Volume: 6 Issue: X Month of publication: October 2018
DOI:

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 6.887
Volume 6 Issue X, Oct 2018- Available at www.ijraset.com

# Multivariate Demand Forecasting of Sales Data

Ajinkya Athlye[1], Angad Bashani[2]
[1,2]Computer Engineering, Savitribai Phule Pune University, Lalit Kala Kendra, Ganeshkhind Road, Aundh, Pune, 411007, Maharashtra, India

***Abstract: Demand Forecasting is vital for any industry that seeks to gain a competitive edge. It's important that an industry manages its inventory and sales in order to maximize profit. This paper's purpose is to propose a forecasting technique that uses Deep Learning, while drawing a comparative analysis with older forecasting methods namely AR and ARIMA. The study is conducted on real world data obtained from a furniture shop based in Pune, wherein the quantity of their most selling product is predicted for the next three months. External factors (temperature, CPI, unemployment rate, holiday) which would affect consumer demand were considered while forecasting future sales. With the help of deep learning models namely ANN, RNN, GRU and LSTMs, it is possible to learn complex non-linear trends and predict future sales accurately by making use of external factors along with the time-series trends. This paper focuses on comparing 6 Artificial Neural Network models (ANN), 36 Recurrent Neural Network models (RNN), and 3 time series models.***

***Index Terms: Demand forecasting, Recurrent neural networks, Artificial neural networks, Machine learning, Statistical learning***

## I. INTRODUCTION

Demand Forecasting is a means to predict future sales of a company in order to manage its inventory effectively. Accurate sales forecasting would mean a company can buy or manufacture its products more efficiently, hence maximizing profits and minimizing losses. The biggest advantage of accurate demand forecasting in a store is the dearth of excessive in stock keeping, and on the other hand, fulfillment of unattended orders due to lack of material in stock. This paper draws observations and subsequent conclusions based on the sales of tables of a furniture store, hence giving the store owners the ability to make informed decisions about how many tables they must order in the upcoming months. Thus, it is important to sustain an accurate demand forecasting tool in the supply chain industry.

Time series forecasting is dependent on the previous patterns of purchases by customers. An observed trend consists of two parts: a systematic part and a random part. Many complex trends exist in the systematic parts, mostly involving yearly trends and intra-year seasonalities. The random part is unaccounted for, and is seemingly hard to predict. Neural Networks have been observed to be useful in demand forecasting due to their ability to tackle non-linear data, and capture subtle pattern shifts and minor functional relationships within the empirical data. For this particular data set, it is observed that Recurrent Neural Networks (RNN) outclass the regular time-series techniques as well as Artificial Neural Networks (ANN) in terms of accuracy of predictions. This paper focuses on comparing several different training methods of the RNN, including the basic RNN Cell, Gated Recurrent Unit RNNs (GRU), and Long Short-Term Memory RNNs (LSTM), ANN, and the traditional time-series methods. [1]

Section 2 presents previous research conducted on demand analysis, and the use of ANN in forecasting. Section 3 explains the data used in all the models for predicting the desired outcome. Section 4 addresses the model performance measures picked for the algorithms used, and explains how they function. Section 5 focuses on the proposed methodologies adopted for the use-case of the paper and explains them. Section 6 puts forth the comparative analysis of all the models along with the performance measure of each training method. Section 7 compares and concludes the result of the proposed models, and suggests future ideas and extensions regarding demand forecasting for the store.

## II. LITERATURE REVIEW

Demand Forecasting is an old idea which has attracted the attention of several researches in the past. Several studies have been conducted which use mathematical tools to predict not only the buying pattern of customers, but also the demand for other entities, such as water. These studies are based on time-series models such as naive forecasting, moving-average, exponential smoothing, Box-Jenkins method and causal models such as auto regressors and econometric models.

[2] conducted a survey to determine the degree of familiarity and usage, accuracy obtained, and evaluation of different time-series forecasting techniques. [3] used space state models and ARIMA to forecast the sales of five different categories of women's footwear. While these methods performed well, they had a serious limitation of not being able to capture non-linear trends.

Moreover, if these methods are not carried out with expertise, there may be a mis-specification of defining the type of variable (independent and dependent), and subsequently end up with poor regression results.

There have been multiple studies conducted in forecasting the retail sales of textiles and clothes, especially in the fashion sector [4], [5], [6]. It cannot be resolutely said that the patterns and external factors observed in forecasting textiles and clothes applies to furniture forecasting, as not only does the quantity of product sold vary drastically, but also the demand criterion and use-cases do not match.

[7] conducted a study on the bullwhip effect, which arises due to the demand variability amplification along a supply chain from retailers to distributors. Ryan, Chen, and Simchi-Levi (2000) studied exponential smoothing forecast and how it affected the retailer on the bullwhip effect. [8] researched how the forecasting models performed on supply chains performance using a computer simulation.

[9] delineated the importance of selecting accurate forecasting techniques as it is proven that the use of naive forecasting, moving average, or demand signal processing will induce the bullwhip effect. Autoregressive linear forecasting, on the other hand, has been shown to diminish bullwhip effects, while outperforming naive and exponential smoothing methods.

The use of neural networks in demand forecasting overcomes many of these limitations, including the bullwhip effect. Neural networks have been mathematically demonstrated to be universal aggregators of functions [10].

Aburto and Weber(2007) showcased a hybrid intelligent architecture combining ARIMA models and NNs for demand forecasting in SCM and made an inventory management system for a Chilean supermarket. Al-Saba et al.(1999) & Beccali, et al (2004), operated on ANNs to forecast short or long term demands for electric load. Chiu and Lin(2004) showed how collaborative agents and ANN could work parallelly to enable collaborative SC planning with a computational framework for mapping the supply, production and delivery resources to the customer orders. [11] made the use of ANNs, RNNs, and SVMs, to forecast distorted demand (bullwhip effect) at the end of the SC. [12] used ANNs to forecast sales for a beverage selling company. Both the above papers displayed results where the forecasting ability of ANNs was better than that of their ARIMA counterparts. Despite the fact that there have been several studies conducted in demand forecasting using ANNs, there are very few who have used multivariate RNNs, along with their different types (LSTM and GRU).

## III. DATA

Several factors affect buying patterns, namely climate and socioeconomic variables. [13] For the walk-in customer, it is logical to presume that a person is more likely to step out of his house to buy luxury goods, such as antique furniture (the data in question), when the environment agrees with him.

For instance, upon observing the sales of tables over a year, it was found that there was a drop in sales during the monsoons. Similarly, socioeconomic variables such as unemployment rate of the city and Consumer Price Index (CPI) also contribute to the trend, especially for people belonging to the mid socioeconomic status.

This study uses a total of 6 variables. The date of the purchase, the total number of sales made in a week, temperature (in Celsius), CPI, Unemployment Rate (in percentage), and a true/false value indicating whether there was any significant holiday/festival in that week.

