Stock price prediction of Apple stock using Recurrent Neural Network

Chan Ka Hing
The Univesity of Adelaide
kahing.chan@student.adelaide.edu.au

Abstract

The Recurrent Neural Network (RNN) is one of the most popular deep neural networks in stock price prediction. It uses hidden states to recognize the data's sequential characteristics making it a good choice for processing sequential data. In this paper, we build different types of Recurrent Neural Networks and perform stock price prediction on Apple stock.

1. Introduction

Stock market prediction is the act of determining the future value of the stock. Inventors attempt to predict the future stock price to make sound investment decisions. One of the successful stock market prediction methods is Recurrent Neural Networks (RNN) in deep learning. The concept of Recurrent Neural Networks was brought up by David Rumelhart in 1986 [3]. The standard RNN is called Vanilla RNN. There are different variations of RNN models invented later on. In 1997, Long Short-Term Memory (LSTM) network was invented by Hochreiter and Schmidhuber [4]. In 2014, Kyunghyun et al. invented Gated Recurrent Unit (GRU) network [5]. The LSTM and GRU networks are the improved versions of the RNN model. RNN is a well-known neural network for sequential data. Apart from stock market prediction, RNN can also be applied to many different sequential modelling problems such as machine translation, voice recognition, text classification, DNA sequencing analysis and so on. In comparing RNN with other neural networks such as Convolutional Neural Network (CNN) in processing sequential data, RNN has much better performance. RNN can handle length varying input and maintain the memory along the long sequential data, while CNN fails to do so.

This paper uses Vanilla RNN, LSTM, GRU and a statistical model for Apple stock prediction. We demonstrate our models using the Apple stock price in the past five years and evaluate their performance by calculating the Mean Absolute Error (MAE) between the predicted price and the actual price. In the experiment, we compare the performance of these four different models. We also study the prediction performance using different optimizers and learning rates.

Regarding the dataset, the Apple stock price record in the past five years is used as our dataset [6]. It starts from 20th November 2017 to 17th November 2022. We only use the daily close price as our dataset for prediction. The dataset is split into the training set and testing set in a ratio of 80% and 20% respectively. The dataset is normalized into a range of 0 to 1 to avoid distortion by extremely large numbers. In the training process, the training set is divided into mini-batches with a batch size of 20.

2. Methodology

The methodology consists of three parts, Recurrent Neural Network (RNN), Limitations and Improvements and Statistical Model.

2.1. Recurrent Neural Network

2.1.1 RNN Architectures

Recurrent Neural Networks (RNN) are the most suitable neural networks for sequential data. The structure of RNN consists of multiple neural networks which take sequential input along with computed hidden state from the previous step [1]. Figure 1 shows a typical RNN structure. Every sequential input creates a hidden state which is computed from the current input and the past inputs [1]. The hidden state and the next input are used to compute the next output and the next hidden state [1]. The hidden state computed from sequence t and the output of sequence t are calculated as follows [1]:

$$a^{<t>} = tanh(W_a a^{<t-1>} + W_x X^{<t>} + b_a) \quad (1)$$

$$Y^{'<t>} = \sigma(W_y a^{<t-1>} + b_y) \quad (2)$$

$a^{<t>}$ is the hidden state computed from sequence t
$Y^{'<t>}$ is output computed at sequence t

$X^{<t>}$ is input at sequence t
W_x is weight vector for input vector
W_y is weight vector for output vector
W_a is weight vector for activation input
b_a is bias for computing hidden state at sequence t
b_y is bias for computing output at sequence t
$tanh$ is tanh activation function
σ is sigmoid activation function

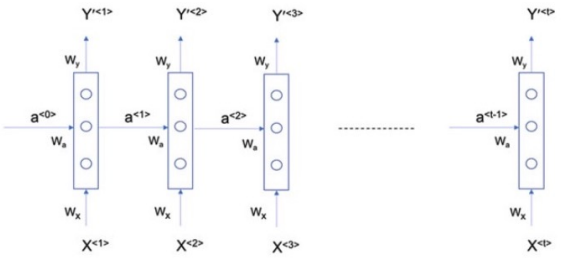

Figure 1. Typical RNN architecture (Rebala, Rav & Churiwala 2019)

At each time step, the hidden state is computed not only based on the current input and corresponding input weights but also based on the past inputs and the corresponding past input weights [1]. The weights W_x, W_y, and W_a are the same for all the time steps. This reduces the number of parameters required by the RNN model. The initial hidden state is initialized as a vector of all zeros.

