# Stock Market Behavior Prediction with Deep Neural Networks

Deepanshu Saini

## Abstract

Stock market trades are historically known to be processed manually. Automated systems are the exception rather than the norm. Humans are limited in their ability to understand the various factors that influence the stock market. With the recent increase in computing power and data gathering techniques, machine learning has become available to a much wider audience. We propose to build a machine learning model using Deep Neural Networks (DNN) to predict the behavior of stock markets. The model would be trained on historic data and would predict whether the stock price would go up or down with a probability guarantee. The first part of the project would involve data pre-processing i.e. collection and cleaning of data. The data would then be split into training and test sets. The model would be trained on the training set and then its accuracy would be tested using the test set. Initially we would start with simple models like Logistic Regression and Shallow Neural Networks and then, at a later stage of the project, work on models like LSTM's and DNN's which are good at predicting sequences. Towards the end we'll choose the best model for stock market prediction by comparing the results in each case.

# Table of Contents

## 1. Introduction

The amount of Money in the US Stock Market is about US$21.3 trillion. US $170 Billion value worth of stocks and funds are traded every day. 3,058 companies trading on the NASDAQ and 2,400 on the NYSE. Stock prices and Indexes are influenced by factors like Quarterly reports, Elections, Interest rates, Oil Prices and an incalculable number of other factors. *Can we predict how the markets would react to changes in these factors?* Machine learning systems have been used frequently in stock markets in the past 10 years. They are used both in automated trading systems and as recommendation engines which give suggestions to market traders.

## 2. Problem Statement

With a fund of 1 Million dollars and 1 year of time, maximize profit while trading on the Dow Jones Index using Machine Learning techniques. Compare the profits obtained using different machine learning methods with the interest that could have been earned with a savings account.

## 3. Data

For our project, we chose the Dow Jones Industrial Average (DJIA). It is a collection of 30 publicly traded stocks and it comprises some of the largest and most successful publicly traded stocks in the United States (e.g. Microsoft, Apple). Currently the DJI is experiencing its highest growth period ever. On November 30, 2017 DJI crossed 24,000 points which is highest amount ever recorded. The DJI value is computed by adding the stock prices of all the constituent stocks and dividing by the Dow Divisor. In case of splits or mergers the Dow Divisor is updated so that the DJI value remains constant.

