Hindawi
Complexity
Volume 2021, Article ID 5511802, 13 pages
https://doi.org/10.1155/2021/5511802

Research Article

Forecasting Volatility of Stock Index: Deep Learning Model with Likelihood-Based Loss Function

Fang Jia [1] **and Boli Yang** [2]

[1]*School of Management, Huazhong University of Science and Technology, Wuhan 430074, China*
[2]*Investment Product Department, Creditease Corp., Beijing 100020, China*

Correspondence should be addressed to Fang Jia; jiafanghust@hust.edu.cn

Received 7 January 2021; Revised 1 February 2021; Accepted 16 February 2021; Published 25 February 2021

Academic Editor: Benjamin Miranda Tabak

Volatility is widely used in different financial areas, and forecasting the volatility of financial assets can be valuable. In this paper, we use deep neural network (DNN) and long short-term memory (LSTM) model to forecast the volatility of stock index. Most related research studies use distance loss function to train the machine learning models, and they gain two disadvantages. The first one is that they introduce errors when using estimated volatility to be the forecasting target, and the second one is that their models cannot be compared to econometric models fairly. To solve these two problems, we further introduce a likelihood-based loss function to train the deep learning models and test all the models by the likelihood of the test sample. The results show that our deep learning models with likelihood-based loss function can forecast volatility more precisely than the econometric model and the deep learning models with distance loss function, and the LSTM model is the better one in the two deep learning models with likelihood-based loss function.

1. Introduction

In finance, volatility refers to the variation degree of asset price, and it measures the uncertainty of the price. It plays an important role in both academic research and financial industry. In risk management and performance measurement, volatility is a risk indicator itself and can be a part of some other indicators, like the Sharpe ratio. In portfolio theory, Markowitz [1] used volatility to measure the risks of assets and the overall risk of the portfolio. Volatility is both the input and the optimization target of the portfolio construction model. In derivative pricing, prices of derivatives can be determined by the volatility of the underlying assets [2].

Since volatility can be useful in different financial areas, and in some situations, the information about future volatility is needed to make financial decisions, it can be valuable to forecast volatility. Econometric methods are widely used to forecast the volatility of financial assets. Engle [3] introduced autoregressive conditional heteroscedasticity (ARCH) model to describe volatility. In this model, the conditional variance is given as a function of the previous variances. The volatility dynamics can be gained by maximizing the likelihood of the model, and then the estimated model can be used to forecast future volatility. Bollerslev [4] extended the ARCH model to be a generalized autoregressive conditional heteroscedasticity (GARCH) model. This model makes conditional variance to be a function of previous errors and previous variances. Nelson [5] further extended the GARCH model and built exponential generalized autoregressive conditional heteroscedasticity (EGARCH) model. An asymmetric response to shocks is allowed in this model. Corsi [6] used intraday high-frequency data to calculate realized volatility and introduced a heterogeneous autoregressive (HAR) model to forecast the volatility.

Using machine learning algorithms is another way to forecast volatility. Compared to econometric models which are based on economic assumptions and statistical logic, machine learning algorithms are more data-driven. A large

number of papers combined neural networks and the GARCH model to be a hybrid model and used the hybrid model to forecast volatility [7–15]. These papers used this type of method and studied different kinds of assets, including stock indices in several nations, metals, oil, and bitcoin. They all concluded that the hybrid neural networks can forecast volatility precisely. Some papers made other kinds of combinations and gained successful predictions. Lahmiri [16] combined neural networks and technical indicators to forecast the volatility of exchange rates. Peng et al. [17] combined support vector regression and GARCH models to forecast volatility of currencies. Ramos-Pérez et al. [18] use a set of machine learning techniques, such as gradient descent boosting, random forest, support vector machine, and neural network, and stack them to forecast volatility of S&P 500.

Some papers used deep learning, which is a special branch of machine learning, to forecast volatility. Since LSTM is an effective machine learning architecture to model time series, some papers combined LSTM and GARCH model to be hybrid model and used the hybrid model to forecast volatility [19–21]. They compared their models with some other models, such as support vector regression (SVR) and GARCH. The hybrid deep learning models showed strong forecasting ability. Xing et al. [22] used text mining to capture the market sentiment from social message streams and incorporated the sentiment signals into the recurrent neural networks (RNNs). Their model defeated nine other models on volatility forecasting. Vidal and Kristjanpoller [23] combined two popular deep learning architectures, which are convolution neural network (CNN) and LSTM, to forecast the volatility of gold. Yu and Li [24] defined extreme value volatility for stock index, which is calculated from daily highest prices and daily lowest prices. They used historical volatilities to forecast future volatility, by LSTM and GARCH separately.

Almost all these machine learning references follow a similar process to forecast volatility. They use historical data as input to forecast volatility for each day, and they use statistical methods to estimate the realized volatility for the same day. Then, they use the distance between the two, for example, mean squared error (MSE) or mean absolute error (MAE), as the loss function, to train the machine learning model. By minimizing the distance function, they can gain effective machine learning models to forecast realized volatility.

This is a common process when researchers try to predict a target by machine learning, but it gains two disadvantages if they follow it to forecast volatility. First, because volatility is unobservable, when researchers use estimated realized volatility as the target, they introduce errors into the forecasting process. These kinds of errors would decrease the accuracy of their prediction because what they try to forecast is not real volatility. Second, econometric models estimate their parameters by maximizing the likelihood of sample data, which is different from the optimization target of these machine learning models. Since these researchers always use distance value to further test the forecasting ability, the training method of the econometric model is not consistent with their testing method. So the comparison between the deep learning model and the econometric model is not fair enough in these papers.

To solve these two problems, we introduce a negative log-likelihood function to be the loss function of the deep learning model in this paper. By using this likelihood-based loss function, we can train the deep learning model without estimating the realized volatility, so the forecasting process can be more straightforward. At the same time, the deep learning model can be compared to the econometric model more fairly, since the training methods of the deep learning model and econometric model are both based on the likelihood function. For consistency, we test the models also by calculating the likelihood function of the test sample.

In addition, the deep learning model we build only uses index returns as the inputs to forecast volatility, including no economic intuition in the model. This is an end-to-end deep learning method since we go straight from the historical return series to volatility prediction. The deep learning model and econometric model use the same inputs to forecast volatility, so the comparison between them is fair enough.

The rest of this paper is organized as follows. Section 2 describes the data and methodology we use. The empirical study and sensitivity analysis are discussed in Section 3. Section 4 makes the conclusion.

2. Materials and Methods

2.1. Data. This research studies three major indices of the US stock market, which are S&P 500 Index, Dow Jones Industrial Average Index, and NASDAQ Composite Index. We obtain closing prices of these stock indices from their start dates to June 30, 2020. In detail, S&P 500 sample data cover from January 2, 1928, to June 30, 2020, Dow Jones sample data cover from May 26, 1896, to June 30, 2020, and NASDAQ sample data are from February 5, 1971, to June 30, 2020. The three sample periods include 23240, 31096, and 12457 trading days separately.

Then we calculate daily returns as the logarithms of relative daily closing prices, using the following equation:

$$r_t = \log\left(P_t/P_{t-1}\right), \tag{1}$$

where r_t is the daily return at time t and P_t is the closing price at time t.

