Forecasting Daily Stock Volatility: A Comparison between GARCH and Recurrent Neuro-networks

Weiting Cai[1, †], Kaiyang Liu[2, †] and Keming Lu[3, *, †]

[1]School of Economics and Statistics, Guangzhou University, Guangzhou, China.

[2]School of Finance, Hebei University of Economics and Business, Shijiazhuang, China.

[3]School of Business, Hang Seng University of Hong Kong, Hong Kong, China.

*Corresponding author: s196263@hsu.edu.hk

[†]These authors contributed equally.

Abstract. Forecasting the volatility of financial derivatives and securities returns has always been the core of financial research. Accurate volatility forecast is integral to financial risk management, which is vital for investors and supervision authorities. Traditional time series models (e.g., ARCH and GARCH models) have been famous tools for volatility forecasting. However, those models cannot capture non-linear correlations, and prediction capability is unsatisfactory. In this paper, we use multifactorial deep learning algorithms RNN, LTSM, and GRU with sentiment data to predict the future volatility of Apple (AAPL), then compare the accuracy of prediction with the GARCH model. According to the analysis, the prediction accuracy of LSTM and GRU significantly improved compared with the GARCH model. These results shed light on guiding further exploration of stock volatility prediction using deep learning algorithms. Besides, it advises investors to choose efficient stock price volatility forecasting and risk management tools.

Keywords: GARCH; Forecasting Volatility; Neuro-network.

1. Introduction

Return and risk are an indispensable part of the investment. For the high-risk and high-return stock market, investors urgently need a systematic and comprehensive scientific method to make accurate predictions on the stock market. Nevertheless, predicting financial asset volatility can be challenging due to the low signal-to-noise ratio of financial information. To address the issue, the modeling tools have changed dramatically in past decades.

Traditional procedures on volatility prediction are mainly about building time series regression models. GARCH and related models are the hotspots of research on volatility prediction in the 20th century. Engle firstly proposed the autoregressive conditional heteroscedasticity (ARCH) model to estimate the inflation variance of the United Kingdom [1]. In 1986, Bollerslev introduced the generalized autoregressive conditional heteroscedasticity (GARCH) model, which solved the problem of overwhelming parameters brought by the ARCH model [2]. Sentana modified the traditional GARCH model and designed the Quadratic GARCH (QGARCH) model, in which non-linear relationships between market data and stock volatility are captured even if those non-linear relationships are far from a complex one [3]. Franses found that QGARCH has a better prediction accuracy than GARCH and GJR models [4].

A novel approach to conducting volatility prediction is artificial neural networks (ANN). Donaldson used basic ANN to forecast stock volatility and concluded that ANN has a better performance as ANN can fit complex non-linear relationships that traditional time series models cannot capture [5]. Mantri et al. used ANN to model the Indian market volatility, but the conclusion indicates a trivial difference in the forecast between ANN and other models [6]. Online sentiments have been essential factors for research in recent years. Liu et al. introduced recurrent neural networks (RNN) combining sentiment data from the online forum, and the results showed a significant improvement in prediction performance [7]. Zhou et al. used a deep learning model to forecast the volatility of SSE50 ETF options. A comparison between the deep learning and GARCH models showed that deep learning performs better [8]. The long short-term memory (LSTM) process was

applied to the stock sequence prediction of China A-stock in the paper of Su et al. The paper found that LTSM can accurately learn the pattern of the China A-stock market [9]. Basic RNNs and LSTM models have memories of past data and tend to perform better volatility prediction.

Put it all together, this paper aims to find the most efficient method for predicting stock market volatility by comparing different models. Combining financial knowledge and technical analysis methods, this paper analyzes the effectiveness of various models in forecasting stock volatility. The remainder of the article is organized as follows. Section 2 describes the sample data collection process. Section 3 demonstrates the measurement of various models. In section 4, we compare the prediction results of different models. A brief analysis is also demonstrated in this section. Concluding remarks with suggestions for future research are given in Section 5.

