2019 5th International Conference for Convergence in Technology (I2CT)
Pune, India. Mar 29-31, 2019

Retail Demand Forecasting: a Comparison between Deep Neural Network and Gradient Boosting Method for Univariate Time Series

Karan Wanchoo
Gurgaon, India
wanchoo.karan85@gmail.com

Abstract—**The traditional retailer had to deal with stocking of inventory and merchandising based on raw estimates of his daily or weekly product sales to make available the right product at the right time for his customers. The modern retailer has to deal with same store operations such as inventory planning and merchandising but with several forecasting techniques at hand such as ARIMA, Holt-Winters, Exponential Smoothing, Neural Networks and Gradient Boosting Methods, he is efficiently and more accurately able to predict store-product sales to replenish inventory with minimum capital requirement. This paper aims at implementation and comparison of two popular and intriguing machine learning techniques, Deep Neural Network (DNN) and Gradient Boosting Method (GBM) for univariate time series sales data at store-day level of a German retail giant. Several studies have leveraged DNN and GBM techniques for forecasting but on a multivariate time series data, that is predicting with the help of causal factors (variables). The paper discusses the feasibility and application of both these techniques on a univariate time series because not every retailer may have access to adequate supply chain metrics to create a multivariate model. The data used for analysis has been picked up from Kaggle. Performance measures used are mean absolute error (MAE) and root mean square error (RMSE). The DNN and GBM models have been developed in python using Keras and Ensemble packages.**

Keywords—deep neural networks, gradient boosting, retail demand forecast, univariate time series, keras

I. INTRODUCTION

Real-world sales series such as retail sales often exhibit non-linear patterns (due to seasonality, trend, new product models, etc.)[5]. Conventional time series methods such as exponential smoothing or ARIMA models are unable to accommodate nonlinearity directly. Therefore, Smoothing/ARIMA model is not able to capture nonlinear patterns that are commonly seen in many business and economic time series. More specifically, ARIMA and AR models require the time series to be stationary, i.e, no seasonality and trend should exist. This is not true in the case of real-world series such as our retail sales time series.

Holt-winter's ability to incorporate seasonality does give us some leverage over ARIMA or Smoothing, but falls short on accuracy and robustness for high fluctuating series.

So in order to capture nonlinearity better with higher accuracy for retail sales time series, it is recommended to explore advanced forecasting methods such as Neural Networks and Gradient Boosting Methods. ARIMA, and smoothing methods assume the model form before the model is built from the data and so a reasonable amount of knowledge about the data set and data creation process is required for forecasting. Neural Networks mathematically emulate part of our biological neural system and with its huge number of parameters utilized in engendering the model, the neural network model effectively represents the input-output relationship of a nonlinear system while being immune to noise. Recent studies demonstrate the effectiveness of neural networks such as recurrent networks as a forecasting method in Electricity Consumption and Wind Speed domain [1, 7]. Feed forward and recurrent neural networks have proven to be useful for business forecasting as well [6, 4].

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of weak prediction models, typically decision trees. The idea behind gradient boosting is to recurrently leverage the patterns in residuals and strengthen or boost a model with weak predictions to make it better.

This study implements the neural network and gradient boosting algorithm to the univariate time series (retail store sales) keeping in view the absence of different features such as store attributes, competition variables, pricing and assortment variables, that could have helped in explaining and forecasting the store sales pattern.

II. DATA REPRESENATION

The data is of a German drug store retailer Rossmann Stores and was available on Kaggle as a part of a sales forecast competition[8]. For the purpose of implementation and comparison of both the techniques, one random store was chosen out of 1115 stores and its data was prepared and split into train, validation and test. Below is the data description.

TABLE I. DATA DESCRIPTION

Data	**Total sales unit of store**
Data granularity level	Daily
Start Date	1/6/2013
End Date	7/31/2015
Total Observations	937
Train Observations	748 (80%)
Test Observations	182 (20%)

The store being closed on Sundays reported zero sales for that day. The data series was hence observed and used in groups of 7 days to capture the trend and cyclicity better as shown in Figure 1 below.

