

Electrical Load Forecasting using Machine Learning Techniques and their comparison

DISSERTATION/THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE AWARD OF THE DEGREE

OF

MASTER OF TECHNOLOGY

IN

POWER SYSTEMS

Submitted by:

Apoorva Mishra

2K18/PSY/02

Under the supervision of

Prof. J.N Rai



DEPARTMENT OF ELECTRICAL ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

**(Formerly Delhi College of
Engineering) Bawana Road,
Delhi-110042**

DEPARTMENT OF ELECTRICAL ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering) Bawana Road, Delhi-110042

CERTIFICATE

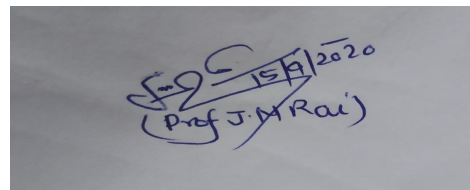
I, APOORVA, Roll No. 2K18/PSY/02 student of M. Tech. (Power System), hereby declare that the dissertation titled “Electrical Load forecasting using Machine Learning Techniques and their Comparison” under the supervision of Prof. J.N Rai of Electrical Engineering Department Delhi Technological University in partial fulfillment of the requirement for the award of the degree of Master of Technology has not been submitted elsewhere for the award of any Degree.

Place: Delhi



APOORVA MISHRA

Date: 15.09.2020



Professor J.N Rai

Electrical Engineering

ABSTRACT

The work presented gives hourly electrical load forecasting as a time series forecasting model using multilayer deep learning Long Short-Term Memory neural network Technique and its detailed comparative study with various Machine Learning Techniques based on their Mean Squared Error, Mean Absolute Percentage Error and Training time. Load Forecasting has immense potential to help in modulating the generation and distribution potentials of our smart grids in accordance to the requirement so that optimum power is generated and supplied through various channels which would be effective in grid management and operations. The MAPE of the model presented below is 0.41.

CONTENTS:

CHAPTER 1: INTRODUCTION

| | |
|-----------------------------------|----|
| 1.1 Need for forecasting | 7 |
| 1.2 Time series forecasting | 8 |
| 1.3 Literature Survey | 9 |
| 1.4 Objective | 11 |

CHAPTER 2: LSTM

| | |
|--|----|
| 2.1 Recurrent neural network(RNN)..... | 12 |
| 2.2 Long short term memory (LSTM)..... | 15 |
| 2.3 LSTM Algorithm..... | 17 |

CHAPTER 3: HYPERPARAMETERS

| | |
|--|----|
| 3.1)Hyperparameters involved in the optimisation of the model during the training process..... | 19 |
| 3.1.1)Splitting estimator of Train ,Test data..... | 19 |
| 3.1.2)Learning Rate..... | 19 |
| 3.1.3)Batch size..... | 20 |

| | |
|---|----|
| 3.1.4)Number of epochs..... | 20 |
| 3.2)Parameters that determine the structure of the model..... | 21 |
| 3.2.1)No. Of hidden units in a layer | 21 |
| 3.2.2)First Hidden layer of the network..... | 21 |
| 3.2.3)Total number of layers in the network..... | 21 |
| 3.3)Hyperparameter optimization..... | 21 |
| 3.3.1)Grid search technique..... | 22 |
| 3.3.2)Random search technique..... | 22 |

CHAPTER 4: ALGORITHMS USED FOR OTHER MACHINE LEARNING TECHNIQUES

| | |
|--|----|
| 4.1)Model used for algorithm..... | 23 |
| 4.2)Description of various regressors..... | 23 |
| 4.2.1)Ridge regressor | 23 |
| 4.2.2)Random forest..... | 24 |
| 4.2.3)Gradient boosting..... | 26 |
| 4.2.4)K-nearest Neighbour..... | 29 |
| 4.2.5)Multi-layer perceptron..... | 32 |
| 4.2.6)Logistic regression classifier..... | 34 |
| 4.2.7)Support vector machine..... | 34 |
| 4.2.8)Extra trees regressor | 35 |
| 4.2.9)Linear regression..... | 36 |

CHAPTER 5 : BACKPROPAGATION AND EVALUATION INDEX EQUATIONS

| | |
|-------------------------------------|----|
| 5.1)Backpropagation..... | 37 |
| 5.2)Key performance Indicators..... | 41 |

CHAPTER 6 : OPTIMIZERS

6.1)Gradient Descent optimizer..... 42

CHAPTER 7: RESULTS 45

CONCLUSION 49

REFERENCES 50

LIST OF FIGURES

| | |
|--|----|
| Fig.1 MLP representation of a neural network..... | 13 |
| Fig 2. RNN network basis idea when all the layers have same weight and go in a recurrent loop..... | 14 |
| Fig 3. Complete structure of RNN..... | 14 |
| Fig 4. LSTM Structure..... | 16 |
| Fig 5. KNN example 1..... | 29 |
| Fig 6. KNN example 2 | 30 |
| Fig 7. RMSE values of all the Regressors | 45 |
| Fig 8. MAPE values of all the Regressors | 45 |
| Fig 9. MAE values of all the Regressors | 46 |
| Fig 10. Training Time of all the Regressors | 46 |
| Fig 11. Comparison of regressors MAE and RMSE | 47 |
| Fig 12. Table for all the values | 47 |
| Fig 13. Output of LSTM Predictions VS original power | 48 |

CHAPTER 1

1.1)INTRODUCTION: Need of forecasting

In many of the business models nowadays, forecasting is used exceedingly and is very crucial whether we are to predict the revenue or load for any organisation or institutional set up. Artificial intelligence has grown up to such extents now that we can predict parameters with an accuracy upto 1%. While using proper technique we can save many resources and a considerable amount of money [1]. Taking an example if we are to suppose an appliance or utility used for load forecasting with 1 gigawatt annual peak load, the risk of oversizing and undersizing would be $0.01 * 1,000MW = 10MW$. Assuming the capital cost of \$10,000/KW, the overnight capital cost would be $\$10,000/KW * 10MW = \10 million. The savings of deferring \$10 million in spending for 1 year with 5% interest rate would be $\$10\text{million} - \$10\text{million}/(1 + 0.05) = \$476,000 \approx \$500,000$. If the utility uses forecasting for obtaining energy from the day ahead market, they might save around \$300,000 per year by improving the accuracy by 1%.

The optimum planning and management of energy distribution is a very crucial task when dealing with smart grids. There has been immense evolution in the smart grid, using the developments of information and communication technologies, and it is becoming a productive and durable system progressively. The smart grid systems are contrived so as to counter the problem of energy management and also to monitor, optimize and control the distribution of power. The decision for the flow or exchange of energy among all the utilities or devices connected to the grid is made after the governance and assessment of demand; which is an indispensable constituent of the energy management system of the electrical grid, thereby assuring the operational functionality, stability and dependency of the entire interconnected electrical system [1]. This will be beneficial in Strategizing the supply according to the demand therefore

the reliability of the grid is increased. It will also be economical as renewable energy sources are integrated with greater efficiency and lower costs.

1.2 TIME SERIES FORECASTING:

Time series forecasting is done over successive time intervals using sequent data points. Over the years many time-series forecasting techniques have been proposed. The most recent, optimized and accurate method of time-series forecasting has been used in this work combined with regression measures[2] .A lot of memory is consumed by the model while using these techniques so many of the useful is lost so most of the useful information is lost. The most recent one recently introduced is the Recurrent Neural Network(RNN) which used recurrent back propagation but was time consuming as well as with increasing information,it has many problems like the vanishing gradient problem.The measurements are arranged in a sequential manner and the forecasting can be univariate or multivariate in nature. In our problem statement we will be using univariate analysis with time as one variable and power the other. The Long Short term memory (LSTM) Technique learns and tracks all the past dependencies of a variable in relation with the new observations. Normally in a neural network input is given to a hidden layer and output is obtained, but LSTM is based on the recurrent neural networks where input is given to a set of hidden layers which contain information of the previous inputs also and then output is obtained.

To overcome these problems Long Short Term Memory cells (LSTM) was introduced [10] which solved complex, artificial long time lags tasks. A variant of LSTM which is widely used in time-series forecasting is introduced which reduces the number of variables used in the Gradient Recurrent Unit (GRU) [20]. Furthermore, the problems that still prevailed in the deeper networks were fixed by the utilization of relu and Xavier initialization ,of vanishing or exploding gradients .

A major problem with the RNN was that the input length was equal to the output length , therefore the number of features could not be reduced in the subsequent operations or layers which might be unnecessary and time consuming . Sometimes the number of outputs might have to be in condensed form. For that generally the output of RNN is connected to a Dense network (a layer with its all element connected to the every input features) but with identity activation and zero biasing (weighted summation). But still the weight of that layer needs to be learned. This added layer could also be used as an extra hidden layer with proper type of activation and initialization. This paper proposes one such type of initialization technique that could be used at this dense layer for faster convergence speed in training. To prove this hypothesis, a simple Feed-forward Long Short Term Memory Neural Network with 3 hidden layers was trained.