Table 1 shows the summary statistics of the three return series. All three indices gain positive average return within the sample period. S&P 500 and Dow Jones experienced more serious single-day loss than NASDAQ. Standard deviations of the three return series are similar. All the return series show negative skewness and high kurtosis. It can be confirmed from the Jarque–Bera test that these return series are not from normal distribution.

2.2. Deep Neural Network. Artificial neural network (ANN) is one of the best known machine learning algorithms. It is designed to imitate the structure of neurons in the human brain. Artificial neurons are connected to each other in this

Table 1: Descriptive statistics of daily returns of stock indices.

	S&P 500	Dow Jones	NASDAQ
Mean	0.022%	0.021%	0.037%
Maximum	15.36%	14.27%	13.25%
Minimum	−22.90%	−25.63%	−13.15%
Std. dev.	1.19%	1.16%	1.25%
Skewness	−0.47	−0.86	−0.38
Kurtosis	22.12	27.72	13.57
Jarque–Bera test	0.00	0.00	0.00
Observations	23239	31095	12456

model, and the networks can acquire a problem-solving ability by adjusting their connection weights through learning.

Deep neural network is ANN with a certain level of complexity. Under general definition, DNN is a neural network with two or more hidden layers. Figure 1 shows the typical architecture of DNN. Inputs are imported into the model through the input layer. Hidden layers and output layer are calculated sequentially by multiplying the previous layer with the connection weights, and the activation function is applied each time.

To make a DNN model work, we need to train it. By using the predicted value that we gain from the output layer and the target value, we can form a loss function. The connection weights in DNN are learned by minimizing the loss function. We mostly use forward and back propagation to deal with the calculation process when optimizing the networks.

2.3. LSTM. LSTM is a particular type of deep neural network developed by Hochreiter and Schmidhuber [25]. By inheriting the benefits from the recurrent neural network (RNN), LSTM has shown great power to model sequential data, like time series. Compared to vanilla RNN, LSTM uses gates to control information flows through the sequence so that it can be capable of learning long-term dependencies and solving the vanishing gradient problem.

Figure 2 shows the typical architecture of LSTM. Like vanilla RNN, inputs are imported into the model at each time step, and outputs can be given at each time step. Especially, LSTM uses memory block to take place of neuron in RNN. A memory block is composed of a memory cell, an input gate, a forget gate, and an output gate.

We can use vector formulas to describe LSTM, as follows:

$$\begin{aligned} X &= \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix}, \\ f_t &= \sigma\left(W_f X + B_f\right), \\ i_t &= \sigma\left(W_i X + B_i\right), \\ o_t &= \sigma\left(W_o X + B_o\right), \\ c_t &= f_t \nabla c_{t-1} + i_t \nabla \tanh\left(W_c X + B_c\right), \\ h_t &= o_t \nabla \tanh\left(c_t\right), \end{aligned} \tag{2}$$

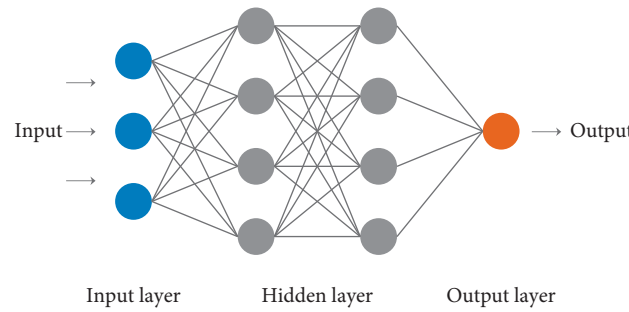


Figure 1: Architecture of DNN.

where h_t is the hidden state at time t and x_t represents the inputs at time t. σ is activation function where logistic sigmoid is popularly used. f_t describes the forget gate, i_t describes the input gate, and o_t describes the output gate. ∇ denotes point-wise multiplication.

2.4. Econometric Model. ARMA-GARCH is the most widely used econometric model when studying volatility. We will use this model to forecast volatility and compare it with deep learning models in this paper. Bollerslev [4] built the GARCH model by generalizing the ARCH model. A general ARMA(m, n)-GARCH(p, q) model is as follows. Firstly, the return is decomposed into AR effects and MA effects:

$$r_t = \mu + \sum_{i=1}^{m} \theta_i \left(r_{t-i} - \mu\right) + \sum_{j=1}^{n} \gamma_j \varepsilon_{t-j} + \varepsilon_t, \tag{3}$$

where μ is the mean of r_t and ε_t is the error term.

It is assumed that ε_t can be given as follows:

$$\varepsilon_t = z_t h_t^{1/2}, \tag{4}$$

where $z_t \sim N(0, 1)$, so h_t is the conditional variance at time t.

Then, the conditional variance is assumed to be a linear function of the errors and its own lags:

$$h_t = \omega + \sum_{i=1}^{p} \alpha_i \varepsilon_{t-i}^2 + \sum_{j=1}^{q} \beta_j h_{t-j}. \tag{5}$$

2.5. Simple Historical Statistics as Benchmark. Besides deep learning models and ARMA-GARCH, we further build a simple method to forecast volatility. Similar to Markowitz [1] which used historical statistics to be the expected risk, we calculate the standard deviation of the return series in an n-trading-day window and use it as the prediction of volatility for the next trading day. This is a simple but intuitive method, and we can use it as a benchmark to evaluate other methods.

The parameter which needs to be estimated is just the window length n. We also use the optimization method similar to econometric models to gain the estimation, which means that the best window length is the value that can maximize the likelihood of sample data.

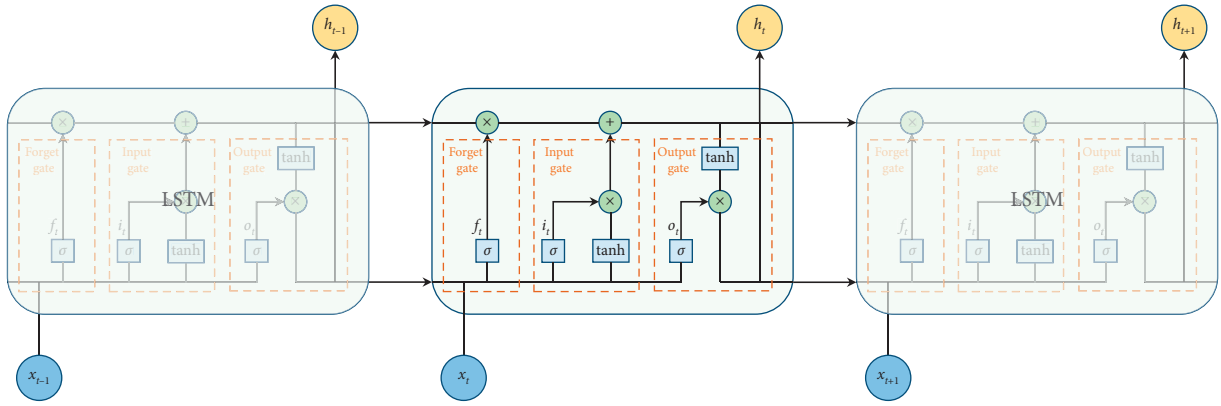


Figure 2: Architecture of LSTM network.