2.1.2 Backpropagation in RNN

The training of RNN includes computing the weights W_x, W_y, and W_a. The loss function is selected based on the application. The loss is computed at each stage of the network sequence. The weights are updated using gradient descent by backpropagation through each time step. Figure 2 shows an illustration of backpropagation through time. The loss function at time step t with respect to W_y, W_x, and W_a are computed as follows:

$$\frac{\partial L^{<t>}}{\partial W_y} = \frac{\partial L^{<t>}}{\partial Y'^{<t>}} \frac{\partial Y'^{<t>}}{\partial W_y} \tag{3}$$

$$\frac{\partial L^{<t>}}{\partial W_x} = \sum_{i=1}^{t} \frac{\partial L^{<t>}}{\partial Y'^{<t>}} \frac{\partial Y'^{<t>}}{\partial a^{<i>}} \frac{\partial a^{<i>}}{\partial W_x} \tag{4}$$

$$\frac{\partial L^{<t>}}{\partial W_a} = \sum_{i=1}^{t} \frac{\partial L^{<t>}}{\partial Y'^{<t>}} \frac{\partial Y'^{<t>}}{\partial a^{<i>}} \frac{\partial a^{<i>}}{\partial W_a} \tag{5}$$

$L^{<t>}$ is the loss at time step t
$Y'^{<t>}$ is output computed at time step t
$a^{<t>}$ is the hidden state computed from time step t
$a^{<i>}$ is the hidden state computed from time step i
W_y is weight vector for input vector
W_x is weight vector for output vector
W_a is weight vector for hidden state input

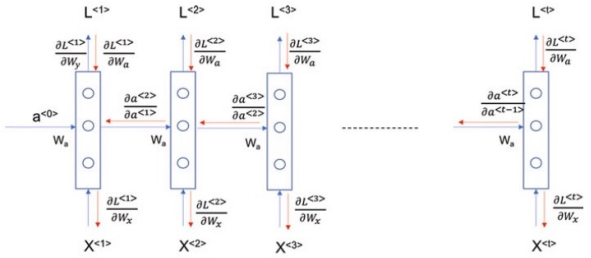

Figure 2. An illustration of backpropagation through time (Rebala, Rav & Churiwala 2019)

Since each of the hidden states $a^{<t>}$ depends on the previous hidden state $a^{<t-1>}$, the gradient of the hidden state flows backwards across the time steps. As a result, the derivatives of all the functions with respect to the weights W_y, W_x, and W_a can be computed.

2.2. Limitations and Improvements

2.2.1 Vanishing and Exploding Gradients

Vanishing and Exploding Gradients are one of the limitations in the RNN. During the backpropagation through time, the gradients are computed t times. The gradient of tanh activation function is in the range of 0 to 1, and the gradients are calculated by the chain rule. This results in multiplying these small numbers t times, causing the vanishing gradient problem. The gradient could reduce exponentially fast and even stop the training process. If under some initialization conditions and activation functions, the initial gradient is large, the gradient could increase exponentially, causing the exploding gradient problem. The vanishing gradient can be addressed by using ReLU activation function instead of tanh or sigmoid activation function and by initializing suitable initial weights. The exploding gradient problem can be addressed by gradient clipping which limits the gradient value at an upper bound. However, a better way to address the vanishing and exploding gradient is using Long Short-Term Memory (LSTM) network and Gated Recurrent Unit (GRU) network, which control the influence of the previous sequence on the current sequence.

2.2.2 Long Short-Term Memory (LSTM) network

LSTM network resolves the problem of vanishing and exploding gradient problem by a memory cell approach and

providing gates that control the influence of the previous sequence on the current sequence [1]. Figure 3 shows an illustration of an LSTM network. The memory state is the long-term memory, and the hidden state is the short-term memory. The forget gate controls the influence of the previous memory state on the current memory state. A candidate memory is computed from the previous hidden state and the current input. The input gate controls the influence of candidate memory on the current memory state. The output gate controls the influence of the current memory state, the previous hidden state, and the current input on the current hidden state. The equation representing the application of LSTM network are as follows:

$$F_t = \sigma(X_t W_{xf} + H_{t-1} W_{hf} + b_f) \tag{6}$$

$$I_t = \sigma(X_t W_{xi} + H_{t-1} W_{hi} + b_i) \tag{7}$$

$$O_t = \sigma(X_t W_{xo} + H_{t-1} W_{ho} + b_o) \tag{8}$$

$$\widetilde{C_t} = tanh(X_t W_{xc} + H_{t-1} W_{hc} + b_c) \tag{9}$$

$$C_t = F_t\, C_{t-1} + I_t\, \widetilde{C_t} \tag{10}$$

$$H_t = O_t\, tanh(C_t) \tag{11}$$

F_t output of the forget gate at sequence t
I_t output of the input gate at sequence t
O_t output of the output gate at sequence t
$\widetilde{C_t}$ candidate memory at sequence t
C_t memory state at sequence t
H_t hidden state at sequence t
X_t is input at sequence t
$tanh$ is tanh activation function
σ is sigmoid activation function
W_{xf}, W_{xi}, W_{xo}, W_{xc} are the weights for input vector at forget gate, input gate, output gate, candidate memory cell respectively
W_{hf}, W_{hi}, W_{ho}, W_{hc} are the weights for hidden state input at forget gate, input gate, output gate, candidate memory cell respectively
b_f, b_i, b_o, b_c are the bias at forget gate, input gate, output gate, candidate memory cell respectively

LSTM networks give a very good long term dependence modelling in many sequential data application. A variation of them is Gated Recurrent Unit (GRU) networks which simplify the network by reducing the number of parameters.

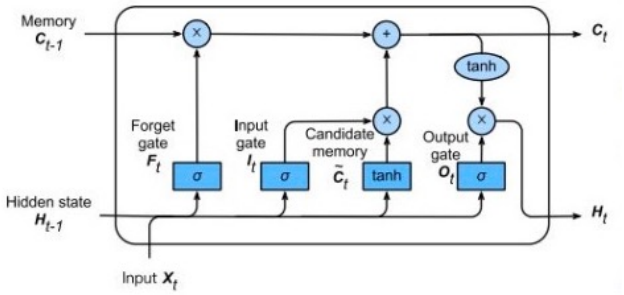

Figure 3. An illustration of a LSTM network (Smola & Li 2019)

2.2.3 Gated Recurrent Unit (GRU) network

The Gated Recurrent Unit (GRU) network is a simplified version of the LSTM network. GRU network controls the influence of the previous sequence on the current sequence by introducing a reset gate and update gate. The reset gate controls the percentage of memory in the previous hidden state to forget. The output is then combined with the current input to compute the candidate hidden state. The update gate controls the percentage influence of the previous hidden state and the candidate hidden state on the current hidden state. The equations representing the application of GRU network are as follows:

$$R_t = \sigma(X_t W_{xr} + H_{t-1} W_{hr} + b_r) \tag{12}$$

$$Z_t = \sigma(X_t W_{xz} + H_{t-1} W_{hz} + b_z) \tag{13}$$

$$\widetilde{H_t} = tanh(X_t W_{xh} + (R_t\, H_{t-1}) W_{hh} + b_h) \tag{14}$$

$$H_t = Z_t\, H_{t-1} + (1 - Z_t)\, \widetilde{H_t} \tag{15}$$

R_t output of the reset gate at sequence t
Z_t output of the update gate at sequence t
$\widetilde{H_t}$ candidate hidden state at sequence t
H_t hidden state at sequence t
X_t is input at sequence t
$tanh$ is tanh activation function
σ is sigmoid activation function
W_{xr}, W_{xz}, W_{xh} are the weights for input vector at reset gate, update gate, candidate hidden state cell respectively
W_{hr}, W_{hz}, W_{hh} are the weights for hidden state at reset gate, update gate, candidate hidden state cell respectively
b_r, b_z, b_h are the bias at reset gate, update gate, candidate hidden state cell respectively

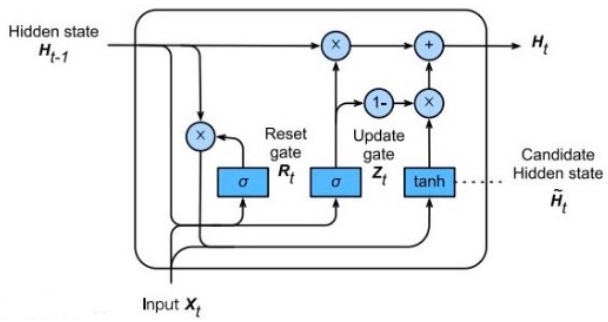

Figure 4. An illustration of a GRU network (Smola & Li 2019)

GRU networks successfully resolve vanishing and exploding gradient problem with less number of parameters compared to LSTM networks.

2.3. Statistical Model

A statistical model is also used to predict the stock price. The result is compared to the RNN, LSTM and GRU models in the experiment part. The statistical model is a Simple Moving Average (SMA) model with 30 days interval. The formula for SMA are as follows:

$$SMA = \frac{1}{k} \sum_{i=s}^{e} p_i \tag{16}$$

k is the number of days in the time interval
e is the last day of the period
s is the start day of the period
p_i is the stock price at the i-th day
SMA is the prediction price of the next day after the time period

The 30 days simple moving average starts with day 1 until day 30 and predicts the price at day 31. The 30 days time interval is then shifted by one day at a time until shifting all the data.

