`*Recurrent Neural Networks for Time Series Prediction*

Luke Robbins   
*School of Natural Sciences and Mathematics*  
*University of Texas at*   
*Dallas*  
Dallas, Texas  
LVR19000  
  
Mia Quinones  
*School of Natural Sciences and Mathematics*  
*University of Texas at*   
*Dallas*  
Dallas, Texas  
MNQ180000  
  
Gabriel Medina  
*School of Engineering and Computer Science*  
*University of Texas at*   
*Dallas*  
Dallas, Texas  
GXM170003  
  
Lan Nguyen  
*School of Engineering and Computer Science*  
*University of Texas at*   
*Dallas*  
Dallas, Texas  
LHN170000

*Abstract*—Stock prediction is a difficult task as it is affected by a myriad of random outside influences. Pattern analysis and sequence prediction are the most productive solutions to the problem and can help with the prediction of stock prices by utilizing a Recurrent Neural Network (RNN). The idea behind using an RNN is to recursively use data to build a prediction based on the patterns found in previous data, which has been useful in the areas of handwriting recognition, time series prediction, image caption generation, etc. In other words, RNN has nodes which pass values through input layers and produces an output layer that is iterated back into a new input layer until the machine has built up enough data to make predictions. Exploring the use of RNNs, we have presented a conceptual study of the algorithm, and implemented these theoretical ideas in a fully functional model. We chose to predict prices for one specific company with the stock ticker ASIANPAINT. Our team constructed an RNN using the ReLU activation function, a 7-epoch strategy, a learning rate of .001, and 2 hidden layers to predict stock prices of the company Asian Paint Ltd. This model performed with a training RMSE of .054 and a test RMSE of 0.0998 scaled to values between 0 and 1. The model predicted closing prices with a training accuracy of 28.05 percent and a test accuracy of 1.7 percent. These findings ultimately show that the RNN should be reformatted with a symbiotic use of CNNs or other helpful prediction strategies to predict closing prices more confidently.

Keywords—RNN, Stock Price, Prediction, Recurrent Neural Net,

# Introduction

Stock price prediction is a field that people have attempted to master for many years. It can be incredibly difficult to predict stock prices because they are influenced by an unknown number of random variables in the world and can change significantly in small amounts of time. Even still, pattern analysis and sequence prediction can be very beneficial for a high degree of accuracy in predicting upcoming stock information. The sequence analysis does not look at extraneous variables, but rather past stock price data collected from the market over a certain amount of time. To analyze this data and make an accurate prediction, a system must be used that can use current data and storing the effects of past patterns and outcomes. One such system is called a recurrent neural network (RNN). An RNN uses neural nodes to pass information recursively and track node hidden states to build a more complete picture of past data and present data combined. This can be extremely helpful for doing time series analysis with respect to sequences. In this paper we explore the benefits of using an RNN model to predict the closing price of a stock efficiently and accurately for the next day based on past data and the sequences contained. We also want to find the accuracy limits of using an RNN alone without other strategies to enhance the model’s performance.

We use a 6-month-old dataset named NIFTY-50 Stock Market Data (2000-2001) [1]. The dataset gives the price history and trading volumes of all 50 stock from the NIFTY index with the following features: *Date, Symbol, Series, Prev Close, Open, High, Low, Last, Close, VWAP, Volume, Turnover, Trades, Deliverable Volume, %Deliverable*. To cater analysis to a specific stock price, we chose to break this dataset down by stock ticker and perform individual stock price prediction using the RNN model. The individual stock data is split into data frames ranging from 3000 to 5000 observations per company.

# Background work

RNN has been used for many machine learning problems concerning pattern recognition and prediction. Some major examples are speech recognition, image description generation, face detection, time-series prediction, and handwriting recognition [2]. These problems are centered around the general idea of recursively using data to build a better prediction, thus showing that RNNs are a useful tool.

1. *CNN-RNN for Handwriting Recognition*

Due to the ability of RNNs to process temporal sequences [3] they have been used frequently in the field of handwriting recognition. Kartik Dutta and Praveen Krishnan show in [4] that improvements can be made to a hybrid convolutional neural network (CNN) and RNN. They find that the CNN can be used for feature extraction to create a more accurate prediction, which then gives those features to a Long Short-term Memory (LSTM) model which is a type of RNN strategy. These authors use synthetic data for pre-training and normalization schemes to improve upon the standard hybrid handwriting recognition process. Normalization focuses on correcting natural slants found in the handwriting to make the letters more recognizable and transforming specific domains to account for distortion and important invariances in the data. This is important to our model because feature selection for temporal sequences could potentially lead to a higher degree of accuracy in stock price prediction. It is also of some importance to consider the impact of combining neural network methodologies to improve a certain part of the learning process. In this case, CNN could help to signal which of the 10 available predictors to use in our dataset.

