Efficient Market Hypothesis from the perspective of deep learning - An Indian context

Vijaya Senthil Kumar Kuppusamy

Chennai, India

1 INTRODUCTION

1.1 Domain Background

Financial forecasting is a complex subject as pricing of an asset class depends on a number of factors. The main focus of this Capstone project is the analysis of equity and derivatives pricing of Indian market using deep learning methods.

There have been quite a few economic theories about market pricing of which efficient market hypothesis (EMH) is the most accepted and widely practiced one [1]. EMH poses that all the available information is fully and instantly incorporated by the markets in its pricing. Thus, any analysis, both technical and fundamental to predict the future direction of prices is fundamentally flawed. If any anomalous behavior exists in the market which enables a market participant to profit, such patterns decay over time.

There are three forms of efficient market hypothesis, weak form which proposes that the information set is limited to past price and its associated statistical parameters. In semi-strong form of EMH the past price of the trading instrument and also the publicly available information about the instrument is included in the information set and in strong form, the information set contains all the public and private information about the instrument.

While there have been many studies to test the EMH on the past data of stock and index prices using machine learning forecasting models, they are predominantly limited to US and European financial markets. The results of these experimental studies are mixed but few poses a challenge to EMH which demonstrates that there are predictable patterns exist in the past which enables a market participant to profit from.

Earlier statistical methods and stochastic analysis were being used to make stock price prediction but machine learning techniques are being used to predict stock price in recent years. In their research, Support Vector Machines for Prediction of Futures Prices in Indian Stock Market, Shom Prasad Das and Sudarsan Padhy [1] discuss Back Propagation Technique and Support Vector Machine Technique to predict futures prices traded in Indian stock market. Then advancement came in the form of recurrent neural nets as depicted in Comparative Study of Stock Trend Prediction Using Time Delay, Recurrent and Probabilistic Neural Networks by Emad W. Saad [2]. Then further improvement was observed using LSTM (long-short term memory) approach by Murtaza Roondiwala[3] et al., to forecast stock index prices.

2 PROBLEM STATEMENT

The purpose of the project is to create a neural network model that can predict stock price by using historical information as a time-series data. The task is to build a stock price predictor that takes daily trading data over a time of certain years as input, and outputs projected estimates for given query dates. The inputs will contain multiple metrics, such as opening price (Open), highest price the stock traded at (High), lowest price of the stock for the day (Low) the developed system will predict the Close price.

The following tasks are performed as part of the Capstone project.

- Explore Nifty and BankNifty index prices
- Implement the basic benchmark model using linear regression
- Implement LSTM using keras
- Compare the results

3 EVALUATION METRICS

The appropriate metrics for this problem is R-squared and root-mean- squared-error. Root-mean-squared-error can provide the average deviation of the prediction from the true value, and it can be compared with the mean of the true value to see whether the deviation is large or small. RMS has nice mathematical properties which makes it easier to compute the gradient. However, mean absolute error (MAE) requires more complicated tools such as linear programming to compute the gradient. Because of the square, large errors have relatively greater influence on RMS than do the smaller error.

The formula for the RMS error is as below.

$$RMSErrors = \sqrt{\frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}{n}}$$

R-squared can provide degree of dependency of dependent variable on the independent variable. The coefficient of determination, denoted R2 or r2 and pronounced "R squared", is the proportion of the variance in the dependent variable that is predictable from the independent variable. The independent variable which is close price of the index for a day depends to some extent on the open price and the previous day's close price. The R squared error will be a good option to check the dependency of close price on open and previous day's close price.

4 ANALYSIS

4.1 Data Exploration

The data that is used in this project is the daily NIFTY 50 index data from year 2001 and the BankNifty index from the year 2005. NIFTY 50 is the index of 50 major Indian stocks trading the National Stock Exchange(NSE). BankNifty is the index that represents the stocks of major banks of India.