The weekly sales data was provided by a furniture company based in Pune, along with the date of purchase. A total of 3 years worth of sales was provided (2015-2017). The temperature for those given dates was obtained from the Indian Meteorological Department (IMD). The temperature for a week is calculated as:

$$t_{avg} = \frac{1}{7}\sum_{i=1}^{7} t_i$$

where, $t_{avg}$ is the average temperature for a week and $t_i$ is the temperature of the day $i$ of the week.

The CPI and Unemployment Rate was obtained from official government bodies (Reserve Bank of India). The final ‘Is Holiday’ variable was manually made and fit into the data.

All major festivities, holidays, events were marked as ‘True.’ This was constructed as it was found that around the time of Diwali, the sales increased substantially every year.

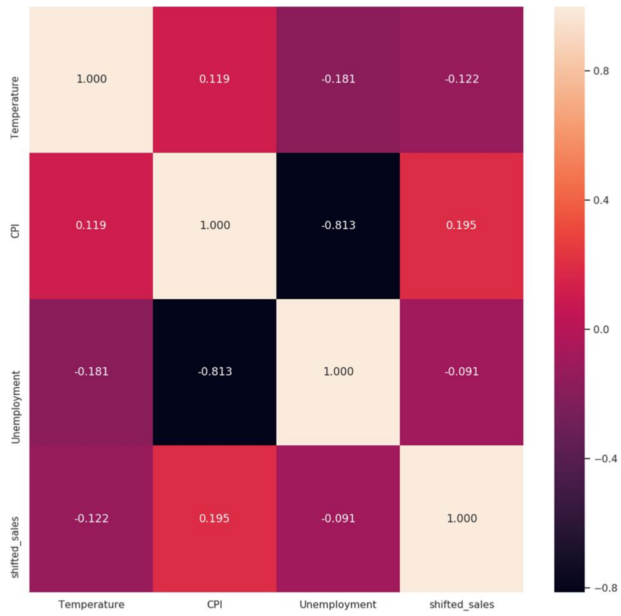


Fig. 1. Correlation of the future sales with Unemployment Rate, CPI, and Temperature.

Finally, all the data was collated into a single file. This data was split into training and test data sets for the models to operate on. The split was equal for all, wherein the last three months were used for the testing data set while the remainder was used for training. Hence, the forecast is made and compared for the last three months of purchase of the year 2017.

## IV. MODEL PERFORMANCE

### *A. Mean Absolute Percent Error (MAPE)*

Error measurement is crucial for tracking forecasting accuracy, monitoring exceptions and benchmarking models against one another. After each of the model structures is trained/tested using the training/testing data set, the performance can then be evaluated in terms of these statistical measures of goodness of fit. In order to provide a metric to gauge 'goodness of fit' between the observed and predicted values, the Mean Absolute Percent Error (MAPE) can be used. MAPE measures accuracy as the average unsigned percentage and is given as follows:

$$MAPE = \frac{1}{N}\sum_{i=1}^{N} \left|\frac{E_i - Q_i}{E_i}\right| \times 100$$

where, $E_i$ is the expected stock quantity; $Q_i$ is the predicted quantity found from AR, ARIMA, ANN, RNN, LSTM and GRU models, respectively. The smaller the value of MAPE, the better is the performance of the model.

### *B. Activation Functions*

When constructing Deep Learning Models, one of the primary considerations is which activation function to choose for the hidden and output layer that makes the model give accurate predictions. Without an activation function an NN would simply be a linear regression model. The activation function does a non-linear transformation on the weighted sum of inputs enabling NN models to fit complex curves that are not possible with simple linear regression. (1.a) exhibits the general formula for an activation function:

$$Y = g(\textstyle\sum_{i=1}^{N} (W_i \times X_i) + b)(1.a)$$

where, $Y$ is the output variable; $g$ is the activation function; $W_i$ and $X_i$ are the weight and input for the $i^{th}$ training example respectively; $b$ is the bias variable.

These activated values are primal in machine learning because backpropagation requires the gradients of the activated value to be supplied along with the error in order to determine the optimal NN parameter updates.

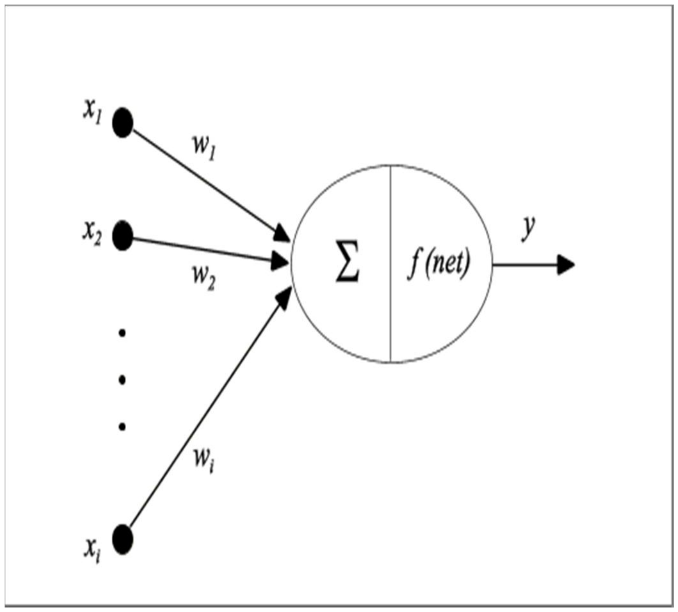


Fig. 2. Flow of an Activation function (Courtesy: www.researchgate.net)

In Figure 2, $x_1$, $x_2$, ..., $x_n$ are input vectors that are multiplied by the weights $w_1$, $w_2$, ..., $w_n$ and a bias is added to the summation of the input vectors and the weights. The bias and weights only linearly transform the input vectors while the activation function allows for non-linear transformation of the input vectors which is what allows machine learning models to learn from complex patterns. One of the most commonly used activation functions is the logistic sigmoid which has an S-shaped curve with values between 0 and 1. However, it has its limitations and is rarely used due to them. This study uses two other commonly used functions: ReLU and Tanh.