1.3 LITERATURE SURVEY

Load forecasting , according to Gross and Galiana (1987), is also concerned with the prediction of hourly, daily, weekly and monthly values of the system load, peak system load and the system energy[2]. Srinivasan and Lee (1995) classified load forecasting in terms of the planning horizon's duration: up to 1 day for short-term load forecasting (STLF), 1 day to 1 year for medium-term load forecasting (MTLF), and 1±10 years for long-term load forecasting (LTLF)[4]. Load forecasting when done properly saves a lot of utilities, resources and money . According to Bunn and Farmer (1985), these savings are realised when load forecasting is used to control operations and decisions such as dispatch, unit commitment, fuel allocation and o-line network analysis[5].The model developed by Papalexopulos and Hesterberg (1990) produces an initial daily peak forecast and then uses this initial peak forecast to produce initial hourly forecasts further again , it uses the maximum of the initial hourly forecast, the most recent initial peak forecast error, and exponentially smoothed errors as variables in a regression model to produce an adjusted peak

forecast[8]. A regression-based daily peak load forecasting method with a transformation technique was introduced by Haida and Muto. They use a regression model which predicts the nominal load and also they predict the residual load using a learning method [9]. Haida also reduced the errors in the prediction of transitional seasons by introducing and designing such processing techniques. Trend cancellation removes annual growth by subtraction or division, while trend estimation evaluates growth by the variable transformation technique. A least-squares approach was used by Varadan and Makram (1996) which was used at power lines and substations so that the different kinds of loads can be identified and quantified[11]. Later an adaptable regression model was developed by Hyde and Hodnett for 1-day-ahead forecasts, which was able to identify and distinguish between weather-insensitive and weather-sensitive load components. Linear regression of past data is used to estimate the parameters of the two components[12]. In order to support future smart grid applications, effective load forecasting techniques for electricity users are gaining increasing interest. Zhang et al. [13] developed a big data architecture that combines load clustering based on smart meter data and decision tree to select corresponding load forecasting models for prediction. Stephen et al. [14] clustered and labelled daily historical data of individual households. The individual households were deemed as label sequences, which are further fit to Markov chains. Then the day ahead label can be sampled, and cluster means at each time points were used for the day ahead prediction. These works all showed that the forecasting errors could be reduced by effectively grouping different customers. However, they all only reported the aggregated load forecasting error at the system or community level where individual customer prediction errors could be offset by the diversity of different end users. In the existing literature, Chaouch's work [15] and the work of Ghofrani et al. [16] are the first two examples that focus on load forecasting for individual users. In these works functional time series forecasting approach was proposed, and the daily median absolute errors were reported. Very recently, deep learning based methods start to emerge in the load forecasting community. Ryu et al showed

that the load forecasting accuracy for industrial customers could be improved by using deep neural networks [17]. In fact, the industrial electricity consumption patterns are much more regular than residential ones, so that much more accurate results are obtained using deep learning techniques. Later in 2016, Marino et al. attempted to solve the load forecasting issue using LSTM [18] and showed good results as all the other methods. However, the effectiveness of the two pioneering works was only verified on the metric of root mean square error (RMSE) instead of the more common metric of mean absolute percentage error (MAPE), which makes it hard to contrast to other works. Later Weicong Kong and Zhao Yang Don proposed a Long Short term(LSTM) Recurrent neural network based load forecasting framework for this extremely challenging task of individual residential load forecasting, because LSTM has been proven to learn the long term temporal connections. The inconsistency in daily consumption profiles generally affects the predictability of the customers. The higher the inconsistency is, the more the LSTM can contribute to the forecasting improvement compared to the simple back propagation neural network [19].

1.4 OBJECTIVE

The objective of this study is to do hourly electrical load forecasting as a time series forecasting model using multilayer deep learning Long Short-Term Memory neural network Technique and its detailed comparative study with various Machine Learning Techniques based on their Mean Squared Error, Mean Absolute Percentage Error and Training time.

CHAPTER 2: LONG SHORT TERM MEMORY

2.1 Recurrent neural network

Normally neural networks could not retain long term information and whenever the training is run it learns from scratch. Recurrent Neural Network(RNN) addresses this issue . The recurrent neural networks can be considered as a loop where each predecessor network passes the previous information to the successor networks .Long short term memory(LSTM) are a special kind of neural networks.

It is an improvement of the Recurrent neural network which can operate and keep a track of long term dependencies. In a RNN network output of the hidden layers is fed back again to the network itself unlike the normal feed forward network which does not work in loop . The RNN can be many to many or many to one but It is to be noted that unlike normal neural networks these RNNs have the same weights for all the time steps as they operate in a loop . The problem that recurrent neural networks face is that due to large scale dependencies on the previous data , if we want our model to learn better, as we increase our time steps , the problem of vanishing or exploding gradients becomes increasingly inevitable .

To understand how RNN works, let's take an example, whenever we write an email, Always the word that could come next is predicted beforehand . So , how do we predict the next word in a sentence? Let's try to analyse using the Multilayer Perceptron network.

In a simple Multilayer perceptron (MLP) there is an input layer , hidden layer and output layer. When we want a deeper network, there are multiple layers of hidden neural networks . It is to be noted that all the hidden layers have their own weights , biases and activation functions which are different in nature. Due to this property the hidden layers can not be combined together unless the weight and biases for all the hidden layers are the same.

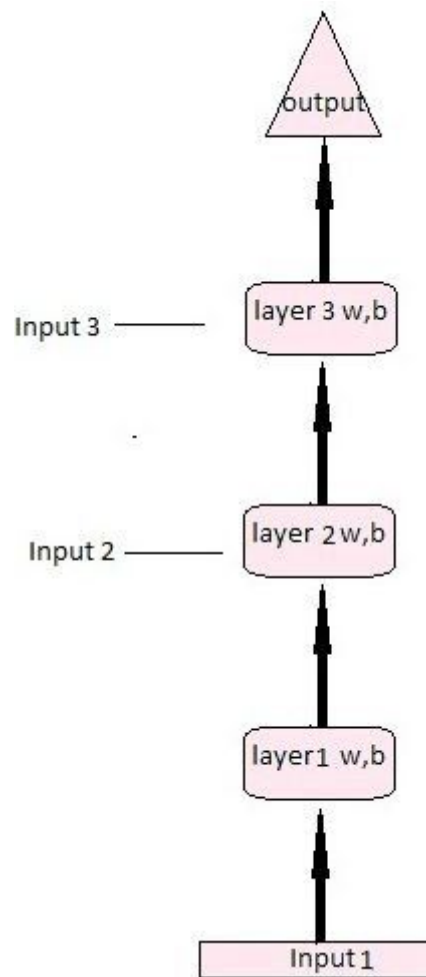


Fig.1 MLP representation of a neural network

When the weights and biases are the same for all the layers . We combine the layers into a single recurrent neural network layer. Therefore this recurrent neural network also stores the information of the previous layer , thereby establishing a relationship between the current and the previous inputs.

Next state is given as :

$$h_t = f(h_{t-1}, x_t)$$

New hidden state is given by h_t and the current input is given by x_t .

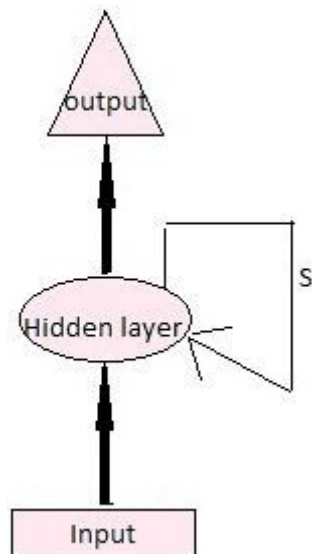


Fig 2. RNN network basis idea when all the layers have same weight and go in a recurrent loop

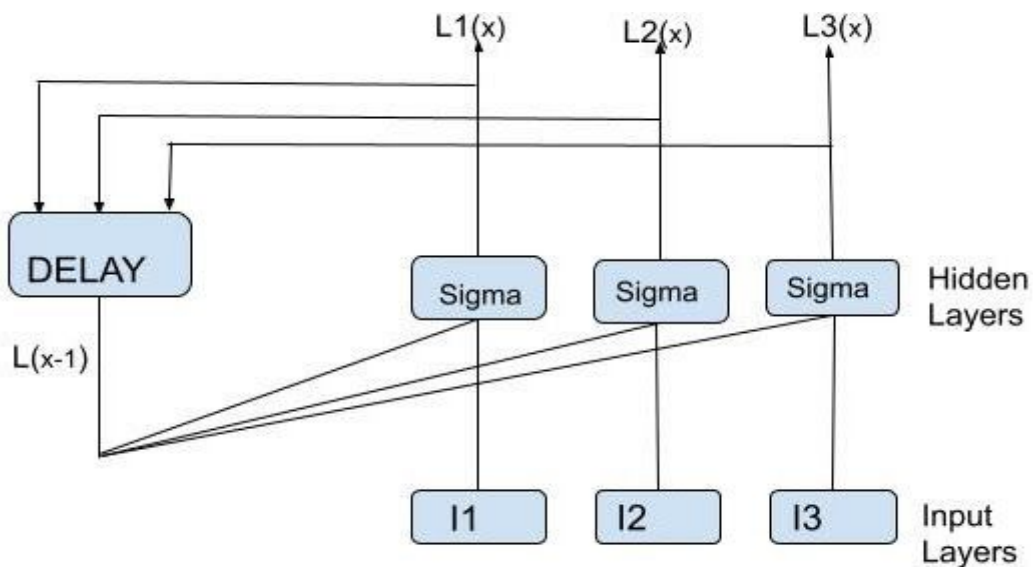


Fig 3. Complete structure of rnn

2.2 Long short-term Memory (LSTM)

It consists of the forget gate, with the purpose of altering the portion of input or the previous stage which is to be passed to the next stage . It helps us to decide what percentage of the data of the previous stage we need to remember in the next stage . The forget gate is made up of a sigmoid layer which outputs a number between 0 and 1 for each state depending on how much to keep and how much to discard. The output of this forget gate can also be thought of as weights to the internal state for each input.