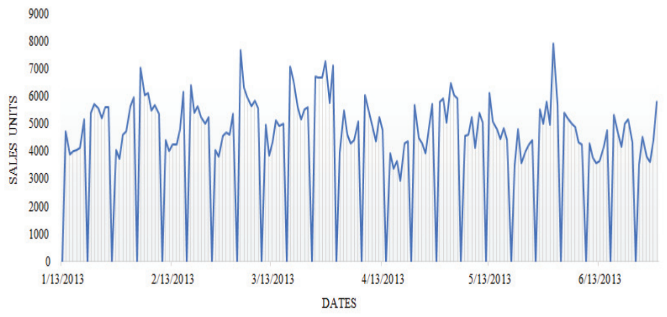

Fig.1. Sample of store sales data exhibiting zero sales on Sundays

The neural network model shall leverage sales in groups of 7 days to train itself to predict the 8th day sales while the gradient boosting method will use the 7-day sales as 7 predictors and 8th day sale as the response variable.

III. APPROACH AND ARCHITECHTURE

The study shall discuss the Deep Neural Network approach first along with its architecture used to forecast the store sales and then elaborate upon Gradient Boosting Method using the same store data.

A. *Deep Neural Network Approach*

With the objective of generating store-day level forecast with high accuracy the setup was created on Python 3.6 using Keras library to leverage astute machine learning frameworks with great customizability and data handling capacity.

- Univariate time series forecasting is performed on a univariate time series as input. A single time series for store has been picked at day level and iteratively for every store, the time series can be subjected to a deep neural network and forecasts can be generated.
- The primary layer of a neural network is known as the input layer and has input neuron count equal to the number of inputs, as shown in Figure 2. Hidden layers add complexity and non-linearity to the system which can be controlled by the forecaster depending upon the time series.
- Since we have a univariate time series at day level, we break the time series into a group of *k* days. Those *k* days are fed in the input layer through *k* neurons.
- The output layer will have one neuron which will be the prediction of the network for (*k*+1)th day. That is, if I have *k*=7, it implies I will input the data for first 7 days in the input layer and forecast for 8th day in the output layer, as shown in Figure 2.
- The window of size *k* is then shifted one step ahead.
- With this approach the network is trained using the training data and single step forecasts are generated.
- Figure 2. Illustrates a feed forward network with four hidden layers and is a deep neural network. All possible connections between nodes have not been shown.

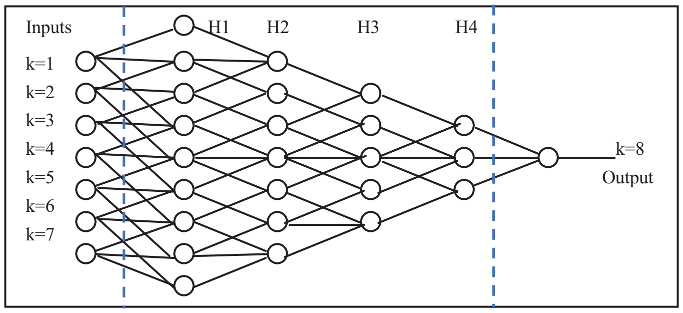

Fig.2. Deep neural network model for univariate time series

With *k* neurons in the input layer, one step ahead, Y_N being the most recent daily sale and Y_{N+1} being the predicted value one step ahead, we obtain (1):

$$Y_{N+1} = f\,(Y_N, Y_{N-1}, \dots , Y_{N-k+1}) \qquad (1)$$

B. *Choosing Parameters for Deep Neural Network*

The list of parameters that drove the accuracy of the network is given below:

- Number of hidden layers and hidden neurons
- Activation functions to be used in each layer
- Weight initialization method
- Optimizer function and learning rate
- Number of epochs
- Batch size for each iteration

The loss function used in this case is mean squared error since we are predicting continuous values. One can switch to cross entropy loss function if the objective is classification. A loss function is a measure of difference between observed and predicted value. Loss functions are minimized to find out optimum weights for the inputs. Gradient descent is an approach to minimize the loss function to determine optimum weights. This is usually a first order derivative of the loss function.