2. Data

The data utilized in this study includes 1511 daily observations of AAPL from May 31, 2016, to May 27, 2022. The former 1258 observations are classified as training set while the other to the testing set. Trading data is obtained from yfinance, an API created by yahoo finance. Market sentiment data and financial data are collected through Google Trend and the SEC database, respectively. Google Trends data is normalized keyword searching volume ranging from 0 to 100, and a higher number means higher searching volume [10]. Google Trend includes data on non-trading days, so we delete those sentiment data to ensure the consistency of the timeline. Our models will predict the next day's volatility using the current day's features. Detailed information on each factor is shown in Table 1.

Table 1. Information on factors

Factor name	Explanation
Volume	Trading volume
Volume_changes	Percentage change of trading volume
Log_returns	Logarithmic returns
Volatility	Annualized daily volatility
Dividends	Dividends on a specific day
Interest	Sentiment index
Interest_changes	Percentage change of sentiment index
PE	Price/earnings per share
Beta	Coefficient of the linear regression between 30 days stock price and NASDAQ composite
Alpha	The intercept of the linear regression between 30 days stock price and NASDAQ composite
MACD	Moving average convergence divergence
RSI	Relative strength index
Bolling_diff	Difference between Bollinger upper band and lower band
KDJ_k	K series of the stochastic oscillator
KDJ_diff	Difference between stochastic oscillator K series and J series
VR	Volume variance, a strength index of volume

3. Methodology

3.1 Volatility Measurement

All the volatility used in the article is annualized daily volatility; following is an explanation of how it is calculated. The method measuring the daily returns is the logarithmic return:

$$r_t = ln\frac{P_t}{P_{t-1}} = lnP_t - lnP_{t-1} \quad (1)$$

The absolute value of daily returns measures daily volatility:

$$v_t = \sqrt{{r_t}^2} \quad (2)$$

There are 252 trading days annually. Thus, the way to calculate annualized daily volatility is as follows:

$$v_t = \sqrt{252{r_t}^2} \tag{3}$$

3.2 GARCH Model

We firstly build a classical GARCH model to predict future stock return volatility. Only history volatility data is needed for GARCH since it is a simple time series regression model. The Python third-party package ARCH is used to build the GARCH model, which can estimate coefficients, conduct significance tests, and forecast volatility using the rolling-window method. A typical $GARCH(p,q)$ model can be expressed as:

$$r_t = \mu + \varepsilon_t \tag{4}$$

$$\varepsilon_t = \sigma_t \cdot e_t \tag{5}$$

$${\sigma_t}^2 = \omega + \sum_{i=1}^{q} \alpha_i {\varepsilon_{t-i}}^2 + \sum_{j=1}^{p} \beta_j {\sigma_{t-j}}^2 \tag{6}$$

$$e_t \sim N(0,1) \tag{7}$$

The item e_t in Eq. (5) is independent and identically distributed. Parameters p and q should be determined by R^2 between the data in the current period and data in each lagging period. After deciding two parameters, one could use maximum likelihood to estimate coefficient ω, α_i, and β_j.

3.3 RNNs

The most common neuro-networks are feedforward, which has the data flow from the input layer to neurons with activation in hidden layers and the output layer, getting the predicted value. For time series data, the data in one period may have connections with data in previous periods, where feedforward neuro-networks cannot reach. This problem can be solved by implementing recurrent layers, whose outputs are affected by both the inputs on the current step and the outcomes on the previous step, giving the network a short memory, which usually decays over time. Figure 1 illustrates a sketch of the RNNs. The mathematical expression of the outputs of each layer on RNNs can be demonstrated as

$$\boldsymbol{Y}_t = \emptyset\left([\boldsymbol{X}_t \quad \boldsymbol{Y}_{t-1}]\begin{bmatrix}\boldsymbol{W}_x \\ \boldsymbol{W}_y\end{bmatrix} + b\right) \tag{8}$$

where $\boldsymbol{X}_t$ is the current input vector, $\boldsymbol{Y}_{t-1}$ is the output vector of the last step, $\emptyset$ is the activation function, $\boldsymbol{W}_x$ and $\boldsymbol{W}_y$ are the weight of two vectors, respectively.