The inner state of the LSTM is defined by a variable s which is delayed by a one-time step and is used by the network to learn the relationship between the inputs and their sequence time thereby creating a recurrent loop. This filtering done by the forget gate also helps to reduce the effect of the vanishing gradient problem encountered in the recurrent neural network.

$$f_{t+1} = \sigma (W_{fo} * [h_t, i_{t+1}] + b_{fo})$$

Then comes the input gate which decides which values are to be updated and what new information is added to the cell state and is also a sigmoid layer .

$$i_{t+1} = \sigma (W_{input} * [h_t, x_{t+1}] + b_{input})$$

The input layer if followed by a tanh layer the results of these two layers are multiplied together to get the new updated scaled value and is then added to the output from the forget gate to make a new cell state. This new state is again fed to tanh which gives values between -1 and 1 to again decide which values are to be passed.

Thus LSTM provides the advantage of adding or removing any information in the cell through gates.

$$C_{t+1} = \tanh (W_{new} * [h_t, x_{t+1}] + b_{new})$$

$$C_{t+1} = f_{t+1} * C_t + i_{t+1} * C_{t+1}$$

$$O_{t+1} = \sigma (W_{output} * [h_t, x_{t+1}] + b_{output})$$

$$h_{t+1} = O_{t+1} * \tanh (C_{t+1})$$

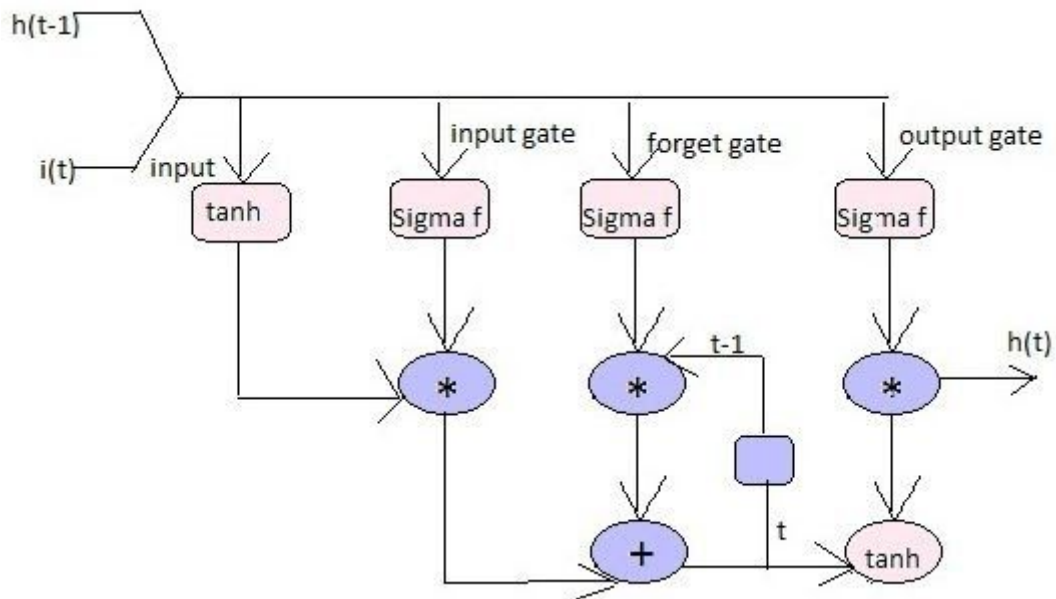


Fig 4. LSTM Structure

LSTM parameters:

a)activation : The default activation function if nothing is given is set as tanh. If we do not want to change any inputs then we can give the activation function as NONE , which implies the activation is linear $f(x)=x$.

b)units : Specifies the dimensions of the output layer.

c)dropout : Dropout value gives the value between 0 and 1. This gives how much value is to be dropped of the input or the layer specified.

d)recurrent_activation : The activation function to be used in the next layers. If we do not specify anything then it is sigmoid by default . If we do not want to change any coming values from the previous layers then we can give the activation function as NONE , which implies the activation is linear $f(x)=x$.

To call the LSTM :

a) input : input is given as a 3D tensor of shape [number of samples ,dimensions of each element , features].

b)training : This function indicates whether the layer will be utilized for training or inference.

2.3 LSTM ALGORITHM

After dividing the entire dataset into training and test data in the ratio 9:1 and reshaping the input in the form (number of samples, dimension of elements) we start building a sequential model.

Layer 1:

Add a LSTM layer with input dimension as 1 and output dimension as 50 with return sequence as True.

Add a Drop out layer to prevent overfitting.

Layer 2:

Add a LSTM layer with output dimension as 50 and return sequence as False.

Add a Dropout layer to prevent overfitting.

Layer 3:

Add a Dense neural network layer with output dimension as 1 and activation linear.

While compiling the model, the optimizer used “rmsprop” and loss to be minimized is specified as “mse”. The gradient descent optimizers are used to reach the global minima by finding the gradient and updating the weights and biases such that the cost functions can attain minimum value. However, sometimes non-convex functions arise due to which the optimizer is stuck in the local minima hence the global minimum value is not achieved by the cost function. Rmsprop optimizer is used as it solves this problem. Rmsprop restricts the oscillations in one direction, basically the vertical direction thereby converging faster to the solution as higher learning rates can be used .

CHAPTER 3: HYPERPARAMETERS

Normally In machine learning we have two types of parameters model and hyperparameters:

1)Model Parameters: Model parameters are those which the model fits after the training from the data. For example, weights of classes , biases, split points or the nodes are the model parameters that the model fits after training.

2)Hyperparameters: Hyperparameters are those which we have to tune in order for the model to perform well and give optimal results.For example , learning rate and number of iterations are something that we tune and provide to the model in order to train according to our requirement.

Some Examples of the hyperparameters available in the scikit learn package:

3.1)Hyperparameters involved in the optimisation of the model during the training process :

3.1.1)Splitting estimator of Train ,Test data:

```
train_test_split( X, y, test_size=0.4, random_state=0)
```

a)Over here, test_size gives the percentage or portion of the data to be included for testing.

b)Random_state is used for random number generation for shuffling the data

3.1.2)Learning Rate:

Learning rate gives the steps to be taken to reach the global minima during optimisation. If we take very small values of learning rate it takes very long times i.e more number of steps to reach the global

minima however if we take a very large learning rate then we may suffer the problem of overshoot and the algorithm might not reach the global minima and hence may not converge .It is very important to keep in mind that the models have lot many parameters and each one has a different kind of curve for the error and not necessarily u-shape so our learning rate has to cater to all them.

3.1.3)Batch size:

Batch size determines a lot of things while training like the requirement of resources , speed , epochs required .

We earlier used to calculate the gradient by feeding the entire data to one particular training epoch, by using the errors of all the data in the data set. Nowadays, we do not feed the entire data at once instead data is fed in bathes as specified by the user.

If we keep a large batch size, we require more memory to run the training but we get the advantage of boosts in computations . If we keep a smaller batch size, it introduces more deviations in the error calculations .However a small batch size is quite instrumental in stopping the optimizer from being stuck at the local minima.

3.1.4)Number of epochs :

The number of epochs or the number of training steps is usually determined by the validation error. We should run our training as long as the validation error keeps on decreasing .We can go for early stopping the training if the error does not decrease for 30-40 epochs .

3.2)Parameters that determine the structure of the model:

3.2.1)No. Of hidden units in a layer :

One of the most complex parameters is the number of hidden units. Neural networks should have enough capacity or we can say enough number of units to learn and fit a particular function which is given by this parameter. When we have to fit a simple function we need very less amount of hidden units but for a larger function the model should have a lot number of hidden units . However it is to be kept in mind that if we give a very large number of units , the model learns everythings and fails to generalise i.e it overfits. So we need to keep an optimal number of hidden layers .

3.2.2)First Hidden layer of the network:

We always need to keep in mind that the first hidden layer should be such that the total number of hidden units is always more than the input. It preserves the input data thus giving a better output .

3.2.3)Total Number of layers in the network:

Normally more number of layers give better results in cases like convolutional neural networks . But in normal neural networks a three layer network does the job just fine .

3.3)Hyperparameter optimisation:

Now we have seen that these hyperparameters are useful only when we find an optimal value for them else, they degrade the model. We find the correct and optimal value for these parameters using various techniques called the hyperparameter optimisation method . These methods include :

3.3.1)Grid search Technique:

Grid search works by training the algorithm with a combination of learning rate and number of layers and evaluates the performance using the cross-validation technique. Also one of the methods is K-fold cross validation technique where a set of randomly taken data is trained on the model for many times . and training and validation sets are different all the time. This is a very useful algorithm unless the data has very high dimensionality in that case it fails and has the problem called curse of dimensionality.

3.3.2)Random search Technique:

Random search randomly allocated data into the validation and training dataset.And selecting randomly the dimensions and parameters, it sometimes does not give good results as it does not take the information from the former models to use.

