Machine Learning Engineer Nanodegree

Capstone Project

Tejash Panchal

August 5th, 2018

1. Definition

1.1. Project Overview

I have been working in supply chain for past many years as a test engineer for a large networking company in bay area. Various functions in supply chain rely on accurate demand forecast to perform their job accurately. As a test development engineer, I heavily rely on accurate demand forecast to develop test processes and test capacities for each of the products I am responsible for. Setting up test process is an expensive and time-consuming task; thus, if I don't setup test process that has enough capacity, orders will be delayed and we will have angry customers that will be willing to move their business to competitor. If I setup test process with excess capacity, we, as company, would waste tremendous amount of money that will have impact on company's bottom line revenue. At my company, we have tried several methods to receive accurate forecast; however, so far, we haven't been able to find an adequate solution that works for different types of products as well as different volume of product shipment.

In this project, I have created a solution that solves the forecasting problem using some of the latest artificial intelligence concepts and technologies that is available today.

1.2. Problem Statement

"Overestimated forecast in supply chain causes excess inventory that stagnant significant dollar amount; on the other hand, underestimated forecast causes unfulfilled orders, angry customers, as well as lost sales and opportunities. The goal of this project is to develop a solution using the latest concepts of neural network that predicts accurate demand forecast for networking products."

The issue described in problem statement is a time series problem that uses observations from previous time steps as input to a regression equation to predict the value at the next time step. The strategy I am proposing is to use neural network especially LSTM concepts to solve the problem using steps below:

- Transform time series dataset using Differencing method
- Transform time series to Supervised Learning using Pandas shift() Function with lag=1
- Transforming the data so that it has the scale -1 to 1 using MinMaxScaler
- Fitting a LSTM network model to the training data using batch_size = 1, epoch = 1000 and neurons = 3. I achieved these values based on lowest RMSE by running the exercise in 30 loops, various different values for each variable.
- Evaluating the static LSTM model on the test data.
- Report the RMSE of the forecasts.

1.3. Metrics

To measure and compare the performance of both, benchmark model and LSTM model, I have used Root Mean Squared Error (RMSE). There are other metrics such as Mean Absolute Error (MAE) could be used as well.

MAE is achieved by using formula below:

MAE = 1/n$\sum_{i=1}^{n} |Yi - Y_{hat}i|$

However, I chose to use RMSE since the errors are squared before they are averaged, the RMSE gives a relatively high weight to large errors. This means the RMSE should be more useful when large errors are particularly undesirable.

RMSE is achieved by first finding the Mean Squared Error (MSE). I am using below formula for MSE:

MSE = 1/n$\sum_{i=1}^{n}(Yi - Y_{hat}i)$^2

Where n is observation count, Yi represents true vector, and $Y_{hat}i$ represents predicted value at current step.

Once I calculate MSE, I take the square-root to derive RMSE:

RMSE = $\sqrt{MSE}$

The lower the RMSE number, the closer the prediction is to the true vector.

2. Analysis

2.1. Data Exploration

For this project, I have created my model and perform all the tests using dataset "historical product demand", readily available at Kaggle website. This dataset has 1048575 data points and 5 different variables. Out of these 5 variables, only three, "Product_Code", "Date", and "Order_Demand" are relevant to the project. Sample of dataset is as below:

	Product_Code	Warehouse	Product_Category	Date	Order_Demand
0	Product_0993	Whse_J	Category_028	2012/7/27	100
1	Product_0979	Whse_J	Category_028	2012/1/19	500
2	Product_0979	Whse_J	Category_028	2012/2/3	500
3	Product_0979	Whse_J	Category_028	2012/2/9	500
4	Product_0979	Whse_J	Category_028	2012/3/2	500

Table 1: Dataset header

There are 2160 different “Product_Code” available in this dataset. To simplify and demonstrate the objective of this project further, I have chosen to use only one “Porduct_Code”, “Product_0001” which has 597 data points. Sample of dataset is as below:

	Date	Product_Code	Order_Demand
0	2011-12-16	Product_0001	0
1	2011-12-20	Product_0001	1
2	2012-01-03	Product_0001	0
3	2012-01-03	Product_0001	2
4	2012-01-04	Product_0001	0

Table 2: Dataset header for Product_0001

Since now I have only one “Product_Code” in my dataset, I have removed the “Product_Code” variable from the dataset and left with only “Date” and “Order_Demand” variables which aligns nicely with time-series dataset format. Sample of dataset is as below:

	Date	Order_Demand
0	2011-12-16	0
1	2011-12-20	1
2	2012-01-03	0
3	2012-01-03	2
4	2012-01-04	0

Table 3: Time series dataset header

Now that the data is in time-series format, I grouped it by month since the forecasting in supply chain of networking equipment typically works in monthly fashion. These gives me five years’ worth of data for my model to train and test with.

	Order_Demand
Date	
2012-01-31	23
2012-02-29	56
2012-03-31	66
2012-04-30	36
2012-05-31	58

Table 4: Time series dataset header grouped by month

2.2. Exploratory Visualization

Augmented Dickey-Fuller test:

Below is the plot of time series data points. Here, I am checking if the data is stationary. If data is stationary, it would show us NO seasonal effects that are dependent on the time index.