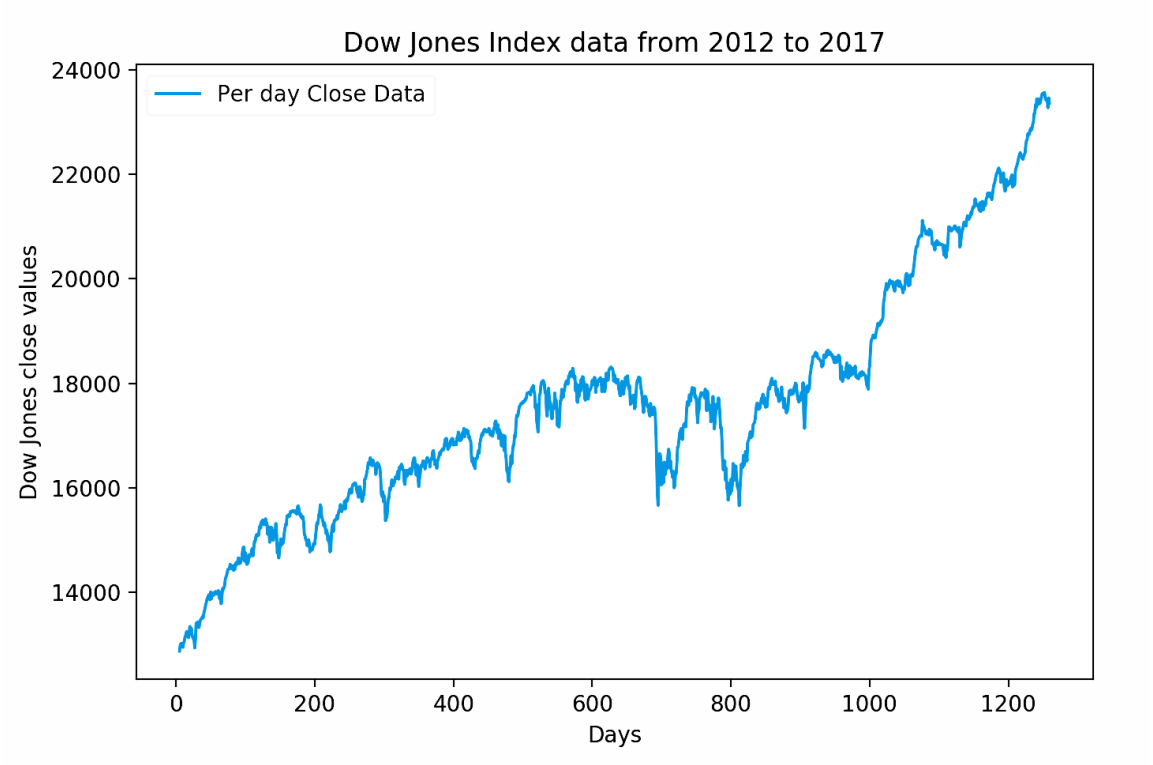
### 3.1 Data Source

We used 5-year historic Dow Jones data retrieved from Yahoo! Finance for training and testing our models. The data consisted of Open, High, Low, Close and volume of stock traded per day data present in a csv format.

### 3.2 Preprocessing

- We used the Pandas library for loading the data from the csv into a pandas data-frame. After loading the data columns other than closing price were dropped.
- Our dataset consisted of 1250 points closing price data of the Dow Jones Index. We split our data for validation as per the thumb rule of N/5 mentioned in class. 1000 days' data is used for training and 250 days for validation.
- We use Minimax scaling to transform our training data to a 0 to 1 scale. We took care to ensure that testing data was scaled separately and unintended data snooping did not occur.

$$X_{sc} = \frac{X - X_{min}}{X_{max} - X_{min}}$$



### 3.3 Feature Extraction

- We remove the open, low, high, volume columns and assemble last 5 days closing prices in a single row. These scaled closing prices will be our features.
- Our output label y is the $6^{th}$ day closing price which these features are supposed to predict.
- At the end, we have X_Train matrix as [1000, 5] Y_Train as [1000, 1]. X_test and Y_test as [250, 5] and [250, 1]

## 4. Tools Used

- *TensorFlow v1.4.0* – Most popular machine learning framework on GitHub. More than 25,000 commits and 40,000 forks. Used for training our Deep learning ANN and LSTM models
- *Scikit-learn v0.19.0* – An open source python machine learning library developed as a Google summer of Code project. Multi-Variable Linear Regression and scaling
- *Matplotlib v2.0.2* - Graphs
- *Numpy v1.13.3* - Matrix algebra and N-d arrays
- *Pandas v0.20.3* - Data preprocessing

## 5. Technical Approach

For each model input label is last 5 day closing price, output label is next day's predicted stock price. If next day predicted price is higher than last day predicted price we buy stock otherwise we sell all the stocks, we currently possess.

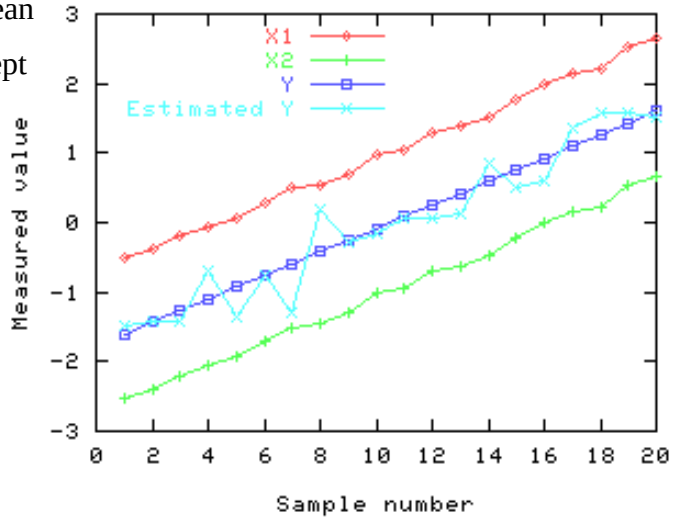
### 5.1 Linear Regression

- Multivariate linear regression is a baseline model which assumes the relation between the output variable y and dependent variables $x_0$ to $x_n$ is linear.
- It can reduce hypothesis to single number with a transposed theta matrix multiplied by x matrix.
- The multivariate multiple linear regression model has the form

