A Project Report on

A Neural Network based Linear Ensemble Framework for Time Series forecasting

Submitted By

Akash Kabra 2016B3A70562P
Ayush Mungad 2016B3A70523P

Submitted to

Prof. NVM Rao

For fulfillment of the course

ECON F266 - STUDY PROJECT
On Nov 26, 2019

Outline

Introduction

Time series forecasting has been a matter of interest to econometricians ever since the last two centuries. This interest is majorly driven by the *huge spectrum* of applications like setting monetary and fiscal policies, state and local budgeting, financial management, and financial engineering, stock market predictions, etc. But the efforts of fitting complex non-linear economic situations by linear econometric models go in vain owing to the complexity of real-life situations.

Combining time series forecasts from several models is a **fruitful alternative to using only a single individual model**. In the literature, it has been widely documented that a combined forecast improves the overall accuracy to a great extent and is often better than the forecast of each component model. The intuition behind combining forecasts is that a single linear model may not fit complex data at all times. It may be possible that the sample looks linear, but the actual behavior of the population is non-linear in nature. This issue can be resolved by **combining the forecasts of multiple models**, which exclusively handle different forms of data behavior so that the final combination made is robust to every possible behavior that the statistic may take.

A linear ensemble is the most common approach to combining multiple forecasts. For the dataset

$$\hat{Y} = [\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_N]^{\mathrm{T}}$$

And it's N forecasts,

$$\hat{Y}^{(i)} = \left[\hat{y}_1^{(i)}, \hat{y}_2^{(i)}, \ldots, \hat{y}_N^{(i)}\right]^{\mathrm{T}} \ (i = 1, 2, \ldots, n),$$

Linear ensemble is:

$$\hat{y}_k = w_0 + w_1\hat{y}_k^{(1)} + w_2\hat{y}_k^{(2)} + \ldots + w_n\hat{y}_k^{(n)} \forall k = 1, 2, \ldots, N.$$

where the sum of all w's is 1.

The most important part of a linear ensemble framework is getting the weights for getting the final forecast. Neural networks have proven successful in the machine learning literature to fit models precisely. Hence, we propose to use **a neural network-based model for deriving the weights of the linear ensemble**, to get a robust and scalable time series forecasting framework. We propose that this framework would be applicable to all kinds of econometric datasets, without tuning any set of parameters for fitting the model.

Objective

Time Series forecasting is an important topic in many areas of research not only in econometric applications but also in many other fields as well. So, building models that are accurate will help in modeling future scenarios to the best.

The generally used method up till now as to check various models either econometric or machine learning models on the in-sample dataset and then selecting the best among them for desired out sample forecasts. But there are a number of serious *limitations* of this approach:

1. Time Series distributions generally do not follow **independence and identical distribution(i.i.d**) property that is generally a fundamental requirement of real-life statistical processes. So, a model that performs better on a sample dataset might not always provide the best forecasts for unseen future values.

2. A forecasting model is specific to the nature of the time series, i.e. whether the series is generated from **a linear or non-linear process**, follows a **stationary or non-stationary distribution**, **contains trend**, **seasonal, or cyclical patterns**, etc. Estimating the exact nature requires a large number of historical observations, but in practice, there is only a very small sample of available data and so the fitted model may be inappropriate

3. Time Series is a **highly dynamic process** that keeps on changing continuously with a high degree of uncertainty. Also, an individual model may be prone to faulty assumptions, implementation error, and biases which considerably affect forecasts.

So, these limitations of single model forecasting models and the high importance of accurate time series model predictions drive us to consider a **combination of multiple forecasts**, and many studies in the existing literature have shown that these typed of combinations improve the forecasting accuracy to a large extent.