Out of the six parameters mentioned above, optimizer function enables the user to specify the type of optimization he would like to apply in order to minimize the cost function and solve for the weights. Keras offers a range of optimizer functions like stochastic gradient descent (SGD), Adagrad, RMSProp, Adadelta, Adam, Adamax and Nadam. As mentioned in [9] Adagrad works well for sparse settings, but its performance has been observed to deteriorate in settings where the loss functions are nonconvex and gradients are dense due to rapid decay of the learning rate in these settings since it uses all the past gradients in the update. Adam, a variant of Adagrad, has been proposed as an efficient method of stochastic optimization in [2] as it proves to be robust and well-suited to a wide range of non-convex optimization problems and has been used as an optimizer in the current forecasting problem.

The learning rate for Adam in this paper has been subjected to variations ranging from 1% to 0.05% to test for better convergence and avoid vanishing and exploding gradient problem.

Epochs and batch size are two parameters that are tested in conjugation. One epoch is when an entire dataset is passed through the neural network only once. However, one epoch can be large enough to be fed into the system at once and so we divide it in several smaller batches. Increasing the number of epochs implies more number of times the weights are changed in the neural network and the curve goes from underfitting to optimal to overfitting curve. Batch size is the total number of training examples present in a single batch and is chosen in multiples of two. Batch size also implies number of samples per gradient update. An iteration would mean the number of batches needed to complete one epoch. For example, we have 2000 training examples that we are going to use. We can divide the dataset of 2000 examples into batches of 500 and then it will take 4 iterations to complete 1 epoch. Several combinations of epochs and batch sizes have been tested to retrieve a good fit.

Activation functions and weight initialization methods are important in order to eliminate saturation within top layers of the deep network. In [10] the paper talks about logistic sigmoid activation being unsuited for deep networks with random initialization because of its mean value, which can drive especially the top hidden layer into saturation. The paper yields several conclusions one of them being that classical neural networks with sigmoid or hyperbolic tangent units and standard initialization fare rather poorly, converging more slowly and apparently towards ultimately poorer local minima. Furthermore, Glorot and Bengio comment that sigmoid activations (not symmetric around 0) should be avoided when initializing from small random weights, because they yield poor learning dynamics, with initial saturation of the top hidden layer. This paper takes cue from the above conclusions and uses ReLu activation function with Glorot Uniform weight initialization method. ReLu or rectified linear unit function (derived from ELU or exponential linear unit) gives an output Y if X is positive and 0 otherwise.

$$Y(x) = \max(0, x) \tag{2}$$

Keras offers the softmax, elu, selu, relu, softplus, softsign, tanh, sigmoid, hard-sigmoid and linear activation functions. The package also renders a good number of weight initializers such as random-normal, random-uniform, truncated-normal, variance-scaling, lecun-uniform, lecun-uniform, glorot-normal, he-normal, he-uniform, identity, orthogonal, constants and zero.

Sigmoid and tanh functions are continuously firing neurons in an analog manner making the activation process dense, unlike ReLu that may cease half of the neurons (in case of negative inputs) making the network light and efficient.

The optimal number of hidden layers to be incorporated into the network is not based on any thumb rule or an axiom but is an outcome of hit and trials which the forecaster has to carefully observe and choose. However, one can rely on the proposition that with increase in number of hidden layers, one increases the complexity in the system and tries to capture complex patterns in the data layer after layer. In this study we finalized three hidden layers after several permutations to give us optimum results.

C. *Gradient Boosting Method Approach*

With the objective of generating store-day level forecast with high accuracy the setup was created on Python 3.6 using Ensemble library from Sklearn package to leverage astute machine learning frameworks with great customizability and data handling capacity.