CHAPTER 4: ALGORITHMS USED FOR OTHER MACHINE LEARNING TECHNIQUES

After dividing the entire dataset into training and testing in the ratio 9:1, we get properties of all the regressors Ridge, K nearest Neighbors, Random forest regressor, Gradient Boosting Regressor, MLP regressor and Extra trees Regressor. We fit the model, take out the training time, testing score and training score, rmse , MAPE and MAE, store them in a `dataframe` for comparison purposes and plot each of them .

4.1 Model used for the algorithms :

Divide the entire dataset into 9:1

```
BEGIN
```

```
    Get the property of the regressor
```

```
    Fit the model
```

```
    Get the properties, take out the training time,  
    testing score and t raining score, rmse , MAPE and  
    MAE
```

```
    Store all the values in a dataframe
```

```
REPEAT
```

```
    Plot and compare each of them
```

```
END
```

4.2 Description of the various kinds of regressors

4.2.1)Ridge regressor : is used in the cases when there is no unique solution available and the answer is to be calculated using approximation.Ridge

regression (RSS) operation is performed using L2 regularization which is the sum of squares of coefficients .

Function to be optimized= $RSS + \alpha * (\text{sum of square of coefficients})$

α is used to give weightage to the term which is to be minimised to greater extent. Various values that can be taken by alpha:

1) $\alpha = 0$:

The function serves the same purpose and the same values of coefficients as simple linear regression are obtained

2) $\alpha = \text{infinity}$:

As non-zero coefficients would make the function infinite, The coefficients will be zero.

3) $0 < \alpha < \text{infinity}$:

Weightage to be given to the L2 regulariser or the cost function to be optimized is decided by alpha.

4.2.2) Random Forest : Random forest, a supervised learning Algorithm works by constructing multiple layers of decision trees, also called ensemble, by training often using the bagging technique which outputs the mode in case of classification and the mean in case of regression problems.

An advantage of the Random forest algorithm is that without any hyper-parameter tuning also, we can achieve required results exploiting the benefits of all the best algorithms available.

Therefore, it works by constructing an ensemble of decision trees and then merge the outputs thereby increasing the accuracy of the prediction. This algorithm when searching through the features while forming the trees instead of selecting the most important feature , selects the best feature from the subset of features while slipping the node thereby adding some randomness.

This algo is one of the best as we can measure the importance of each feature at every prediction. It calculates the score as to how much is the impurity reduced

by the nodes in the complete forest which uses that particular feature . This score for all the features is scaled and added such that the sum is equal to 1.

Now , in machine learning since we generally try to keep the number of features really low in order to avoid overfitting, the feature importance detection provided by random forest algorithms really contributes a lot in deciding which feature to drop depending upon its contribution to the model.

However it is to be noted that there is difference between decision trees and random forest, they should not be confused to be the same

The first difference is that the decision tree collects the labels and the training data and on that basis makes some rules, and generates the predictions abiding by those rules , however , the random forest gives the average results by building many decision trees by taking random features and observations ..therefore not particular set of rules are followed.

The second difference is that because of this feature of random forest algorithm of introducing so much of randomness, a big problem of the deep decision trees,overfitting is solved. The algorithm takes subsets and build many small trees using those, thus solving the problem of deep decision trees and combining those subsets further. Though it solves many problems giving better results, makes computation a little slow and consumes some time.

The hyperparameters used in ransom forest are those which are normally used to increase the predictive power of the model or to make the model faster:

a)N_estimator :One of the hyperparameters which gives the number of trees built by the algorithm before taking the average of the predicted values is n_estimator. Higher the number of trees better is the prediction but it slows down the computation so n_estimator assigns an optimum number of trees which gives good results under considerable time.

b)max_features: max_features gives the number of features that the random forest decides to select while splitting a node.

c)min_sample_leaf: min_sample_leaf gives the min number of leafs that the decision trees makes while splitting a node

d) n_jobs :The n_jobs hyperparameter alots the number of processors which can be used. For example if the value is 2 , it can use 2 processors but if it is -1, it can use all of it

e)random_state : If we give the same training data, observations, hyperparameters and labels the model will produce the same predictions and outputs everytime if a definite srandom_state is set every time.

f)oob_score: One of the methods often used is cross-validation in which one-third of data is set aside for evaluation and testing the results ,its accuracy and the performance , it is also called oob_sampling

Therefore, the random forest algorithm is versatile in that it works on both classification and regression problems and solves the problem of overfitting if there are optimum trees in the forest. But due to many trees it makes the algorithm slow in predictions which can be inefficient in some real-time applications.It has many uses like in medicine it is used to examine the medical history of patients and determine the type or cause of disease.Also ,it is used in banks and e-commerce trades.

4.2.3) Gradient boosting : The Gradient boosting algorithm works on the method of minimization of error. The prediction model given by the gradient boosting algorithm is generally the ensemble of many models arranged in the sequence which may produce the minimum error, mostly a set of decision trees.

It works like the adagrad algorithm which trains a decision tree by assigning equal weights to all the observations. Further as we proceed, after the first iteration and evaluation of the first decision tree, we assign higher weights to those data which are difficult to classify and lower weight to the data that is easy to classify so that we can improve upon the predictions from the first iteration. Our model now is the combination of the both 1st tree and the second tree, then, the errors in classification from this iteration of the 2-tree model are calculated and a third tree is formed to decrease the error further by adjusting the weights on the observations as we had done for the 2nd tree. Repeating this for a particular number of iterations, each coming tree focuses on the data that was not classified correctly by the previous tree. Therefore the model finally gives the weighted sum of the predictions .

However the gradient boosting algorithm proceeds to train the models in a sequential, gradual and additive way. The two algorithms adagrad and gradient boosting are different in way such that adagrad acknowledged the weak decision trees by giving them a higher weight, the gradient boosting algorithm acknowledges them by introducing a gradient in the loss function.

The ability of the model coefficients to fit the data is measured by the loss function. Gradient boosting comes with an advantage of optimising the cost functions given by the user. For example , if we predict the prices of a commodity using regression methods , the loss function will be formed by the difference of the true and predicted prices of the commodity.

The algorithm follows as :

```

BEGIN
    Calculate average of the target label

    Calculate the error Actual - Predicted

    REPEAT (until n_estimators)
        Construct a decision tree whose leaves give the
        error

        Use all trees in ensemble to predict the target
        label

        Calculate new errors

    Use all the trees to make final prediction
END

```

Parameters used for a decision tree:

a) Over here, `criterion` gives the parameter which decides the quality of the split. This means that we set a particular threshold for the purity or the parameter beyond which we do not want our node to split. Like we specified entropy so those nodes will not split after which more information gain could not be achieved. Also we use gini impurity.

b) `splitter` is the strategy to split, i.e it can be either the best split or random split of the nodes

c) `max_depth` gives the depth of the tree. If we do not give any value for `max_depth` then the tree will expand upto the point where either the leaves are pure or contain samples less than specified `min_samples_split`.

d)Min_samples_split gives the minimum number of samples that are required to split a node. If the samples are less , the node does not split.

e)Min_sampe_leaf if specifies that at a node, a leaf will be considered for training only if it has the specified number of samples

f)Min_impurity_split gives a threshold value after which we need the tree growth to stop early if the impurity level exceeds the given value. The node will not split further and remain as a leaf .

4.2.4)K-Nearest Neighbor : K-Nearest Neighbor works on the measured Euclidean distances between the query and the data points and then operating on a particular number (K) of data samples then finds the most suitable class for it in case of classification or takes the average of the points in case of regression.It works on the idea that similar things are placed together .

To understand how it works let's take a classification model with two sets of data points one is positive denoted by red and the other is negative denoted by green :

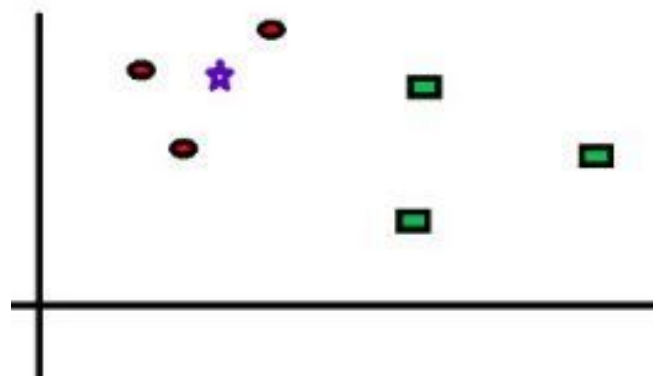


Fig 5. KNN example 1

We have to find out the label of the point in blue denoted by the star taking the value of k as 3.

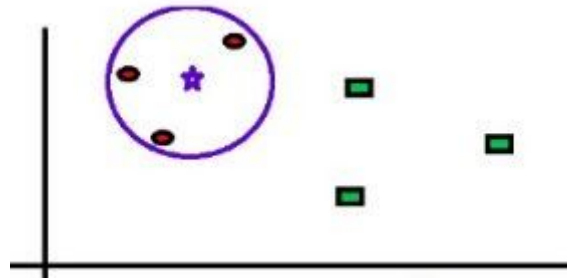


Fig 6. KNN example 2

Algorithm of K-Nearest Neighbour:

BEGIN

Load the data

Initialize k to your chosen number of neighbours

REPEAT

Calculate distance between the current chosen data point and the query

Keep appending the distance and data point index into a list

Sort this list in ascending order of distances

Choose first k distances

Take the labels of these chosen entries

IF regression

RETURN mean of labels

IF classification

RETURN mode of labels

END

Choosing the correct K value is very important in order for the algorithm to work properly. We train the dataset many times with different values of k and find those values of k at which the error is minimum when the algorithm is subjected to unseen data, i.e the data at which it has not been trained.