1. *RNN for Time Series Prediction*

Because RNNs can maintain a “short term memory”, the model has a unique ability to bridge any temporal gap that some data might have to create relationships and provide a more comprehensive prediction [2]. An example of applying an RNN model to time series data could be using SARS-CoV-2 data to determine the rate of mutation in the RNA virus over time to determine the risk of the mutated disease [5]. Pathan, et. al. show that an LSTM RNN model can be useful in detecting the mutation rate for Covid-19 from data consisting of the genomic sequence from a patient and comparing these to other countries at different times. In this example, they find that the RNN model has a Root Mean Squared Error (RMSE) of 0.06 in testing and 0.04 in training [5]. This leads to a conclusion of a 0.1% mutation rate for mutating nucleotides of a certain sequence and a 0.1% decrement of others. We draw on this knowledge for our model application to stock prices because, much like the sequential nature of a genomic sequence across temporal space, stock prices can have a temporal sequence dependent on seasons, holidays, etc. This means that the same implementation of an LSTM RNN on genomic sequence may be useful for our model of stock price prediction.

1. *RNN for Image Caption Generation*

RNN models are distinctly able to create useful relationships between assets and apply recurrent information to the relationship easily. This is the case in image captioning, as a bi-directional relationship is needed between the description and the image being captioned. Chen and Zitnick use this property to generate a visual representation of images as a scene is being read [6]. Using only RNNs, the team was able to provide captions to images that human users preferred about 21% of the time. The RNN used chooses a contextual approach and passes the visual features to the sentence relationship, and then back to visual features to create a complete picture. This can be useful to understand for our stock price model considering there is a relationship between the closing price of previous days and their opening price, daily high, etc. Being able to build a contextual model may be helpful in determining the correct closing price for the next day.

# Rnn conceptual study

Recurrent Neural Networks are very useful for problems that require a model to “remember” previous data. Simply put, RNNs store relations between variables and their influence on outcomes to continually influence the final outcome or prediction. This is done by using a neural network combined with loops that iterate through the nodes in the network while storing the result of the previous iteration as context. To build our stock prediction RNN, we explored the math behind the model and implementation.

Firstly, it is necessary to state how a neural network works. A traditional neural network is given input to an input node and passes the input value through to a layer of nodes called the hidden layer. This hidden layer performs some transformation of the input value with a specific “activation” function and passes the new value to the next node in the line. Once the value makes it to the output node, the predicted value is compared against the correct value and error is measured. This gives the model insight on how to “weight” the value as it passes from node to node to create a bias that helps predict the correct outcome [7]. The weight is updated by the following equation:

|  |  |
| --- | --- |
|  | (1) |

where is the old weight, is the learning rate, and Error is the derivative of the error function with respect to weight.

Ultimately, this leads to a model that has static weights used for predicting a value based off some input. In the case of time series data, this can be improved further by understanding that some data has happened prior to a current “time-step”, meaning that we can use the previous calculation result to help predict the next step in a pattern. This is where RNNs help solve the problem of “remembering” information about our predictor’s relationships.

An RNN can be thought of as a node that takes an input, sends it to the hidden layer node, and generates an output, but then also feeds both this output and the next data point in the series into the input node that is trained on the last data that was passed through. We show this network in Figure 1.

The compressed representation (top) is shown, as well as the unfolded network (bottom). Here we can see the generalized d-dimensional feature vector (x) being given to the input recurrent layer along with the vector or hidden units (h) at some time t. This shows how each subsequent recurrent layer is fully contingent on the previous values influencing the hidden unit vector. represents the bias for each node. This representation is for 3 recurrent steps before passing the final output to the feedforward layer that gives our final prediction. It is worth noting that this is a one-to-many relationship; a vector of x values can be used to predict one output but can also be extrapolated to predict multiple outputs.