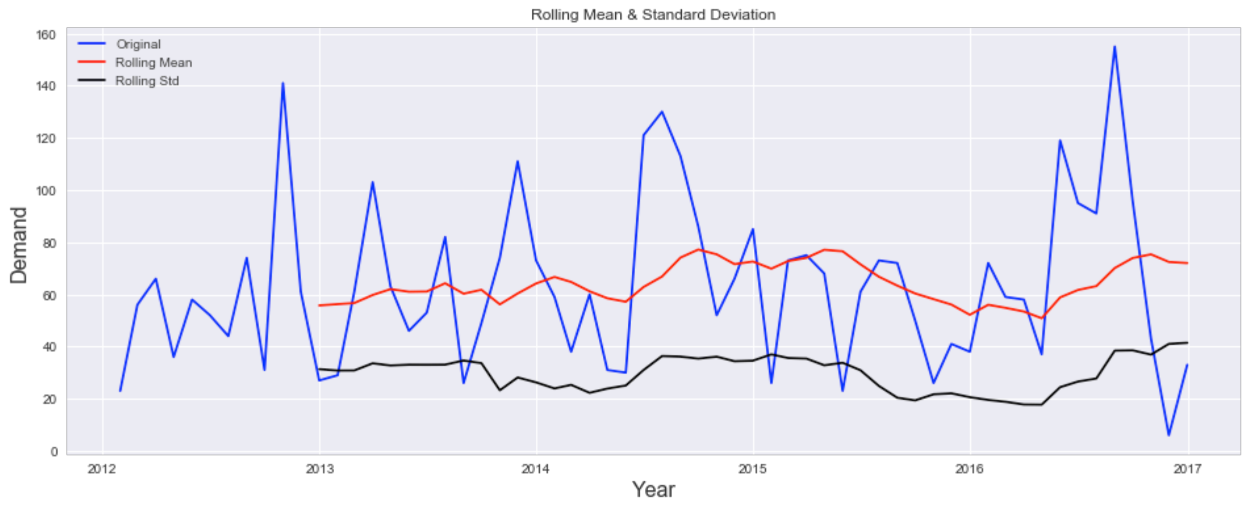

Figure 1: Time series data plot

Below is the output from Dickey-Fuller test of my dataset.

```
Results of Dickey-Fuller Test:
Test Statistic                 -5.034089
p-value                         0.000019
#Lags Used                      1.000000
Number of Observations Used    58.000000
Critical Value (5%)            -2.912837
Critical Value (1%)            -3.548494
Critical Value (10%)           -2.594129
```

- **Null Hypothesis (H0)**: If failed to be rejected, it suggests the time series has a unit root, meaning it is non-stationary. It has some time dependent structure.
- **Alternate Hypothesis (H1)**: The null hypothesis is rejected; it suggests the time series does not have a unit root, meaning it is stationary. It does not have time-dependent structure.

We interpret the above result using the p-value from the test. A p-value below a threshold (such as 5% or 1%) suggests we reject the null hypothesis (stationary), otherwise a p-value above the threshold suggests we fail to reject the null hypothesis (non-stationary).

- **p-value > 0.05**: Fail to reject the null hypothesis (H0), the data has a unit root and is non-stationary.

- **p-value <= 0.05**: Reject the null hypothesis (H0), the data does not have a unit root and is stationary.

Dickey-Fuller test above also prints the test statistic value of -5. The more negative this statistic, the more likely we are to reject the null hypothesis. Dickey-Fuller test ADF statistic also shows that our statistic value of -5 is less than the value of -3.548 at 1%. This suggests that the data is stationary and we can reject the null hypothesis with a significance level of less than 1%.

Seasonal Decompose:

In chart below, we can see that the trend and seasonality information extracted from this series does seem to indicate any trend or seasonal pattern. The residuals are showing periods of high variability most of the years of the series. A residual refers to the amount of variability in a dependent variable (DV) that is "left over" after accounting for the variability explained by the predictors in your analysis.

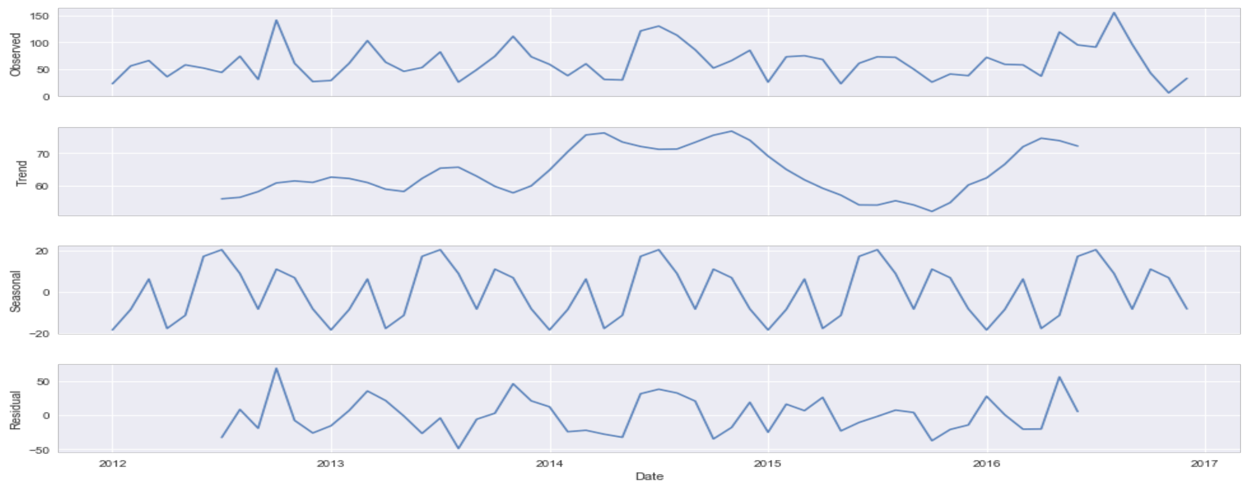

Figure 2: Time series data seasonal decompose plot

Stationarity of residuals:

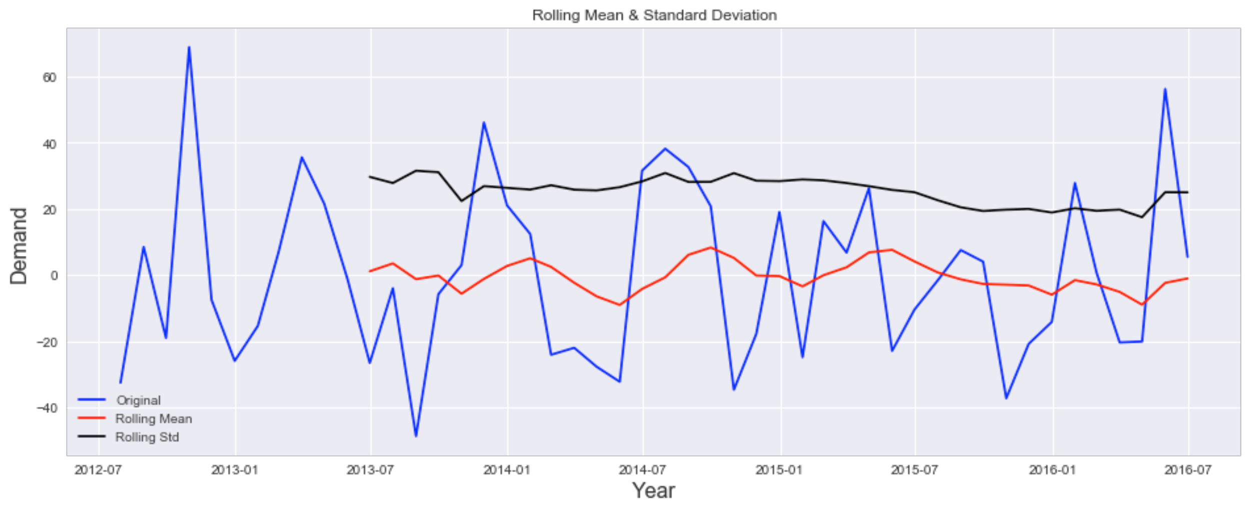

Figure 3: Time series data residuals plot

Residuals Dickey-Fuller Test:

```
Results of Dickey-Fuller Test:
Test Statistic                  -4.671160
p-value                          0.000095
#Lags Used                       5.000000
Number of Observations Used     42.000000
Critical Value (5%)             -2.933297
Critical Value (1%)             -3.596636
Critical Value (10%)            -2.604991
```

Dickey-Fuller test above for residuals also prints the test statistic value of -4. The more negative this statistic, the more likely we are to reject the null hypothesis. Dickey-Fuller test ADF statistic also shows that our statistic value of -4 is less than the value of -3.596 at 1%. This suggests that the residual data is stationary and we can reject the null hypothesis with a significance level of less than 1%.

2.3. Algorithms and Techniques

In time series predictive modeling field, Recurrent Neural Networks (RNN) is a type of neural network designed to handle sequence dependent input variables.

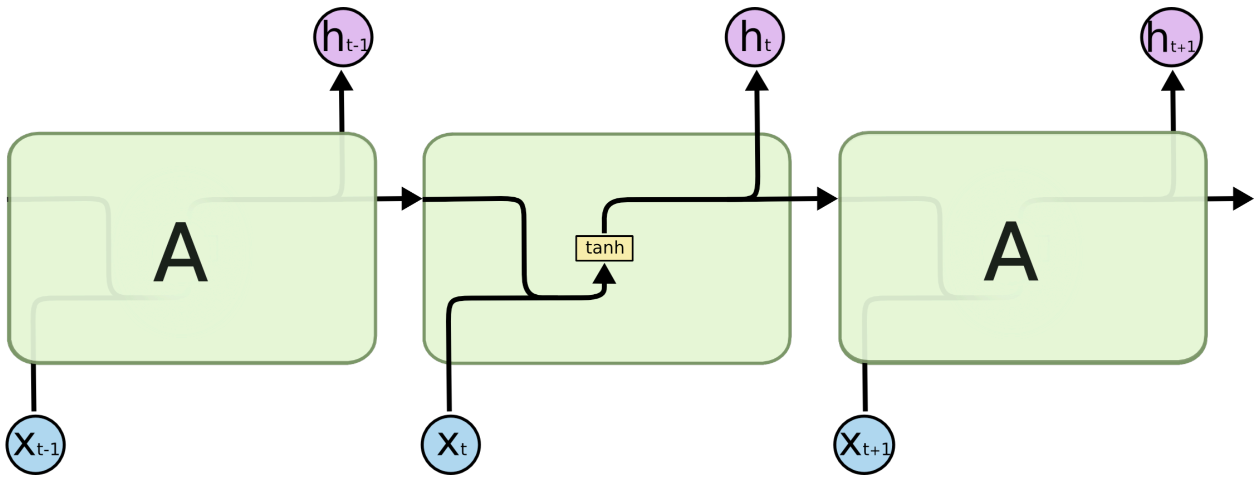

Figure 4:The repeating module in a standard RNN contains a single layer

The Long Short-Term Memory (LSTM) network is a type of recurrent neural network used in deep learning because very large architectures can be successfully trained using Backpropagation. LSTMs are a special kind of RNN, capable of learning long-term dependencies. Like RNN, LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.