1) *ReLU:* Given the nature and shape of the sigmoid(x) and tanh(x), they will cause almost all neurons to fire. This makes training the model computationally expensive. Though the ReLU is non-linear in nature, it does not activate all the neurons at the same time. You can see in Figure 3 if the input is negative it will convert it to zero and the neuron does not get activated. This makes the network light and consequently efficient and easy for computation. However, like sigmoid(x), ReLU also falls a prey to the gradients moving towards zero i.e. if a negative input given to ReLU(x) is mapped to zero. This may cause the NN model to get stuck since the gradient will be 0 and the weights will not be updated during backpropagation. This can create dead neurons which never get activated. [14]

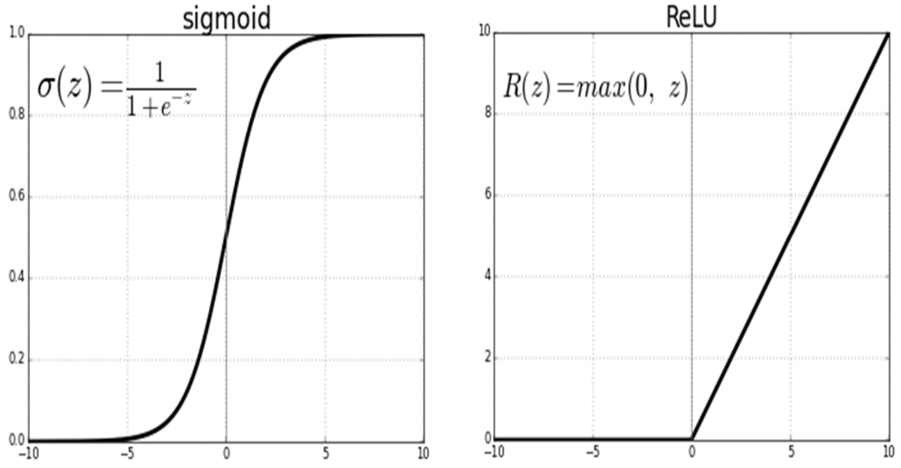
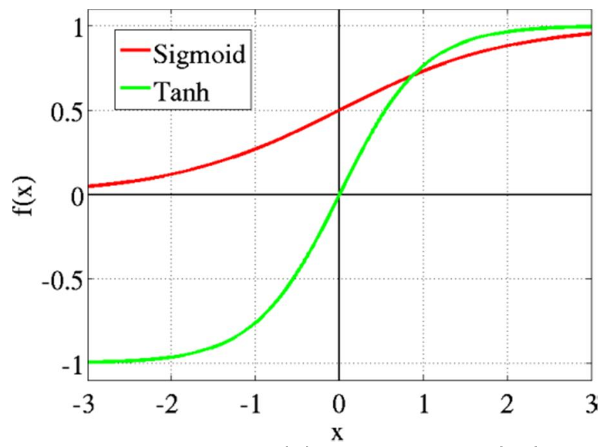


Fig. 3. Sigmoid vs ReLU (Courtesy: www.towardsdatascience.com)

2) *Tanh:* Tanh is just a scaled version of the sigmoid activation function and can be represented as follows:



3) $tanh(x) = 2 \times sigmoid(2x) - 1$

Fig. 4. Tanh vs Sigmoid (Courtesy: www.towardsdatascience.com)

As we can see in Figure 4, tanh(x) is scaled such that its range is from -1 to 1. This prevents the NN from getting 'stuck' at a given state since even if a strongly negative input is provided to tanh(x), it is mapped to a negative output and not 0, which is not the case for sigmoid(x).

## C. *Weight Assignment*

Initializing a deep neural net with the right weights can be the difference between converging accurately with a reasonable amount of time or giving inaccurate predictions with tremendous loss. If the weights of the network are too small, then as the model progresses, its weight values shrink and taper off until it's too tiny to be useful. On the other hand, if the weights are too large then the variance of the input tends to grow rapidly as it passes through each layer until it is too big to be useful.

Since we don't know anything about the data, we are not sure how to assign weights that would work well for particular problems. One good way is to assign the weights from a Normal distribution. This would mean that the weights are picked from values that have a zero mean and a finite variance. Consequently, this ensures that as the layers progress in the model the variance remains the same. This helps us keep the signal from exploding to a high value or vanishing to zero.

## D. *Batching*

Stochastic Gradient Descent (SGD) is a variation of the Gradient Descent algorithm. SGD calculates the error and updates the model for every example in the training data set. An improvement to SGD is Batch Gradient Descent. Batch Gradient Descent calculates the error for every example in the training data set and only updates the model after all the examples have been computed. The weights in the model are updated after every iteration of the training data set i.e., after every epoch.

It is still likely that our model overfits the data for both SGD and Batch Gradient Descent instead of giving a good generalized fit. Therefore, we decided to use Mini-batch Gradient Descent method which is a variation of the Gradient Descent algorithm that splits the training data set into small batches that are used to calculate model error and update model coefficient after each batch. This way the most optimal weights and biases can be found and a very good fit to the data is achieved.

A few advantages of Mini-batch Gradient Descent are:

1) The frequency with which the model updates the weights is higher than Batch Gradient Descent and therefore has better convergence and avoids local minimums.
2) It is computationally more efficient compared to SGD since feeding the data as mini-batches allows not having all the training data in memory at once.

3) $\theta := \theta - \eta \sum_{i=1}^{k} \frac{\partial E}{\partial \theta}$

where, $\theta$is the weight; $\eta$is the learning rate; *k* is the minibatch; *E* is the error function.

### *E. Optimizer*

In Stochastic Gradient Descent a single learning rate is maintained for all weight updates. A learning rate is used to train a network and consequently is updated to better fit the network once the training is done. Adam (Adaptive Moment Estimation) on the other hand maintains adaptive learning rates for each weight (parameter) in the network, takes into account the momentum, which ensures the model does not get stuck at the local minimum, and also addresses the problem of vanishing gradients. Adam gleans its properties from 2 other byproducts of the Stochastic Gradient Descent algorithm, namely:

1) *Adaptive Gradient Algorithm (AdaGrad):* AdaGrad maintains a different learning rate for every weight (parameter) in the model which improves model performance when encountered with sparse gradients.
2) *Root Mean Square Propagation (RMSProp):* This also maintains a different learning rate per parameter in the model that adapts according to how quickly the magnitudes of the gradients for the weights are changing. Adam not only adapts the parameter learning rates according to the mean of the first moments, it also incorporates the use of the mean of the second moments of gradients. In effect, Adam calculates the exponential moving average of the gradients and squared gradient and has ways to control the decay rates of these moving averages. Adam has become a popular optimizer in deep learning because it quickly achieves good results. In the original paper presented by Diederik Kingma from OpenAI and Jimmy Ba [15], Adam was applied to the MNIST character recognition problem and the IMBD sentiment analysis datasets. It was seen that Adam converged accurately and faster than the other optimizers it was compared with.