However the important points that we need to keep in mind are :

- a) K can not be taken as 1, for example if we have an entire portion of area surrounded by red, our guess should be that the answer is the label indicated by the red points, but instead, if we have a green point in close proximity to the query, and it has the shortest distance , then our answer will be the label corresponding to the green point which is incorrect .
- b) However as we keep on increasing the value of k , the errors are reduced and we get better results by taking the majority voting or average label whichever method we tend to choose.
- c) It is to be noted, after a particular value of K , the error starts increasing again. This is the point where we should understand that we have taken the value of k too much and it is not just taking the neighbours , but a large amount of data points beyond the considered area as well .
- d) Now, it is to be kept in mind when the problem of classification is considered, we take the majority vote or mode of the labels. We should always take an odd value of k.

```
KNeighborsClassifier(n_neighbors=5,p=2,metric='minkowski',n_jobs,algorithm
```

a)Over here, n_neighbours is the number of neighbours, or the value of K that we use in most the nearest neighbour algorithm.

b)Minkowski is the formula for the distance calculation .

c) p gives the power for the distance formula , if it is 1, that means our metric for distance calculator is the manhattan distance and if it two, it means the metric for calculator is the euclidean distance.

d) n_jobs gives the number of parallel processors or jobs that we want to run to search for the value of k .

e)algorithm gives the algorithm which is used to determine k ..it can be brute search or kd-tree etc.

Therefore K-NN is a versatile model which can be used for classification and regression and even in searching problems and also it is very easy to implement without having to tune and train many hyperparameters and make numerous assumptions . But sometimes it becomes really slow as the data points increases. There are many applications of k-nn in the real world like finding whether a review on an ecommerce site for a review is positive or negative .

4.2.5)Multi-layer Perceptron (MLP): MLP regressor is a multi-layer perceptron supervised learning, non-linear training technique that uses backpropagation to update weights.

Firstly let's describe how does an neural network layer is formed:

1. We have a perceptron which has 4 parts , an input layer, weights and biases, summation and an activation function

2. This perceptron receives the input, It take a set of features Let's say $X=X_1, X_2, \dots, X_n$ and a label or an output Y , this regressor can also form a model for both classification and regression which can be non-linear in nature. However, it is not to be mistaken for logistic regression, it can have many non-linear layer in between input and output layers.
3. The input layer mostly consists of neurons $\{x_i \mid x_1, x_2, \dots, x_m\}$ where m gives the number of features or dimensions of the input. The hidden layer works in the manner such that each neuron gives the output which is the weighted linear sum of the inputs from the previous layers, given as:

$$\omega_1 * x_1 + \omega_2 * x_2 + \dots + \omega_m * x_m$$
4. This weighted sum is then passed to an activation function whose role is to map the input according to our requirements between $(0,1)$ or $(-1,1)$. The activation function is usually non-linear in nature.
5. These layers of perceptrons are added to create a Neural network. This is known as a multi-layer perceptron network.
6. Learning in the MLP is generally done through the backpropagation method, where the weights are updated after each iteration.

Though it serves a great advantage of learning non-linear functions and it has many uses in speech recognition and natural language processing, It has many disadvantages too.

1. It has a loss function which is non-convex in nature in the hidden layers and it might have more than 1 local minima. Hence, if we give different or random weights each time, it gives different validation accuracy.
2. It gives different outputs and is very sensitive to scaling of the features.

3. A lot of hyperparameter tuning is required . The different parameters would be the number of hidden layers, the number of hidden neurons in each hidden layer and the total number of epochs.

4.2.6) Logistic regression classifier:

`LogisticRegression(C=1000.0, random_state=0)`

a) Over here, C is the inverse of regularization strength. The smaller the value higher is the regularization strength.

b) `penalty` gives the l1 or l2 norm used as a penalizer

c) `fit_intercept` gives whether we want to add a constant value to the deciding function whether it is a bias or an intercept

d) `class_weight` gives the balance or the weights that we provide to each class...i.e which one is more important . If we do not give this then it is presumed that all the classes have the same weights and are treated as equal.

e) `max_iter` gives the number of iterations that the solver requires to converge .

4.2.7) Support vector machine classifier

`SVC(kernel='linear', C=1.0, random_state=0)`

a) Over here, the `kernel` gives the type of kernel, linear or non-linear that we want . Linear is used for linear classification and rbf for non-linear.

b) C is the regularization or the penalty parameter used whether we want the L1 or L2 norm.

c)degree gives the degree of the kernel polynomial .

4.2.8)Extra Trees Regressor: Extra Trees Regressor is also a variation of the Random Forest where instead of selecting a optimal feature we select a random feature for split .It is built from random forest with the following changes :

- a)Each decision branch is built by using all the training data.
 - b)To form any node we take a subset of random features and then determine their best split.
 - c) Maximum depth of 1 is allowed for the decision branch .
- It consumes more computational power than random forest.

The major hyperparameters used are :

- a)n_estimators: number of trees that we want in the forest.
- b)criterion: it is the measure by which we decide to reduce the error and optimise out function
- c)max_depth gives the depth of the tree. If we do not give any value for max_depth then the tree will expand upto the point where either the leaves are pure or contain samples less than specified min_samples_split.
- d)Min_samples_split gives the minimum number of samples that are required to split a node. If the samples are less , the node does not split.
- e)Min_sampe_leaf if specifies that at a node, a leaf will be considered for training only if it has the specified number of samples
- f)Min_impurity_split gives a threshold value after which we need the tree growth to stop early if the impurity level exceeds the given value. The node will not split further and remain as a leaf .

g)bootstrap: Whether we want to build the trees using bootstrap samples or the entire dataset.

h)max_Samples: IT gives the number of samples of the training data..that is X which can be used to train the estimators , if the value of bootstrap is true.

Therefore , It is a more randomised form of a decision tree.

4.2.9)Linear regression : basically is represented by a linear equation which takes a set of linear inputs and predicts corresponding outputs. It works similar to a linear relation in statistics. In machine learning since we have a larger number of dimensions of the inputs we call it a hyperplane instead of a line. This algorithm works on the assumption that the relationship between your input and input is linear in nature and does not have any noises in data. Also we need to make sure that we remove all the collinearity in data as those may overfit our model. For better predictions through linear regression we can normalize our input. The regularization methods of the linear regression include lasso and ridge regularization depending upon whether we want L1 or L2 regularization

CHAPTER 5: BACKPROPAGATION AND EVALUATION INDEX EQUATION

5.1) Backpropagation:

The backpropagation algorithm uses the chain rule to train a neural network efficiently. Whenever we move in the forward direction, after completion of each forward movement the algorithm goes backwards also, altering the weights and biases.

Discussing the backpropagation in detail along with the training procedure :

Let's say we have n-layer neural network modelling . This would mean that the input layer has n neurons and also the hidden layer has n neurons. The output has 1 neuron . The input can be of any type scalar ,vector or complex. The input values are the activation of the first layer . The activation function which is normally non-linear when applied to the input gives activation values ..i.e which values and to what extent is to be passed through the layer . We normally use sigmoid , tanh or Relu functions .

The hidden layer gives the output after multiplication of weights and inputs.

The output layer has one single neuron which produces the final value.

Here we show how the forward propagation is evaluated :

$$\begin{aligned}X^1 &= i^1 \\Z^2 &= W^1 * X^1 + b^1 \\i^2 &= f(Z^2) \\Z^3 &= W^2 * i^2 + b^2 \\i^3 &= f(Z^3) \\S &= W^3 * i^3\end{aligned}$$

Now , we determine the relation between this predicted output value S and the expected value y from the corresponding training data. We generally give a cost function which is either MSE value or RMSE value depending on our requirements . Now based upon the value that we get from this cost function the model determines how much is the predicted value deviated from the input value and accordingly the model adjusts the weights and biases to get the output value close to the actual value . This process is done through the back propagation process which is explained below.

The aim of backpropagation is to minimise the error or the cost function value. Now how much change or adjustment is needed is calculated using the derivatives of the error function with respect to all the inputs of the layer.

The gradient or derivative of a function gives the change or the partial derivative of the target function in relation to the parameter with which it is calculated .