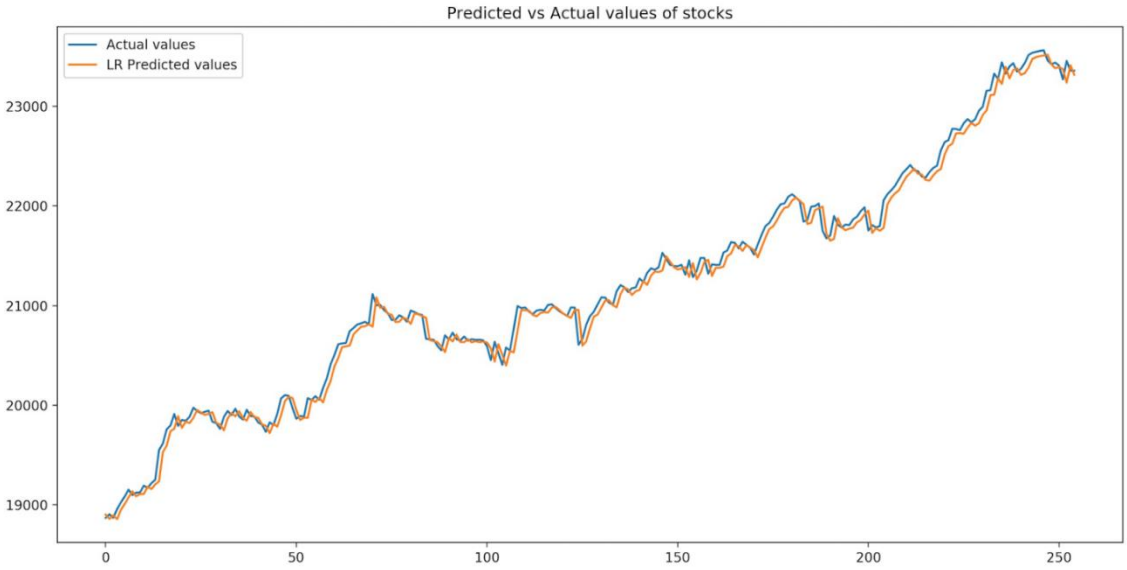
$$y_i = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \ldots + \beta_{p-1} x_{i,p-1} + \epsilon_i.$$

- **Y** is a matrix with series of multivariate measurements (each column being a set of measurements on one of the dependent variables)
- **X** is a matrix of observations on independent variables that might be a design matrix (each column being a set of observations on one of the independent variables)
- **B** is a matrix containing parameters that are usually to be estimated and
- **E** is a matrix containing errors (noise). The errors are usually assumed to be uncorrelated across measurements, and follow a multivariate normal distribution.
- For out Regression model we used mean squared error (MSE) to fit the intercept for our model. Multivariable linear regression is an excellent model when you are working with less data.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$



## Validation and Results




- Root Mean Squared error = 97.07, Profit earned = $152,000 for input fund of $1,000,000
- Predicted values lag the actual values. Not a true prediction.
- Model weights calculation is exponentially faster compared to the other models

## 5.2 Artificial Neural Network

- ANN is a computational model based on the structure and functions of biological neural networks. It's a great tool that can be used for modeling the statistical data.
- It is an effective way for dealing with the non-linear relations between system inputs and outputs. A neural network can handle the complex relationships inside the data pattern, generalize the data and come by a solution with a plausible error.

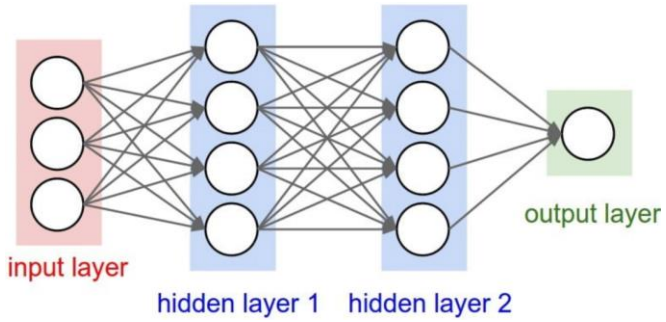



**Figure 1**. ANN with 2 Hidden Layers

- Our network is a 2-layer feed-forward network with backpropagation.
- The signal, in a typical feed-forward ANN, is introduced to the input layer neurons and transmitted to the next layers through the connections the neurons until the signal reaches the output layer.