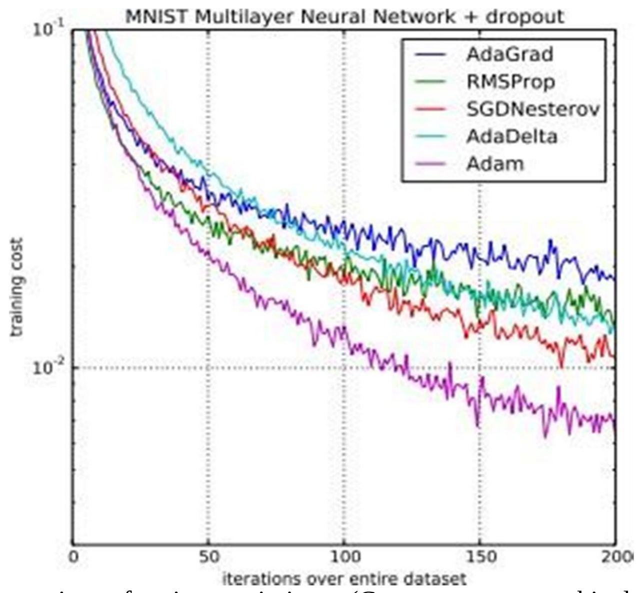


Fig. 5.Comparison of various optimizers. (Courtesy: www.machinelearningmastery.com)

## V. PROPOSED METHODOLOGIES

### *A. Neural Networks*

1) *Artificial Neural Network (ANN):* Artificial Neural Networks (NNs) are versatile, non-linear, and data-driven structures that have suitable properties for making predictions. Linear statistical methods are usually efficient for data having seasonal or trend patterns, whereas artificial neural algorithms can accommodate the data influenced by special cases like promotion or extreme crisis demand fluctuation (Nikolaos Kourentzes, 2013). ANNs are proven to be efficient in structuring complex and seemingly "patternless" problems, provided there is plenty of data (Dhar & Stein, 1997).
   In this study, a feed-forward error back-propagation type of ANN is used. In these networks, there is a fixed number of neurons in each layer, and these individual neurons are organized such that output signals from the neurons of a given layer are passed to all of the neurons of the next layer. Thus, there is a forward unidirectional flow of activations, layer by layer. There may be many to none number of layers between the input and output layer, and a certain number would be better than another for reasons involving optimal accuracy and minimal overfitting.

The basic element of an ANN is a neuron. The model of a neuron is depicted in Figure 6. A neuron $k$ can be described as in (1) and (2)

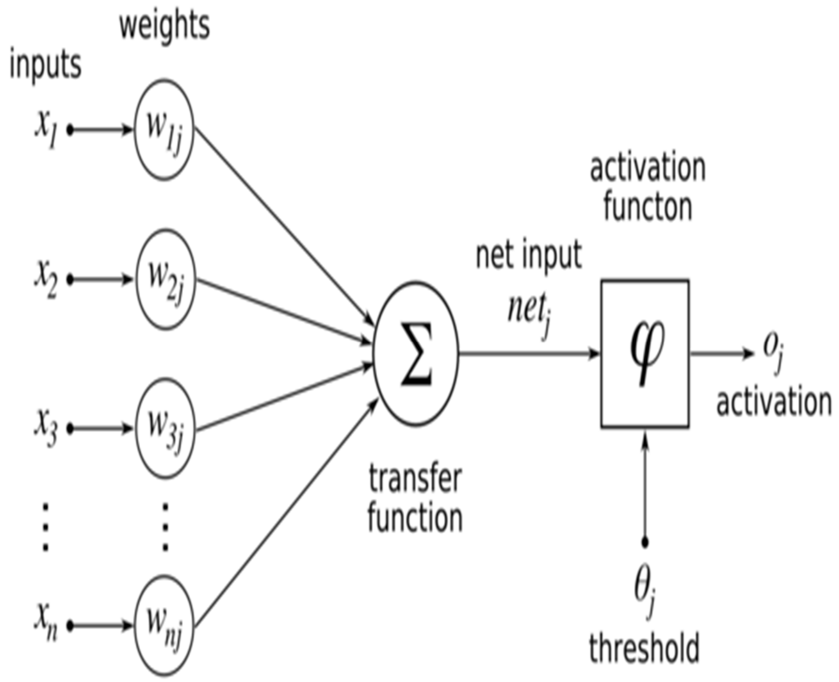


Fig. 6. Architecture of a neuron in an Artificial Neural Network.

$$net_j = \sum_{k=1}^{n} w_{jk}x_k (1)$$
$$o_j = \varphi(net_j - \theta_j) (2)$$

where, $x_1$, $x_2$, ..., $x_n$ are the input variables; $w_{j1}$, $w_{j2}$, ..., $w_{jn}$ are the weights for all the input variables for neuron $j$; $net_j$ is the linear combiner output; $\theta_j$ is the threshold; $\varphi$ is the activation function; $o_j$ is the output value of the neuron. Neural networks are tuned to obtain the desired outcome by a mapping of inputs to the outputs, using training algorithms that minimize loss per epoch. In this project, 'Error Back propagation' [16] is used. Error Back propagation is a supervised learning model where the error between the expected output (target) and the calculated output is calculated and minimized by adjusting the weights between two connection layers starting backwards, all the way from the output layer to one layer before the input layer. It does so by minimizing the error per layer, not unlike the way gradient descent does for the target. There are multiple layers involved, and each layer has an 'error' metric that is dependent on the layer(s) before it.

2) *Recurrent Neural Network (RNN):* Recurrent neural networks add the explicit management of order between observations when learning a mapping function from inputs to outputs. The biggest difference between RNN and ANN is that the output obtained from the activation function is fed as the input for the next "layer", which is actually another "time step" in the future.
The addition of sequences is a new dimension to the output function being approximated. Instead of mapping inputs to outputs alone, the network learns a mapping function for the inputs over time to an output. This makes the RNN capable of handling time-series data. Additionally, RNNs can also learn the Temporal Dependence from the data. Learned Temporal Dependence is defined as the context of observations learned over time. That is, the network is shown one input observation at a time from a time-series sequence, which then learns from the previously relevant observations that can be used for forecasting.
The promise of RNNs is that the Temporal Dependence in the input data can be learned, or in other words, a fixed set of lagged observations does not need to be specified. Moreover, a Temporal Dependence that varies with circumstance can also be learned.
This leads us to the conclusion that while removing systematic structures from time-series data makes the problem easier to model, the general potential of these models does not make it a necessity. Traits such as trend and seasonality are automatically picked up upon. [5]

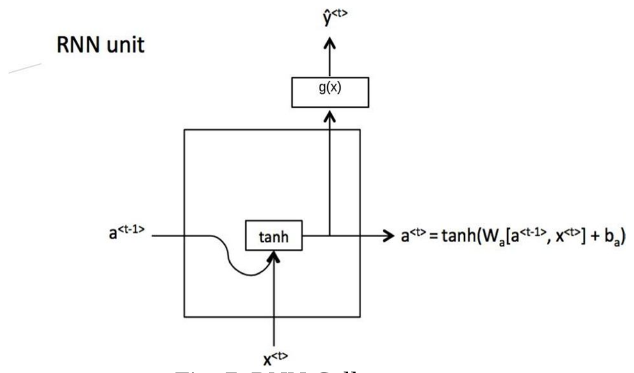


Fig. 7. RNN Cell

$$a^{<t>} = f(W_a[a^{<t-1>}, x^t] + b_a)$$

where, $a^{<t>}$ are the activation values for the current time step *t*, which is obtained by applying an activation function *f(x)* to a parameter $W_a$ , times the activation values from the previous time step $a^{<t-1>}$ and current input values $x^t$ .

$$\hat{y}^{<t>} = g(W_y[a^{<t>}, x^t] + b_y)$$

where, $y^{<t>}$ is the forecast value obtained after one time step; *g(x)* is an activation function applied to get the output.