2.6. Experimental Setup of Our Deep Learning Models. As discussed in the Introduction, we use a likelihood-based loss function when training our DNN and LSTM models. If r_t is the return series that we observe from time 1 to time T and σ_t is the volatility that our models forecast at the same time step, we can calculate the sample likelihood under the assumption of normal distribution:

$$L = \prod_{i=1}^{T} \frac{1}{\sqrt{2\pi}\sigma_t} \exp\left(-\frac{r_t^2}{2\sigma_t^2}\right). \tag{6}$$

Then, we can have the log-likelihood function:

$$\log L = \sum_{i=1}^{T} \left(-\frac{1}{2}\log(2\pi) - \log\ \sigma_t - \frac{r_t^2}{2\sigma_t^2}\right). \tag{7}$$

The log-likelihood function should be maximized through optimization, while the deep learning models are always trained by minimizing their loss function. So we need to use the negative log likelihood to be the loss function in deep learning models. To further reduce the computational cost of deep learning models, we use a simplified function as the loss function of our DNN and LSTM models:

$$\text{loss} = \sum_{i=1}^{T} \left(2\ \log\ \sigma_t + \frac{r_t^2}{\sigma_t^2}\right). \tag{8}$$

When we test all the models we discuss in this paper, we will use equation (7) to calculate the log-likelihood function of the test sample and compare the values.

No matter in the training process or in the testing process of all the methods, the forecasting procedure is implemented day by day. It means that we forecast the volatility in a rolling window, and the window moves forward one trading day at each time. After we forecast the volatility day by day and get the volatility series σ_t from time 1 to time T, we can use it to calculate equations (7) and (8) in different processes.

The other settings of our DNN model are as follows. The unit number of the input layer is set to be 10, which means that we use 10-day return series as input each time. The number of hidden layers is two. The first hidden layer has 40 units, and the second hidden layer has 80 units. The activation function of hidden layers is chosen to be ReLU, and the dropout of these two layers is set to be 0.3. The activation function of the output layer is set to be sigmoid. RMSprop is used as the optimizer to train the model. The batch size is set to be 2048. There will be early stopping if the loss function of the validation set does not go down anymore.

Our LSTM model combines LSTM with a fully connected layer in the last time step. Its architecture is shown in Figure 3. The length of the input series is set to be 10. The unit number is 20 at each layer of LSTM. The fully connected layer has 40 units, and its activation function is chosen to be ReLU. For the fully connected layer, we set the dropout to be 0.5, while there is no dropout in the LSTM layer. The activation function of the output layer is set to be sigmoid. RMSprop is used as the optimizer to train the model. The batch size is set to be 2048. There will be early stopping if the loss function of the validation set does not go down anymore.

It is noteworthy that the activation function of the output layer in our two deep learning models is an important setting, when we introduce the likelihood-based loss function. If the loss function is the distance function that other research studies use, linear activation function can be good enough. But when we use the likelihood-based loss function as equation (8), some activation functions like linear function cannot help the models to converge when training. Sigmoid activation function is a good choice in the output layer, and our models can be trained successfully.

All the deep learning models in this paper, which include the DNN model and the LSTM model that we have mentioned above, and two more models with different loss functions that are used for comparison are implemented by the Keras framework.

For DNN and LSTM models, we divide the full sample into three parts: train set (70%), validation set (15%), and test set (15%). ARMA-GARCH model and the simple method do not need validation when training, so we combine the train set and validation set for these two methods and estimate their parameters on this full train set (85%). Train-test split for the ARMA-GARCH and simple method and train-validation-test split for the deep learning models are both in

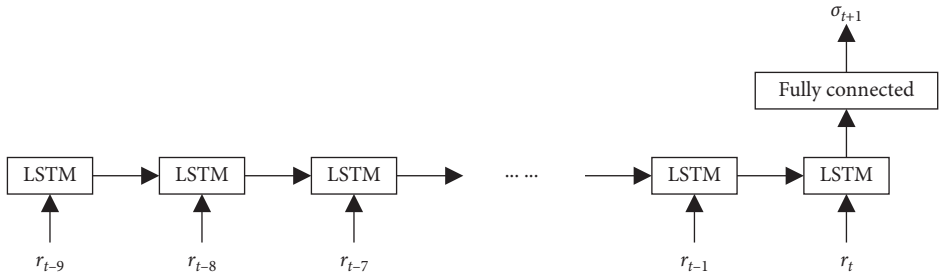


Figure 3: Architecture of our LSTM model.

time order. They share the same test set, so they can be compared directly, by comparing the log-likelihood values of this test sample.

2.7. Experimental Setup of Models for Comparison. As we have discussed, in almost all the reference papers which use machine learning to forecast volatility, the distance between estimated realized volatility and forecasted volatility is used as the loss function to train the models. To compare this kind of common method to our deep learning models with likelihood-based loss function, we further test DNN and LSTM models with distance loss function like them. By doing so, we can also study whether the deep learning models can still have good forecasting ability, when we train them by minimizing the distance loss function and test them by calculating the likelihood value of the test sample. Since their training method is not consistent with their testing method, deep learning models with distance loss function are in a disadvantageous position when we compare them with other models in this paper.

The distance loss function of DNN and LSTM is chosen to be MSE. This loss function is the most chosen one of the related researches. Similar to these researches, the realized volatility is calculated by the standard deviation of the return series in a 21-trading-day window, which is close to the average number of trading days in a month. The activation function of the output layer is set to be the linear function, which is also a common choice of deep learning models with distance loss function, when forecasting volatility. All the other settings are the same as our deep learning models with likelihood-based loss function.

When training ARMA(m, n)-GARCH(p, q) model, we set m, n, p, and q to be 1 to 3 and estimate all different combinations. The model with minimum BIC is chosen and will be used for prediction on the test set. Simple method just needs to choose the best window length on the train set, as we have discussed in the previous section.

3. Results and Discussion

3.1. Empirical Results. When we train the LSTM model with likelihood-based function, we record the loss function values of train samples and validation samples at each epoch and show the learning curves in Figure 4. For S&P 500, the model gains minimum validation loss function at epoch 723, and the value of validation loss function is −15241.51 at this epoch. For Dow Jones, the model gains minimum validation loss function at epoch 568, and the value of validation loss function is −15755.87 at this epoch. For NASDAQ, the model gains minimum validation loss function at epoch 1212, and the value of validation loss function is −14600.25 at this epoch.

When we train the DNN model with likelihood-based loss function, we also record the loss function values of train samples and validation samples at each epoch and draw the learning curves as in Figure 5. For S&P 500, the model gains minimum validation loss function at epoch 1318, and the value of validation loss function is −15197.80 at this epoch. For Dow Jones, the model gains minimum validation loss function at epoch 622, and the value of validation loss function is −15735.40 at this epoch. For NASDAQ, the model gains minimum validation loss function at epoch 688, and the value of validation loss function is −14586.70 at this epoch.

When we train the LSTM model with MSE loss function, the optimization process stops at epoch 90 for S&P 500, at epoch 422 for Dow Jones, and at epoch 70 for NASDAQ. When we train the DNN model with MSE loss function, the optimization process stops at epoch 271 for S&P 500, at epoch 309 for Dow Jones, and at epoch 174 for NASDAQ. It is obvious that the deep learning models with MSE loss function converge faster than the deep learning models with likelihood-based loss function, under the same learning rate.