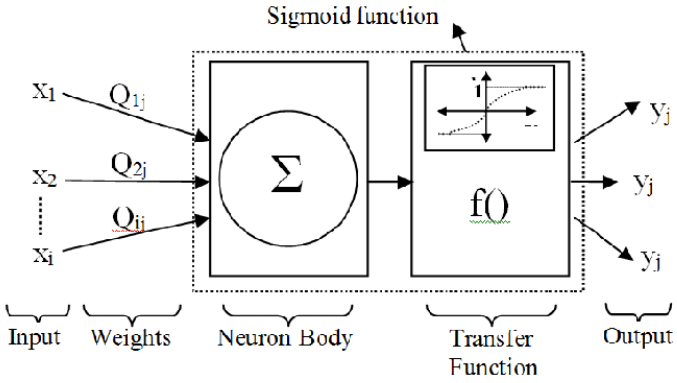
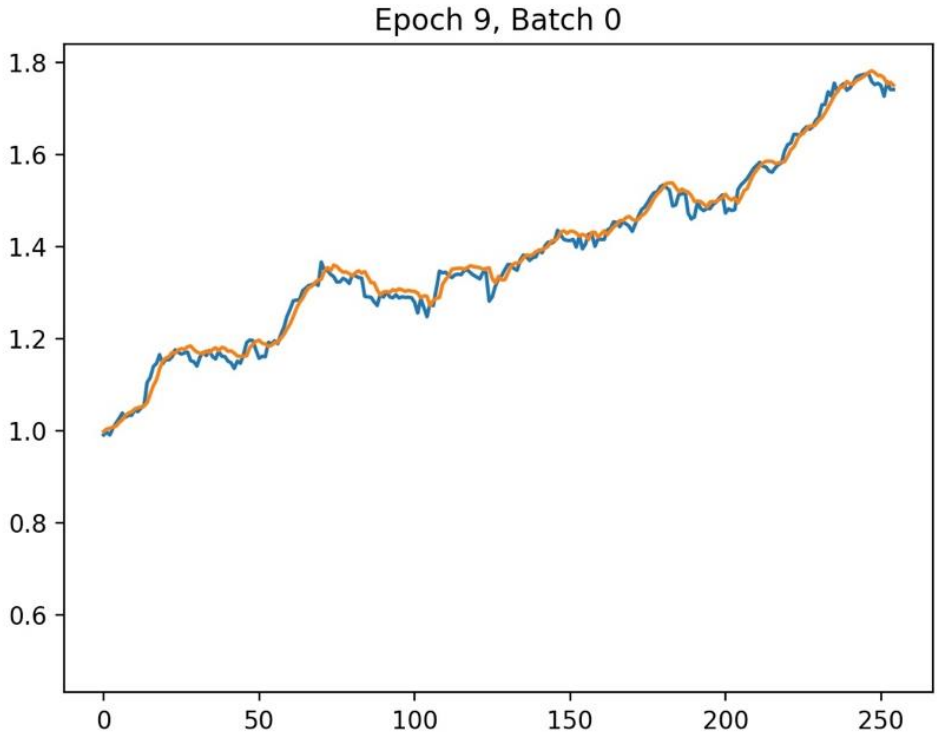



**Figure 2**. Detailed view of an ANN neuron

- Each layer is connected to the next layer neurons and there exist no interconnectivity among the neurons of same layer. The intensity of the passing signal is modified with the multiplication of the connection weights while transferring through the network and a network output is obtained from the neurons of output layer.
- Basically, neurons can be considered as the basic processing units of a neural network
- For our model, we used the tf.train.AdamOptimizer method to control our learning rate. It uses moving averages of the parameters which allow it to have a larger step size.
- We used the tf.nn.relu ('Relu") activation function for all the neurons.

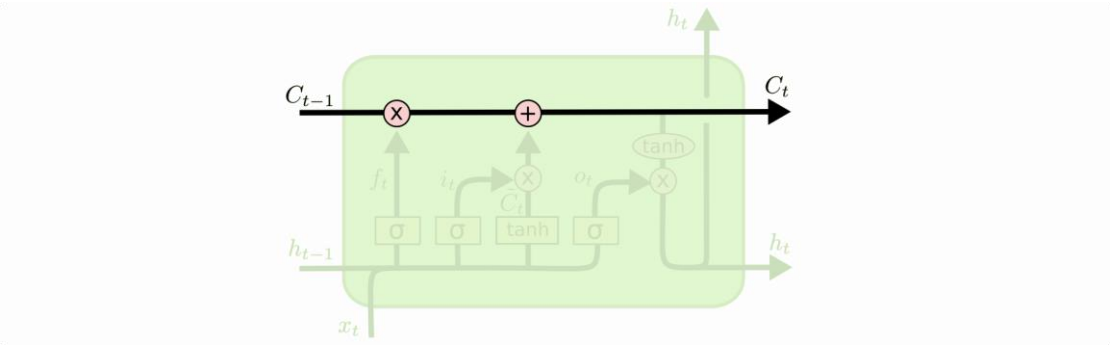
**Validation and Results**




- Root Mean Squared Error = 615, Profit = $ 150,000
- Model computation time is slower than Linear regression and faster than LSTM
- Predicted values follow actual values accurately rather than lag behind
- Model loses accuracy towards the end of the testing period