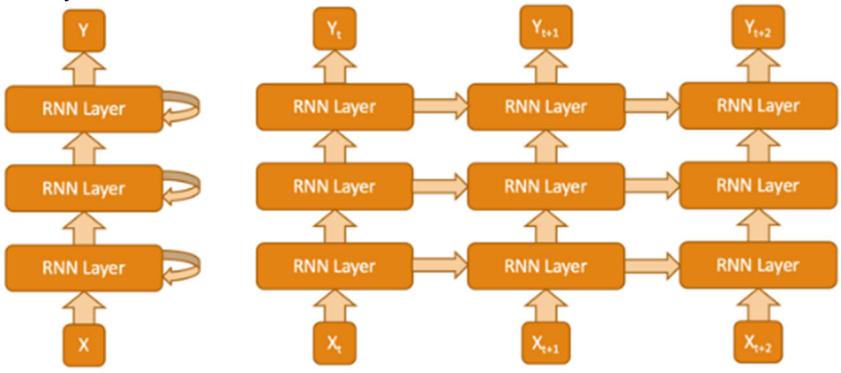

Fig. 1 Illustration of Simple RNN.

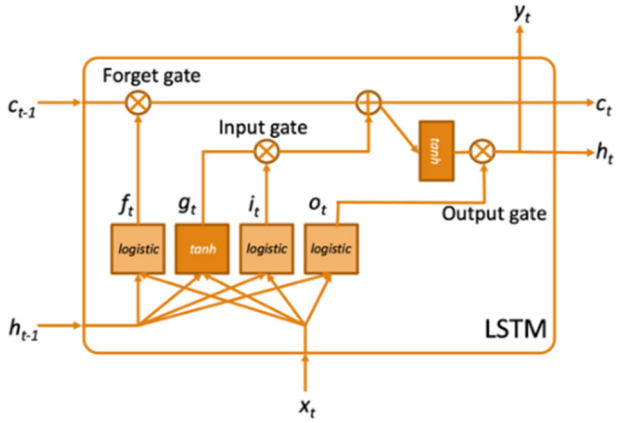

Fig. 2 Structure of an LSTM cell.

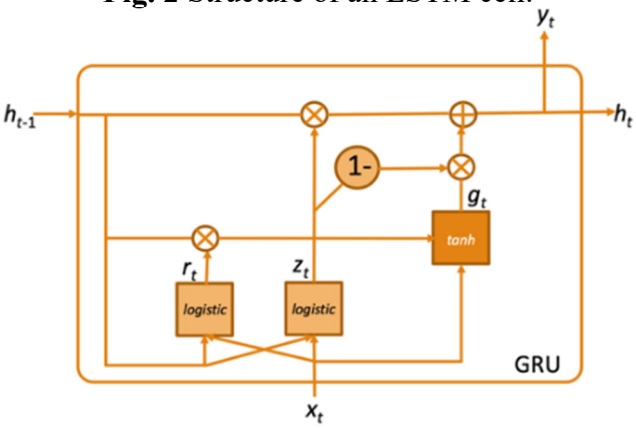

Fig. 3 Structure of a GRU cell.

3.4 LSTM Cells

Basic RNNs do not have long-term memory as the memory decays quickly over time. Long short-term memory is a cell that can capture long-term relations between data, which was innovated by Hochreiter et al. in 1997 [11]. Fig. 2 shows the structure of an LSTM cell. Here, $\boldsymbol{x}_t$ is the input vector, $\boldsymbol{c}_t$ is the long-term state vector that contains long memory and $\boldsymbol{h}_t$ is a short memory vector. Vector $\boldsymbol{c}_{t-1}$ drops some information after element-wise multiplication with $\boldsymbol{f}_t$, the output of the logistic function ranges from 0 to 1, and then embrace some new information after adding data from the input gate, finally forming the new state vector $\boldsymbol{c}_t$. Output gate is used to calculate the output of the neuron and the new short memory vector $\boldsymbol{h}_t$. Eqs. (9) to (14) display the detailed calculation of each vector, where σ means logistic activation and $\otimes$ means element-wise multiplication. The most impressive feature of the LSTM cell is that it can keep a very long memory of essential information through $\boldsymbol{c}_t$ compared with simple RNNs because only simple linear calculations are applied to $\boldsymbol{c}_{t-1}$.