3) *Gated Recurrent Unit (GRU):* GRU has a new unit called c which is a memory cell. The memory cell provides a bit of memory that helps the GRU remember a state.

$$\hat{c}^{<t>} = g(W_c[c^{<t-1>}, x^t] + b_c)$$

where$\hat{c}^{<t>}$is the candidate value for $c^{<t>}$ and is considered for update at every time step *t.*

$$\Gamma_u = \sigma(W_u[c^{<t-1>}, x^t] + b_u)$$

where,$\Gamma_u$iis the update gate and is obtained by applying the sigmoid activation function to weights $W_u$ times the memory cell of the previous time step $c^{<t-1>}$ and the current input values $x^t$ . Since most of the possible ranges for the input to the sigmoid function are either very close to 0 or very close to 1 we can consider the update gate $\Gamma_u$to have values either 0 or 1. The role of the update gate $\Gamma_u$is to decide when the memory cell value $c^{<t>}$ will be updated. Therefore, the equation of $c^{<t>}$ can be given by:

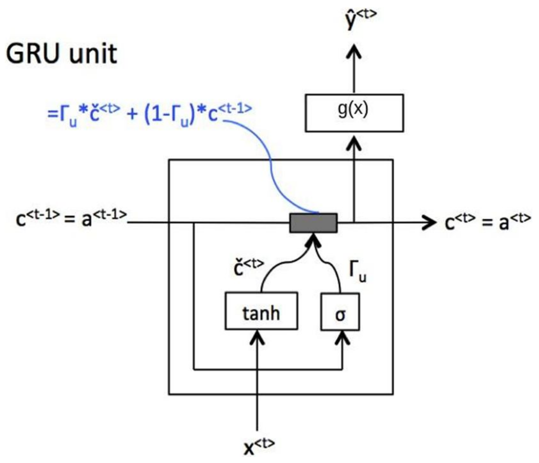


Fig. 8. GRU Cell

$$\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u)$$

$$c^{<t>} = \Gamma_u \times \hat{c}^{<t>} + (1 - \Gamma_u) \times c^{<t-1>} \quad (1)$$

If the update gate $\Gamma_u = 1$, then (1) becomes:

$$c^{<t>} = \Gamma_u \times \hat{c}^{<t>}$$

i.e., the current memory cell $c^{<t>}$ is updated with the candidate value $\hat{c}^{<t>}$. If the update gate $\Gamma_u = 0$, then (1) becomes :

$$c^{<t>} = (1 - \Gamma_u) \times c^{<t-1>}$$

The current memory cell will not be updated and will be set to the old value $c^{<t-1>}$. [17]

4) *Long Short-term Memory Cell (LSTM):* Like the GRU, LSTM is a type of RNN. LSTMs use backpropagation through time to find optimal learning weights which makes the model learn sequences from input data sets. Hence, they are frequently used to tackle complex sequence problems that have many features contributing to the pattern.

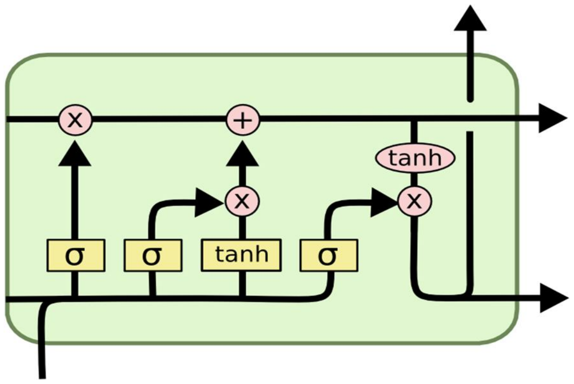


Fig. 9. LSTM Cell. (Courtesy: colah.github.io)

Just as the name suggests, LSTMs have memory blocks that make it smarter and more apt for sequence to sequence problems than a normal Multilayer Perceptron network. Each block contains a memory unit/state which stores the current state that the LSTM network is in and also a number of gates that manages this state and the output of the LSTM unit. A sigmoid activation function is used to control whether the gates are triggered or not making the change of state in a block conditional. There are 3 gates within a unit:

a) *Forget Gate:* Conditionally determines what information is to be discarded from the block. It's represented as:

$$\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f)$$

b) *Update Gate:* Conditionally determines which value the current state will be updated with. It's represented as:

$$\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u)$$

c) *Output Gate:* Conditionally determines what the unit will output based on the memory and input. It's represented as:

$$\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o)$$

## B. *Time-Series*

1) *Autoregressive Models (AR):* An autoregressive model forecasts values in the future, based on the past behavior. It's used for prediction if a pattern or some correlation exists between values at that instant and the values that precede and succeed that time lag. Past data is used to give shape to the behaviour, therefore the name autoregressive (the Greek prefix auto– means "self."). This process can be considered as a linear regression of the data in the current time lag against one or more historical values in that same series. [6]

In an AR model for time-series forecasting, the value of the target($y$) at any point $t$ in time is not unlike regular linear regression, where it is directly related to the input variable($x$). AR models and regular linear regression differ where $y$ is dependent on $x$ and previous values for $y$.

The AR model has degrees of uncertainty or randomness built in, and hence can be considered as a stochastic process. Even though the randomness means that you might be able to forecast future patterns pretty well with the help of historical data, it's impossible to get 100 percent accuracy. Normally the model gets close enough only to pass off as a rough estimator. AR($x$) is a model that is autoregressive in nature, wherein specific lagged values of $y_t$ are used as predictor variables.

Lags are defined as the parameter where output from one time period affects the following periods. The value for $x$ in AR($x$) is called the order. For instance, an AR(1) would be of the first order. The output for a first order AR model at some point $t$ is related only to time periods that are one time lag apart (i.e. the value of the variable at $t$–$1$). Similarly, a second or third order AR process would be related to data two or three time periods apart. The AR($x$) model is defined by:

$$y_t = \delta + \Phi_1 y_{t-1} + \Phi_2 y_{t-2} + \ldots + \Phi_x y_{t-x} + A_t$$

where, $y_{t-1}$,$y_{t-2}$,...,$y_{t-x}$ are past time-series values, or lags; $A_t$ is the white noise (randomness); $\delta$is:

$$\delta = (1 - \sum_{i=1}^{x} \Phi_i) \times \mu$$

where, $\mu$is the process mean.

2) *ARIMA:* ARIMA models are another way to tackle time series forecasting. ARIMA stands for Autoregressive Integrated Moving Average. It aims to describe the autocorrelations in the data and capture a suite of different standard temporal structures in time series data.