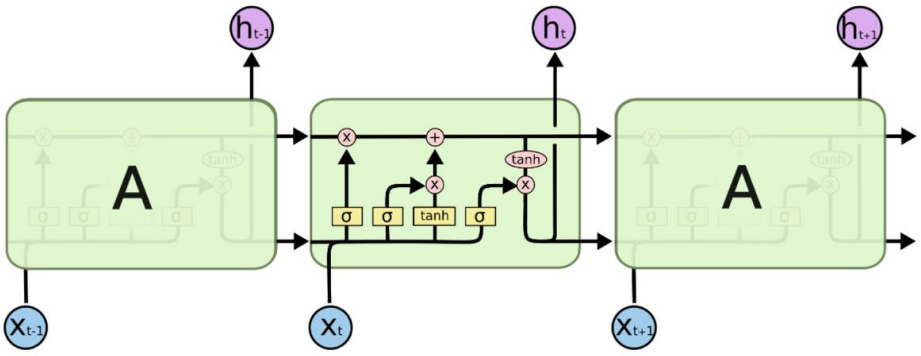
### 5.3 Long Short-Term Memory Network

- LSTMs are a special kind of RNN designed for remembering information for long periods of time. *Gated cells* are used to maintain and modify memory.
- They are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior.



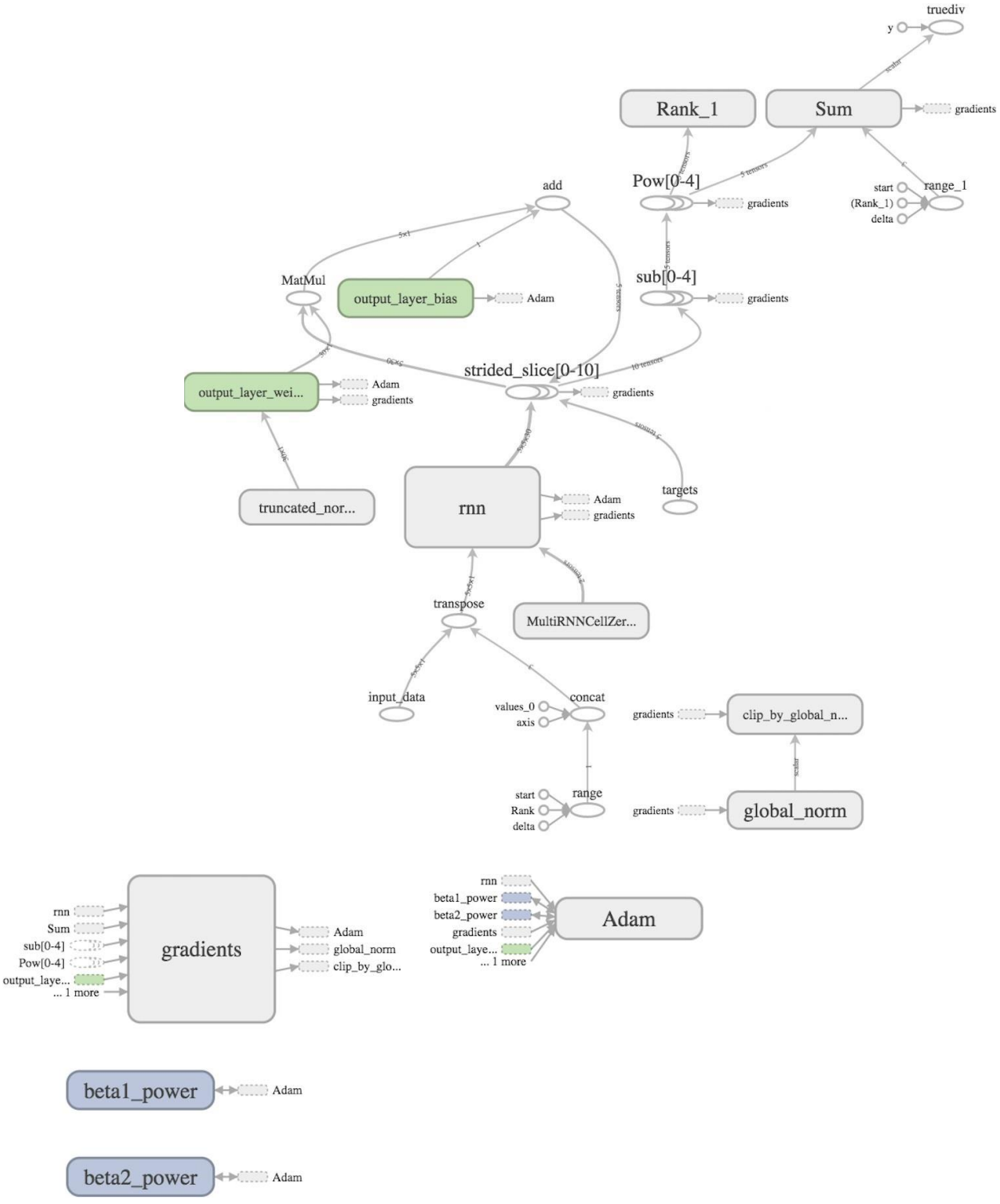

The key to LSTMs is the cell state, the horizontal line running through the top of the diagram above. It's kind of like a conveyor belt that runs straight down the entire chain, with only some minor linear interactions.