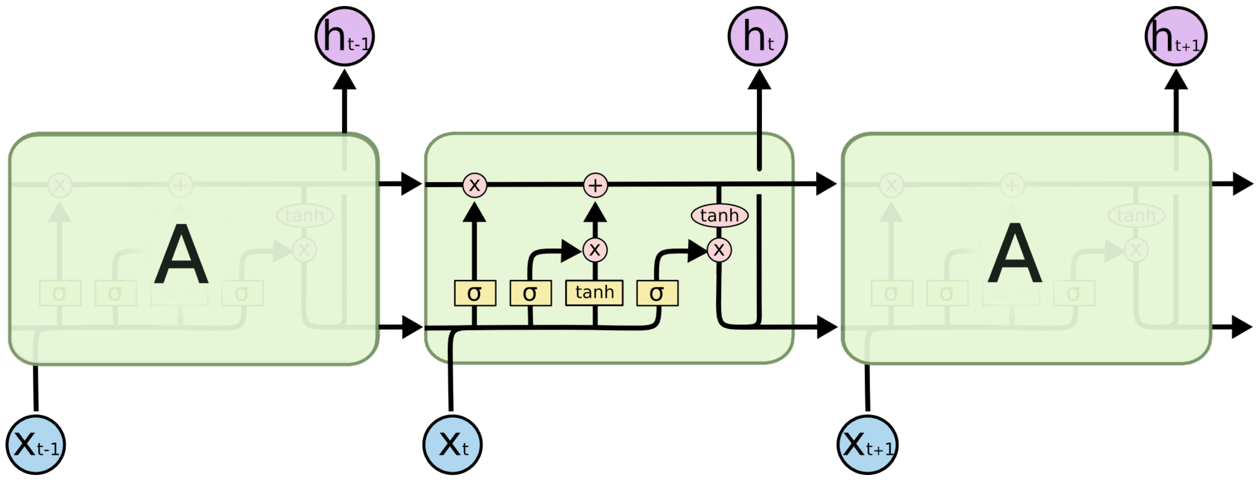

Figure 5: The repeating module in an LSTM contains four interacting layers

In the above diagram, each line carries an entire vector, from the output of one node to the inputs of others. The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers. Lines merging denote concatenation, while a line forking denotes its content being copied and the copies going to different locations.

There are three types of gates within a unit:

- **Forget Gate**: Conditionally decides what information to throw away from the block.

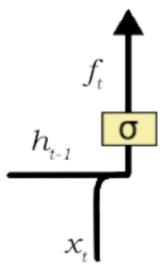

Figure 6: Forget Gate

- **Input Gate**: Conditionally decides which values from the input to update the memory state.

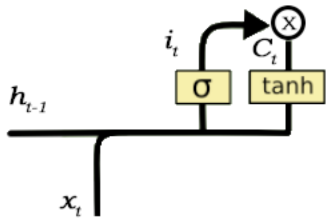

Figure 7: Input Gate

- **Output Gate**: Conditionally decides what to output based on input and the memory of the block.

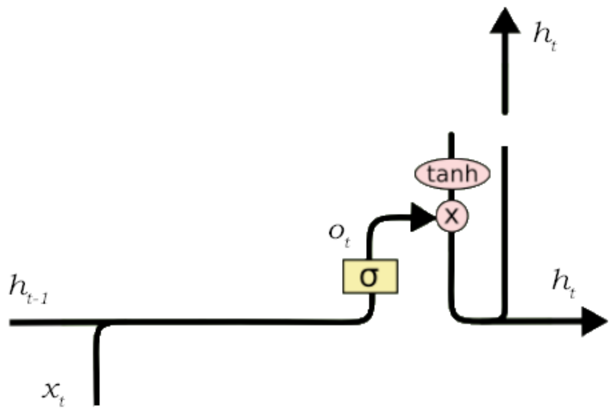

Figure 8: Output Gate

Each unit is like a mini-state machine where the gates of the units have weights that are learned during the training procedure.

With above architecture in mind, in this project, for the given time series data, I am following the steps below:

- Transform time series dataset using Differencing method
- Transform time series to Supervised Learning using Pandas shift() Function with lag=1
- Transforming the data so that it has the scale -1 to 1 using MinMaxScaler
- Fitting a LSTM network model to the training data using batch_size = 1, epoch = 1000 and neurons = 3. I achieved these values based on lowest RMSE by running the exercise in 30 loops, various different values for each variable.

- Evaluating the static LSTM model on the test data.
- Report the RMSE of the forecasts.

2.4. Benchmark

I have chosen ARIMA model as benchmark to compare my model against. ARIMA is an acronym that stands for Autoregressive Integrated Moving Average. It provides a simple yet powerful method for making skillful time series forecasts.