A glimpse of a typical Nifty 50 Index data is shown below:

```
dataset = pd.read_csv('nifty_50_1day.csv')
print(dataset.head())
```

```
        Date     Close      Open      High       Low  Volume  Change
0  19-Dec-01  1060.75   1082.50   1086.35   1057.10  49.99M  -1.99%
1  20-Dec-01  1062.00   1060.45   1064.80   1047.50  55.79M   0.12%
2  21-Dec-01  1050.85   1061.80   1061.80   1043.65  47.17M  -1.05%
3  24-Dec-01  1048.50   1051.00   1055.00   1032.60  41.90M  -0.22%
4  26-Dec-01  1034.25   1048.90   1058.30   1030.50  42.61M  -1.36%
```

There was no abnormality observed in the datasets, i.e, no feature is empty and does not contains any incorrect value as negative values.

The mean, standard deviation, maximum and minimum of the data are depicted below:

```
dataset.describe()
```

	Close	Open	High	Low
count	4252.000000	4252.000000	4252.000000	4252.000000
mean	5216.385360	5218.379198	5253.489699	5177.484537
std	2853.714012	2857.376872	2863.256538	2844.730809
min	922.700000	922.550000	931.050000	920.000000
25%	2804.775000	2803.912500	2842.725000	2762.975000
50%	5185.050000	5185.775000	5219.725000	5117.650000
75%	7617.125000	7631.812500	7675.837500	7580.450000
max	11738.500000	11751.800000	11760.200000	11710.500000

Statistics of the Nifty data

A glimpse of a typical Banknifty Index data is shown below:

```
bn_data = pd.read_csv('banknifty_1day.csv')
print(bn_data.head())
```

```
        Date     Close      Open      High       Low Vol  Change
0  10-Jun-05  3593.55   3640.05   3665.80   3585.80   -  -1.18%
1  13-Jun-05  3642.90   3593.90   3648.40   3556.70   -   1.37%
2  14-Jun-05  3689.80   3649.05   3696.35   3637.70   -   1.29%
3  15-Jun-05  3683.85   3700.80   3703.75   3662.85   -  -0.16%
4  16-Jun-05  3618.80   3684.40   3685.75   3607.20   -  -1.77%
```

```
bn_data.describe()
```

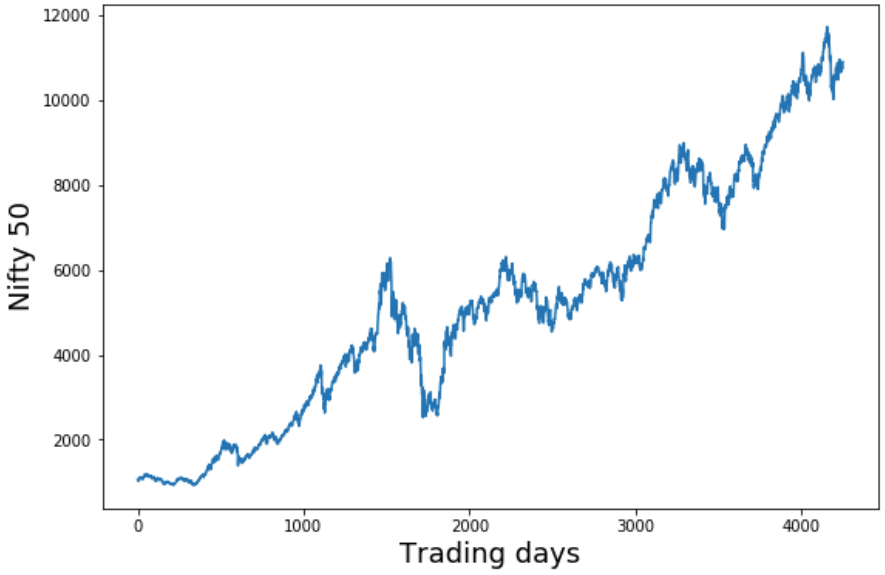
	Close	Open	High	Low
count	3374.000000	3374.000000	3374.000000	3374.000000
mean	12529.082869	12534.881283	12640.945984	12414.988826
std	6701.389841	6704.893778	6722.931446	6678.286703
min	3339.700000	3385.400000	3446.750000	3314.550000
25%	7106.175000	7114.500000	7227.800000	6995.875000
50%	10742.125000	10753.675000	10873.600000	10658.750000
75%	17600.325000	17622.275000	17714.825000	17485.625000
max	28320.000000	28379.900000	28388.650000	28171.650000

Statistics of the Banknifty data

For time series prediction, the open and low of the price cannot be considered as input because the open and low can be determined at the end of the day. So the relevant inputs for our model to predict the Close price is the Open price and the Close price of the previous day. We utilize these two inputs for our model.