- It's very easy for information to just flow along it unchanged.
- The LSTM does have the ability to remove or add information to the cell state, carefully regulated through G*ates*.

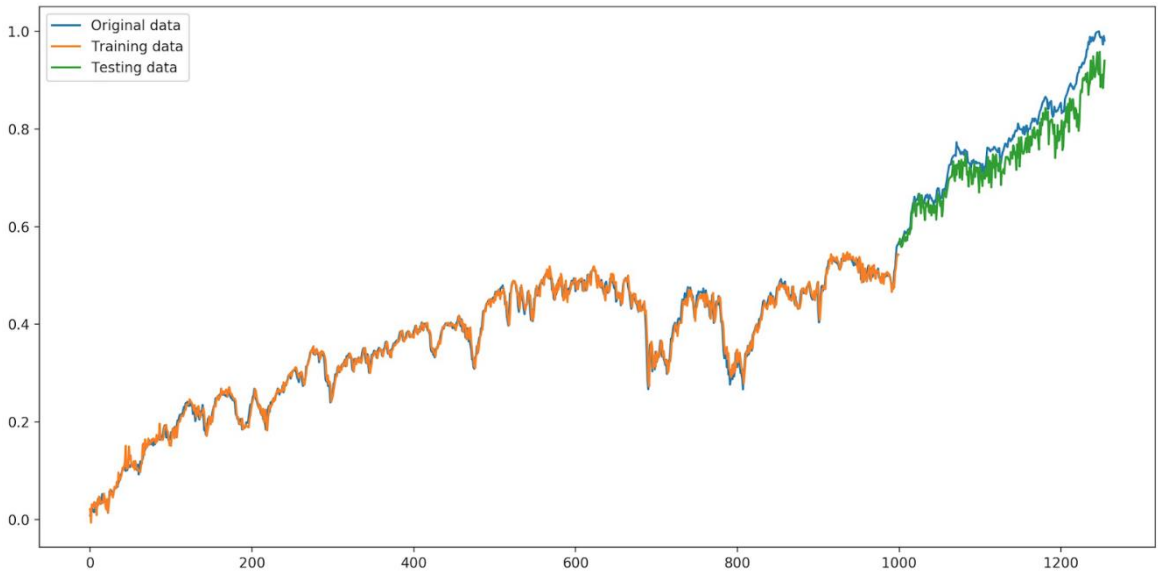



- For our model, we use a single layer RNN with 32 LSTM cells.
- We employed a high dropout rate to ensure our network does not overfit.
- The tf.learn.AdamOptimizer was used for the LSTM model as well to compute and minimize loss while dynamically varying the learning rate.