- AR: Autoregressive. A model that uses the dependent relationship between an observation and some number of lagged observations.
- I: Integrated. The use of differencing of raw observations (e.g. subtracting an observation from an observation at the previous time step) in order to make the time series stationary.
- 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.

A standard notation is used of ARIMA(p,d,q) where the parameters are substituted with integer values to quickly indicate the specific ARIMA model being used. The parameters of the ARIMA model are defined as follows:

- p: The number of lag observations included in the model, also called the lag order.
- d: The number of times that the raw observations are differenced, also called the degree of differencing.
- q: The size of the moving average window, also called the order of moving average.

When two out of the three terms are zeros, the model may be referred to based on the non-zero parameter, dropping "AR", "I" or "MA" from the acronym describing the model. For example, ARIMA (1,0,0) is AR(1), ARIMA(0,1,0) is I(1), and ARIMA(0,0,1) is MA(1). ARIMA (p, d, q)(P, D, Q)m where (p, q, d) is Non-seasonal part of the model, (P, D, Q) is seasonal part of the model, and m is number of periods per season.

To achieve desired results with ARIMA model, the data has been split into train and test. To train the model, this program is using the data from 2012 to 2015. For test, we have allocated the data for the year 2016. Using training dataset, we have identified best pdq (0, 2, 2), best seasonal pdq (1, 2, 0, 12), and best AIC 256.0286.

Once best pdq, and seasonal pdq is identified, we feed that information to SARIMAX package from statesmodels library function to create and fit the ARIMA model.

Model summary:

```
arima_results.aic: 386.94861182
arima_results.summary:
==============================================================================
=
                   coef    std err          z      P>|z|      [0.025
0.975]
```

```
--------------------------------------------------------------------------------
-
ar.L1         -1.6345      0.241     -6.770      0.000      -2.108      -
1.161
ar.L2         -0.6538      0.208     -3.138      0.002      -1.062      -
0.245
ma.L1         -0.0005     10.322  -4.38e-05      1.000     -20.232
20.231
ma.L2         -0.9995      0.599     -1.667      0.095      -2.174
0.175
ar.S.L12      -0.1718      0.211     -0.816      0.414      -0.584
0.241
ar.S.L24       0.7359      0.282      2.609      0.009       0.183
1.289
sigma2      1445.4681      0.007   2.01e+05      0.000    1445.454
1445.482
================================================================================
=

residuals:
                0
count   60.000000
mean     3.367278
std     84.487628
min   -219.512360
25%    -34.196651
50%     -0.669671
75%     42.416196
max    286.546396
```

Figure below displays the ARIMA diagnostics plot:

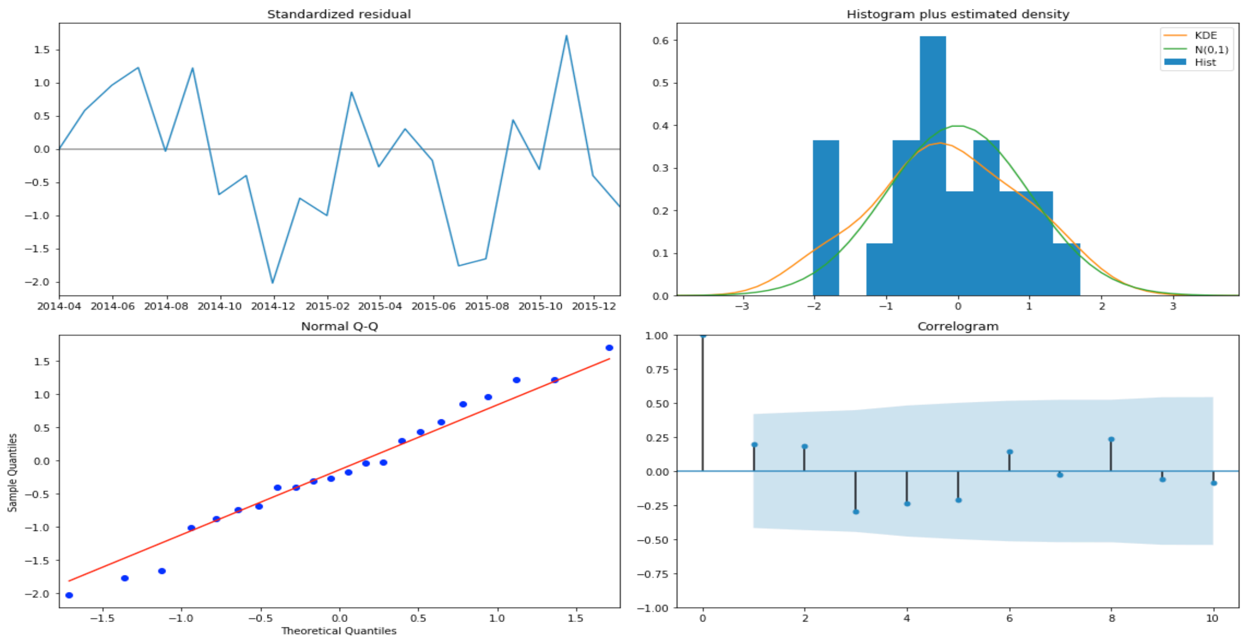

Figure 9: Time series data ARIMA diagnostics plot

Forecast with ARIMA:

For the completeness of the project, I am using three different ways, one-step ahead prediction, out-of-sample prediction, and long-term forecasting, to predict the forecast using ARIMA model. For one-step ahead prediction and out-of-sample prediction, the model returns MSE:11502.32 and RMSE: 107.25. Below is the graph for one of the method:

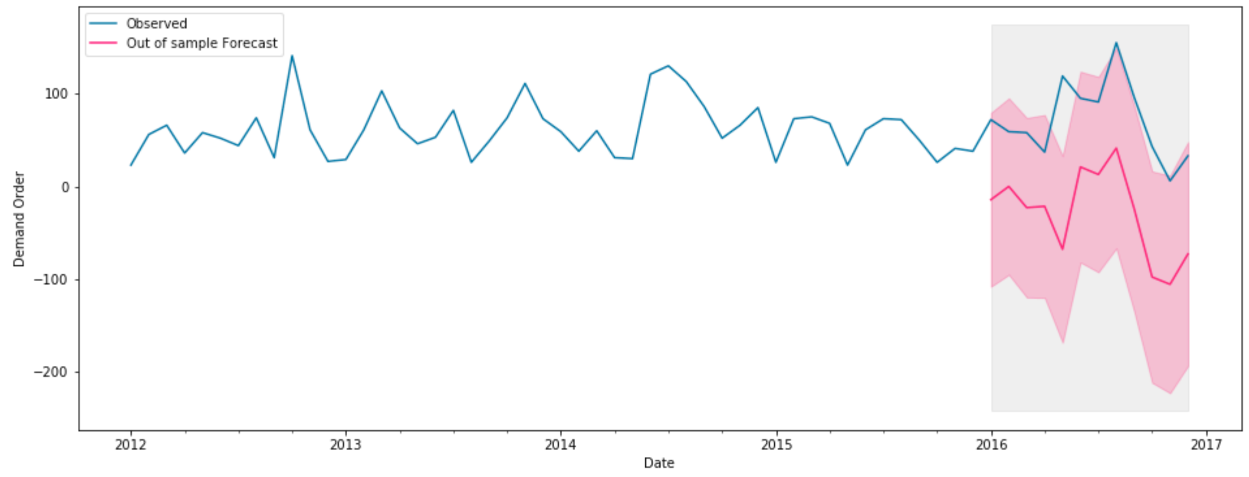

Figure 10: Out-of-sample prediction chart

3. Methodology

3.1. Data Preprocessing

In this project, Data preprocessing is done by using three steps below:

3.1.1. Transform Time Series to Stationary

3.1.1.1. Given dataset is non-stationary because there is a structure in the data that is dependent on the time. Stationary data is easier to model and will very likely result in more skillful forecasts. The trend can be removed from the observations, then added back to forecasts later to return the prediction to the original scale and calculate a comparable error score.

3.1.2. Transform Time Series to supervised learning

3.1.2.1. The LSTM model in Keras assumes that your data is divided into input (X) and output (y) components. For a time series problem, we can achieve this by using the observation from the last time step (t-1) as the input and the observation at the current time step (t) as the output. We can achieve this using the shift() function in Pandas that will push all values in a series down by a specified number places. We require a shift of 1 place, which will become the input variables. The time series as it stands will be the output variables.

3.1.3. Transform Time Series to Scale

3.1.3.1. Like other neural networks, LSTMs expect data to be within the scale of the activation function used by the network. The default activation function for LSTMs is the hyperbolic tangent (tanh), which outputs values between -1 and 1. This is the preferred range for the time series data.

3.2. Implementation

In this project, implementation is done in stages:

3.2.1. Split data into train and test sets
I have five years' worth of data from year 2012 to 2016. For this project, I grouped the data by month and sorted by date in ascending order. Then I split the data into train and test sets. For train set, I am using first 48 data points and for test set, I am using last 12 set. This will give me train set from 2012 to 2015 and test set for the year 2016.

3.2.2. Create LSTM model
To create LSTM model, my code has to comply with the requirements of the LSTM layer. The LSTM layer expects the input to be in a matrix with dimensions such as samples, time steps, and features. Samples are the rows of data. Time steps are separate time steps of a given variable for a given observation. Features are separate measures observed at the time of observation. The shape of the input data must be specified in the LSTM layer using the "batch_input_shape" argument with parameters batch_size and dimension of train data set. We also need to provide number of neurons. The network requires a single neuron in the output layer with a linear activation to predict the forecast demand at the next time step. Now that we have specified the network, we must compile into an efficient symbolic representation using a TensorFlow backend mathematical library. In the compile network, I have use "mean_squared_error" as loss function since it closely matches RMSE that I am interested in. I have also use ADAM optimization algorithm.

3.2.3. Train model using train data
Once compiled, it can be fit to the training data. Now we can fit the training data for defined epoch numbers one at a time. Since the network is stateful, we must control the internal state to reset after each epoch.

3.2.4. Prediction for test data
To make a forecast, we can call the predict() function on the model. This requires a 3D numpy array input as an argument. The predict() function returns an array of predictions, one for each input row provided. Because we are providing a single input, the output will be a 2D numpy array with one value.

3.3. Refinement

3.3.1. The benchmark model can be refined as below:

3.3.1.1. Use stationary scaled data with ARIMA model to achieve better RMSE.

3.3.2. LSTM model can be refined as below:

3.3.2.1. Initially, I used the windows method; however, with windows you lose several data points in train and test sets. Also, during test prediction, it's not using any test data point in account when predicting the next one; thus, predictions are very inaccurate.