$$\boldsymbol{i}_t = \sigma\left(\boldsymbol{W}_{xi}{}^{\top}\boldsymbol{x}_t + \boldsymbol{W}_{hi}{}^{\top}\boldsymbol{h}_{t-1} + \boldsymbol{b}_i\right) \tag{9}$$

$$\boldsymbol{f}_t = \sigma\left(\boldsymbol{W}_{xf}{}^{\top}\boldsymbol{x}_t + \boldsymbol{W}_{hf}{}^{\top}\boldsymbol{h}_{t-1} + \boldsymbol{b}_f\right) \tag{10}$$

$$\boldsymbol{o}_t = \sigma\left(\boldsymbol{W}_{xo}{}^{\top}\boldsymbol{x}_t + \boldsymbol{W}_{ho}{}^{\top}\boldsymbol{h}_{t-1} + \boldsymbol{b}_o\right) \tag{11}$$

$$\boldsymbol{g}_t = tanh\left(\boldsymbol{W}_{xg}{}^{\top}\boldsymbol{x}_t + \boldsymbol{W}_{hg}{}^{\top}\boldsymbol{h}_{t-1} + \boldsymbol{b}_g\right) \tag{12}$$

$$\boldsymbol{c}_t = \boldsymbol{f}_t \otimes \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \otimes \boldsymbol{g}_t \tag{13}$$

$$\boldsymbol{y}_t = \boldsymbol{h}_t = \boldsymbol{o}_t \otimes tanh(\boldsymbol{c}_t) \tag{14}$$

3.5 GRU Cells

In 2014, the gated recurrent unit (GRU) cell was innovated by Cho et al. [12]. GRU cells can be recognized as simplified variants of LSTM cells. Many researchers found that GRU cells perform as well as LSTM cells. The main difference between LSTM cells and GRU cells is that vector $\boldsymbol{h}_t$ stores both the long-term memory and the short-term one. Two inputs of a GRU cell are the previous state

vector $\boldsymbol{h}_{t-1}$ and current input vector $\boldsymbol{x}_t$. Forget gate and input gate are all manipulated using vector $\boldsymbol{z}_t$, forget gate is manipulated directly. Still, the signal sent to the input gate is $\mathbf{1}-\boldsymbol{z}_t$, which means the useless information should be deleted before adding new information. To impose more control on the previous vector $\boldsymbol{h}_t$, new gate $\boldsymbol{r}_t$ and $\boldsymbol{g}_t$ are added to GRU cells. In short, GRU cells use one state vector to manage the flow of information between periods. Compared with LSTM cells, GRU cells enable a perfect performance with less computation. Fig. 3 illustrates how GRU cells works, and the mathematical expression of the mechanism of the cell is shown in equation (15) to (18).

$$\boldsymbol{z}_t = \sigma\left(\boldsymbol{W}_{xz}{}^{\top}\boldsymbol{x}_t + \boldsymbol{W}_{hz}{}^{\top}\boldsymbol{h}_{t-1} + \boldsymbol{b}_z\right) \tag{15}$$

$$\boldsymbol{r}_t = \sigma\left(\boldsymbol{W}_{xr}{}^{\top}\boldsymbol{x}_t + \boldsymbol{W}_{hr}{}^{\top}\boldsymbol{h}_{t-1} + \boldsymbol{b}_r\right) \tag{16}$$

$$\boldsymbol{g}_t = tanh\left(\boldsymbol{W}_{xg}{}^{\top}\boldsymbol{x}_t + \boldsymbol{W}_{hg}{}^{\top}(\boldsymbol{r}_t \otimes \boldsymbol{h}_{t-1}) + \boldsymbol{b}_g\right) \tag{17}$$

$$\boldsymbol{h}_t = \boldsymbol{z}_t \otimes \boldsymbol{h}_{t-1} + (1-\boldsymbol{z}_t) \otimes \boldsymbol{g}_t \tag{18}$$

3.6 Performance Measure

Mean absolute error (MAE) and mean squared error (MSE) are metrics to measure the loss of the model or the model's error to predict in this paper. The calculation of MAE and MSE can be expressed as:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}\left|Y_i - \hat{Y}_i\right| \tag{19}$$

$$loss = MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{20}$$

Here, Y_i is the actual value, $\hat{Y}_i$ is the predicted value, and n is the sample size. MSE is used as a loss function during the training section.