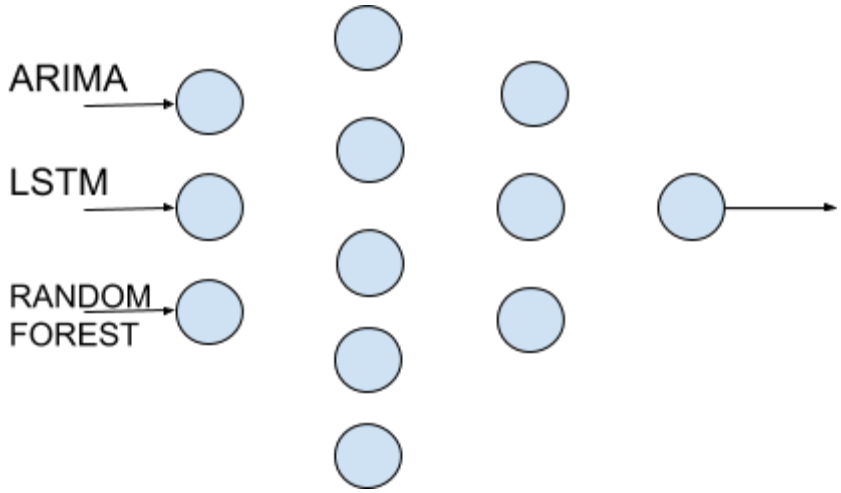
Now, there have been multiple econometric as well as machine learning-based methods for modeling time series. In the method of combining multiple models, we aim to use both of them i.e take some econometric based models and some machine learning models and forming an ensemble of both the techniques. The advantage it gives that econometric model that captures the relationships which a machine learning model might not have captured and vice versa. **We use the AR, ARMA, ARIMA models** from the econometric theory, and **LSTM based Recurrent Neural Networks and Random Forest Models** from the machine learning domain.

Methodology

We used **Airport Passenger monthly dataset** for the years 1949 to 1969. We divided the dataset into train, validation and test sets of size 100, 44 and 25 respectively. Now the following methods were used to get forecasts on the validation dataset:

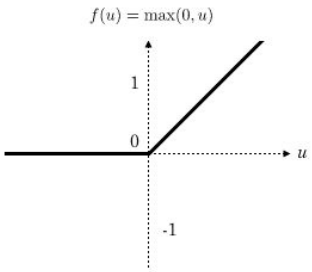
1. ARIMA
2. LSTM
3. Random Forest

After getting the (ŷ) estimated of all 3 models, they are fed into a neural network to get the final ensemble output. The architecture of neural network is as shown below:



Between the two hidden layers, the non-linear **"ReLu" activation** function is applied. Mean Average Percentage loss function is used and **Adam optimizer** is used to get the results.

Basically, all three results are passed into a non-linear ensemble network to get the final result.



The layers are specified as:

```
# define the keras model
model = Sequential()
model.add(Dense(5, input_dim=2, activation='relu'))
model.add(Dense(3, activation='relu'))
model.add(Dense(1, activation='relu'))
```

```
model.compile(loss='mean_absolute_percentage_error',
optimizer='adam', metrics=['accuracy'])
model.fit(x, y, epochs=300, batch_size=1)
```

Algorithm for the same can be written as follows:

- Divide the dataset into Train(*tr*), Validation(*vd*) and Test(*ts*) dataset.
- For every y^{tr} in train dataset, fit all three models on them, to get predictions y^{LSTM} , y^{ARIMA} , y^{RF} .
- $for\ (i = 0;\ i\ < N^{tr};\ i++)\{$

  Feed these outputs as input to the neural network.

  $net\ = \Sigma w^{(i)} * x^{(i)}$

  $out\ = f(net),\ where\ f\ is\ the\ activation\ function$

  In the case of ReLu, $f(x)\ = x\ (if\ x\ >\ 0),\ f(x)\ =\ 0\ otherwise$

  Repeat for all neurons.

  Backpropagate the error by computing gradients dL/dw where L = loss value and w = weights of neural network.

  $w\ =\ w\ -\ \alpha * (\ dL/dw\ )$

$\}$

  /* Training is over now */

- Feed Forward the same network to get predictions for the ensemble framework.

Techniques Used

The accuracy of a linear combination of forecasts primarily depends on the associated combining weights.

So, the technique that we deploy uses a neural network to adjust these weights in the ensemble that is to be formed. We use the results from two types of techniques to form the ensemble:

1. Statistical or Econometric forecasting models.
2. Machine Learning-Based models.

The statistical model that we use for forecasting is the ARIMA model.
The machine learning model that we use is the RNN(Recurrent Neural Network) and LSTM(Long Short Term Memory) based model.