Then we try to find the suitable ARMA(m, n)-GARCH(p, q) models for each index. After we estimate all ARMA-GARCH models with m, n, p, and q from 1 to 3, we choose the best one with minimum BIC. For S&P 500, it is ARMA(1, 1)-GARCH(1, 2), for Dow Jones, it is ARMA(1, 1)-GARCH(1, 2), and for NASDAQ, it is ARMA(1, 2)-GARCH(1, 1). Table 2 shows the chosen models with their BIC values for each index.

Table 3 shows the parameter estimation results of the chosen ARMA-GARCH models for each index. For S&P 500 and Dow Jones, the AR effect is negative and the MA effect is positive. For NASDAQ, the AR effect is positive and the MA effect is negative. The parameters of ARCH term are all around 0.11 for three indices. The sum of ARCH and GARCH parameters is near 1 for each index. It can be concluded from the t-tests that almost all the parameters are significant at 5% level. Only ω of ARMA(1, 2)-GARCH(1, 1) for NASDAQ is significant at 10% level.

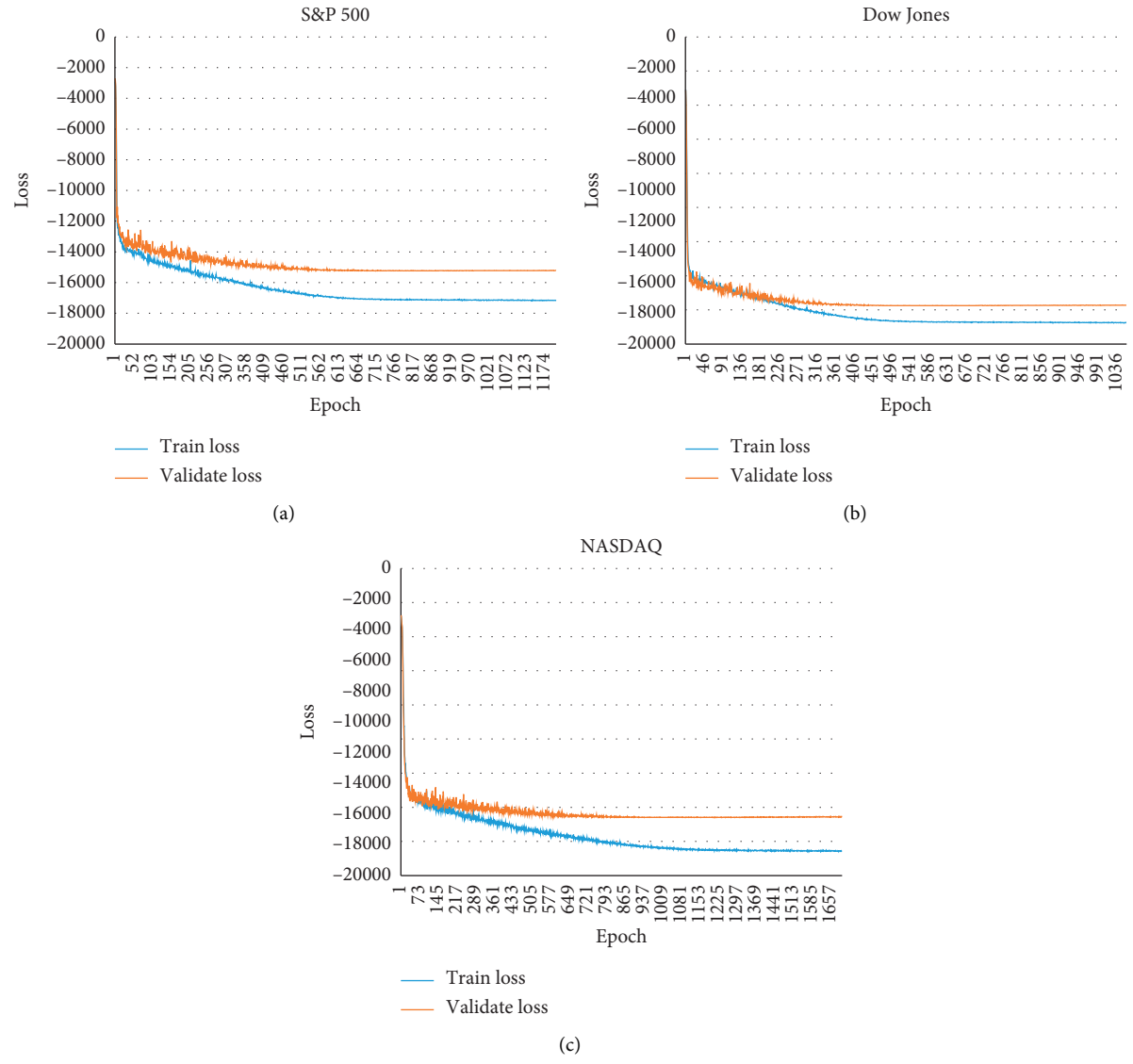


Figure 4: The learning curves when training LSTM (likelihood-based loss).

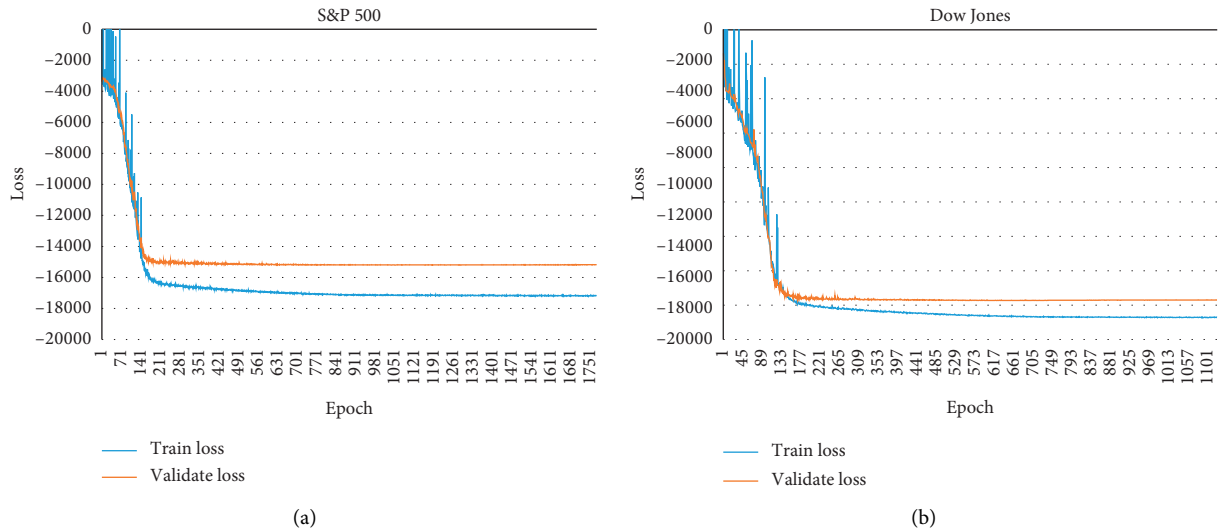
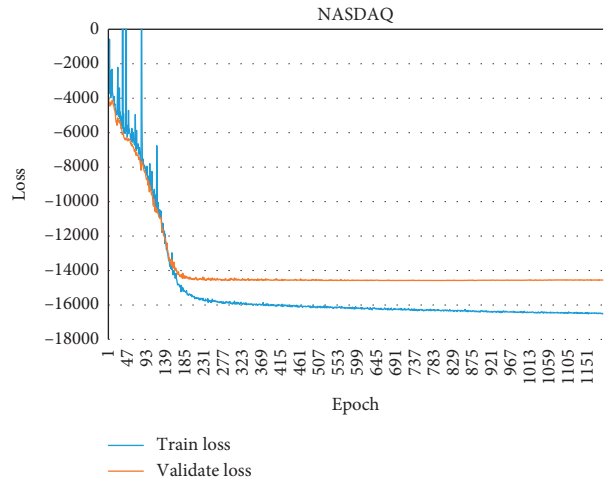


Figure 5: Continued.