4. Empirical analysis

This section will discuss the training and testing of models and optimization methods and compare the performance of different models. The parameter estimation of the GARCH model is done through the python third-party package ARCH. The neural networks in this article are all built by TensorFlow.

The current volatility data strongly correlates with the lag 1 to 2 period data. However, the parameter estimates of the GARCH(2,2) model cannot pass the significance test, so the GARCH(1,1) model is selected. The parameters of the GARCH model are obtained by maximum likelihood estimation. The estimated values of ω, α and β parameters are 0.0848, 0.1352, 0.8267, respectively. The estimated values of the three parameters all passed the significance test at the 0.05 significance level.

Before molding with neuro-networks, we use min-max normalization to set the data range to 0 to 1 to avoid unstable gradient problems. The calculation method of min-max normalization is shown in equation (21).

$$\boldsymbol{X} \leftarrow \frac{\boldsymbol{X}-\boldsymbol{X}_{min}}{\boldsymbol{X}_{max}-\boldsymbol{X}_{min}} \tag{21}$$

Two optimizers are adopted to train neuro-networks. Stochastic gradient descent (SGD) is one of the most classical optimizers in machine learning. SGD searches the lowest point of the loss function through partial derivatives; if the gradient is a downturn, the weight will increase and verse versa. It was proven to have perfect optimization performance in most cases. However, SGD has a slow convergence speed, and the optimization quality is unsatisfactory in some circumstances. The process of SGD is displayed in Eqs. (22) and (23), where $\boldsymbol{w}$ is the weights vector, η is the learning rate, and $g(\boldsymbol{w})$ is the vector that contains partial derivatives.

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \cdot g(\boldsymbol{w}) \tag{22}$$

$$g(\boldsymbol{w}) = \begin{bmatrix} \frac{\partial}{\partial w_0} MSE(\boldsymbol{w}) \\ \frac{\partial}{\partial w_1} MSE(\boldsymbol{w}) \\ \cdots \\ \frac{\partial}{\partial w_n} MSE(\boldsymbol{w}) \end{bmatrix} \tag{23}$$

Another optimizer utilized here is adaptive moment estimation (Adam). The optimizer has a memory of gradients and squared gradients in both current and previous steps stored in vector $\boldsymbol{m}$ and $\boldsymbol{s}$, respectively, and the memory can decay over time through parameters β_1 and β_2. $\widehat{\boldsymbol{m}}$ is the momentum vector and vector $\hat{\boldsymbol{s}}$ makes the learning rate adaptive to the magnitude of the gradient. All the factors make Adam reaches the global optimum more accurately and faster. Eqs (23) to (27) show the detailed algorithm of the optimizer.

$$\boldsymbol{m} \leftarrow \beta_1 \boldsymbol{m} - (1 - \beta_1) \cdot g(\boldsymbol{w}) \tag{24}$$

$$\boldsymbol{s} \leftarrow \beta_2 \boldsymbol{s} - (1 - \beta_2) \cdot g(\boldsymbol{w}) \otimes g(\boldsymbol{w}) \tag{24}$$

$$\widehat{\boldsymbol{m}} \leftarrow \frac{\boldsymbol{m}}{1-\beta_1^{\intercal}} \tag{26}$$

$$\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1-\beta_2^{\intercal}} \tag{27}$$

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta \cdot \frac{\widehat{\boldsymbol{m}}}{\sqrt{\hat{\boldsymbol{s}}+\boldsymbol{\varepsilon}}} \tag{28}$$

In 2015, Ioffe et al. proposed batch normalization (BN) to tackle unstable gradient problems and over-fitting [13]. The idea of batch normalization is to normalize the data of inputs in each batch before the optimization process. In Eq. (29), input vector $\boldsymbol{x}$ is normalized to $\widehat{\boldsymbol{x}}$ with a mean of 0 and a standard deviation of 1; $\boldsymbol{z}$ is the final output of the BN process, which is the result of the linear calculation on $\widehat{\boldsymbol{x}}$.