- Similar to previous approach, univariate time series forecasting is performed on a univariate time series as input. A single time series for store has been picked at day level and iteratively for every store, the time series can be subjected to a deep neural network and forecasts can be generated.
- The input data is prepared in the same way as was for the neural network forecasting. First seven days sales are used as seven inputs/regressors and sale for eighth day serves as the output/response.
- Gradient boosting regressor in python builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage a regression tree is fit on the negative gradient of the given loss function.
- Boosting is engendered on the principle that a collection of weak learners can be combined to produce a strong learner. Boosting methods use additive training methods that add a new weak learner to the model in each step. So, if $f_t(X)$ represents the complete model after *t* rounds and $g(X)$ is the new tree to be added to the model, we can represent the relationship as:

$$f_t(X) = f_{t-1}(X) + g(X) \tag{3}$$

D. *Choosing Parameters for GBM*

The list of parameters that built the gradient boosted trees is:

- Maximum depth of a tree

- Subsample: Fraction of observations to be selected for each tree
- Number of sequential trees to be modeled
- Learning rate
- Random number seed
- Loss function to be minimized in each split

The maximum depth of the individual regression estimators limits the number of nodes in the tree [3]. This parameter controls high variance as higher depth will allow model to learn relations very specific to a sample. This study experimented with maximum depths ranging from 3 to 6.

Subsample is the fraction of samples to be used for fitting the individual base learners or individual trees. The selection of samples is done by random sampling. Values slightly less than 1.0 are preferred because that leads to reduction in variance. This study experimented with subsamples ranging from 0.25 to 1.

Number of sequential trees to be modeled is the number of boosting stages to perform [3]. Since gradient boosting is robust to over-fitting so a large number of sequential trees usually results in better performance. This study experimented with number of trees ranging from 40 to 300.

Learning rate determines the impact of each tree on the outcome. There is a decent trade-off between the boosting stages and learning rate. This study prefers using learning rate 0.1.

Loss function to be minimized in each split can have various values for classification and regression case. Loss can be least square regression, least absolute deviation, combination of the two known as huber and quantile regression. This study uses least square regression as the loss function.

Other parameters which can be considered for tweaking the GBM model are minimum sample required to split an internal node, minimum samples required to be at a leaf node, minimum impurity split and maximum leaf nodes.

IV. RESULTS AND DISCUSSION

We have discussed so far the approach and parameters for our two forecasting techniques and now we shall begin with the results recorded and comparison of the techniques.

A. *Deep neural network results*

Table II describes the deep neural network architecture used in this study. Neural network was experimented with different number of hidden layers, hidden neurons, batch size and number of epochs to finally come up with the architecture represented in Table II. The input layer shall always have neurons equal to the number of inputs. Other parameters include Glorot-Uniform weight initializer, Adam optimizer function, learning rate of 0.05%, 70 epochs, batch size of 4 and mean squared error as the loss function.

Fig. 3 shows how the model loss decreases exponentially with increase in epochs as the iterations progress and weights are updated in the neural network.
Fig. 4 shows the actual vs predicted store sales graph for the test data set.

TABLE II. DNN ARCHITECTURE

	Neurons	**Outputs**	**Activation Function**
Input Layer	7	21	Rectified Linear Unit (ReLu)
Hidden Layer-1	21	14	ReLu
Hidden Layer-2	14	10	ReLu
Hidden Layer-3	10	10	ReLu
Output Layer	1	1	