2 Data Visualization

We want to predict the values of closing prices of the index and trend of the data is as shown below:



Plot of Nifty 50

The above plot is the Nifty 50, index of 50 stocks trading in Indian stock exchange vs trading days from (2001). As we can see from the plot that the trend is predominantly bullish with a major decline at after around 1500th day (during financial crisis of 2008). A lot of short-term volatility can be observed in the index price. Though the data needs to be stationary for

statistical methods to work, this won't be a bottleneck for neural network models like LSTM which can take in raw input data. One obvious limitation we have to keep in mid while analysin the result is that in the present analysis we are considering an index which is predominantly in bullish trend. Including different indexes/stocks with many different phases like bear trend, mean reverting trend can make the model robust, but this is not the scope of the project we are implementing now. The analysis with many different instruments is considered for the future work.

5 Algorithms and Techniques

The goal of this project was to study time series data i.e. when the data has a certain kind of sequence or dependence on the previous value and explore various possible ways to predict with a decent accuracy. LSTM, a modification of the RNN is a popular model which predicts the time series data with great accuracy and is widely considered in stock prediction.

The LSTMS are networks which contain memory cells which is controlled with the help of gates. A much more detailed explanation is as given below:

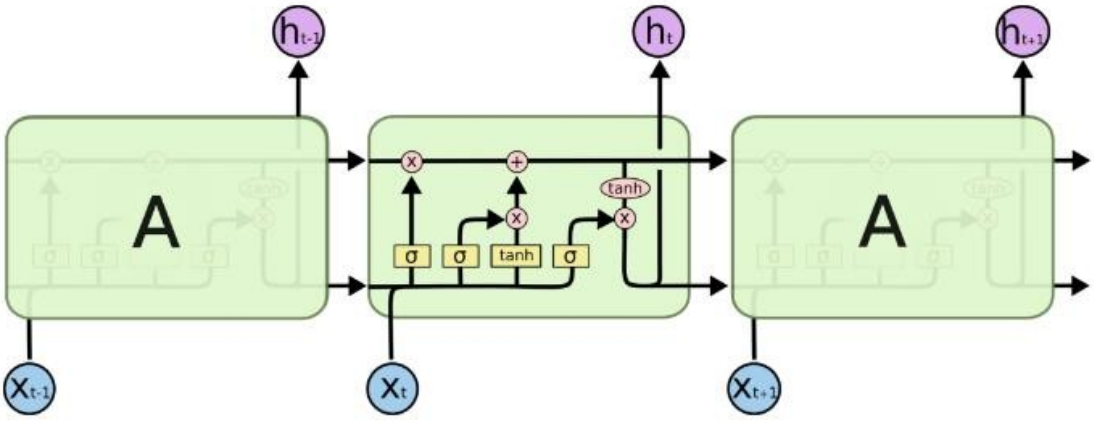
• Recurrent Neural Networks

The basic idea behind RNNs is to make use of sequential information. In a traditional neural network, we assume that all inputs are independent of each other. But in time series data, the output depends on the previous outputs. So to predict such sequences, RNNs, also called recurrent neural networks are used because they perform the same task for every element of a sequence, with the output being depended on the previous computations. The interesting feature by which this is accomplished is that they have a "memory" which captures information about what has been calculated so far. But RNN has a few drawbacks like exploring gradient and vanishing gradient problem.

• Long Short Term Memory

In order to solve the drawback of RNN, there came LSTMs which don't have different architecture from RNNs, but use a different function to compute the hidden state. The memory in LSTMs are called cells that take as input the previous state and current input. Internally these cells decide what to keep in (and what to erase from) memory. They then combine the previous state, the current memory, and the input. It turns out that these types of units are very efficient at capturing long-term dependencies hence it will be suitable for stock prediction problem.

The basic LSTM unit is composed of a cell, an input gate, an output gate and a forget gate as shown below:



FORGET GATE

A forget gate is responsible for removing information from the cell state. The information that is no longer required for the LSTM to understand things or the information that is of less importance is removed.