The details about these models are explained below:

ARIMA:

The Autoregressive Integrated Moving Average (ARIMA) models, pioneered by Box and Jenkins, are the backbone of almost all statistical forecasting methods.

– **AR** Autoregression. 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 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.

Each of these components is explicitly specified in the model as a parameter. Note that AR and MA are two widely used linear models that work on stationary time series, and “I” is a preprocessing procedure to “stationarize” time series if needed.

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.
– **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.

A value of 0 can be used for a parameter, which indicates to not use that element of the model. We are dealing with monthly data, so we used 12 lags in the explanatory variable and the parameters used are:
AR = 12, I = 1, MA = 3

We are finally able to get an **R-Squared of 89.97%** on validation data.

RNN with LSTMs:

To understand the significance of an LSTM, let us consider an example: Sales of Mangoes, Y_t depends on T_t-12 primarily, because of the seasonal behavior of sales of Mangoes. If we model this Y_t using RNNs, we would not be able to get accurate predictions as long term dependencies are lost in RNNs. Here comes LSTM. LSTM stands for Long Short term memory. LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn!

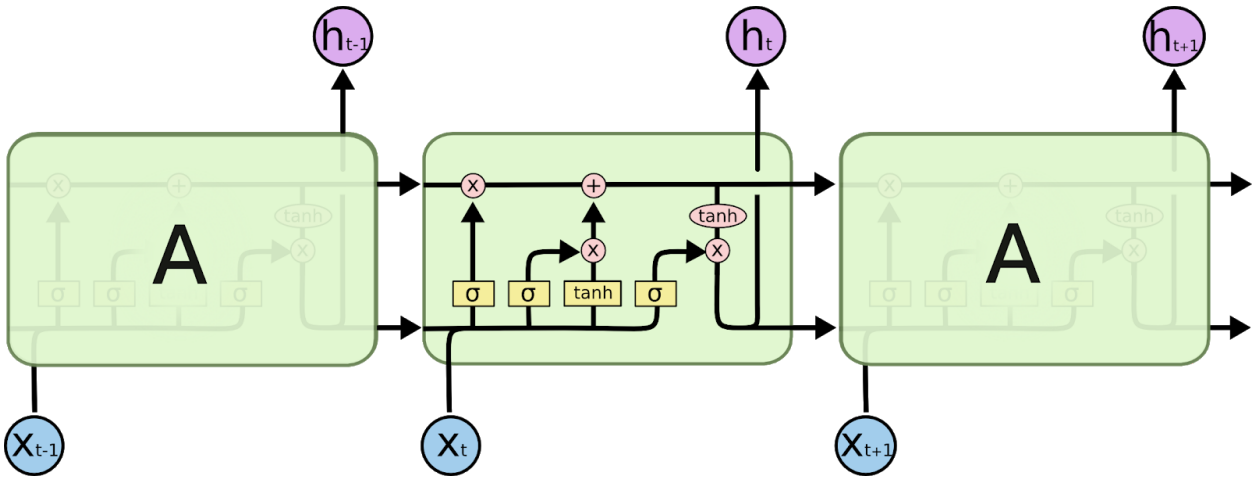


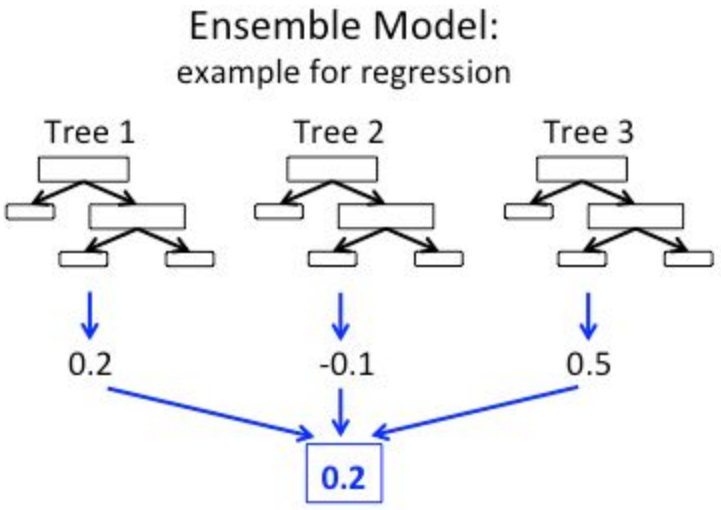
Fig: Structure of LSTM:

LSTM has 3 types of gates: Input gate, Forget gate and output gate. LSTMs are widely used for time series forecasting.