It is a generalisation of the AR model and Moving Average model and supplements the notion of integration. The acronym is descriptive, capturing the key aspects of the model itself:

*a) AR: Autoregression.* A model that uses the relationship between an observation at a defined time instance and some number of lagged observations from that time.

*b) I: Integrated.* The use of difference of raw observations (e.g. subtracting an observation from an observation at the previous time step) in order to make the time series stationary.

*c)* MA: *Moving Average*. A model that uses the dependency between an observation and a residual error from a moving average model applied to lagged observations.

Each of these components is explicitly specified in the model as a parameter. A standard way to represent it is ARIMA(p, d, q) where the parameters are replaced with integers. The parameters of the ARIMA model are defined as follows:

*p*: The lag order, that is, the number of lag observations included in the model.

*d*: The degree of difference, or the number of times the raw observations are subtracted.

*q*: The order of moving average window, or, the size of the moving average window.

A linear regression model is devised including all the parameters mentioned above, and the data is prepared by setting a value for *d* in order to make it stationary, i.e. to remove any trend and seasonal structures that negatively affect the regression model. It is to be noted that if one wishes to not include any, or all of the parameters, the respective parameter(s) can be set to 0. This way, the ARIMA model may function as AR, I, MA, ARI, ARMA, or IMA. [5]

The ARIMA model equation may be generalized as:

$$(1 - \sum_{i=1}^{p} \phi_i L^i)(1 - L)^d X_t = \delta + (1 + \sum_{i=1}^{q} \theta_i L^i)\varepsilon_t$$

where, $L$ is the lag operator; $\phi_{i}$ are the parameters of the AR model; $\theta_{i}$ are the parameters of the MA model; $\epsilon_{t}$ is the error term.

There are two important functions associated with ARIMA models:

*Autocorrelation Function (ACF):* Autocorrelation is a correlation coefficient. However, instead of correlation between two different variables, the correlation is between two values of the same variable at times $t_i$ and $t_{i+k}$ . ACF represents the degree of persistence over respective lags of a variable.

*Partial Autocorrelation Function (PACF):* The partial correlation between two variables is the amount of correlation between them which is not explained by their mutual correlations with a specified set of other variables. For example, if we're regressing a variable Y on other variables X1, X2, and X3, the partial correlation between Y and X3 is the amount of correlation between Y and X3 that is not explained by their common correlations with X1and X2. Partial correlation measures the degree of association between two random variables, with the effect of a set of controlling random variables removed.