INPUT GATE:

The input gate is responsible for the addition of information to the cell state. This addition of information is accomplished basically in three-steps:

- Regulating what values need to be added to the cell state by involving a sigmoid function.
- Creating a vector containing all possible values that can be added
- Multiplying the value of the regulatory filter to the created vector and then adding this useful information to the cell state.

OUTPUT GATE:

This job of selecting useful information from the current cell state and showing it out as an output is done via the output gate.

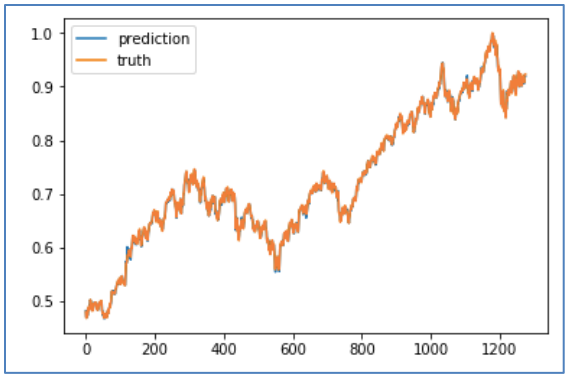
BENCHMARK MODEL

The benchmark model for this project would be using linear regressions since it's is a pretty simple model.

The results of the benchmark model are as follows,

```
from sklearn.metrics import r2_score, mean_squared_error
r2_score(y_test,predicted_price),np.sqrt(mean_squared_error(y_test,predicted_price))
```

```
(0.9996469447592496, 0.002385694020564292)
```



METHODLOGY

1 Data Pre-Processing

The first step of the project would be to acquire the data set in the form of the .csv file from www.investing.com . Remove unimportant features (date) from the acquired data.

In order to make neural network converge faster, there is a need to scale the inputs. The next step involves the normalizing of data which is a requirement for the LSTM networks which will be done using the function MinMaxScaler of the sklearn library.

Split the dataset into the training (70%) and test (30%) datasets for linear regression model. The split was of following shape:

```
print(X_train[1])
print(y_train[1])
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
[0.01273403 0.01235092 0.01181595]
[0.01287931]
(2976, 3)
(1276, 3)
(2976, 1)
(1276, 1)
```

Split the dataset into the training (70%) and test (3%) datasets for LSTM model. The split was of following shape:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(train_X, train_Y, test_size = 0.3,shuffle = False,random_state = 0)
print(y_train[1])
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
```

```
[0.01181595]
(2976, 3)
(1276, 3)
(2976, 1)
```

For the LSTM model, sequence generation is tried i.e., the concept of window to make a improved model. Rolling window of length 5 days is set, moving this window along the time series, and recording the data in each step as one row of input sequence.

2 IMPLEMENTATION

The benchmark model is implemented using the sklearn library as:

1) The first step is to import sklearn library which enable us to implement linear regression algorithm,

2) In the second step, the object for the above model is created so that the methods can be invoked.

3) Using the above model, we train the data by using fit function like model.fit() and to this function the input are feature vectors (X_train) and label vector(y_train).

4) The next step is to predict the values for the part of dataset which is not seen by the model while training so for that we use the function predict like model.predict() And the input to this function is the feature vector of unseen data.

5) Then since linear regression basically models the data using a line we can get its slope and intercept and for that the functions are:

Slope: **model.coef_** and Intercept : **model.intercep**

b) INITIALISING MODEL

```
In [27]: from sklearn.linear_model import LinearRegression
         model= LinearRegression()
         model =model.fit(X_train,y_train)
         predicted_price = model.predict(X_test)
         predicted_train = model.predict(X_train)
         print(model.coef_)
         print(model.intercept_)
```

The LSTM model's specifications are as follows:

1) This model will be implemented using Keras module which contains deep learning modules. After importing keras, the Sequential model needs to be imported with Dense, Activation and Dropout layer modules.

2) Next step is to create an object for the model by calling Sequential function of keras.

3) Then we add layers to the model using the function add

4) In this network, a LSTM layer is added which takes the 2-dimensional array as input and has dimension 300 i.e the number of neurons. The parameter return_sequences is set as true for all the layers except the last because we want full output sequence.

5) Then a dropout layer was added with a value of 0.2 as a parameter to the function which means that the probability of dropping a neuron is 0.2.

6) Then, again a LSTM and a dropout layer have been added.

7) The final layer is the output dense layer containing a single neuron, with a sigmoid function

8) Then function is compiled using loss function parameter 'mae', the 'nadam' optimizer and the metrics 'Mean Squared Error'.

9) At last, function to fit (i.e., train) the model using the dataset and input of number of epochs, batch size and validation split i.e., are defined as below.