$$\widehat{\boldsymbol{x}} = \frac{\boldsymbol{x}-\boldsymbol{\mu}}{\sqrt{\boldsymbol{\sigma}^2+\varepsilon}} \tag{29}$$

$$\boldsymbol{z} = \boldsymbol{\gamma} \otimes \widehat{\boldsymbol{x}} + \boldsymbol{\beta} \tag{30}$$

In the same year, Laurent et al. and other researchers believed that BN is imperfect at optimizing RNNs, but layer normalization can do the job [14]. However, for our data and model, the RNNs with BN consistently outperform RNNs with layer normalization. The structure of the best RNNs we found is summarized in Table 2. We drop 20% neurons per layer to avoid over-fitting with training data. A 20% dropout rate tends to have the best results for our model. Figure 4 depicts the performance of simple RNNs, measured by MAE of the testing set, with different normalization methods and optimizers. According to the results, applying optimizer Adam and implementing batch normalizations can give simple RNNs the best performance.

Batch normalization can also have a positive effect on the LSTM models. Table 3 lists the best structure for the LSTM model, having a 20% dropout rate and the implementation of BN. Figure 5 exhibits performance with different optimizers. It indicates that Adam has a better performance on the predictions on the test set but a poorer performance on the train set. The possible reason for this phenomenon is the over-fitting caused by SGD. The emergence of this phenomenon has made us think that optimizer significantly impacts training effects, and the choice of optimizer should be adapted to local conditions.

Table 2. Structure of best RNNs

Type	Neuron quantity	Dropout rate	Activation
RNN	25	20%	SeLu
RNN(BN)	30	20%	SeLu
RNN(BN)	30	20%	SeLu
RNN(BN)	30	20%	SeLu
RNN(BN)	25	20%	SeLu

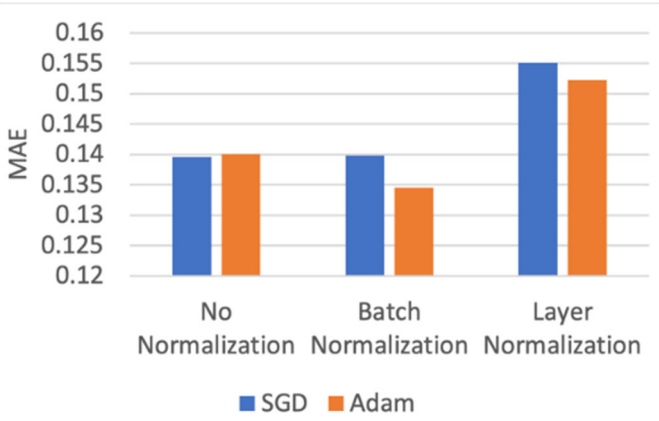

Fig. 4 Simple RNNs performance with different normalization methods and optimizers.

Table 3. The best structure of the LSTM model

Type	Neuron quantity	Dropout rate
LSTM	20	20%
LSTM(BN)	20	20%
LSTM(BN)	20	20%
LSTM(BN)	20	20%
LSTM(BN)	20	20%

Fig. 5 LSTM performance with different optimizers

SGD tends to have better performance on GRU models. Figures 6 displays the different performance of the GRU model with different optimizers. Table 4 illustrates the best GRU model structure. By comparison, it can be found that the SGD optimization method fits the train set better when training the GRU model, and the Adam optimization method's prediction accuracy of the test set is also slightly improved.

Table 5 shows the MAE of different models. The table shows that GRU has the best predictive accuracy for test set data. The prediction accuracy of LSTM is second only to GRU. Compared with the GARCH model, the accuracy of GRU and LSTM improved by 11% and 14%, respectively. There is no significant improvement for the simple RNN model.