## VI. COMPARATIVE ANALYSIS

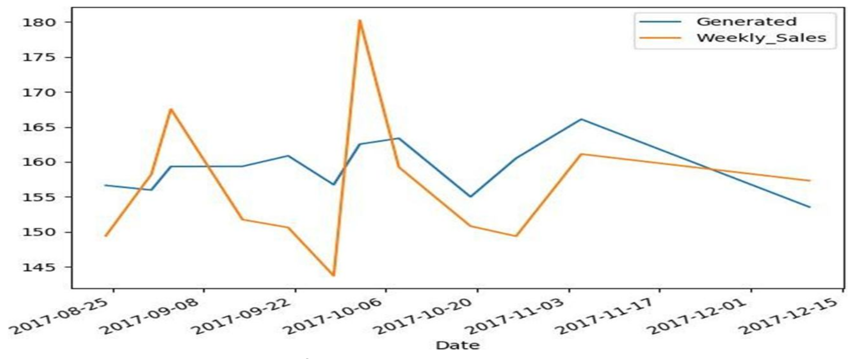


Fig. 10. Basic RNN for LR=0.001, Activation=tanh, time-step=6

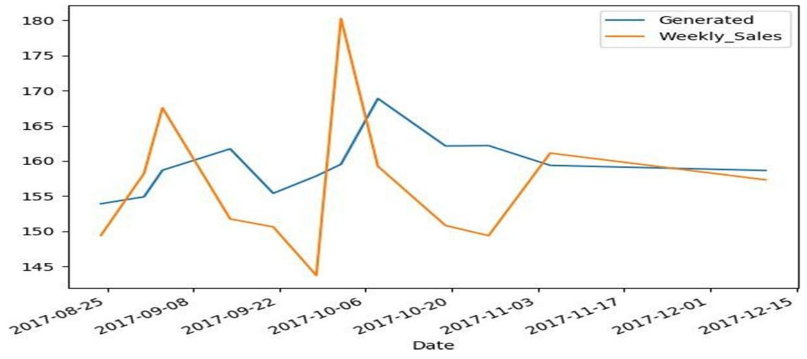


Fig. 13. ANN for LR=0.01, Activation=Tanh, Hidden Layers=15, Neurons=100

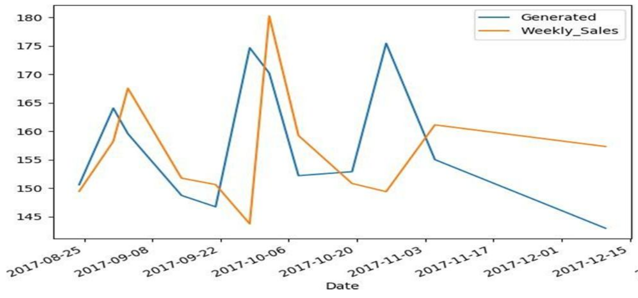


Fig. 11. GRU for LR=0.01, Activation=tanh, time-step=1

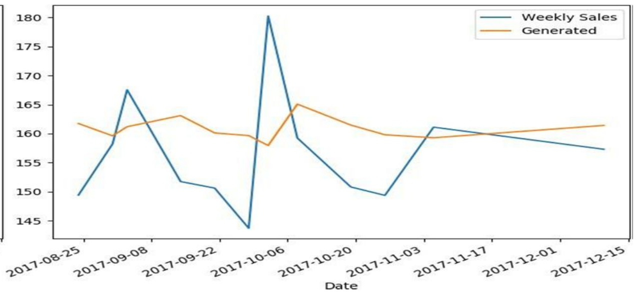


Fig. 14. ARIMA(6,1,1) model

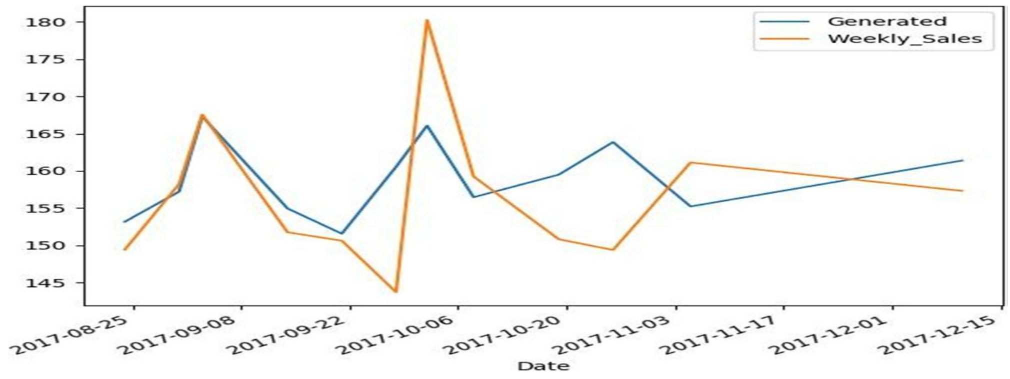


Fig. 12. LSTM for LR=0.01, Activation=ReLU, time-step=1

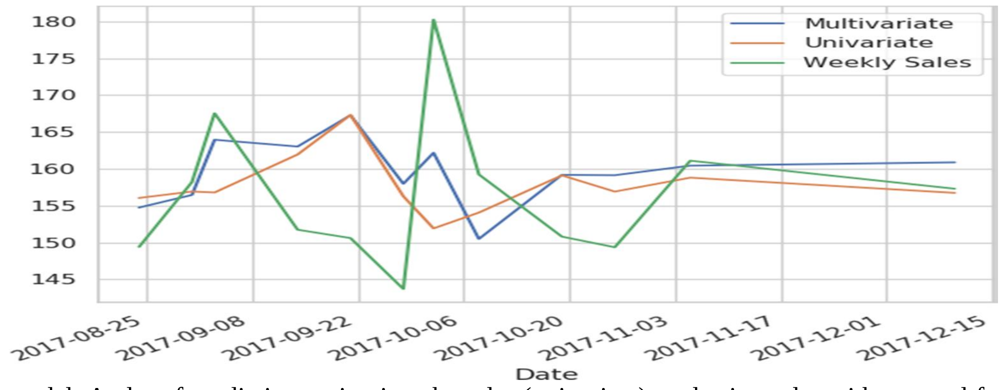
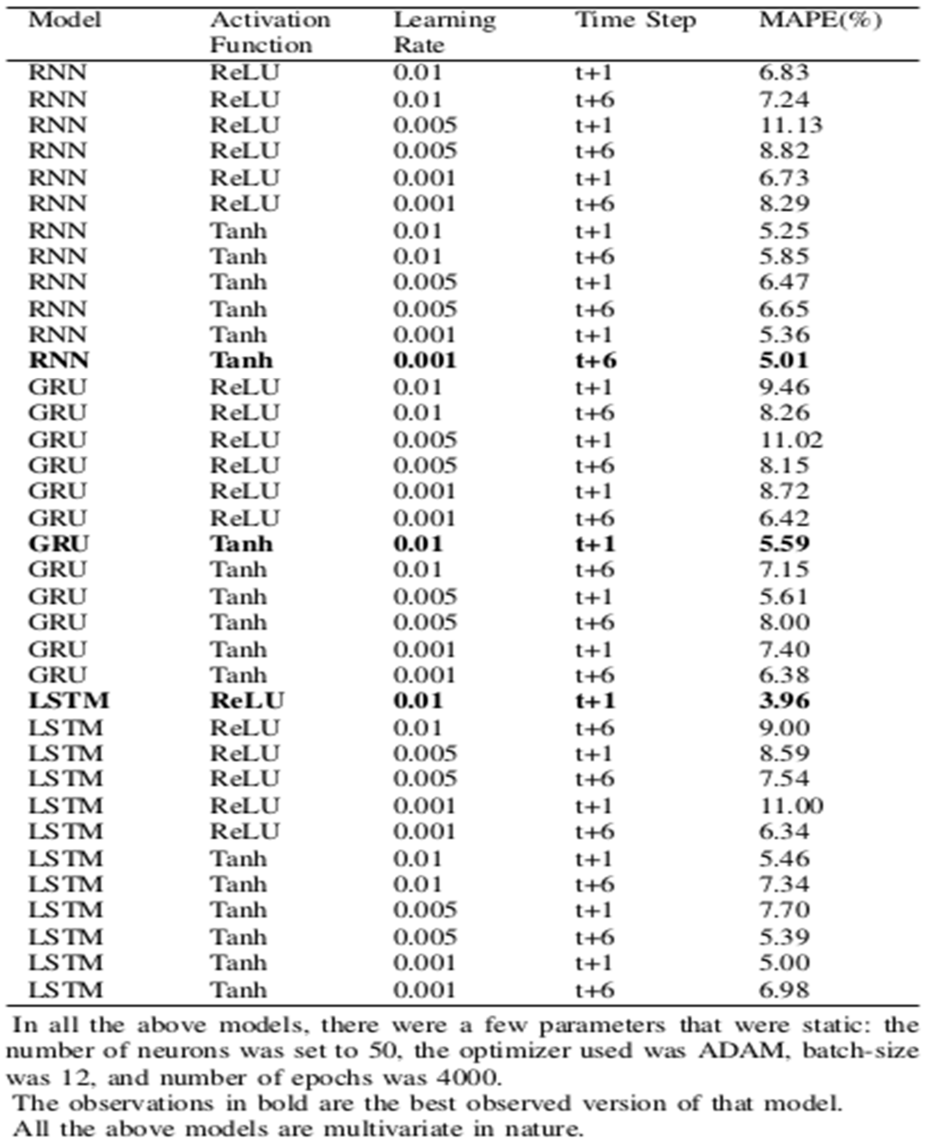


Fig. 15. AR model: A plot of predictions using just the sales (univariate), and using sales with external factors (multivariate)