```
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.layers import LSTM,Dropout

model = Sequential()


model.add(LSTM(units=300, return_sequences=True, input_shape = (X_train.shape[1], 2)))
model.add(Dropout(.2))


model.add(LSTM(units=150, return_sequences=False))
model.add(Dropout(.2))




model.add(Dense(units=1, activation='sigmoid'))

# Compiling the RN
model.compile(loss='mae', optimizer='nadam', metrics=['mean_squared_error'])

# Train :)
history = model.fit(X_train,y_train, epochs=100,validation_split=0.1, batch_size=32)

############ Save & load Trained Model ############
# Save Trained Model
model.save('TICKER-RNN.h5')
```

The loss function is chosen to be mean absolute error, and the optimizer is chosen to be Nesterov Adam, which is Adam RMSprop with Nesterov momentum and a track of the validation loss is kept so that we are able to prevent the model from over-fitting.

REFINEMENT

A new model has also been created using windowing:

```
def windowing(data, window):

    sequence = []
    for index in range(len(data) - window):
        sequence.append(data[index: index + window])
    return np.asarray(sequence)
```

```
window = 5
X_train2 = windowing(X_train, window)
X_test2 = windowing(X_test, window)
y_train2 = y_train[-X_train2.shape[0]:]
y_test2 = y_test[-X_test2.shape[0]:]

print("x_train", X_train2.shape)
print("y_train", y_train2.shape)
print("x_test", X_test2.shape)
print("y_test", y_test2.shape)
```

```
x_train (2970, 5, 2)
y_train (2970, 1)
x_test (1271, 5, 2)
y_test (1271, 1)
```

For windowing, a function windowing is written which takes the window length as input and then iterates through the whole data and for every data unit append the data of window i.e., contain=ing the next data points/units into one single array.

```
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.layers import LSTM,Dropout

model = Sequential()


model.add(LSTM(units=200, return_sequences=True, input_shape = (X_train2.shape[1], 2)))
model.add(Dropout(.2))


model.add(LSTM(units=100, return_sequences=False))
model.add(Dropout(.2))




model.add(Dense(units=1, activation='sigmoid'))

model.compile(loss='mae', optimizer='nadam', metrics=['mean_squared_error'])

history = model.fit(X_train2,y_train2, epochs=20,validation_split=0.1, batch_size=32)
```

Hyper-parameter tuning:

1) SEQUENCE LENGTH:

The sequence length of 5, 10, 20, 50(it is approximately the number of trading days in a month) was attempted and it turns out the result of 5 days was the best.

2) NUMBER OF LAYERS:

The neural network structure of 3 LSTM layers with hidden units of 200-100-100 and 2 LSTM layers described above was tried initially, and it was observed that the result with 2 layers was better than 3 layers and the best score was observed at 300-150.

3) THE LOSS FUNCTION.

Both mean absolute error and mean squared error were attempted, the mean absolute error was found to be more appropriate and gave best prediction.

4) THE OPTIMIZER

Among the optimizer, RMSprop, Adam and Nesterov Adam, and the result are pretty similar with only marginal difference. Nestrov Adam was utilized

5) THE NUMBER OF EPOCHS.

The result converged after using 50-100epochs. Larger number of epochs didn't make a difference.

RESULTS

Model Evaluation and Validation

Final parameters of the model are

1) NORMAL LSTM MODEL:

The number of neurons in layers are 300 -150

Batch size =32

Number of epochs = 100

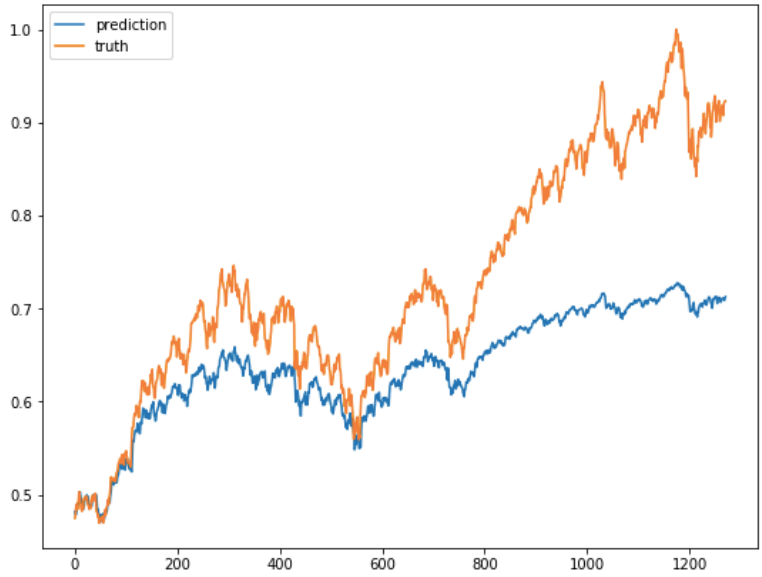
Results are:

```
from sklearn.metrics import r2_score, mean_squared_error
r2_score(y_test2,testPredict),np.sqrt(mean_squared_error(y_test2,testPredict))
```

```
(0.17096111760040789, 0.11493431021397163)
```

Plot of the prediction vs actual index data:

```
plt.figure(figsize=(9,7))
plt.plot(testPredict, label='prediction')
plt.plot(y_test2, label='truth')
plt.legend()
plt.show()
```



Nifty 50 test data perfomance

2) IMPROVED LSTM MODEL:

The number of neurons in layers are 200 -100

Batch size =32

Number of epochs = 20

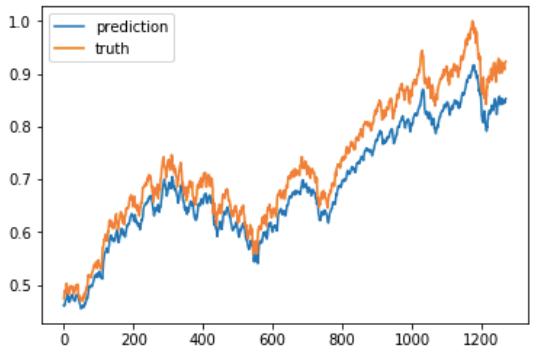
With sigmoid activation

CALCULATING ERRORS

```
from sklearn.metrics import r2_score, mean_squared_error
r2_score(y_test2,testPredict),np.sqrt(mean_squared_error(y_test2,testPredict))
```

```
(0.868274570637904, 0.04581387803265262)
```

```
plt.plot(testPredict, label='prediction')
plt.plot(y_test2, label='truth')
plt.legend()
plt.show()
```



The improved LSTM model with 5-day windowed input data have more predictive power than the normal LSTM model

MODEL ROBUSTNESS

In order to show that my model is robust I would like to give two arguments.

First that for my improved LSTM model, I am getting a R2 score of 0.86 which is a quite high value and shows that model predicts the correct values upto 86 %

Though the benchmark model have better R2 score and MSE value, it is highly likely that the linear regression may overfit for the out of sample data.

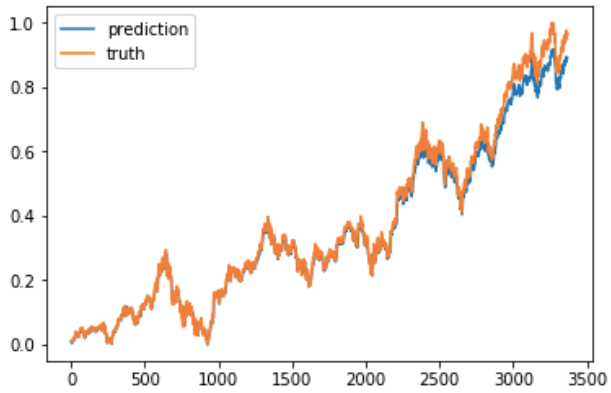
JUSTIFICATION

The model obtained by training the Nifty 50 data was applied to predict the Banknifty data, and the model could predict the direction and the price of the Banknifty pretty well. This key result tells that the price variation pattern among different trading instruments isnot very different. This give credibility to the idea of technical analysis of the asset prices which involves the study of chart to predict future prices.