At each node, an activation function is still used to cater input to the next node, like an Artificial Neural Network (ANN). It is important to understand what an activation function does in the neural network to choose a beneficial option for our RNN. Firstly, an activation function helps to keep the magnitude of the values being passed restricted to a certain limit so that the bias and weights in the network are reasonable and computationally efficient [9]. Second, patterns in data are rarely linear in nature, and thus using values given from the weight vector multiplied by the d-dimensional feature vector added to some bias (as a linear classifier) will likely not be able to adequately follow the pattern in the data. To give a higher degree of complexity to the model a non-linear function is used to transform the data and help to allow a closer interpretation of the underlying pattern. Our model uses the Rectified Linear Unit (ReLU) function:

|  |  |
| --- | --- |
|  | (2) |

We use the ReLU function because it does not suffer from the Vanishing Gradient problem [10] and is not computationally expensive. The drawback to the ReLU function is that all negative values are assigned 0, which allows some nodes to “die” and thus won't contribute to the network. In our model, we make sure to check that values given after the activation function are within a reasonable ceiling as positive infinity is the function limit and can create very computationally expensive iterations.

All these steps are taken in the forward pass of an RNN and provide the first node with a scaled and contextualized output. After the network reassigns weights similarly to an ANN, the RNN iterates through another pass in the network. As we can see in figure one, we hold on to the “hidden state” of the previous pass, which influences the model again. This can be described with the following equations.

|  |  |
| --- | --- |
|  | (3) |
|  | (4) |

In the case of our model, f(x) in (3) and (4) represents the ReLU function. (3) describes the process of getting “+1” hidden state with respect to a certain time-step: passing the feature vector, previous hidden state, weights for both respectively, and the bias of the hidden state to the ReLU function provides an updated hidden state for a specific node. (4) describes the final forward pass of the RNN to five the predicted output after some number k steps [8]. At the end of the iterative process, the RNN has predicted an outcome or class that can be interpreted by the user.

# Process and Results

## Data Preproccesing

To fit a model more clearly for stock prediction, we chose to focus on a specific stock ticker (ASIANPAINT), or the representative name of the company Asian Paints Ltd. on the stock market and apply the RNN to this data. To have clean and descriptive data, we used a few techniques to create a catered dataset to our analysis. As a first step, we looked at the number of missing values for each predictor given and found that *Trades* had more than 50% of the values missing, and *Deliverable Volume* and *%Deliverable* had 509 missing values. Because such a large amount of *Trade* data was

Table

Description automatically generated

Figure 2: Predictor Description

missing and the range between the minimum and maximum values was so large (figure 2), we chose to drop this variable. A Likely reason for the high range in the variables is the time frame for the data set of over 20 years. For *Deliverable Volume* and *%Deliverable,* the range in values was such that it made sense to use a mean column value to fill the empty spots. The predictors *Series* and *Symbol* were dropped from the data frame as they held no significance to our analysis. *Series* describes the section of the market the stock is held in and is of no importance unless comparing to stocks in a different series since it is the same for all values in our analysis. *Symbol* is the same for all values as well since we are using data from only one company’s stock price. All other variables were deemed important to the model.

To help with the time-series analysis, the *Date* variable was converted to a datetime from an Object, and we assigned the index of each row to the respective date. This allowed all variables to be casted to a type of Float32 for easier use in the model, while still retaining the time-series nature of predicated analysis. We also normalized the dataset for all values to be between 0 and 1 to give each predictor an equalized weight. Lastly, we added a column *predc* which consisted of an upward shifted index from the *Close* column. This provided an easy way to have a prediction value for the next day’s closing stock price, the goal of our model.

## Performing Exploratory Analysis of The Dataset

To better understand the scope of the data we used exploratory analysis techniques to show relationships and important information about patterns in the data. Although other analysis was done, the most important and interesting for the RNN model was in respect to the closing price as this was the prediction goal of the model. Firstly, looking at the pattern of closing price over time, we can see a large drop in price around 2013.



Figure 3.1: Closing Price Over Time

We should note this drop-off to make sure the RNN is correctly predicting values during this period of the data. Similarly, we want to look at the average closing price on a yearly and weekly basis.

|  |  |
| --- | --- |
| Chart, histogram  Description automatically generated | Chart, bar chart  Description automatically generated |
| Figure 3.2 | Figure 3.3 |

The yearly average close price follows the same general pattern as the daily close prices, and we can see that all weekdays have similar closes as well. This gives confidence in the idea that no significant outliers are dependent on year or weekday. It is important to note that markets are closed on Saturday and Sunday except for special occasions, so these observations were removed from the dataset as outliers. Lastly, we plotted the rolling 30-day average to give a sense of prediction accuracy only using the mean.