On our airline passenger dataset, LSTMs gave an **R-Squared of 74.58%**

Random Forest:

More recently a new class of regression models have been developed to address the challenges associated with the classic models. This method starts by **creating decision trees**. Decision trees **recursively partition data** in the regression space until the amount of variation in the subspace is small. A predictor for the subspace can then be created simply by taking the average value of the dependent data corresponding to the independent data in the subspace. The recursive partitioning step can be visualized as a tree, hence the name. Predictions for new data are obtained by finding the predictor corresponding the partition where the new input variable resides.



The *partition process* for Random Forest is greedy and, as a result, does not generally converge to the globally optimal tree.

To compensate for this a collection or ensemble of **locally optimal trees**, each tree is generated by sampling uniformly at random from the original subset, a procedure termed bagging.

Furthermore, after creating the resampled version of the data set, all but a small number of features are sampled. With the sampling procedure complete a new tree is trained.

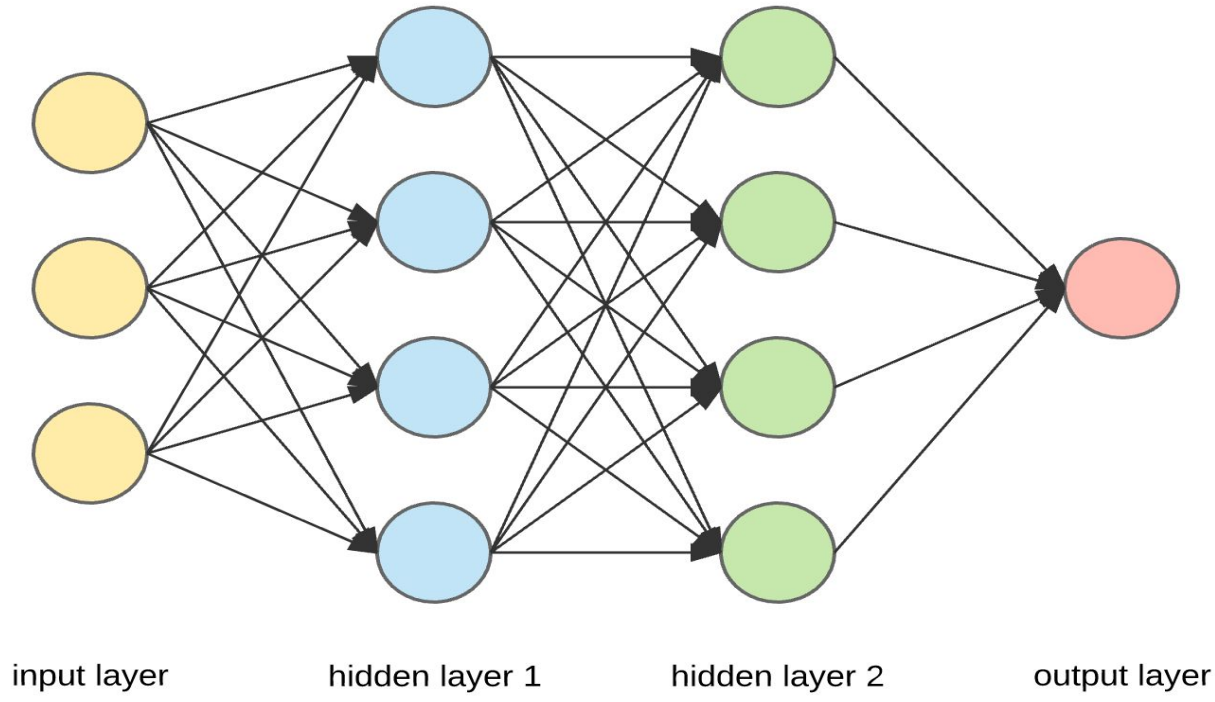
The collection or ensemble of trees is termed a **forest**. Predictions can then be made based on an aggregate of the individual predictions made by each of the trees, a procedure termed **“voting”**. An aggregate prediction obtains better accuracy than any of the constituent trees

.