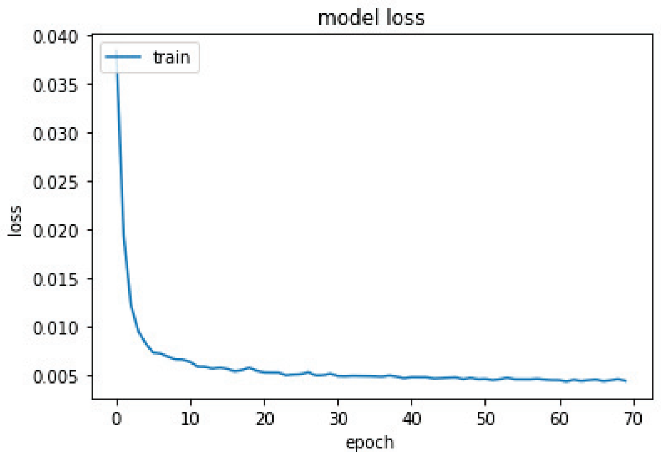

Fig.3. Model loss vs epoch plot

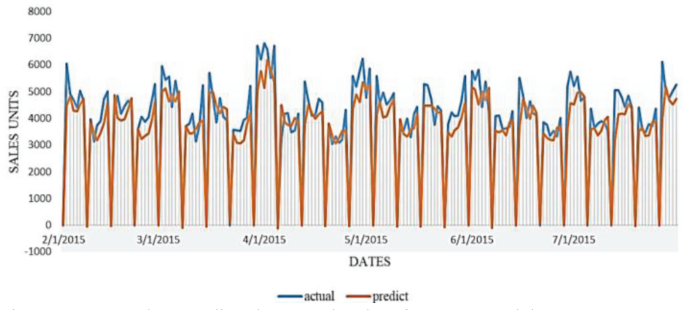

Fig.4. Actual vs predicted store sales data for DNN model

B. *Gradient Boosting Method Results*

The results recorded in Table II exhibit the impact of different parameters have on the forecast generated by gradient boosted method. We notice runs 1,9,12 and 14 are high bias runs, that is under fitting, in which the train rmse and train mae are the poorest among the rest but the test errors are not too far off from the train. Runs 4,6,7,10 and 11 are high variance runs in which train rmse and mae are quite low but far off from the test errors exhibiting overfitting of model. Runs 2,13 and 15 are high bias and high variance runs because of poor train errors and great difference between train and test errors. Runs 3,5 and 8 are good fit by the model because of low train errors and test errors being not too far off in range.

Increasing the number of sequential trees helps in dropping the train error magnitude as can be seen from runs 1,3 and 10,11.

Increasing the max depth from 3 to 4 resulted in decrease in train errors but increase in variance as evident from runs 3,4 and 5,6 whereas decreasing the max depth from 5 to 4 resulted in increase in train errors.

This study prefers the best parameter configuration to be for runs 3,5 and 8 and the actual vs predicted graph for run 8 for test data has been shown in Fig. 5.

TABLE III. IMPACT OF PARAMETERS ON GBM

Run	Learning Rate	Trees	Max Depth	Sub Sample	Train RMSE	Test RMSE	Train MAE	Test MAE
1	0.1	100	3	1	0.047	0.040	0.037	0.016
2	0.05	100	3	1	0.062	0.042	0.049	0.016
3	**0.1**	**300**	**3**	**1**	**0.028**	**0.041**	**0.022**	**0.016**
4	0.1	300	4	1	0.014	0.041	0.012	0.016
5	**0.1**	**100**	**5**	**1**	**0.023**	**0.041**	**0.018**	**0.015**
6	0.1	100	6	1	0.015	0.043	0.011	0.016
7	0.1	200	5	1	0.011	0.040	0.008	0.015
8	**0.1**	**100**	**5**	**0.5**	**0.028**	**0.039**	**0.023**	**0.015**
9	0.1	100	5	0.25	0.044	0.039	0.036	0.015
10	0.1	100	5	0.75	0.022	0.043	0.018	0.016
11	0.1	200	5	0.75	0.010	0.043	0.008	0.016
12	**0.1**	**50**	**5**	**0.8**	**0.038**	**0.044**	**0.029**	**0.016**
13	0.1	50	4	0.8	0.051	0.037	0.040	0.014
14	0.1	40	5	0.8	0.046	0.044	0.036	0.017
15	0.1	40	4	0.8	0.061	0.039	0.049	0.015