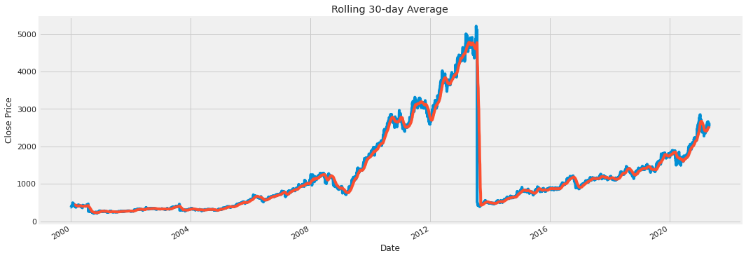


Figure 3.4

## Methodology

After researching Neural Network activation functions, our team decided to use the ReLU function discussed in our conceptual study as the node activation function and the final activation function for the last pass. We wrote the model to take a certain shape of data (number of records x length of sequence x types of sequences) and used the 3-dimensional input to give a 2-dimensional output. We also used an 80/20 split of the data, which results in 4225 observations in the train data and 1056 in the testing data. We also decided to use Root Mean Squared Error (RMSE) as our cost metric, which was implemented by a loss function that calculated loss on each epoch of the model calculations. We used NumPy for the storage of weight values, layer values, and some calculations to hold information correctly.

Finding the parameters with the lowest test error was the focus of the model implementation. We decided to try different values for the epoch, learning rate, and hidden layer parameters, and found results for the following parameters: 5 and 7 epochs, .0001, .001, and .1 learning rates, and 1 and 2 hidden layers. These were selected based on frequency of use in the field as well as some light testing while developing the algorithm. All models assume the same sequence length and output dimension.

## Results

Overall, the results of the RNN on this data were unimpressive. Appling a rolling 30-day mean seemed to have a much higher accuracy then the RNN. With this said, the best model, shown in figure 4.2, had the following parameters: 7 epochs, learning rate of 0.001, and 2 hidden layers.

Chart, line chart

Description automatically generated

Figure 4.1: Training Predictions

Chart, histogram

Description automatically generated

Figure 4.2: Test Predictions

In the above models the blue line represents the actual close price, and the green represents the predicted close price. We can easily see that at some points of the training data the predictions are very far off; this is mainly at the end of the data. In the test data we can also see that the predictions are generally lower than the actual closing price and seem to be very unstable. This model returned a train RMSE of .0536 and a test RMSE of .0998. These error rates are on the scale of 0 to 1, so they are only mediocre. Also, the training accuracy was 28.05% while the testing accuracy was 1.7%. The accuracy being so low is somewhat expected since there are quite a few variables being input, but this coupled with the RMSE seems to show that the model is not performing extremely well.

# Future work

Moving forward with this project proposes some identifiable objectives. Firstly, we can clearly see that the performance of the RNN could potentially be increased by using a hybrid model instead of only a RNN. Combining the model with Convolutional Neural Network (CNN) or another more symbiotic model could largely improve the results. As stated in the conceptual study, these technologies can work together extremely well and would be beneficial for time series data like stock data. Second, our team would like to pursue the application of the model to a wider variety of data. Unfortunately, stock data tracking can warrant many different types of cleaning and processing to use the same model on general data. Our team can pursue the automation of such a task so that a wider variety of data can be used efficiently and easily. Third, there are many theoretical modifications that could be made to our model that may improve the prediction accuracy. Some such improvements would be using a 30-day rolling scope to train the model or using a different activation function that fits more closely to the data. Lastly, we would like to compare the results of our RNN to the results of a 3rd party RNN to see if the results are comparable. If they are not matching, it stands to reason that investigating the results would be worth our time. If the results match closely, we would be confident to start pursuing the addition of other ideas to the model, such as a CNN.