The gradient is given as :

$$\frac{\partial C}{\partial i} = \left[\frac{\delta C}{\delta i_1}, \frac{\delta C}{\delta i_2}, \frac{\delta C}{\delta i_3} \dots \dots \dots \frac{\delta C}{\delta i_m} \right]$$

This derivative gives the measure as to how much is the function sensitive when the output changes with respect to the change in the inputs. Therefore the value of the derivative will give the direction and the value by how much the input needs to change in order to keep the error as low as possible .

$$\frac{\delta C}{\delta W_{jk}^t} = \frac{\delta C}{\delta Z_j^t} \frac{\delta Z_j^t}{\delta W_{jk}^t} \dots \dots \dots \text{equation 6}$$

Here m gives the number of neurons in the t-1 layer.

$$\delta Z_j^t = \sum_{k=1}^m w_{jk}^t * i_k^{t-1} + b_j^t \dots\dots\dots \text{equation 7}$$

When we differentiate the equation 7 with respect to weight:

$$\frac{\delta Z_j^t}{\delta W_{jk}^t} = i_k^{t-1} \dots\dots\dots \text{equation 8}$$

Putting values of $\frac{\delta Z_j^t}{\delta W_{jk}^t}$ from equation 8 to equation 6, we get the error or the cost function for weights as :

$$\frac{\delta C}{\delta W_{jk}^t} = \frac{\delta C}{\delta Z_j^t} i_k^{t-1} \dots\dots\dots \text{equation 9}$$

Similarly when we differentiate for bias:

$$\frac{\delta C}{\delta b_{jk}^t} = \frac{\delta C}{\delta Z_j^t} \frac{\delta Z_j^t}{\delta b_{jk}^t} \dots\dots\dots \text{equation 10}$$

$$\frac{\delta Z_j^t}{\delta b_{jk}^t} = 1 \dots\dots\dots \text{equation 11}$$

Substituting the value of gradient from equation 11 into equation 10 we get the partial derivative as :

$$\frac{\delta C}{\delta b_{jk}^t} = \frac{\delta C}{\delta Z_j^t} 1 \dots\dots\dots \text{equation 12}$$

Taking the values from the equation 12 and 9 , our cost function becomes:

$$\frac{\delta C}{\delta W_{jk}^t} = \frac{\delta C}{\delta Z_j^t} i_k^{t-1} + \frac{\delta C}{\delta Z_j^t} \dots \dots \dots \text{equation 13}$$

Now, these gradients when subjected to a particular condition or a particular error value help in updating the function and optimizing it. This loop is terminated once the cost function archives its minimum value :

WHILE (condition)

$$W = W - \epsilon \frac{\delta C}{\delta W}$$

$$b = b - \epsilon \frac{\delta C}{\delta b}$$

END

The initial value of weights and biases are randomly chosen at the starting of the training process and then brought down to the correct value after the optimization is complete .The W and b mentioned in the loop are the matrices.

Over here , ϵ gives the learning rate which was discussed earlier in the hyperparameters chapter.It determines how much weightage is to be given to gradient so as to decrease the value of weights or biases less or more.

Therefore this is how we optimize and minimise the error using backpropagation.

5.2)KPI(Key performance Indicator)

Key performance Indicator gives the measure of the performance of our model on the data. The key performance indicators that we use over here are :

MAPE: In forecasting MAPE or Mean absolute percentage error is the most commonly used KPI to measure accuracy . We take out absolute errors for each data point individually and divide it by total in that demand period. It can also be called the average of errors of percentages. However MAPE can be skewed sometimes ,because we calculate each error individually.

$$\text{MAPE} = (1/n) * \Sigma [\text{abs} (\text{actual} - \text{predicted})/\text{actual}]$$

MAE: Mean absolute error is one of best measures for forecast accuracy. It is the absolute error divided by all the data points .It is always good to get the percentage values of these indices as they are not scaled to the demand so you would not know whether it is good or bad.

$$\text{MAE} = [\Sigma \text{abs} (\text{actual} - \text{predicted})] / n$$

RMSE: Root mean square error is given by the square root of the average of individual squares of the errors.Many algorithms of machine learning are based on MSE which is related to RMSE. However , if we have even one big error, RMSE will give a very big value for it.

$$\text{RMSE} = [\Sigma(\text{predicted} - \text{actual})^2 /n]^{1/2}$$

CHAPTER 6 : OPTIMIZERS

Gradient Descent Algorithm

The gradient descent optimizers are used to reach the global minima by finding the gradient and updating the weights and biases such that the cost functions can attain minimum value.

The algorithm for gradient descent :

BEGIN

- Prediction values of the neural network are taken
- The loss function is used to calculate the losses
- The loss functions are differentiated and the gradients are obtained; these are also the partial derivatives.
- These gradients update the values of weights and biases

REPEAT

Whenever we update the weights and biases after finding the gradients we reach closer to the optimum value. Therefore as the training of neural networks progresses the function that is to be minimised moves closer to its global minima. However, sometimes the function gets stuck in local minima for non-convex function therefore preventing it from reaching its global minima. To come out of such situations we use methods like learning rate, momentum, etc. The learning rate, being the most crucial part of the optimizers, can be thought of as the steps we take to reach the bottom of the global minima. Depending on the learning rate the weights and biases are updated, if we take larger values of the learning rate it is a possibility that we might skip the global minima and move about it thereby never converging. But, Taking a smaller value of learning rate

takes a lot of time to converge. Also we need to keep in mind that it does not get trapped in local minima.

Gradient descent with momentum:

When the gradient descent algorithm is used with momentum, it converges fast. The momentum function works such that we move fast or take larger steps in the x coordinate direction therefore restricting and slows down movement in the y coordinate direction, unlike the standard algorithm which takes larger steps in the y coordinate direction thereby proceeding slowly in the x direction.

RMSprop Optimizer:

RMSprop optimizer is the combination of the gradient descent algorithm with momentum. When we restrict the movement and take shorter steps in the vertical direction, we can increase our learning rate therefore it can converge fast. The momentum is generally denoted by beta (b) and its value is 0.9 generally.

Values calculated for gradient descent momentum:

$$v_{dw} = b \cdot v_{dw} + (1-b) \cdot dw \dots\dots\dots \text{equation 14}$$

$$v_{db} = b \cdot v_{db} + (1-b) \cdot db \dots\dots\dots \text{equation 15}$$

$$W = W - \alpha \cdot v_{dw} \dots\dots\dots \text{equation 16}$$

$$b = b - \alpha \cdot v_{db} \dots\dots\dots \text{equation 17}$$

The values calculated for RMSprop optimizer:

$$v_{dw} = b \cdot v_{dw} + (1-b) \cdot dw^2 \dots\dots\dots \text{equation 18}$$

$$v_{db} = b \cdot v_{db} + (1-b) \cdot db^2 \dots\dots\dots \text{equation 19}$$

The values obtained from v from equations 18 and 19 of the rmsprop optimizer are used in equations 20 and 21 to update weights and biases during the training.

$$W = W - \alpha \cdot \frac{dw}{\sqrt{v} + \epsilon} \dots\dots\dots\text{equation 20}$$

$$b = b - \alpha \cdot \frac{db}{\sqrt{v} + \epsilon} \dots\dots\dots\text{equation 21}$$

The following notations are used here :