Table. 4 Best structure of the GRU model

Type	Neuron quantity	Dropout rate
GRU	25	10%
GRU(BN)	25	10%
GRU(BN)	25	10%
GRU(BN)	25	10%
GRU(BN)	25	10%

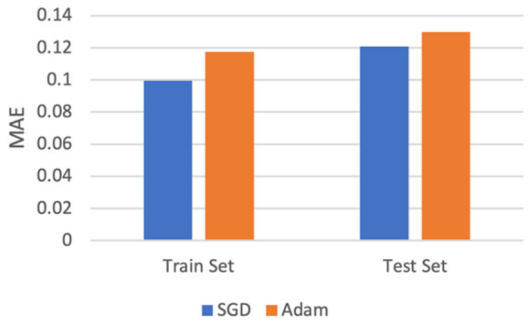

Fig. 6 GRU performance with different optimizers

Table 5 Comparison of performance.

Model	Data	MAE
GARCH	Train set	0.11265
	Test set	0.13978
Simple RNN	Train set	0.11571
	Test set	0.13448
LSTM	Train set	0.10543
	Test set	0.12469
GRU	Train set	0.09950
	Test set	0.12087

Figs 7 to 10 present the distribution of actual volatility and the distribution of volatility predicted by different models. A few negative values appeared in the picture because of the application of the kernel density estimation, and the bandwidth was large. One can observe that simple RNN and LSTM predicted volatility are most like the actual distribution. Therefore, they have the most robust ability to predict extreme values. The forecasting ability of the GARCH model is not ideal. The data obtained from the forecast are all distributed in a small range, and there is no ability to predict large and small volatility accurately. The distribution characteristics of GRU's predicted data differ from the distribution characteristics of actual data. GRU's prediction of extreme values may not be as good as LSTM and simple RNN. However, compared with the GARCH model, it has a significant improvement. Based on empirical analysis, it is believed that each neuro-network model has its advantages, and the prediction accuracy is better than the GARCH model.