##### References

1. National Stock Exchange India, “NIFTY-50 Stock Market Data (2000-2021) March, 2021 Distributed by Kaggle. https://www.kaggle.com  
   /rohanrao/nifty50-stock-market-data
2. Mishra, Devdarshan. “Applications of Recurrent Neural Networks (Rnns).” *OpenGenus IQ: Computing Expertise & Legacy*, OpenGenus IQ: Computing Expertise & Legacy, 4 Dec. 2019, https://iq.opengenus.org/applications-of-rnn/.
3. Koutnik, J., Greff, K., Gomez, F. &amp; Schmidhuber, J.. (2014). A Clockwork RNN. *Proceedings of the 31st International Conference on Machine Learning*, in *Proceedings of Machine Learning Research* 32(2):1863-1871 Available from https://proceedings.mlr.press/v32/koutnik14.html.
4. K. Dutta, P. Krishnan, M. Mathew and C. V. Jawahar, "Improving CNN-RNN Hybrid Networks for Handwriting Recognition," 2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR), 2018, pp. 80-85, doi: 10.1109/ICFHR-2018.2018.00023.
5. Pathan, Refat Khan, et al. “Time Series Prediction of COVID-19 by Mutation Rate Analysis Using Recurrent Neural Network-Based LSTM Model.” *Chaos, Solitons & Fractals*, vol. 138, 2020, p. 110018., https://doi.org/10.1016/j.chaos.2020.110018.
6. C. Xinlei, L. Zitnick, “Mind's Eye: A Recurrent Visual Representation for Image Caption Generation” Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp.2422-2431
7. “Neural Network Foundations, Explained: Updating Weights with Gradient Descent & Backpropagation.” *KDnuggets*, https://www.kdnuggets.com/2017/10/neural-network-foundations-explained-gradient-descent.html.
8. Saeed, Mehreen. “An Introduction to Recurrent Neural Networks and the Math That Powers Them.” *Machine Learning Mastery*, 22 Sept. 2021, https://machinelearningmastery.com  
   /an-introduction-to-recurrent-neural-networks-and-the-math-that-powers-them/.
9. Jain, Vandit. “Everything You Need to Know about ‘Activation Functions’ in Deep Learning Models.” Medium, Towards Data Science, 30 Dec. 2019, https://towardsdatascience.com/everything-you-need-to-know-about-activation-functions-in-deep-learning-models-84ba9f82c253.
10. Brownlee, Jason. “How to Fix the Vanishing Gradients Problem Using the Relu.” Machine Learning Mastery, 25 Aug. 2020, https://machinelearningmastery.com/how-to-fix-vanishing-gradients-using-the-rectified-linear-activation-function/.

Tabulated Results from Model Iteration

|  | **Epochs** | **Learning Rate** | **Hidden Layers** | **Train RMSE** | **Test RMSE** | **Training Accuracy** | **Test Accuracy** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 5 | 0.0001 | 1 | 0.302888 | 0.278767 | 4.36 | 0.00 |
| **1** | 5 | 0.0001 | 2 | 0.256331 | 0.219285 | 8.57 | 0.00 |
| **2** | 5 | 0.0010 | 1 | 0.301602 | 0.251889 | 8.07 | 0.09 |
| **3** | 5 | 0.0010 | 2 | 0.302770 | 0.278767 | 7.86 | 0.00 |
| **4** | 5 | 0.0100 | 1 | 0.302888 | 0.278767 | 4.36 | 0.00 |
| **5** | 5 | 0.0100 | 2 | 0.182671 | 0.371380 | 8.69 | 3.50 |
| **6** | 7 | 0.0001 | 1 | 0.302888 | 0.278767 | 4.36 | 0.00 |
| **7** | 7 | 0.0001 | 2 | 0.268346 | 0.237430 | 4.45 | 0.00 |
| **8** | 7 | 0.0010 | 1 | 0.374372 | 0.359931 | 0.88 | 0.00 |
| **9** | 7 | 0.0010 | 2 | 0.053605 | 0.099761 | 28.05 | 1.70 |
| **10** | 7 | 0.0100 | 1 | 0.482246 | 0.738288 | 2.23 | 0.95 |
| **11** | 7 | 0.0100 | 2 | 2.077172 | 2.303813 | 0.00 | 0.00 |
| **12** | 15 | 0.0001 | 1 | 0.275079 | 0.239893 | 8.05 | 0.00 |
| **13** | 15 | 0.0001 | 2 | 0.302888 | 0.278767 | 4.36 | 0.00 |
| **14** | 15 | 0.0010 | 1 | 0.302752 | 0.277595 | 4.71 | 0.00 |
| **15** | 15 | 0.0010 | 2 | 0.302886 | 0.278767 | 4.36 | 0.00 |
| **16** | 15 | 0.0100 | 1 | 0.302888 | 0.278767 | 4.36 | 0.00 |
| **17** | 15 | 0.0100 | 2 | 0.302888 | 0.278767 | 4.36 | 0.00 |