W : weights

b : biases

dw : partial derivative of the weights

db : partial derivative of the biases

β : momentum value

α : learning rate

ϵ : constant epsilon

CHAPTER 7 : RESULTS

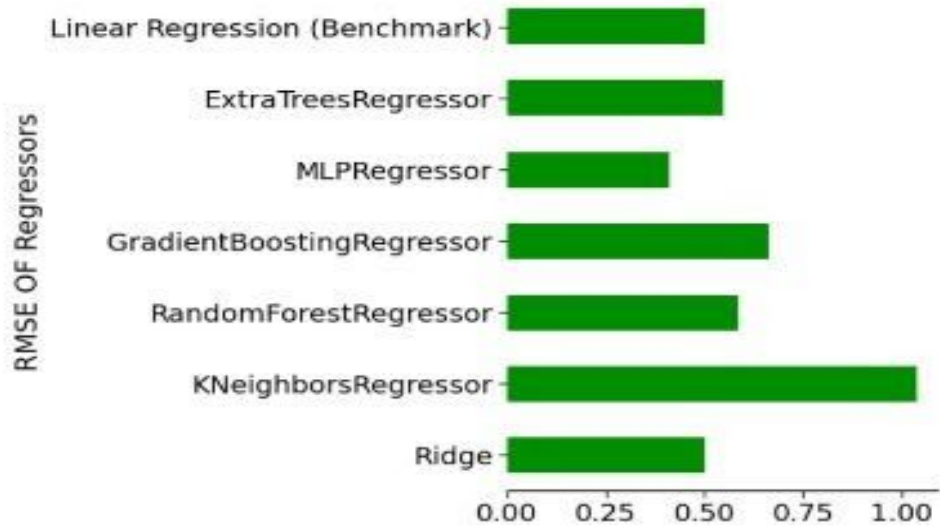


Fig 7. RMSE values of all the Regressors



Fig 8. MAPE values of all the regressors

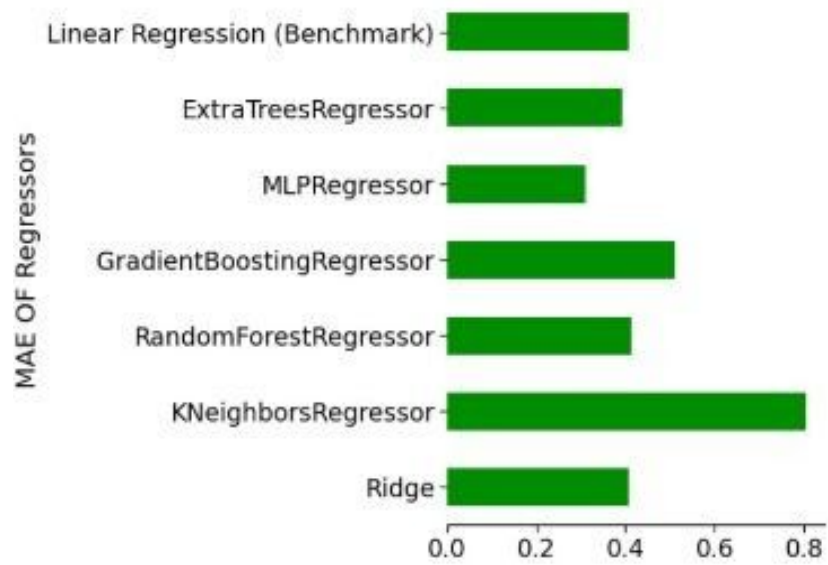


Fig 9. MAE of Regressors

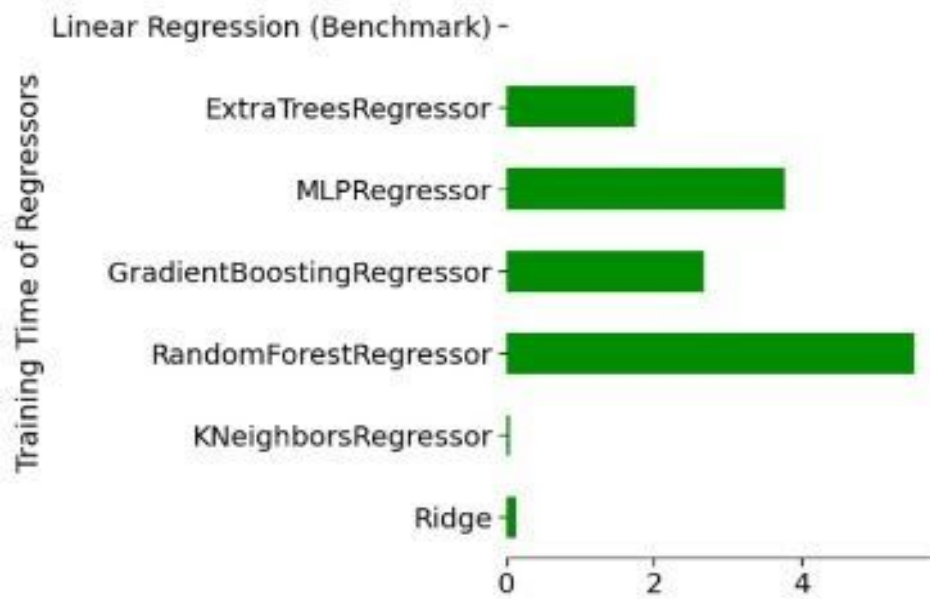


Fig 10. Training time of Regressors

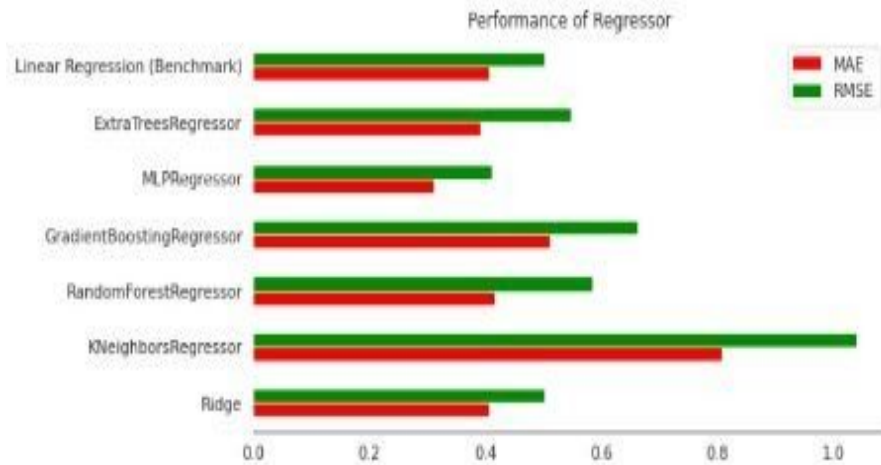


Fig 11. All regressors MAE and RMSE

| | Training Times | Training Scores | Testing Scores | RMSE | MAE | MAPE |
|-------------------------------|----------------|-----------------|----------------|----------|----------|----------|
| Ridge | 0.131169 | 0.984576 | 0.983428 | 0.501693 | 0.407092 | 0.407092 |
| KNeighborsRegressor | 0.0487008 | 0.980788 | 0.9289 | 1.03915 | 0.806351 | 0.806351 |
| RandomForestRegressor | 5.56136 | 0.998458 | 0.977519 | 0.584323 | 0.414538 | 0.414538 |
| GradientBoostingRegressor | 2.68983 | 0.987917 | 0.971156 | 0.661867 | 0.509457 | 0.509457 |
| MLPRegressor | 3.78176 | 0.992753 | 0.988944 | 0.40978 | 0.311283 | 0.311283 |
| ExtraTreesRegressor | 1.75971 | 1 | 0.98026 | 0.547547 | 0.391871 | 0.391871 |
| Linear Regression (Benchmark) | 0.00402188 | 0.984576 | 0.983454 | 0.5013 | 0.406677 | 0.406677 |

Fig 12. Comparison table of all the regressors

Seaborn and matplotlib are used to plot the results. From the Figures 7,8,9 and 10 where we are comparing the RMSE, MAE and MAPE value of all the regressors it is clear that the lowest RMSE, MAPE, Testing score and MAE value is for MLP regressor. Training score is not the highest but not too low for MLP regressor. Therefore, the MLP regressor is better than the other techniques which signify that a perceptron based network or a neural network would give the best performance on forecasting problems. Now we know LSTM is also composed of neural networks and is an improvement of the MLP networks. Comparing the MLP regressor to the LSTM network

LSTM results:

TestMAPE: 0.41 MAPE

TestMSE: 0.27 MSE

Test_RMSE: 0.51 RMSE

MLP results:

TestMAPE: 0.311283 MAPE

TestMSE: 0.311283 MSE

Test_RMSE: 0.40978 RMSE

Though MAPE and RMSE values of MLP are better than LSTM . But, for the purpose of forecasting we would require the properties of learning information from the previous layers and a long chain , offered by LSTM . In Fig.13 we can see that the dotted black line represents the original data for power and the green line represents the predicted power using the LSTM algorithm. Hence our model forecasts power correctly.

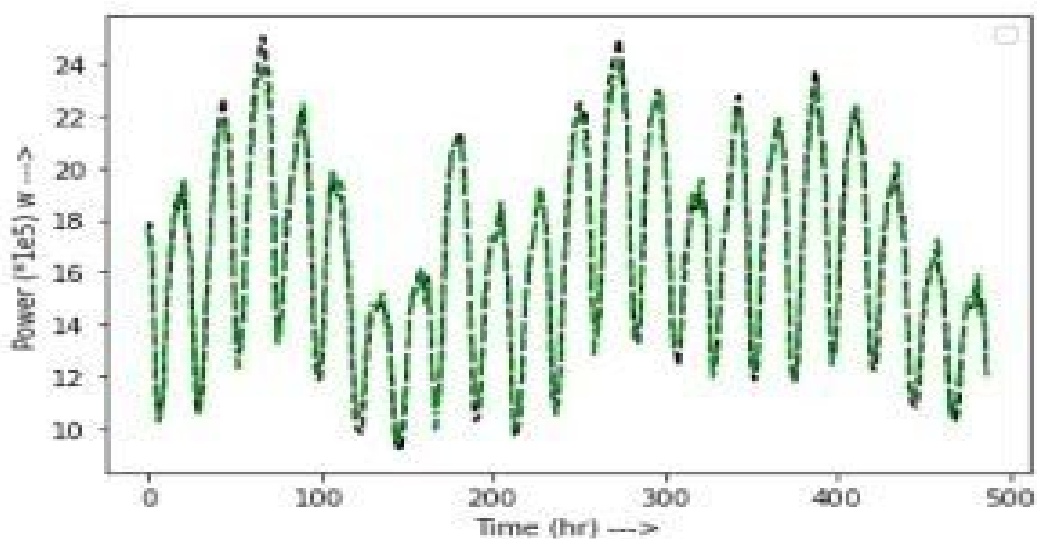


Fig 13. output of LSTM Network predictions vs original power.

CONCLUSION

We can see from the data that MLP is always a better choice as compared to all other algorithms for forecasting as the key performance values of MLP method are better than the rest of the algorithms .

However, MLP alone does not support the operation of remembering long term datas . Therefore we introduced a looping or recurrent loop method where all the weights and biases for all the layers were the same unlike MLP . Recurrent Neural Network was an improvement of the MLP network.

This Recurrent Neural Network However faces the problem of vanishing gradient when the data to be remembered becomes too large. To counter this problem LSTM was introduced which provided the benefits of MLP and Recurrent neural networks and is an improvised form of both the networks which support larger data memory and we can anytime use the forget gate and input gate to remove or add new data as per our requirements .

Hence We have successfully implemented LSTM neural networks to forecast load power studying and implementing all the other algorithms that can be used for forecasting . Concluding that LSTM is the best algorithm till now for load forecasting purposes.