3. Experiments and analysis

The experiments and analysis section consists of two parts, general analysis of Vanilla RNN Model and the experiments.

3.1. General Analysis of Vanilla RNN Model

The Vanilla RNN model has a sequence length of 10. The input is the daily close price of the stock. We use 10 days price to compute the predicted price for the next day. The number of RNN layers is 2. The loss function is the Mean Square Error (MSE) loss with Adam optimizer as our optimization algorithm. We demonstrate our Vanilla RNN model using the Apple stock price in the past 5 years with a learning rate of 1×10^{-3} for 50 iterations. The Mean Absolute Error (MAE) of training and testing data are 4.54 and 8.46 respectively. As shown in Figure 5, the Vanilla RNN model shows a good prediction performance in both training and testing data. Therefore, the Vanilla RNN model successfully predicts the stock price for Apple stock.

3.2. Comparison between Vanilla RNN, LSTM, GRU and Simple Moving Average models

In this experiment, we compare the prediction performance of Vanilla RNN, LSTM, GRU and Simple Moving Average models. The experiment aims to find the best performance model among Vanilla RNN, LSTM and GRU models and compare these machine learning models with the statistical model. Table 1 shows the Mean Absolute Error (MAE) of training and testing data of different prediction models. Figures 5,6,7, and 8 show the actual price and the predicted price of Vanilla RNN, LSTM, GRU and Simple Moving Average model respectively. As shown in figure 5,6,7, the Vanilla RNN, LSTM and GRU models show good prediction results in both training and testing data. The GRU model has the lowest testing data Mean Absolute Error (MAE) of 4.94, as illustrated in table 1. The testing data MAE for the LSTM model is 5.19. The Vanilla RNN model has the highest testing data MAE of 8.46. The LSTM and GRU models have a better prediction performance than the Vanilla RNN model due to different gates design which properly controls the influence of the previous sequence to the current sequence. The GRU model outperforms the LSTM model which might be due to the simpler gate structure design and less number of parameters. The Simple Moving Average model shows a fairly good prediction for training and testing data, as shown in figure 8. The testing data MAE of the Simple Moving Average model is 8.03 which is higher than the GRU and LSTM models, but slightly lower than the Vanilla RNN models. The result shows that GRU and LSTM models outperform the Simple Moving Average model. However, the performance of the Vanilla RNN model is slightly worse than the Simple Moving Average model.

3.3. Comparison between different optimizers

In this experiment, we study three different optimizers in the GRU model as GRU model has the best performance among all the models. The three different optimizers are Adaptive Moment Estimation (Adam), Stochastic Gradient Descent (SGD) with momentum and Root Mean Squared Propagation (RMSprop). The experiment aims to find the best optimizer among these three optimizers which has the lowest mean absolute error (MAE). The SGD with momentum is an extension of SGD which helps accelerate the gradient vectors in the right directions leading to faster convergence. The momentum factor is set as 0.8 in our experi-

Figure 5. (left) The actual price and predicted price of training data of Vanilla RNN model. (right) The actual price and predicted price of testing data of Vanilla RNN model.

Figure 6. (left) The actual price and predicted price of training data of LSTM model. (right) The actual price and predicted price of testing data of LSTM model.

Figure 7. (left) The actual price and predicted price of training data of GRU model. (right) The actual price and predicted price of testing data of GRU model.

Figure 8. (left) The actual price and predicted price of training data of Simple Moving Average model. (right) The actual price and predicted price of testing data of Simple Moving Average model.

Experiment 1 (Different models)		
Model	Mean absolute error (Training data)	Mean absolute error (Testing data)
Vanilla RNN	4.54	8.46
LSTM	5.40	5.19
GRU	3.36	4.94
Simple Moving Average	3.92	8.03

Table 1. The Mean Absolute Error (MAE) of training and testing data of different prediction models with learning rate = 1×10^{-3} for 50 iterations.