TABLE I COMPARATIVE ANALYSIS OF ALL RNN MODELS



| Model | Activation Function | Learning Rate | Time Step | MAPE(%) |
|---|---|---|---|---|
| RNN | ReLU | 0.01 | t+1 | 6.83 |
| RNN | ReLU | 0.01 | t+6 | 7.24 |
| RNN | ReLU | 0.005 | t+1 | 11.13 |
| RNN | ReLU | 0.005 | t+6 | 8.82 |
| RNN | ReLU | 0.001 | t+1 | 6.73 |
| RNN | ReLU | 0.001 | t+6 | 8.29 |
| RNN | Tanh | 0.01 | t+1 | 5.25 |
| RNN | Tanh | 0.01 | t+6 | 5.85 |
| RNN | Tanh | 0.005 | t+1 | 6.47 |
| RNN | Tanh | 0.005 | t+6 | 6.65 |
| RNN | Tanh | 0.001 | t+1 | 5.36 |
| **RNN** | **Tanh** | **0.001** | **t+6** | **5.01** |
| GRU | ReLU | 0.01 | t+1 | 9.46 |
| GRU | ReLU | 0.01 | t+6 | 8.26 |
| GRU | ReLU | 0.005 | t+1 | 11.02 |
| GRU | ReLU | 0.005 | t+6 | 8.15 |
| GRU | ReLU | 0.001 | t+1 | 8.72 |
| GRU | ReLU | 0.001 | t+6 | 6.42 |
| **GRU** | **Tanh** | **0.01** | **t+1** | **5.59** |
| GRU | Tanh | 0.01 | t+6 | 7.15 |
| GRU | Tanh | 0.005 | t+1 | 5.61 |
| GRU | Tanh | 0.005 | t+6 | 8.00 |
| GRU | Tanh | 0.001 | t+1 | 7.40 |
| GRU | Tanh | 0.001 | t+6 | 6.38 |
| **LSTM** | **ReLU** | **0.01** | **t+1** | **3.96** |
| LSTM | ReLU | 0.01 | t+6 | 9.00 |
| LSTM | ReLU | 0.005 | t+1 | 8.59 |
| LSTM | ReLU | 0.005 | t+6 | 7.54 |
| LSTM | ReLU | 0.001 | t+1 | 11.00 |
| LSTM | ReLU | 0.001 | t+6 | 6.34 |
| LSTM | Tanh | 0.01 | t+1 | 5.46 |
| LSTM | Tanh | 0.01 | t+6 | 7.34 |
| LSTM | Tanh | 0.005 | t+1 | 7.70 |
| LSTM | Tanh | 0.005 | t+6 | 5.39 |
| LSTM | Tanh | 0.001 | t+1 | 5.00 |
| LSTM | Tanh | 0.001 | t+6 | 6.98 |

In all the above models, there were a few parameters that were static: the number of neurons was set to 50, the optimizer used was ADAM, batch-size was 12, and number of epochs was 4000.
The observations in bold are the best observed version of that model.
All the above models are multivariate in nature.

TABLE II

COMPARATIVE ANALYSIS OF ANN MODELS

| Activation | Learning Rate | Hidden layers | Neurons | MAPE(%) |
|---|---|---|---|---|
| Tanh | 0.001 | 4 | 25 | 13.79 |
| Tanh | 0.005 | 10 | 50 | 7.43 |
| **Tanh** | **0.01** | **15** | **100** | **5.54** |
| ReLU | 0.001 | 4 | 25 | 10.71 |
| ReLU | 0.005 | 10 | 50 | 8.89 |
| ReLU | 0.01 | 15 | 100 | 6.45 |

In all the above models, there were a few parameters that were static: the optimizer used was ADAM, a mini-batch was maintained of size 10, and the number of epochs was 1000.
The observation in bold is the best observed version of ANN.

TABLE III
COMPARATIVE ANALYSIS OF TIME-SERIES MODELS

| Model | MAPE(%) |
|---|---|
| AR - U | 5.84 |
| AR - M | 5.30 |
| ARIMA(1,1,1) | 5.91 |
| ARIMA(6,1,1) | 5.79 |

U - Univariate; M - Multivariate; ARIMA(x, y, z) - where x, y, and z correspond to p, d, and q values respectively.

## VII. CONCLUSION

Comparing the best configuration of each model, we have the following result:

TABLE IV
COMPARATIVE ANALYSIS OF ALL MODELS

| Model | MAPE(%) |
|---|---|
| AR | 5.30 |
| ARIMA | 5.79 |
| ANN | 5.54 |
| **RNN** | **3.96** |

We observe that an RNN gives us the least error, hence being the most accurate model. With more data, it is likely that the accuracy improves for our RNN, while remaining more or less static with time-series models.
In the future, the model can be further ameliorated with the inclusion of batch normalization and drop out techniques during training, cross-validation sets, inclusion of more, relevant, external factors such as cost of the product, rainfall, and social media ad campaigns. Different models such as radial basis neural networks may also be used in order to improve the accuracy of the predictions. The scope of the product can be broadened further by predicting the sales of more furniture items the shop sells (cupboards, chairs, home decor items).

## VIII. ACKNOWLEDGMENT

We would like to thank Sanskriti Lifestyle for providing their data set, helping us understand their supply chain, and providing access to their inventory for purposes of this project.

## REFERENCES

[1] C. Chu and G. P. Zang, “The Comparative Study of Linear and Nonlinear Models for Aggregate Retail Sales Forecasting, Int,” J. Economics, vol. 86, pp. 217–231, 2003.
[2] J. T. Mentzer and J. E. Cox, “Familiarity, Application, and Performance of Sales Forecasting Techniques,” Journal of Forecasting, vol. 3, pp. 27–36, 1984.
[3] P. Ramos, N. Santos, and R. Rebelo, “Performance of state space and ARIMA models for consumer retail sales forecasting,” Robotics and Computer-Integrated Manufacturing, vol. 34, pp. 151–163, 2014.
[4] K. Tanaka(2010, “A sales forecasting model for new-released and nonlinear sales trend products, Expert Systems with Applications 37,” pp. 11–7387.
[5] Z. Sun, T. Choi, K. Au, and Y. Yu, Sales Forecasting using Extreme Learning, 2008, vol. 46.
[6] C. Frank, A. Garg, A. Raheja, and L. Sztandera, “Forecasting women’s apparel sales using mathematical modeling,” International Journal of Clothing Science and Technology, vol. 2, no. 15, p. 125, 2003.
[7] H. L. Lee, V. Padmanabhan, and S. Whang, “The bullwhip effect in supply chains,” Sloan Management Review, vol. 38, pp. 93–102, 1997.
[8] X. Zhao, J. Xie, and J. Leung, “The impact of forecasting model selection on the value of information sharing in a supply chain,” European Journal of Operational Research, vol. 142, pp. 321–344, 2002.
[9] J. Dejonckheere, S. M. Disney, M. R. Lambrecht, and D. R. . Towill, “Measuring and avoiding the bullwhip effect: A controltheoretic approach,” European Journal of Operational Research, vol. 147, no. 3, pp. 567–590.
[10] M. Garetti and Taisch, “Neural networks in product ion planning and control,” Production Planning and Control, vol. 10, no. 4, pp. 324–339, 1999.
[11] R. Carbonneau, K. Laframboise, and R. Vahidov, “Application of machine learning techniques for supply chain demand forecasting,” European Journal of Operational Research, vol. 3, no. 184, pp. 1140– 1154, 2008.
[12] R. J. Kuo and K. C. Xue, “A decision support system for sales forecasting through fuzzy neural networks with asymmetric fuzzy weights ,” Decision Support Systems, vol. 24, no. 2, pp. 105–126, 1998.
[13] W. T. Anderson and W. H. Cunningham, “The Socially Conscious,” Consumer, vol. 36, pp. 23–31, 2015.
[14] X. Glorot, Y. Bengio, and A. Bordes, “Deep Sparse Rectifier Neural Networks.”
[15] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization, International Conference for Learning Representations.” San: Diego, 2015.
[16] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, Learning internal representations by error propagation, Parallel distributed processing. Cambridge, MA: MIT Press, 1986, vol. 1.
[17] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, Emperical Evaluation of Gated Recurrent, 2014.