## Architectural Graph Diagram



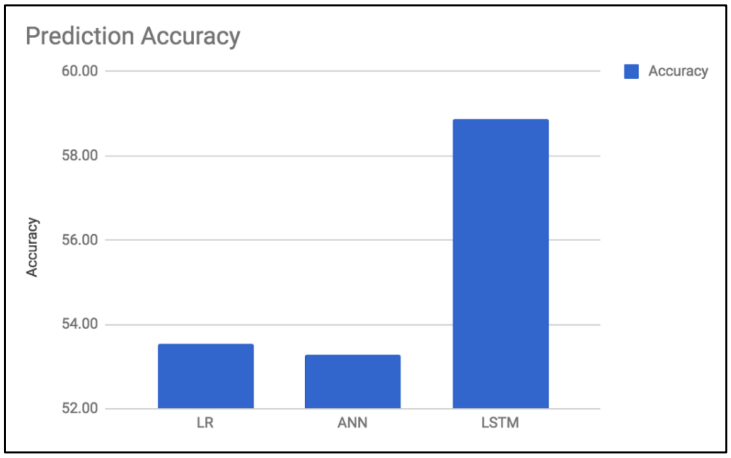
## Validation and Results



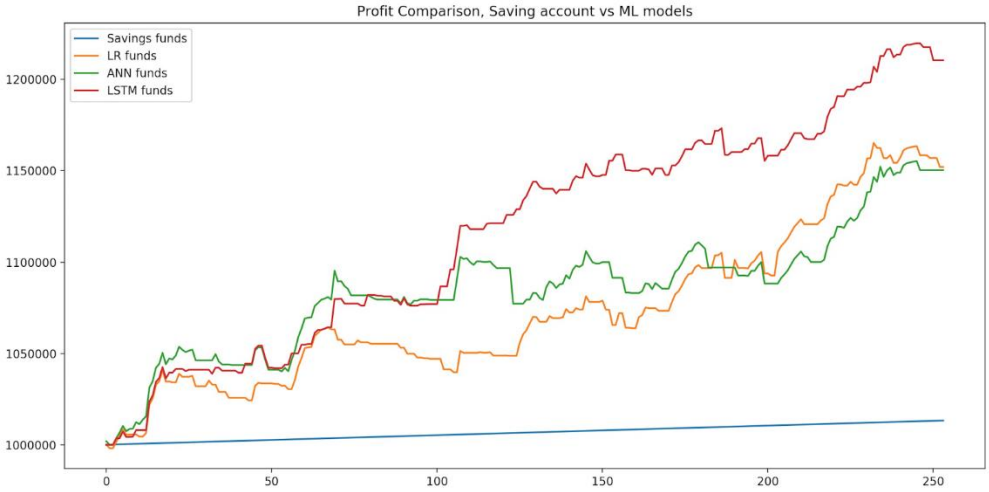

- Root Mean Squared Error = 800, Profit = $ 210,000, Slowest Model to compute
- Predicted values are accurate for the first 30%, model slowly loses accuracy after that
- Need to retrain model often to maintain accuracy
- Longer memory is a negative as data older than a few weeks is not relevant

# 6. Results Comparison

## 6.1 Prediction and fund growth comparison



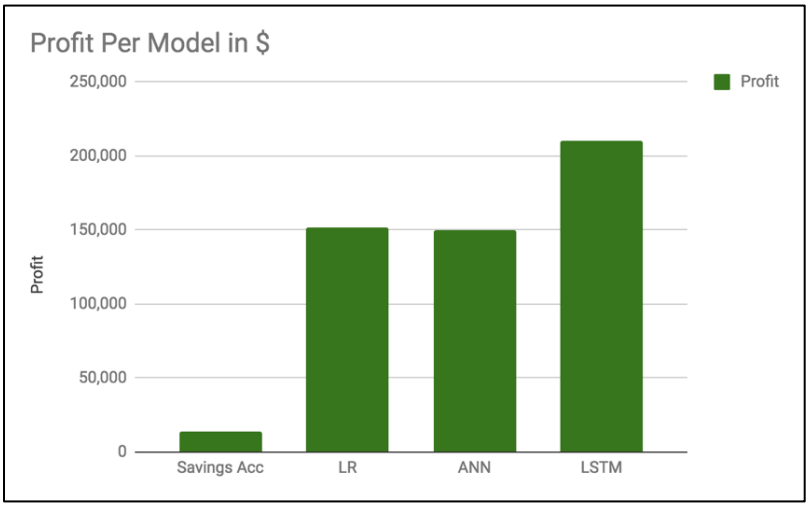

As expected Long short-term memory networks have the highest prediction accuracy at 58% i.e. they can predict whether the market will go up or down correctly 58% of the time. Such a system would make acceptable profits if used in the market as the profit calculation below shows.