Experiment 2 (Different Optimizers)		
Optimizer	Mean absolute error (Training data)	Mean absolute error (Testing data)
Adam	3.36	4.94
SGD with momentum	2.88	6.66
RMSprop	7.63	10.01

Table 2. The mean absolute error of training and testing data of GRU model with different optimizers with learning rate = 1×10^{-3} for 50 iterations.

ment. RMSprop is also an extension of SGD. It penalizes the gradient vectors that cause oscillation by a normalization factor to achieve faster convergence. Adam optimizer combines the advantages of both SGD with momentum and RMSprop. Table 2 shows the mean absolute error (MAE) of training and testing data among these three optimizers. As shown in Table 2, the Adam optimizer has the lowest testing data MAE (4.94). The testing data MAE of SGD with momentum is 6.66. RMSprop has the highest testing data MAE (10.01). The Adam optimizer has the best prediction performance as it combines the advantages of both SGD with momentum and RMSprop. SGD with momentum has a better prediction performance than RMSprop in our experiment.

3.4. Comparison between different learning rates

In this experiment, we study six different learning rates ranging from 5×10^{-3} to 1×10^{-5} on the GRU model as it has the best performance in the previous experiment. The experiment aims to find the best learning rate among these six learning rates which has the lowest mean absolute error (MAE). Table 3 shows the mean absolute error (MAE) of training and testing data among these six learning rates. As shown in Table 3, the MAE of testing data decreases

Experiment 3 (Different learning rates)		
Learning rate	Mean absolute error (Training data)	Mean absolute error (Testing data)
5×10^{-3}	16.43	51.31
1×10^{-3}	3.36	4.94
5×10^{-4}	2.67	4.93
1×10^{-4}	2.27	5.19
5×10^{-5}	2.87	6.80
1×10^{-5}	5.59	15.08

Table 3. The mean absolute error of training and testing data of GRU model with different learning rates in 50 iterations.

to a minimum, then increases again as the learning rate increases. The learning rate of 5×10^{-3} has testing data MAE of 51.31 which means the learning rate is too high causing overshooting of the minima in gradient descent. The learning rate of 1×10^{-5} has testing data MAE of 15.08 which means the learning rate is too low and requires more epochs to converge. The best learning rate is 5×10^{-4} which has the lowest testing data MAE of 4.93.

4. Code

https://github.com/ChanKaHing/RNN.git

5. Conclusion

This paper presents different Recurrent Neural Network models and a statistical model for Apple stock prediction. In the experiment, we compare the performance of the Vanilla RNN, LSTM, GRU and Simple Moving Average models. We also study different optimizers and learning rates on the GRU model. After the experiment and analysis, we have the following findings. The LSTM and GRU models have better performance than the Vanilla RNN model. It might be due to the gates structure in LSTM and GRU which more accurately controls the influence of the past memory on each hidden state and the final output. Among LSTM and GRU, GRU outperforms LSTM in prediction results. The reasons behind the better performance of GRU might be the simpler gates structure and fewer training parameters which results in faster training and better performance in a fixed number of iterations. In comparing the machine learning models with the Statistical model, GRU and LSTM models perform better than the Simple Moving Average model. The result shows that GRU and LSTM successfully perform stock prediction with better accuracy than a simple statistical method. The Vanilla RNN model performs slightly worse than the Simple Moving Average model. In the experiment of different optimizers, we found that the best optimizer to our model is Adam optimizer which combines the advantages of both SGD with momen-

tum and RMSprop. The SGD with momentum performs better than RMSprop. In the experiment of different learning rates, we found that the optimum learning rate is around 5×10^{-4} which provides a good learning speed without causing divergence in accuracy. A higher learning rate may overshoot the gradient minima, while a lower learning rate requires more iterations to converge. Future work may include using transformer for stock price prediction. The transformer uses self attention module for feature extraction of the time series data. It can also include multi-head attention to extract more information about the relationship between the data.

References

[1] Rebala, G, Ravi, A Churiwala, S 2019, *An Introduction to Machine Learning*, 1st ed. 2019., Springer International Publishing, Cham.

[2] Smola, A Li, M 2019 *Introduction to Deep Learning 19: Recurrent Neural Networks*, lecture notes, The University of California, Berkeley, delivered in Spring 2019.

[3] Rumelhart, DE, Hinton, GE Williams, RJ 1986, 'Learning representations by back-propagating errors', Nature (London), vol. 323, no. 6088, pp. 533–536.

[4] Hochreiter, S Schmidhuber, J 1997, 'Long Short-Term Memory', Neural Computation, vol. 9, no. 8, pp. 1735–1780.

[5] Cho, K, Bart van Merrienboer, Bahdanau, D Bengio, Y 2014, 'On the Properties of Neural Machine Translation: Encoder-Decoder Approaches', arXiv.org.

[6] Yahoo Finance, *Apple Inc. (AAPL)*, Yahoo Inc., n.d. <https://finance.yahoo.com/quote/AAPL/history/>