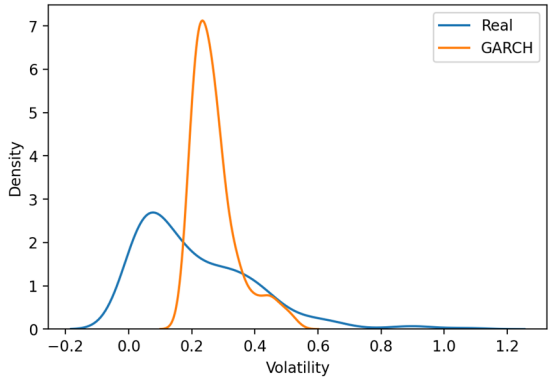

Fig. 7 Distribution of actual volatility and volatility predicted by GARCH

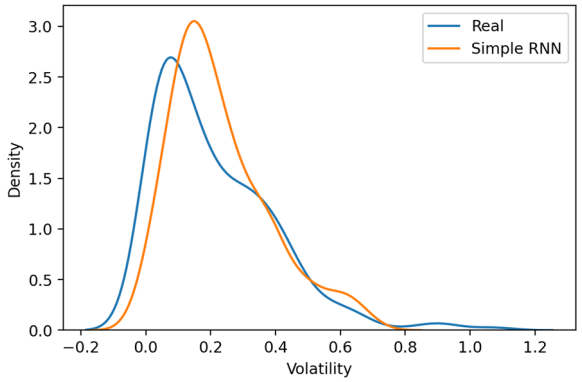

Fig. 8 Distribution of actual volatility and volatility predicted by simple RNN

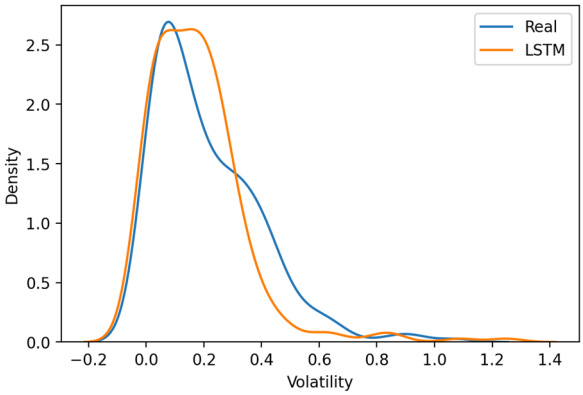

Fig. 9 Distribution of actual volatility and volatility predicted by LSTM

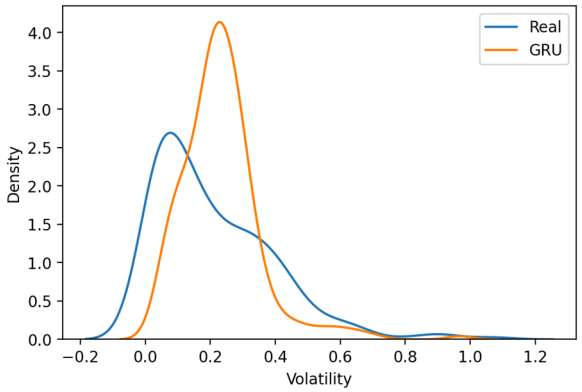

Fig. 10 Distribution of actual volatility and volatility predicted by GRU

5. Summary

In conclusion, this paper investigates stock price prediction based on GARCH and RNN. The primary contribution of this paper is to study and analyze the performance and respective advantages of the GARCH model and various neural network models integrating market sentiment indicators for stock volatility forecasting. Our forecasting models include not only technical indicators but also market sentiment data. In brief, neuro-networks with LSTM and GRU cells obtained perfect performance on volatility prediction. According to the analysis, simple RNN and LSTM predicted volatilities have the distribution characteristics closest to the actual situation, while traditional time series regression method GARCH has the poorest prediction performance. Although other studies have shown that the effect of layer normalization is better for RNNs, adding batch normalization to all models has a better effect on performance improvement for the data of this study. The optimizer will affect the model's performance, depending on the specific data and model, and the optimizer suitable for different models or data is also different.

This study has two significant limitations. First, the small amount of data may lead to limited generalization ability. Secondly, the search volume cannot reflect market sentiment fully and objectively and may deviate from the actual market. In future research, it is necessary to improve the quality and quantity of data and apply more objective market sentiment indicators to obtain more accurate predictive models that can be used for financial risk management. Overall, the results demonstrate several models with better forecasting performance and can provide important references for the future application of machine learning in stock market volatility forecasting.

References

[1] Engle Robert F. Autoregressive Conditional Heteroscedasticity with Estimates of the Variance of United Kingdom Inflation. Econometrica, 1982, 50(4): 987–1007.

[2] Bollerslev T. Generalized autoregressive conditional heteroskedasticity. Journal of econometrics, 1986, 31(3): 307-27.

[3] Sentana E. Quadratic ARCH models. The Review of Economic Studies, 1995, 62(4): 639-61.

[4] Franses P H, Van Dijk D. Forecasting stock market volatility using (non-linear) GARCH models. Journal of forecasting, 1996, 15(3): 229-235.

[5] Donaldson R G, Kamstra M. Forecast combining with neural networks. Journal of Forecasting, 1996, 15(1): 49-61.

[6] Mantri J K, Gahan P, Nayak B B. Artificial neural networks–an application to stock market volatility. Soft-Computing in Capital Market: Research and Methods of Computational Finance for Measuring Risk of Financial Instruments, 2014, 179.

[7] Liu Y, Qin Z, Li P, et al. Stock volatility prediction using recurrent neural networks with sentiment analysis. International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, Springer, 2017, 192-201.

[8] Aimin Z, Rui G. Research on Volatility Prediction Based on Deep Learning Algorithm—Example of SSE 50ETF Option. Huabei Finance, 2021, (09): 7.

[9] Su Z, Xie H, Han L. Multi-factor RFG-LSTM algorithm for stock sequence predicting. Computational Economics, 2021, 57(4): 1041-1058.

[10] Google. FAQ about Google Trends data. Trends Help, 2022, Retrieved from: https://support.google.com/trends/answer/4365533?hl=en&ref_topic=6248052

[11] Hochreiter S, Schmidhuber J. Long short-term memory. Neural computation, 1997, 9(8): 1735-1780.

[12] Cho K, Van Merriënboer B, Gulcehre C, et al. Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078, 2014

[13] Ioffe S, Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift, International conference on machine learning. PMLR, 2015: 448-456.

[14] Laurent C, Pereyra G, Brakel P, et al. Batch normalized recurrent neural network. 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2016: 2657-2661.