(c)

Figure 5: The learning curves when training DNN (likelihood-based loss).

Table 2: ARMA-GARCH model with minimum BIC.

	S&P 500	Dow Jones	NASDAQ
Best model	ARMA(1, 1)-GARCH(1, 2)	ARMA(1, 1)-GARCH(1, 2)	ARMA(1, 2)-GARCH(1, 1)
BIC	−6.6148	−6.5230	−6.5385

Table 3: ARMA-GARCH estimation results.

	S&P 500		Dow Jones		NASDAQ	
	Estimate	T-test	Estimate	T-test	Estimate	T-test
μ	$4.59e-4$	8.10^{***}	$4.45e-4$	8.52^{***}	$6.20e-4$	6.16^{***}
θ_1	−0.150	-2.41^{**}	−0.155	-2.02^{**}	0.866	11.50^{***}
γ_1	0.259	4.26^{***}	0.235	3.12^{***}	−0.668	-8.65^{***}
γ_2					−0.147	-6.43^{***}
ω	$1.23e-6$	3.07^{***}	$2.10e-6$	2.57^{**}	$1.03e-6$	1.73^{*}
α_1	0.115	15.96^{***}	0.113	9.38^{***}	0.103	9.40^{***}
β_1	0.504	59.49^{***}	0.543	6.73^{***}	0.892	86.09^{***}
β_2	0.379	15.04^{***}	0.336	5.05^{***}		

Table 4: Best window length of the simple method.

	S&P 500	Dow Jones	NASDAQ
Best window	39	39	35
Log likelihood	64332.12	84973.79	33611.08

We also train the simple statistical method on the three index return series. As Table 4 shows, the best window length is 39 trading days for S&P 500, 39 trading days for Dow Jones, and 35 trading days for NASDAQ. Under these settings of window length, we can gain maximum log likelihood of the train samples.

After we train all the models, we use them to forecast volatility day by day on the test set. We calculate the log likelihood of the test sample for each model and compare the values of the log-likelihood function, which are shown in Table 5. We can conclude from Table 5 that, for all the three indices, LSTM (likelihood-based loss) gains the largest log likelihood and can be the best model to forecast volatility. The value of log likelihood when using DNN (likelihood-based loss) is not as good as LSTM (likelihood-based loss), but is larger than the value when using ARMA-GARCH. LSTM performs better with likelihood-based loss function than with MSE loss function, and DNN also performs better with likelihood-based loss function than with MSE loss function. LSTM (MSE loss) gains larger log likelihood than ARMA-GARCH in two cases of index, while DNN (MSE loss) gains smaller log likelihood than ARMA-GARCH for all the three indices. Simple historical statistics is the method that gains smallest log likelihood for all the indices.

To intuitively exhibit the forecasting ability of the deep learning models and econometric model, we use the log-likelihood value that simple method gains as the benchmark and calculate the improvements which the five models can produce upon it. Figure 6 shows the percentage of improvements that LSTM (likelihood-based loss), DNN

Table 5: Log likelihoods of the test sample.

	S&P 500	Dow Jones	NASDAQ
LSTM (likelihood-based loss)	11406.25	15448.84	6074.32
DNN (likelihood-based loss)	11346.34	15392.89	6062.46
LSTM (MSE loss)	11348.36	15329.30	6036.95
DNN (MSE loss)	11275.18	15259.39	5999.94
ARMA-GARCH	11291.16	15383.54	6012.85
Simple	11061.59	15123.93	5887.78

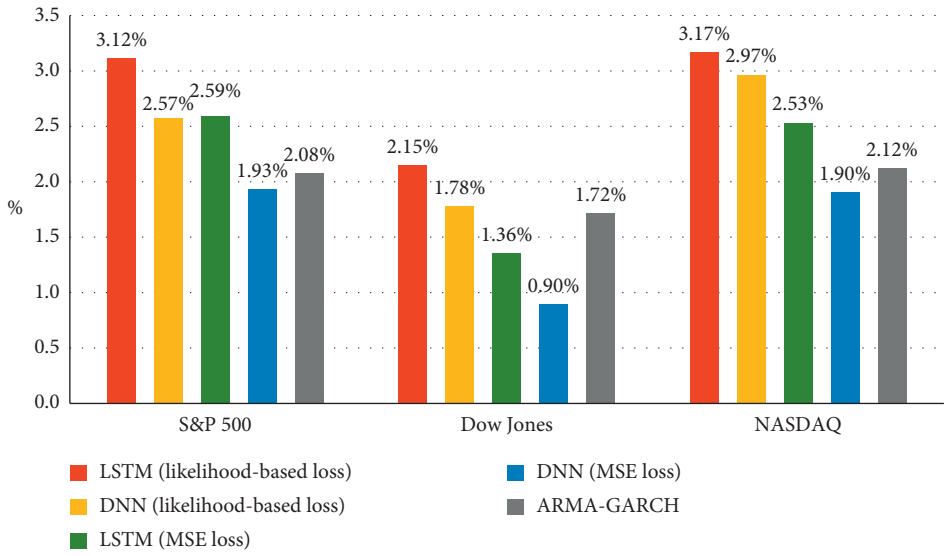


Figure 6: Log-likelihood improvements upon the simple method.