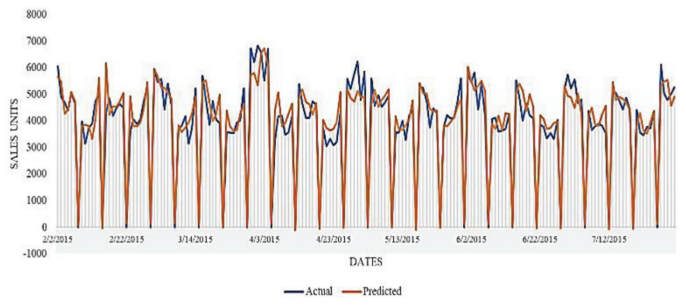

Fig.5. Actual vs predicted store sales data for GBM model

C. *Comparison of Results*

Gradient boosting performs better than deep neural network in this study but is very sensitive to over fitting and if parameters are scrutinised and adjusted properly, gradient boosted trees can yield exemplary results. Neural networks on the other hand often suffer from vanishing and exploding gradient problems and that can prevent the forecaster from reducing the prediction error further. GBM also yields a relative variable importance table that shows relative importance of input variables in predicting the output variable. However in our study causal factors are not in scope and hence we omit it.

TABLE IV. COMPARISON OF RMSE AND MAE

S.No.	**Model**	**Train RMSE**	**Test RMSE**	**Train MAE**	**Test MAE**
1	DNN	0.064	0.092	0.043	0.069
2	GBM (run 8)	0.028	0.039	0.023	0.015

V. CONCLUSION

This study successfully demonstrates the application and comparison of two prominent machine learning techniques, deep neural network and gradient boosting method for forecasting a univariate sales time series. Forecasting in real world is used to fulfil challenging business objectives such as efficient inventory planning or establishing product-location relationships and hence holds immense importance across industries especially retail industry. When we fall short of data and are unable to gather causal factors impacting store sales or product sales, we can rely on the study made in this paper and go ahead with a univariate sales forecast using either of the two techniques or may even create an ensemble of both these techniques. Gradient boosting trees have performed better in this case but are subject to overfitting whereas neural networks are vulnerable to vanishing and exploding gradients. The aim of this paper was to illustrate the application of these two techniques, apprise of python as an efficient coding language, keras and ensemble packages.

REFERENCES

[1] A. Marvuglia and A. Messineo, "Using Recurrent Artificial Neural Networks to Forecast Household Electricity Consumption", in Energy Procedia, 14, 2nd International Conference on Advances in Energy Engineering (ICAEE), Bangkok, Thailand, 2011, pp. 45–55.

[2] D.P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”, in Proceedings of 3rd International Conference on Learning Representations, 2015.

[3] J. Friedman, Greedy Function Approximation: A Gradient Boosting Machine, The Annals of Statistics, Vol. 29, No. 5, 2001.

[4] M. Orra and G. Serpen, "An Exploratory Study for Neural Network Forecasting of Retail Sales Trends Using Industry and National Economic Indicators", in Proceedings for the 4th Workshop on Computational Intelligence in Economics and Finance, Salt Lake City, USA, 2005, pp. 875–878.

[5] N. Morritz, L. Stefan, and V. Stefan, “Sales forecasting with partial recurrent neural networks: empirical insights and benchmarking results,” 48th Annual Hawaii International Conference on System Sciences, January 2015.

[6] P. Das and S. Chaudhury, "Prediction of retail sales of footware using feedforward and recurrent neural networks", Neural Computing & Applications, 16 (4–5), 2007, pp. 491–502.

[7] Q. Cao, B.T. Ewing, and M.A. Thompson, "Forecasting wind speed with recurrent neural networks", European Journal of Operational Research, 221, 2012, pp. 148–154.

[8] Rossmann Store Sales. Kaggle. Web. https://www.kaggle.com/c/rossmann-store-sales.

[9] S. Reddi, S. Kale, and S. Kumar, “On the convergence of Adam and Beyond”, in Proceedings of 3rd International Conference on Learning Representations, 2018.

[10] X.Glorot, Y, Bengio, “Understanding the difficulty of training deep feedforward neural networks”, in Proceedings of 13th International Conference on Artificial Intelligence and Statistics, 2010.