3.3.2.2. Then I tried current method; however, finding correct value of epochs, neurons, and batch size was very challenging.

3.3.2.3. I overcame these challenge by keeping two values constant and used multiple third values and ran in loop for 50 times each. Number of neurons was determined by keeping batch size and number of epochs constant and running different neurons [1, 2, 3, 4, 5] in loop for 50 times each. For each neuron at each run, I accumulated RMSE and created box and whisker plot. I picked a value of neuron that has the best RMSE value. Similar experiment was done for epochs with the values of [100, 500, 1000, 3000, 5000] in loop for 50 times each. I picked a value of epochs that has the best RMSE value. I kept batch size constant at 1.

4. Results

4.1. Model Evaluation and Validation

Once I refined the model as described in above refinement section, I started to get desired RMSE outcome. I feel very comfortable with the results I achieved with the final model I have presented here. I tested the model with various inputs of unseen test data and the predictions are reasonable. I feel very strongly that with few enhancements I have mentioned in refinement section, this model can be a robust and trustworthy to use at my work place.

4.1.1. Demand Forecast chart with LSTM model:

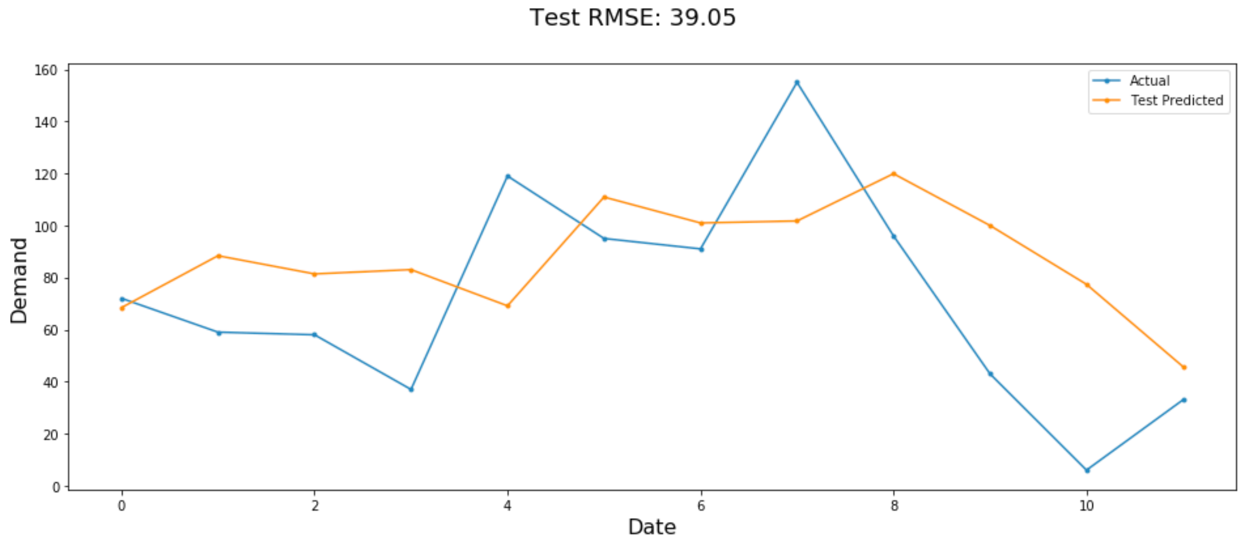

Figure 11: Demand forecast with LSTM model

4.2. Justification

With benchmark model, I get RMSE of 107.25. With my proposed LSTM model, I was able to improve that to 39.05. That is significant enhancement over benchmark model. As I have discussed in earlier section of this paper, I believe I can further enhance the model by suggested enhancements. I feel extremely comfortable to propose this solution to my work place to solve the accurate demand forecast problem.