While Random Forest models often provide superior prediction accuracy, they are often challenging to interpret, generally people term algorithmic models including Random Forests as a **“black box”**. For Random Forests specifically, each of the constituent trees trains on a potentially non-linear regression space and is then combined with others.

On our airline passenger dataset, Random Forest gave an **R-Squared of 84.26%**

Ensemble:



Now, since we have the results from both models, now we are in a position to make the ensemble using these results.
The ensemble is basically another neural network, in the input layer are the output values obtained from the component models. Then we have hidden layers and the output is the ground truth value of y, the neural network is trained on the basis of the given data.

So, after the training is over now our neural network has adjusted the weights that are to be given to the component models while making predictions using the ensemble model. For the upcoming years now we can predict the results using the ensemble network.

RESULTS: Using the ensemble model, we have obtained an **R-Squared of 87.77%.**

We can see that our framework is robust and is a fruitful alternative to using only a single individual model.

Rationale behind techniques used:

It can be observed that using correct methods as input to ensemble are as important as the hyperparameters of ensemble itself. It is necessary that the model is able to reproduce a robust framework which can take any form of behavior that data takes. It should contain **mutually exclusive** techniques to forecast every possible behavior.

- ARIMA is remarkably efficient in forecasting time series, generated from **purely linear processes.** But, due to this linearity restriction, their accuracy is not so impressive for nonlinear time series datasets
- LSTM are self-adaptive and data-driven, as they do not require any prior assumption about the data generation process of the time series. They can capture long term dependencies and non-linearities that may not be covered by ARIMA.
- LSTM also helps in performing nonlinear **time dependent mappings** with introducing a context layer and feedback connections. But, this feature comes with an additional computational overhead.
- Random forest is a completely different approach. overcomes a major shortcoming of neural networks by always producing the **unique global optimal solution.** Computational cost of Random forest is also low, in contrast to vanilla RNN and LSTMs.

These three approaches together are able to cover each other's shortcomings and make a **self-supportive framework** robust to every complex situation which may be required to be forecasted.

It must be noted that our approach is **completely data independent.** The strength of this approach lies in the fact that the same algorithm/approach may be used to forecast any possible time series model. It may not be necessary to adjust any hyperparameters or try different approaches to different models.

Another necessary point is that this approach may be used for **long term forecasting.** Accurate and reliable estimates would be provided by this approach.

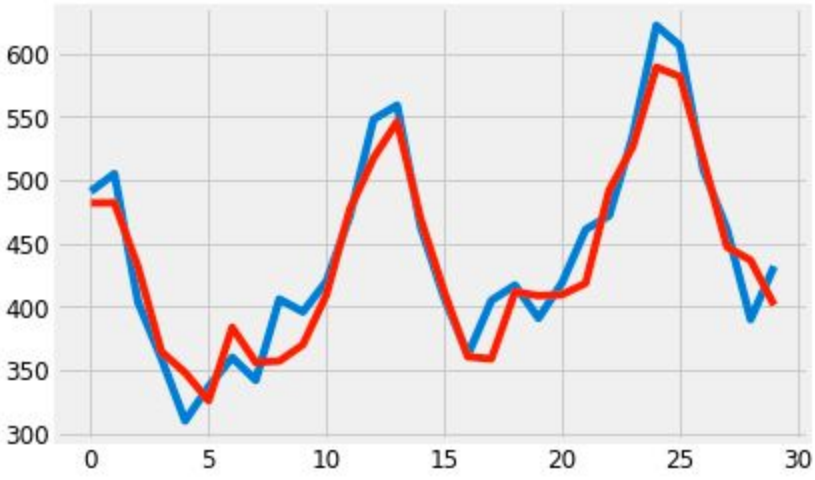
RESULTS AND CODE SNAPSHOTS

ARIMA

Code snapshot:

```
# ARIMA
from statsmodels.tsa.arima_model import ARIMA
from sklearn.metrics import mean_squared_error
from math import sqrt

model_ARIMA = ARIMA(y.passengers, order=(9, 1, 3))
model_ARIMA_fit = model_ARIMA.fit()
start_index = y.index[13]
end_index = y.index[-1]

#Predictions
ARIMA_pred = model_ARIMA_fit.predict(start=start_index,
end=end_index, typ= 'levels')
# ARIMA_pred contains predictions on unseen dataset
plt.plot(y.passengers)
plt.plot(ARIMA_pred, color='red')
```