REFERENCES

1. Fangchen Su, Yinliang Xu, Xiaoying Tang , “Short and mid-term load forecasting using machine learning techniques”, IEEE 2017 china International Electrical and Energy Conference. 21 June 2018 .
2. Gross, G., and Galiana, F. D., 1987, “Short term load forecasting”, Proceedings of the IEEE, 75, 1558±1573.
3. Tomas Vantuch, Aurora Vidal, Alfonso P. Ramallo-Gonzalez, Antonio F. Skarmeta, “Machine learning based electric load forecasting for short and long-term period”, 2018 IEEE 4th World Forum on Internet of Things(WF-IoT).
4. SRINIVASA N, D., and LEE, M. A., 1995, “Survey of hybrid fuzzy neural approaches to electric load forecasting”, Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Part 5, Vancouver, BC, pp. 4004±4008.
5. BUNN, D. W., and FARMER, E. D., 1985, “Review of Short-term Forecasting Methods in the Electric Power Industry” ,New York:Wiley, pp. 13±30
6. Stefan Hosein, Patrick Hosein, “Load forecasting using deep neural networks”, 2017 IEEE Power and Energy Society Innovative Smart Grid Technologies Conference(ISGT).
7. Wan He, “Load Forecasting via Deep Neural Networks”, Published in Procedia Computer Science, volume 122, 2017, pages 308-314, Elsevier.
8. Papalexopoulos, A. D., and Hesterberg, T. C., 1990, “ A regression-based approach to short-term load forecasting”, IEEE Transactions on Power Systems, 5, 1214±1221.
9. Haida, T., Muto, S., Takahasi, Y., and Ishi, Y., 1998, “Peak load forecasting using multiple-year data with trend data processing techniques”, Electrical Engineering in Japan, 124 , 7±16
10. Chujie Tian, Jian Ma, Chunhong Zhang and Panpan Zhan, "A Deep Neural Network Model for Short-Term Load Forecast Based on Long Short-Term Memory Network and Convolutional Neural Network", Energies 2018,11,3493; doi:10.3390/en11123493.
11. Vardan, S., and Makra M, E. B., 1996, “Harmonic load identification and determination of load composition using a least squares method”, Electric Power Systems Research, 37, 203±208.

12. HYDE, O., and HODNETT, P. F., 1997b, "Adaptable automated procedure for short-term electricity load forecasting", IEEE Transactions on Power Systems, 12, 84-94.
13. P. Zhang, X. Wu, X. Wang, and S. Bi, "Short-term load forecasting based on big data technologies," CSEE Journal of Power and Energy Systems, vol. 1, pp. 59-67, 2015
14. B. Stephen, X. Tang, P. R. Harvey, S. Galloway, and K. I. Jennett, "Incorporating Practice Theory in Sub-Profile Models for Short Term Aggregated Residential Load Forecasting," IEEE Transactions on Smart Grid, vol. PP, pp. 1-8, 2016.
15. M. Chaouch, "Clustering-Based Improvement of Nonparametric Functional Time Series Forecasting: Application to Intra-Day Household-Level Load Curves," IEEE Transactions on Smart Grid, vol. 5, pp. 411-419, 2014.
16. M. Ghofrani, M. Hassanzadeh, M. Etezadi-Amoli, and M. S. Fadali, "Smart meter based short-term load forecasting for residential customers," in North American Power Symposium (NAPS), 2011, pp. 1-5.
17. S. Ryu, J. Noh, and H. Kim, "Deep neural network based demand side short term load forecasting," in 2016 IEEE International Conference on Smart Grid Communications (SmartGridComm), 2016, pp. 308-313.
18. D. L. Marino, K. Amarasinghe, and M. Manic, "Building energy load forecasting using Deep Neural Networks," in IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society, 2016, pp. 7046-7051.
19. Weicong Kong, Zhao Yang Dong, David J. Hill, Yuan Zhang, "Short-Term Residential Load Forecasting based on LSTM Recurrent Neural Network", DOI 10.1109/TSG.2017.2753802, IEEE Transactions on Smart Grid.
20. Sumit Kumar, Lasani Hussain, Sekhar Banarjee, Motahar Reza, "Energy Load Forecasting using Deep Learning Approach-LSTM and GRU in Spark Cluster", 2018 Fifth International Conference on Emerging Applications of Information Technology. DOI 10.1109/EAIT.2018.8470406.
21. Long C. Nguyen, H. Nguyen-Xuan, "Deep learning for computational structure optimization", ISA Transactions Volume 103, August 2020, Pages 177-191.
22. N. Amral, C.S. Ozveren, D. King, "Short term load forecasting using Multiple Linear Regression", 2007 42nd International Universities Power Engineering Conference.

23. Grzegorz Dudek, "Short-Term Load Forecasting Using Random Forests", *Intelligent Systems 2014* pp 821-828, Part of *Advances in Intelligent systems and computing*, Elsevier.
24. Xianlong Lv, Xingong Cheng, YanShuang, TANG Yan-mei, "Short-term Power Load Forecasting Based on Balanced KNN", *IOP Conf. Series: Materials Science and Engineering* 322 (2018) 072058 doi:10.1088/1757-899X/322/7/072058.
25. Grzegorz Dudek, "Multilayer perceptron for short-term load forecasting: from global to local approach", *Neural Computing and Applications* 32, 3695-3707(2020), 14th March 2019, Elsevier.
26. Muhammad Waseem Ahmad, Monjur Mourshed, Yacine Rezgui, "Tree-based ensemble methods for predicting PV power generation and their comparison with support vector regression", *Energy*, Volume 164, 1 December 2018, Pages 465-474, Elsevier.
27. Chung Ming Cheung, Rajgopal Kannan, Viktor K. Prasanna, "Temporal ensemble learning of univariate methods for short term load forecasting", 2018 *IEEE Power and Energy Society Innovative Smart Grid Technologies Conference(ISGT)*.
28. Park, Dong C. et al. "Electric load forecasting using an artificial neural network". *IEEE transactions on Power Systems*, Vol. 6, Issue 2, 1991, pp 442-449.
29. Hecht-Nielsen, Robert. "Theory of the backpropagation neural network". *Neural networks for perception*, 1992, pp 65-93.
30. Bakirtzis, A. G., et al. "A neural network short term load forecasting model for the Greek power system". *IEEE Transactions on power systems*, Vol 11, Issue 2, 1996, pp 858-863.
31. Park, Dong C., et al. "Electric load forecasting using an artificial neural network". *IEEE transactions on Power Systems*, Vol 6, Issue 2, 1991, pp 442-449.
32. Hao, Alex D. Papalexopoulos Shangyou. "An implementation of a neural network based load forecasting model for the EMS". *IEEE transactions on Power Systems*, Vol 9, Issue 4, 1994.
33. Paatero, Jukka V., and Peter D. Lund. "A model for generating household electricity load profiles". *International journal of energy research*, Vol 30, Issue 5, 2006, pp 273-290.

34. Pardo, Angel, Vicente Meneu, and Enric Valor. "Temperature and seasonality influences on Spanish electricity load". *Energy Economics*, Vol 24, Issue 1, 2002, pp 55-70.
35. Hong, Tao, and Shu Fan. "Probabilistic electric load forecasting: A tutorial review". *International Journal of Forecasting*, Vol 32, Issue 3, 2016, pp 914-938.
36. Soares, Lacir J., and Marcelo C. Medeiros. "Modeling and forecasting short-term electricity load: A comparison of methods with an application to Brazilian data". *International Journal of Forecasting* Vol 24, Issue 4, 2008, pp 630-644.
37. Amaral, Luiz Felipe, Reinaldo Castro Souza, and Maxwell Stevenson. "A smooth transition periodic autoregressive (STPAR) model for short-term load forecasting". *International Journal of Forecasting*, Vol 24, Issue 4, 2008, pp 603-615.
38. Antoniadis, Anestis, et al. "A prediction interval for a function-valued forecast model: Application to load forecasting". *International Journal of Forecasting*, Vol 32, Issue 3, 2016, pp 939-947.
39. Wang, Pu, Bidong Liu, and Tao Hong. "Electric load forecasting with recency effect: A big data approach". *International Journal of Forecasting*, Vol 32, Issue 3, 2016, pp 585-597.
40. Berk, K., A. Hoffmann, and A. Müller. "Probabilistic forecasting of industrial electricity load with regime switching behavior". *International Journal of Forecasting*, Vol 34, Issue 2, 2018, pp 147-162
41. Quan, Hao, Dipti Srinivasan, and Abbas Khosravi. "Short-term load and wind power forecasting using neural network-based prediction intervals". *IEEE transactions on neural networks and learning systems*, Vol 25, Issue 2, 2014, pp 303-315.
42. Jurado, Sergio, et al. "Hybrid methodologies for electricity load forecasting: Entropy-based feature selection with machine learning and soft computing techniques". *Energy* Vol 86, 2015, pp 276-291.
43. Chae, Young Tae, et al. "Artificial neural network model for forecasting sub-hourly electricity usage in commercial buildings". *Energy and Buildings* Vol 111, 2016, pp 184-194.

44. Panapakidis, Ioannis P., and Athanasios S. Dagoumas. "Day-ahead electricity price forecasting via the application of artificial neural network based models". *Applied Energy* Vol 172, 2016, pp 132- 151.
45. Hippert, H. S., D. W. Bunn, and R. C. Souza. "Large neural networks for electricity load forecasting: Are they overfitted?". *International Journal of forecasting* Vol 21, Issue 3, 2005, pp 425-434.
46. Tripathi, M. M., K. G. Upadhyay, and S. N. Singh. "Short-term load forecasting using generalized regression and probabilistic neural networks in the electricity market". *The Electricity Journal* Vol 21, Issue 9, 2008, pp 24-34.