```
from sklearn.metrics import r2_score, mean_squared_error
r2_score(train_Y[1:],trainPredict),np.sqrt(mean_squared_error(train_Y[1:],trainPredict))
```

```
(0.9888321362085575, 0.028326446746622394)
```

```
plt.plot(trainPredict, label='prediction')
plt.plot(train_Y, label='truth')
plt.legend()
plt.show()
```



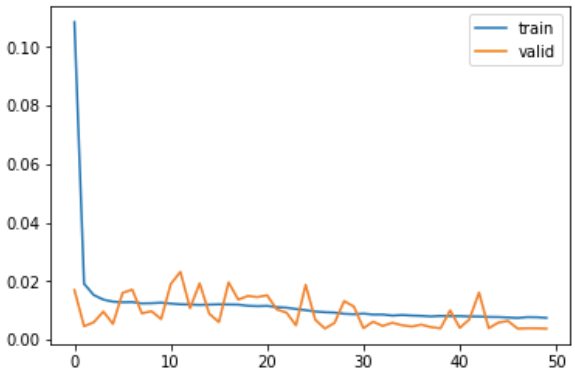
Model performance in the bank nifty data.

CONCLUSION

FREE-FORM VISUALIZATION

A very important topic for neural networks is its convergence of loss function. Not only the convergence will show whether the model is successfully trained, it will also tells you whether the model is under-fitting or over-fitting. The figure below shows the learning curve of the training process of the model. We can see from the curve that both the loss for training and validation set is decreasing as the number of epoch increases, and converges to almost zero. This indicates the model is successfully training, at least is converging into a local minimum. The validation loss is decreasing and is less volatile as the number of epoch increase, which indicates there is no over-fitting or under-fitting problem in the training process.

```
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='valid')
plt.legend()
plt.show()
```



The key result of the study is that deep neural networks show a promise in predicting the index prices of Indian Market and thus question the basic assumption of efficient market hypothesis that the stock market has no memory. More study with various index of the world stock markets are required to conclusively disprove efficient market hypothesis.

REFLECTION

To recap, the process undertaken in this project

- Incorporate data from www.investing.com
- Pre-process the requested data according to the requirements of the models
- Develop function for normalizing data
- Performing 70/30 split on training and test data across all models
- Develop Benchmark Model
- Setting up basic Linear Regression model with Sklearn library
- Develop Basic LSTM Model
- Calibrate hyperparameters
- Set up basic LSTM model with Keras and improving it to increase the accuracy.
- Analyze and understanding the results.

This project gave exposure and learning in a completely new model, LSTM neural networks, and also to explore the modelling of real time series data sets.

The major problem I faced during the implementation of project was exploring the data and preprocessing it and understanding how to appropriately shape and split the data for the two models: Linear regression and LSTM.

IMPROVEMENT

The modelling of stock/index prices could be further improved by using deep LSTM model or Stacked LSTM model and using GRU as a modification of recurrent neural networks which has shown more predictive power in time series prediction in recent literature.

REFERENCES

[1] B. G. Malkeil, The efficient market hypothesis and its critics, Journal of economic perspectives 17 (2003) 59–82.

[2[A. Timmermann, C. W. Granger, Efficient market hypothesis and forecasting, International Journal of Forecasting 20 (2004) 15–27.

[3] Das, Shom Prasad, and Sudarsan Padhy Support vector machines for prediction of futures prices in Indian stock market. International Journal of Computer Applications 41.3 (2012).

[4] Emad W. Saad, Student Member, IEEE, Danil V. Prokhorov, Member, IEEE, and Donald C. Wunsch, II, Senior Member, IEEE Comparative Study of Stock Trend Prediction Using Time

Delay, Recurrent and Probabilistic Neural Networks, IEEE TRANSACTIONS ON NEURAL NETWORKS, VOL. 9, NO. 6, NOVEMBER 1998

[5] Murtaza Roondiwala, Harshal Patel, Shraddha Varma Predicting Stock Prices Using LSTM, International Journal of Science and Research (IJSR),2017.