Results:



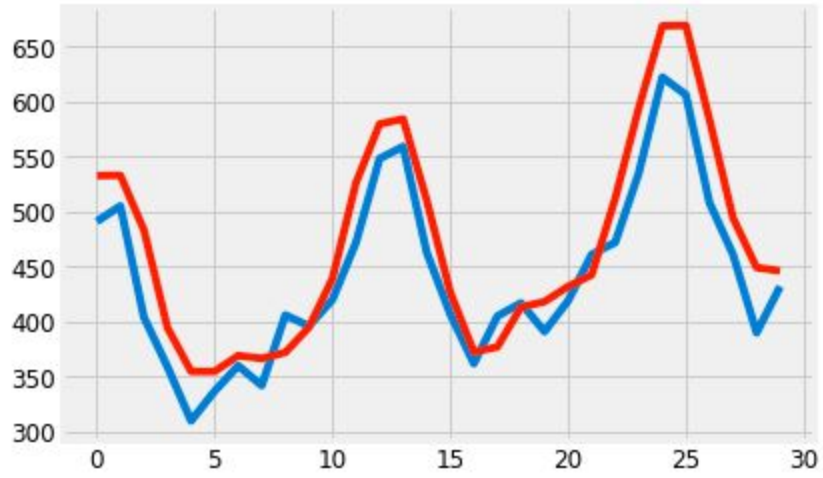
$R^2 = 89.97\ \%$

LSTM :

Code snapshot:

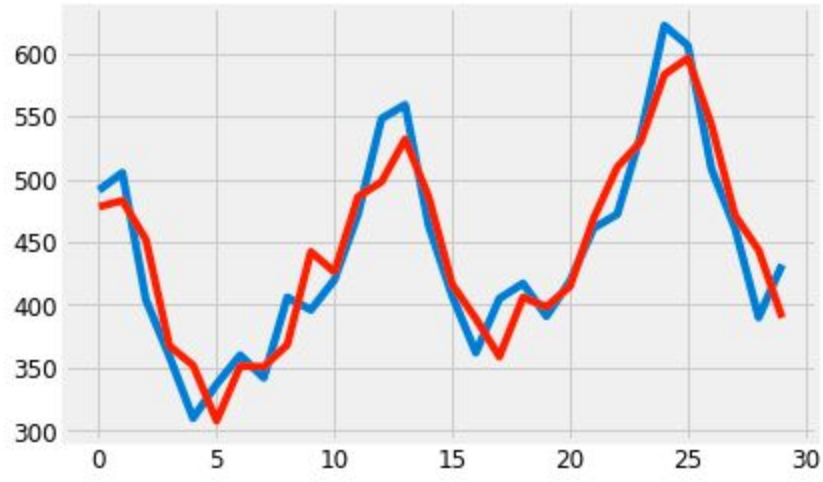
```
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.layers import LSTM
model_k = Sequential()
model_k.add(LSTM(100, input_shape=(1,12)))
model_k.add(Dropout(0.1))
model_k.add(Dense(1))
model_k.compile(loss='mean_squared_error', optimizer='adam')

history = model_k.fit(x_t_reshaped, y_train,
validation_data=(x_val_reshaped, y_val),epochs=100, batch_size=12,
verbose=1)
x_test = x_test.values
x_te_reshaped = x_test.reshape((x_test.shape[0], 1, x_test.shape[1]))
res_test = model_k.predict(x_te_reshaped)
# res_test contains predictions on unseen data
```

Result:



$R^2 = 74.58\ \%$

Random Forest *:



$R^2 = 84.26\ \%$

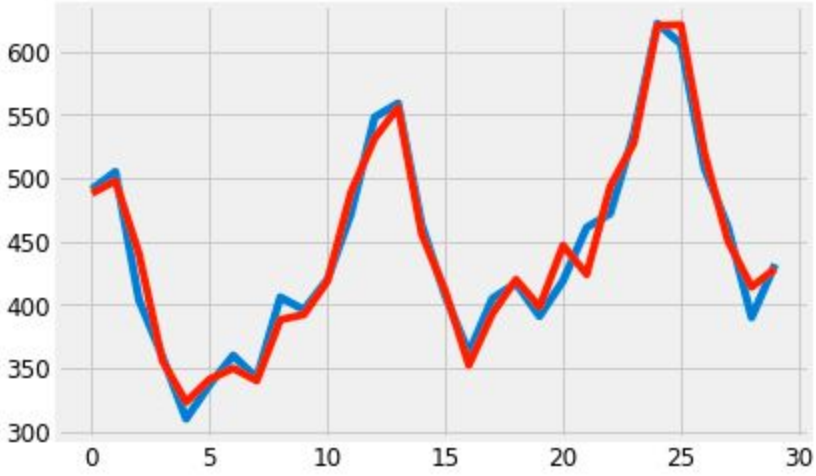
**code not added due to similarity with ARIMA*

Ensemble Framework:

Code snapshot:

```
Import pandas as pd
df = pd.read_csv("train.csv")
# train.csv contains forecasts for all 3 methods and the ground truth
x1 = df['1']
x2 = df['2']
x3 = df['3']
y = df['gt']
# define the keras model
model = Sequential()
model.add(Dense(5, input_dim=2, activation='relu'))
model.add(Dense(3, activation='relu'))
model.add(Dense(1, activation='relu'))
model.compile(loss='mean_absolute_percentage_error',  optimizer='adam',
metrics=['accuracy'])
model.fit(x, y, epochs=300, batch_size=1)
loss, acc = model_load.evaluate(x_val, y_val, verbose=0)
plt.plot(y_val)
plt.plot(y_pred)
```

Results:



$R^2 = 87.77\ \%$

CONCLUSIONS

Time series analysis and forecasting is a dynamic research area, having fundamental importance in numerous practical fields.

Improving the accuracy of time series forecasts is a challenging task that has been gaining continuous research attention from the past few decades. Extensive works have been performed on combining forecasts from several time series models with the general conclusion that this practice improves the **forecasting accuracy** to a large extent.

In this project, we tried to develop an effective technique for forming an **ensemble** using a novel **neural network based architecture:**

So, we First apply the individual methods separately and calculate the predictions of each method.
Secondly, the neural network method is useful in determining the **intrinsic weights** to be given to each of the component models. It also captures any non-linearity that might be among the combination of component models.

The empirical study with these 3 models and airport passenger time series data clearly shows that the proposed combination method has achieved significantly better accuracies in the long run and on an average would provide better results than the component models in the long run.

FURTHER SCOPE:

1. This method can also be explored for other large collections of time series and for an extended set of component models. Moreover, other variety of neural networks can be utilized as well, for recognizing the in-sample weight patterns.

2. Extensive experimentation can be done to come up with novel architectures that can model the ensemble in a more precise way.

3. Other econometric as well as machine learning methods can be added to the ensemble. The methods which independently capture different properties of the data should be chosen, as we get to capture the various relationships within the data.

BIBLIOGRAPHY:

1. https://www.sciencedirect.com/science/article/pii/S0925231215000338
2. https://colah.github.io/posts/2015-08-Understanding-LSTMs/
3. https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/
4. https://towardsdatascience.com/lstm-for-time-series-prediction-de8aeb26f2ca
5. J.M. Bates, C.W.J. Granger, The combination of forecasts
6. https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/
7. https://towardsdatascience.com/machine-learning-part-19-time-series-and-autoregressive-integrated-moving-average-model-arima-c1005347b0d7
8. https://petolau.github.io/Ensemble-of-trees-for-forecasting-time-series/
9. http://proceedings.mlr.press/v39/oliveira14.pdf
10. https://link.springer.com/chapter/10.1007/978-3-319-48317-7_13