**Fund growth comparison per model**

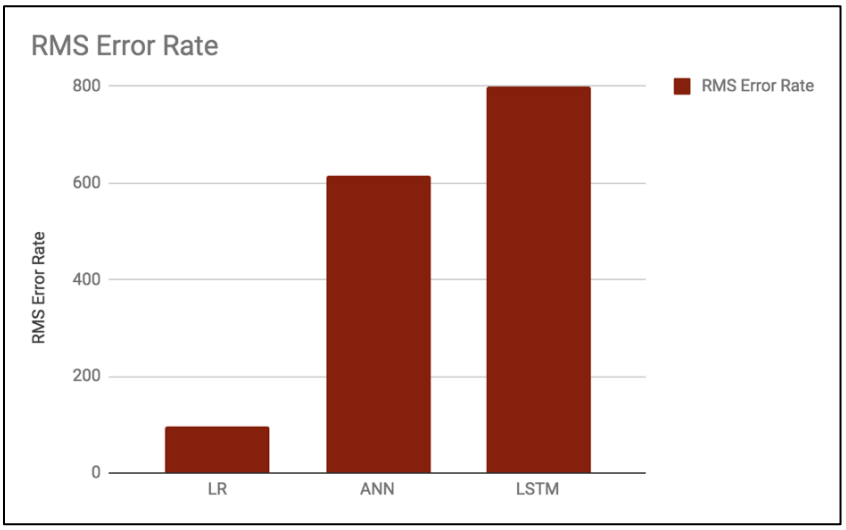
We can see that ANN has the advantage until the middle of the testing set where the DJI experienced unexpected growth. Our LSTM model was able to keep up with that and raise profits accurately. Our savings account in comparison grows at a steady rate of 1.3% per annum.

### 6.2 Profit Comparison




The above chart shows the amount of money we would have made if we had invested 1 Million dollars using our machine learning strategies and compares it to an ordinary saving account. Our savings account would have made a measly $13,000 compared to our LSTM model at $210,000. A portion of this credit belongs to the growth the market is currently experiencing but this shows that our model works nonetheless.

### 6.3 Error Comparison



**Linear Regression has the lowest error rate when comparing the profit**

Here we present a comparison of the error produced when our models tried to predict the absolute value instead of market trends. Linear regression being a local average provides the most accurate values. LSTM's memory proves to be a negative here as they remember the average stock prices over the training period and cannot adjust to sudden growths. However absolute values cannot help us make buy/sell decisions, we would need to know market trends up/down to do that. That is where LSTMs shine.

**Chart comparing predictions, total profits and error values for absolute accuracy**

| Feature | Savings Acc. | LR | ANN | LSTM |
|---|---|---|---|---|
| Prediction Accuracy | NA | 53.54% | 53.29% | 58.87% |
| Profit for $1,000,000 fund | $13,500 | $152,000 | $ 150,000 | $ 210,000 |
| RMS Error Rate | NA | 97 | 615 | 800 |

## 7. Discussion

- Stock Prices can be predicted to some extent using Machine Learning.
- A simple approach like Linear Regression is useful for validation.
- Deep Learning Models can predict price trends accurately compared to actual stock value.
- Accuracy loss in later half of testing set is due to testing data being far away date wise from training data.
- 90% of time is spent in data preprocessing and tuning hyperparameters.
- Past week data has more impact on prediction compared to months of older data.
- Models must be periodically re-trained with updated data to maintain accuracy

## 8. Other Paths

- Logistic Regression could have been used as a baseline approach had we needed a classifier instead of a regressor.
- The sigmoid activation function could have been used for outputting the probability of stock trends rather than absolute values
- The ANN we choose was a simple two hidden layer model. We could have instead opted for more layers if we had the computing capacity.
- We trained the ANN using ReLU as the activation function. We could have instead used sigmoid, tanh or maxout.
- All our models used Adam Optimizer we could have instead gone for a simpler Gradient Descent Optimizer as we had less data
- Recent stock data was much more useful compared to older data. Simple RNN could be used instead of LSTM
- Individual stocks could have been easier to predict instead of an Index which depends on its constituent stocks
- Data was trained only once and tested on one-year worth of data. As recent stock is more relevant we should have trained the model repeatedly and then asked it to predict just the next week’s data. This would have resulted in a much higher accuracy.

## 9. Future Scope

- The features we used in training were just the last 5 days closing prices. Because they were not averaged/ smoothened out a lot of noise would have crept in. We could have taken a moving 3-5-day average which would have gotten rid of a lot of noise and help prevent overfitting.
- Our dataset consisted of per day index data only we would have missed out on variance on intraday trading, volatility in individual stocks which make up the index and several other features. More time could be spent on feature design and extraction
- The long-term memory of the LSTM proves to be a short-coming as it remembers too much. This is a detriment when there is sudden growth or decline periods. We can use an RNN here
- For training, we used a MacBook with a quad core processor and an Intel graphics card. Currently state of the art Machine Learning models is trained on NVidia GPU clusters. Using these GPUs and CUDA api would have allowed us to train on a much bigger dataset and allowed the model to figure out more patterns.