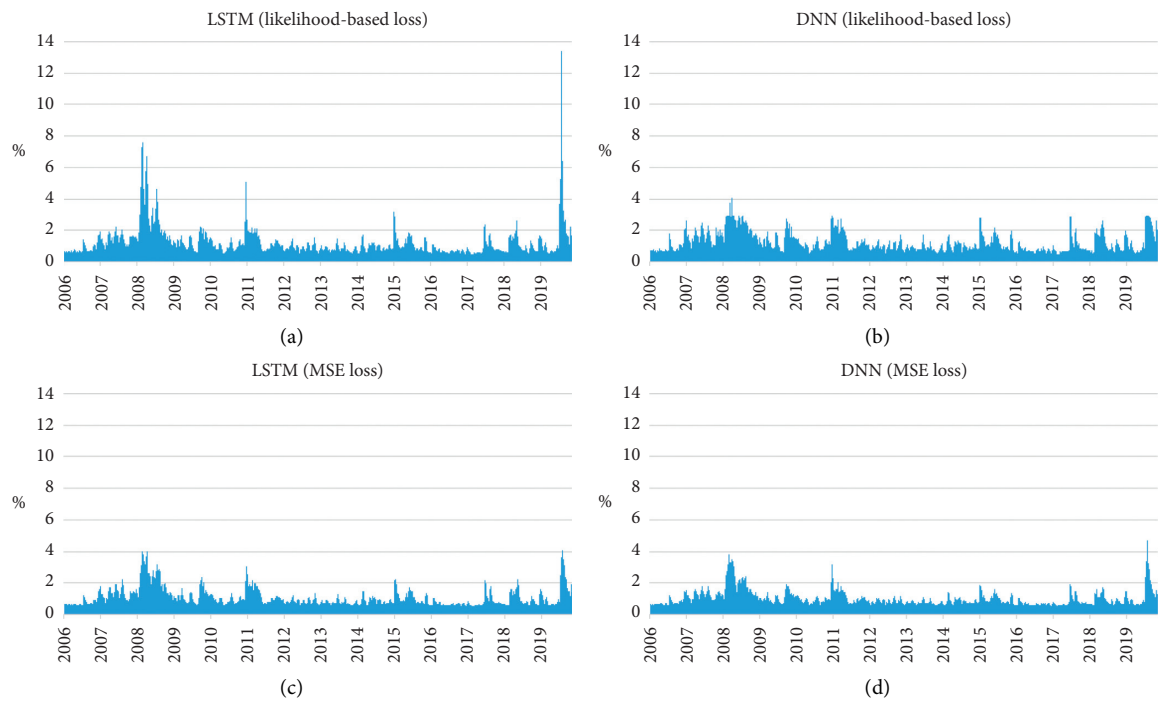


Figure 7: Continued.

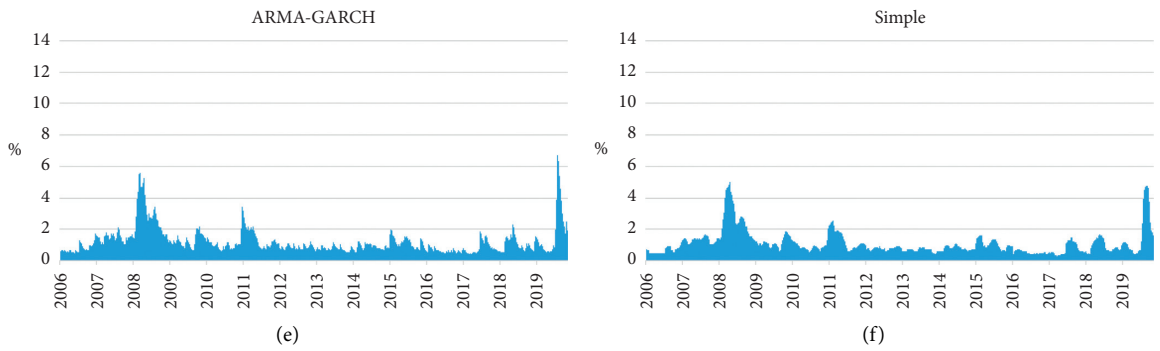


Figure 7: Volatilities of S&P 500 that the models forecast on the test period.

(likelihood-based loss), LSTM (MSE loss), DNN (MSE loss), and ARMA-GARCH make. For S&P 500, the improvements of the five models are 3.12%, 2.57%, 2.59%, 1.93%, and 2.08%. For Dow Jones, the improvements of the five models are 2.15%, 1.78%, 1.36%, 0.90%, and 1.72%. For NASDAQ, the improvements of the five models are 3.17%, 2.97%, 2.53%, 1.90%, and 2.12%.

Figure 7 shows the volatilities that the six models forecast day by day, when we use the trained models on the test sample of S&P 500. They show very similar trends in the long term. The volatilities which LSTM (likelihood-based loss) forecast have extreme peak values. The volatilities which DNN (likelihood-based loss) and LSTM (MSE loss) forecast have relatively low peak values. Although these three models are all deep learning methods, they show different characteristics in this aspect. The volatilities which the simple method forecast change more smoothly than the other five.

To see more details of the volatilities that the models forecast, we plot the six volatility series within the latest one year, which covers from July 1, 2019, to June 30, 2020, and show them together in Figure 8. It is clear in this figure that the forecast of the simple method is smoother than the other five. The forecasts of deep learning models and econometric models have similar values when volatility is relatively low. But when the market is in extreme condition around March 2020, they act very differently. Since the US stock market is under historic shock during this period of time, LSTM (likelihood-based loss) seems to capture the property better.

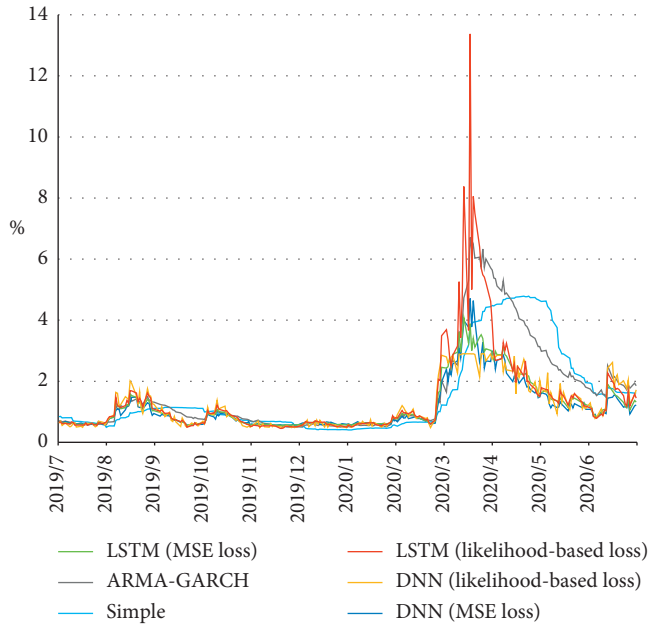


Figure 8: Volatilities of S&P 500 that the models forecast within the latest one year.

3.2. Sensitivity Analysis. In order to test the robustness of the forecasting result, we further use two different train-test sample splits and make the same kind of forecasting. The size of the test sample is set to be 10% and 20% separately, and we make the validation set of the deep learning models always have the same size as the test set. We can check whether the deep learning models with likelihood-based loss function still gain good performance among all the methods.

Table 6 shows the length of the train set and test set in trading days, under different train-test sample splits. When the test set is 10% of the full sample, we train the models in a relatively longer period and test them in a shorter range. When the test set is 20% of the full sample, we train the models in a relatively shorter period and test them in a longer range.

When the test sample size is set to be 10%, we train the ARMA(m, n)-GARCH(p, q) models and choose the best one with minimum BIC. As Table 7 shows, for S&P 500, it is ARMA(1, 1)-GARCH(1, 2), for Dow Jones, it is ARMA(1, 1)-GARCH(1, 2), and for NASDAQ, it is ARMA(1, 2)-GARCH(1, 1). The selections of m, n, p, and q are totally the same as in the previous section.

Then we also train the simple statistical method on the three index return series. As Table 8 shows, the best window length is 39 trading days for S&P 500, 39 trading days for Dow Jones, and 48 trading days for NASDAQ. The best window length only changes for NASDAQ.