5. Conclusion

5.1. Free-Form Visualization

5.1.1. Prediction chart with benchmark model:

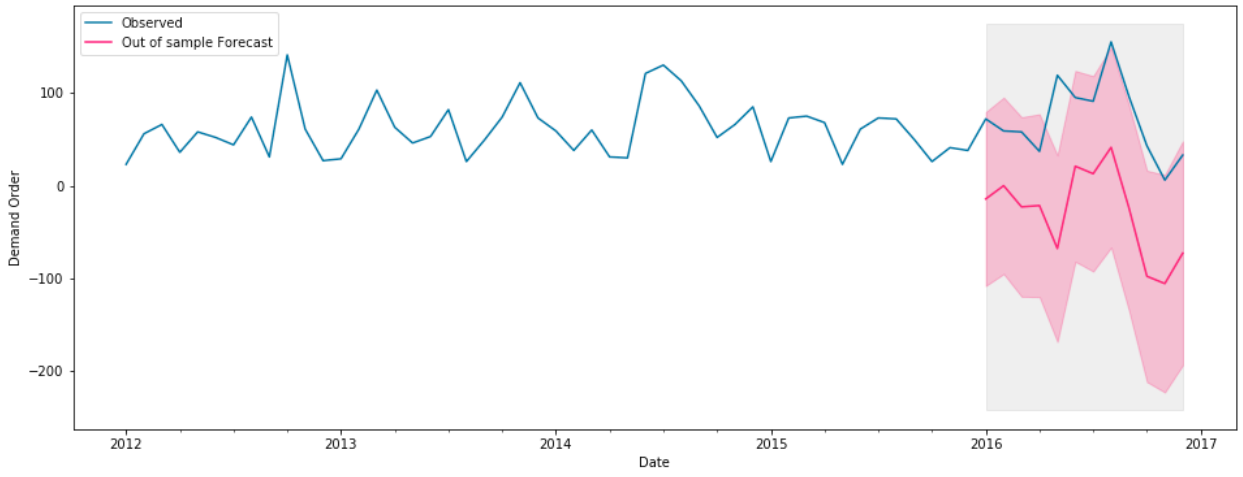

Figure 12: Demand forecast with benchmark model

5.1.2. Prediction chart with proposed model:

Test RMSE: 39.05

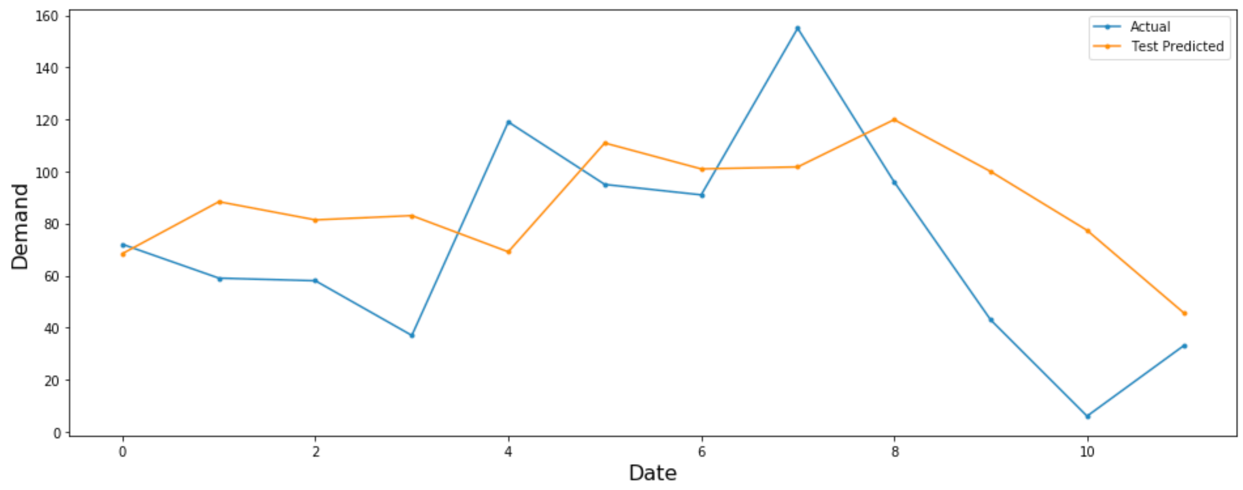

Figure 13: Demand forecast with proposed model

5.2. Reflection

I began the project with below problem statement:

"Overestimated forecast in supply chain causes excess inventory that stagnant significant dollar amount; on the other hand, underestimated forecast causes unfulfilled orders, angry customers, as well as lost sales and opportunities. The goal of this project is to develop a solution using the latest concepts of neural network that predicts accurate demand forecast for networking products."

The process I used to solve the problem can be summarized with steps below:

5.2.1. Relevant datasets were found at Kaggle site.
5.2.2. The data was downloaded and preprocessed.
5.2.3. Data was analyzed for stationary.

5.2.4. A benchmark model was created.
5.2.5. Data was converted for supervised learning and scaled.
5.2.6. LSTM model was created with appropriate inputs and compiled.
5.2.7. Train predictions were performed.
5.2.8. Model was validated using the test set of data.

I had most difficulties and took long time to fine tune steps 5.2.6, 5.2.7, and 5.2.8. Once I got through those steps, the model provided the results to my expectation.

5.3. Improvement

Besides the enhancements I mentioned in refinement section, I would also like to try below improvements:

5.3.1. The benchmark model can be improved to achieve better RMSE as below:
5.3.1.1. Use stationary data instead of series data
5.3.1.2. Scale the data to improve the outlier points.
5.3.1.3. Use stationary scaled data with ARIMA model and see how the RMSE fairs against LSTM.
5.3.2. LSTM model can be improved to achieve better and stable RMSE as below:
5.3.2.1. Each time I train the model, it gives different RMSE. I need to find a way to get constant RMSE. May be one way to apply random number seeding to Keras.
5.3.2.2. Another way to achieve this by running the experiment multiple times and taking average of RMSE.
5.3.2.3. Even though values of epoch and neurons have been fine-tuned, I also like to add support for variable batch size, fine-tune for that, and optimize it further for epoch and neurons.
5.3.2.4. Try various lag values and observe the effect of RMSE values for each.
5.3.2.5. Try adding dropout layer to the model
5.3.2.6. Try with sigmoid activation and see the difference

6. Reference

- http://colah.github.io/posts/2015-08-Understanding-LSTMs/
- https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/
- http://www.business-science.io/timeseries-analysis/2017/08/30/tidy-timeseries-analysis-pt-4.html
- https://machinelearningmastery.com/time-series-forecasting-performance-measures-with-python/
- https://www.analyticsvidhya.com/blog/2018/02/time-series-forecasting-methods/
- https://www.kaggle.com/sydjaffy/historical-product-demand
- https://www.ijraset.com/fileserve.php?FID=381