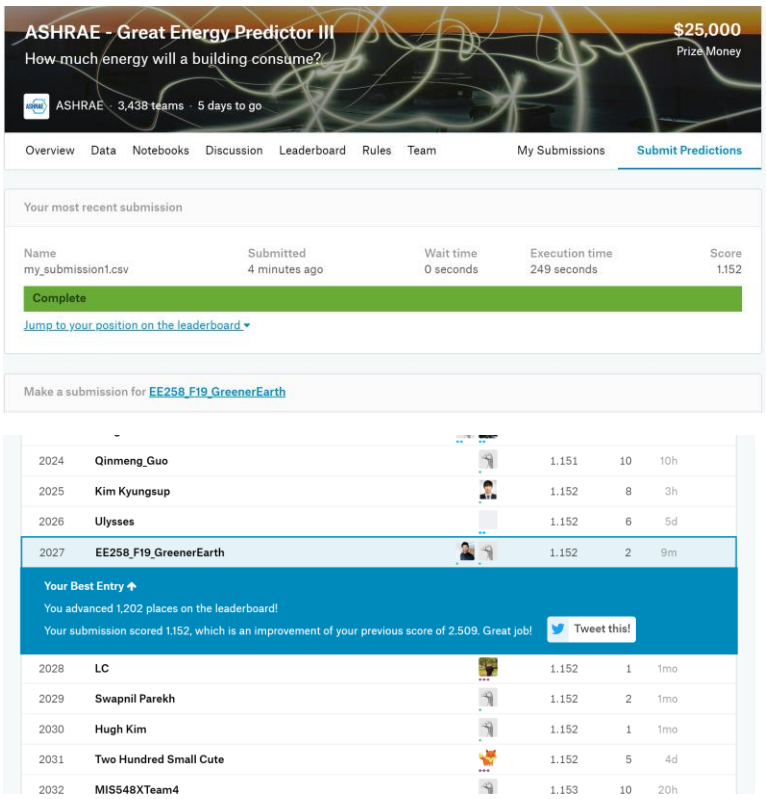

Fig6.5 -Results for Embedding Method

```
Training until validation scores don't improve for 200 rounds
[200]   training's rmse: 0.90602        valid_1's rmse: 1.50683
[400]   training's rmse: 0.860636       valid_1's rmse: 1.50277
[600]   training's rmse: 0.835886       valid_1's rmse: 1.50041
[800]   training's rmse: 0.819647       valid_1's rmse: 1.50033
Early stopping, best iteration is:
[664]   training's rmse: 0.830981       valid_1's rmse: 1.49965
```

Fig. 6.6 – LGBM method- RMSE- Training=0.83, Validation=01.49

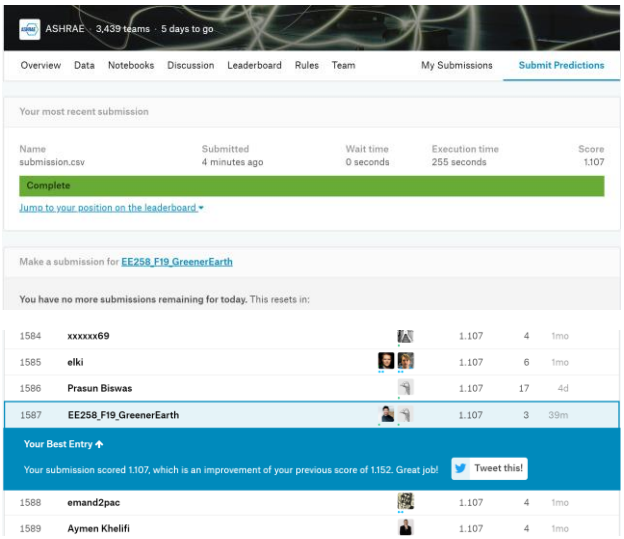

Fig 6.7 -Best results with A light GBM Model

NAME OF MODEL	Training RMSLE	Test RMSLE	SCORE	RANK
LSTM	**2.01**	**1.722**	**2.50**	**3165**
EMBEDDINGS NN	**1.01**	**0.98**	**1.152**	**2027**
Light- GBM	**1.49**	**0.83**	**1.107**	**1587**

Table-6.1 – Overall comparison of the networks

Conclusion- The given problem is a continuous time series regression problem so a Recurrent Neural Network is a preferred choice however its an interesting finding that Light GBM which is a tree based learning algorithm performs very well on our dataset.The data set is very large and so high accuracy can be obtained but provided that a good selection of model is made. With huge dataset comes the problem of preprocessing the data so we now know the importance of preprocessing the data before jumping into creating a model. The training time increases with such a huge data so some memory optimizers can be used. The problem of overfitting is observed and hence we can always use any one of the regularization techniques. In future improvement we are going to create possible models and then combine them together towards prediction of the output, this method is also called as ensemble. Kaggle is a good platform for solving complex Neural Network and Machine Learning problems as many data science enthusiasts can share their work and create a good model but we observed a problem of kernel abruptly stopping and the whole model needed to be run again, this caused a lot of delays in creation of the model .we used Jupyter notebook instead of the Kaggle kernel to save our time and overcome the difficulties faced on the Kaggle kernel. There is good scope of improvement in the RMSLE and that can be done by analyzing the different submissions done on the Kaggle website, creating a model based on all the good models and combining best of the models to get a improved performance.

REFERENCES-

[1]Hochreiter, S. (2019). *Long Short Term Memory*. [online] Bioinf.jku.at. Available at: https://www.bioinf.jku.at/publications/older/2604.pdf [Accessed 11 Dec. 2019].

[2] Medium. (2019). *Adam—latest trends in deep learning optimization.*. [online] Available at: https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c [Accessed 11 Dec. 2019].

[3] Hochreiter, S. (2019). *Long Short Term Memory*. [online] Bioinf.jku.at. Available at: https://www.bioinf.jku.at/publications/older/2604.pdf [Accessed 11 Dec. 2019].

[4] Amidi, A. (2019). *A detailed example of data generators with Keras*. [online] Stanford.edu. Available at: https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly [Accessed 12 Dec. 2019].

[5] Hoffman, R. (2019). *ASHRAE_DataGenerator_LSTM | Kaggle*. [online] Kaggle.com. Available at: https://www.kaggle.com/drcapa/ashrae-datagenerator-lstm [Accessed 12 Dec. 2019].

[6] Lopum, C. (2019). *ASHRAE -Start Here: A GENTLE Introduction | Kaggle*. [online] Kaggle.com. Available at: https://www.kaggle.com/caesarlupum/ashrae-start-here-a-gentle-introduction/output [Accessed 12 Dec. 2019].

[7] m, o. (2019). [online] Available at: https://www.kaggle.com/bablooBokya/ashrae-half-and-half/edit [Accessed 12 Dec. 2019].

[8] Nugyen, M. (2019). *Illustrated Guide to LSTM's and GRU's: A step by step explanation*. [online] Medium. Available at: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21 [Accessed 12 Dec. 2019].

PLEASE NOTE- CODE REFERENCES ARE IN README FILE SUBMITTED