After we train all the models, we can use them to forecast volatility day by day on the test set, which is set to be 10% of the

Table 6: Length of the train set and test set in trading days under different train-test sample splits.

	S&P 500		Dow Jones		NASDAQ	
	Train	Test	Train	Test	Train	Test
Test 10%	20917	2322	27987	3108	11212	1244
Test 15%	19755	3484	26433	4662	10590	1866
Test 20%	18594	4645	24878	6217	9967	2489

Table 7: ARMA-GARCH model with minimum BIC when the test set is 10%.

	S&P 500	Dow Jones	NASDAQ
Best model	ARMA(1, 1)-GARCH(1, 2)	ARMA(1, 1)-GARCH(1, 2)	ARMA(1, 2)-GARCH(1, 1)

Table 8: Best window length to maximize the log likelihood when the test set is 10%.

	S&P 500	Dow Jones	NASDAQ
Best window	39	39	48

Table 9: Log likelihoods of the test sample when the test set is 10%.

	S&P 500	Dow Jones	NASDAQ
LSTM (likelihood-based loss)	7901.28	10255.62	3973.68
DNN (likelihood-based loss)	7888.22	10198.73	3963.67
LSTM (MSE loss)	7873.73	10156.86	3942.13
DNN (MSE loss)	7828.06	10089.74	3914.47
ARMA-GARCH	7812.65	10187.24	3923.81
Simple	7631.45	9958.84	3802.13

full sample. We calculate the log likelihood of the test sample for each model and compare the values of the log-likelihood function, which are shown in Table 9. For all indices, LSTM (likelihood-based loss) still gains the best performance among all the methods. DNN (likelihood-based loss) is always ranked second. LSTM performs better with likelihood-based loss function than with MSE loss function, and DNN also performs better with likelihood-based loss function than with MSE loss function. LSTM (MSE loss) gains larger log likelihood than ARMA-GARCH in two cases of index, while DNN (MSE loss) gains larger log likelihood than ARMA-GARCH just for S&P 500. The simple method always gains smallest log likelihood for all the three indices.

To intuitively exhibit the forecasting ability of the deep learning models and econometric model, we also calculate the percentage of improvements that LSTM (likelihood-based loss), DNN (likelihood-based loss), LSTM (MSE loss), DNN (MSE loss), and ARMA-GARCH make upon the simple method and show the result in Figure 9. For S&P 500, the improvements of the five models are 3.54%, 3.36%, 3.17%, 2.58%, and 2.37%. For Dow Jones, the improvements of the five models are 2.98%, 2.41%, 1.99%, 1.31%, and 2.29%. For NASDAQ, the improvements of the five models are 4.51%, 4.25%, 3.68%, 2.95%, and 3.20%.

When the test sample size is set to be 20%, we train the ARMA(m, n)-GARCH(p, q) models and choose the best one with minimum BIC. As Table 10 shows, for S&P 500, it is ARMA(1, 1)-GARCH(1, 2), for Dow Jones, it is ARMA(1, 1)-GARCH(1, 2), and for NASDAQ, it is ARMA(1, 2)-GARCH(1, 2). The unique change also comes from NASDAQ. It may be because the sample size of NASDAQ is the smallest among the three indices.

We also train the simple statistical method and show the best window length for each index in Table 11. The best length is 39 trading days for S&P 500, 39 trading days for Dow Jones, and 35 trading days for NASDAQ. The best window lengths are the same as in the previous section.

After we train all the models, we can use them to forecast volatility day by day on the test set, which is set to be 20% of the full sample. We calculate the log likelihood of the test sample for each model and compare the values of the log-likelihood function, which are shown in Table 12. For all the three indices, LSTM (likelihood-based loss) performs better than DNN (likelihood-based loss), DNN (likelihood-based loss) performs better than LSTM (MSE loss), and LSTM (MSE loss) performs better than DNN (MSE loss). Deep learning models with MSE loss function gain relatively weak forecasting ability, since they cannot beat ARMA-GARCH in most cases. The simple method still makes smallest log likelihood for all the three indices.

We also calculate the percentage of improvements that LSTM (likelihood-based loss), DNN (likelihood-based loss), LSTM (MSE loss), DNN (MSE loss), and ARMA-GARCH make upon the simple method and show the result in

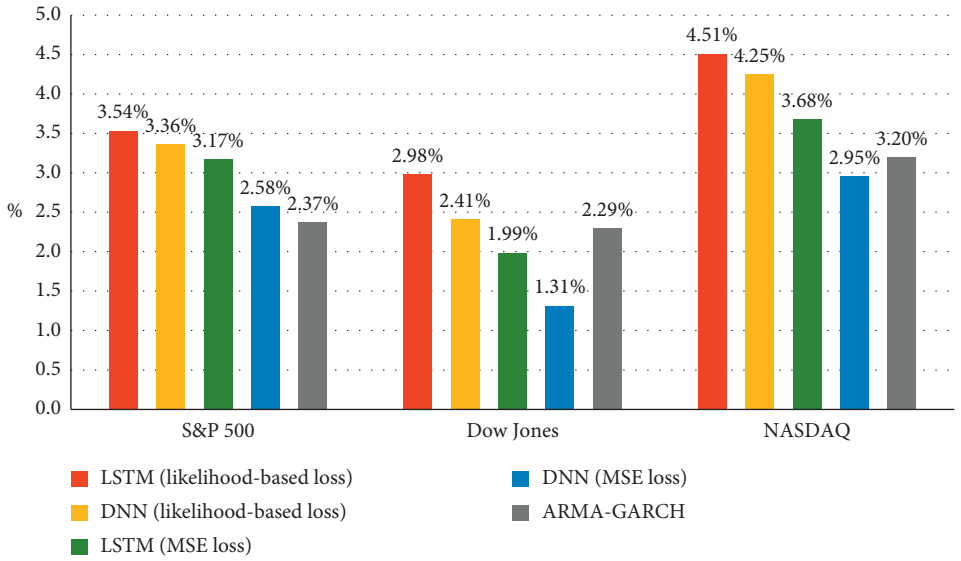


Figure 9: Log-likelihood improvements upon the simple method when the test set is 10%.

Table 10: ARMA-GARCH model with minimum BIC when the test set is 20%.

	S&P 500	Dow Jones	NASDAQ
Best model	ARMA(1, 1)-GARCH(1, 2)	ARMA(1, 1)-GARCH (1, 2)	ARMA(1, 2)-GARCH(1, 2)

Table 11: Best window length to maximize the log likelihood when the test set is 20%.

	S&P 500	Dow Jones	NASDAQ
Best window	39	39	35

Table 12: Log likelihoods of the test sample when the test set is 20%.

	S&P 500	Dow Jones	NASDAQ
LSTM (likelihood-based loss)	15197.71	20290.54	8001.75
DNN (likelihood-based loss)	15193.05	20236.08	7969.07
LSTM (MSE loss)	15140.80	20174.46	7913.75
DNN (MSE loss)	15087.88	20114.91	7856.39
ARMA-GARCH	15124.33	20203.73	7929.03
Simple	14883.33	19902.60	7787.29

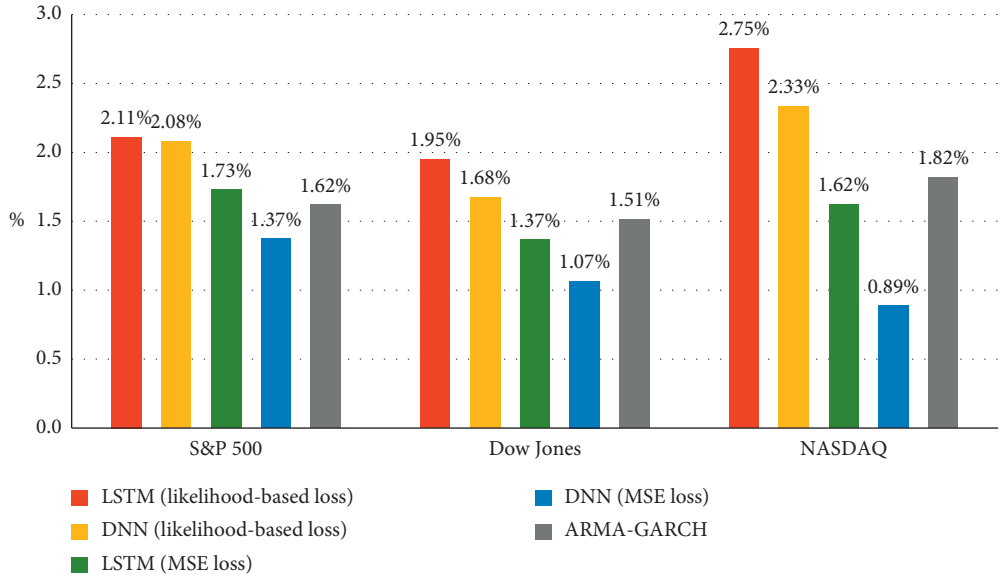


Figure 10: Log-likelihood improvements upon the simple method when the test set is 20%.

Figure 10. For S&P 500, the improvements of the five models are 2.11%, 2.08%, 1.73%, 1.37%, and 1.62%. For Dow Jones, the improvements of the five models are 1.95%, 1.68%, 1.37%, 1.07%, and 1.51%. For NASDAQ, the improvements of the five models are 2.75%, 2.33%, 1.62%, 0.89%, and 1.82%.

The results in this section show that the forecasting abilities of our deep learning models with likelihood-based loss function are robust when we use different sample split settings. Considering the samples of three indices cover different periods, we can further state that their forecasting abilities are not sensitive to different choices of sample periods. The deep learning models with likelihood-based loss function are robustly better in forecasting volatility than the other models, and LSTM (likelihood-based loss) is always the best one among all the methods.

Considering the results when the test set is 10%, 15%, and 20% of the full sample, it is clear that the deep learning models with MSE loss function are not as good as the deep learning models with likelihood-based loss function. The main reason is the inconsistency that we have discussed earlier. But LSTM (MSE loss) still can beat ARMA-GARCH in more than half of the cases, even though it suffers from this disadvantage. DNN (MSE loss) is the weaker one, and it cannot beat ARMA-GARCH in most cases.

4. Conclusions

In this paper, we use deep learning models, an econometric model, and a simple statistical method to forecast the volatility of three US stock indices. Different from related research studies, we further introduce a likelihood-based loss function to train the deep learning models and test all the methods by the likelihood of the test sample. By doing so, we can incorporate fewer errors into the process of volatility forecasting when using deep learning models. At the same time, we can make a fairer comparison among the models we study.

The results of the empirical study show that our deep learning models with likelihood-based loss function forecast volatility more precisely than the econometric model, and LSTM (likelihood-based loss) is the better one in these two deep learning models. The volatility series forecasted by the six models show very similar trends in the long term. LSTM (likelihood-based loss) seems to capture the property better under the extreme condition of the US stock market, around March 2020.

Then we change the setting of the train-test sample split and do the sensitivity analysis. We can conclude that the deep learning models with likelihood-based loss function are robustly better in forecasting volatility than the other models, and LSTM (likelihood-based loss) is always the best. LSTM (MSE loss) can beat ARMA-GARCH in more than half of the cases, even though it suffers from the disadvantage of inconsistency. DNN (MSE loss) is weaker and it cannot beat ARMA-GARCH in most cases. At the same time, all the deep learning models and the econometric model gain positive improvement upon the simple method.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon reasonable request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

References

[1] H. Markowitz, "Portfolio selection," *The Journal of Finance*, vol. 7, no. 1, pp. 77–91, 1952.

[2] F. Black and M. Scholes, "The pricing of options and corporate liabilities," *Journal of Political Economy*, vol. 81, no. 3, pp. 637–654, 1973.

[3] R. F. Engle, "Autoregressive conditional heteroscedasticity with estimates of the variance of United Kingdom inflation," *Econometrica*, vol. 50, no. 4, pp. 987–1007, 1982.

[4] T. Bollerslev, "Generalized autoregressive conditional heteroskedasticity," *Journal of Econometrics*, vol. 31, no. 3, pp. 307–327, 1986.

[5] D. B. Nelson, "Conditional heteroskedasticity in asset returns: a new approach," *Econometrica*, vol. 59, no. 2, pp. 347–370, 1991.

[6] F. Corsi, "A simple approximate long-memory model of realized volatility," *Journal of Financial Econometrics*, vol. 7, no. 2, pp. 174–196, 2009.

[7] M. Bildirici and Ö. Ö. Ersin, "Improving forecasts of GARCH family models with the artificial neural networks: an application to the daily returns in Istanbul Stock Exchange," *Expert Systems with Applications*, vol. 36, no. 4, pp. 7355–7362, 2009.

[8] E. Hajizadeh, A. Seifi, M. H. Fazel Zarandi, and I. B. Turksen, "A hybrid modeling approach for forecasting the volatility of S&P 500 index return," *Expert Systems with Applications*, vol. 39, no. 1, pp. 431–436, 2012.

[9] W. Kristjanpoller, A. Fadic, and M. C. Minutolo, "Volatility forecast using hybrid Neural Network models," *Expert Systems with Applications*, vol. 41, no. 5, pp. 2437–2442, 2014.

[10] W. Kristjanpoller and E. Hernndez, "Volatility of main metals forecasted by a hybrid ANN-GARCH model with regressors," *Expert Systems with Applications*, vol. 84, pp. 290–300, 2017.

[11] W. Kristjanpoller and M. C. Minutolo, "Gold price volatility: a forecasting approach using the Artificial Neural Network-GARCH model," *Expert Systems with Applications*, vol. 42, no. 20, pp. 7245–7251, 2015.

[12] W. Kristjanpoller and M. C. Minutolo, "Forecasting volatility of oil price using an Artificial Neural Network-GARCH model," *Expert Systems with Applications*, vol. 65, pp. 233–241, 2016.

[13] W. Kristjanpoller and M. C. Minutolo, "A hybrid volatility forecasting framework integrating GARCH, Artificial Neural Network, technical analysis and principal components

analysis," *Expert Systems with Applications*, vol. 109, pp. 1–11, 2018.

[14] T. H. Roh, "Forecasting the volatility of stock price index," *Expert Systems with Applications*, vol. 33, no. 4, pp. 916–922, 2007.

[15] A. Baffour, J. Feng, and E. Taylor, "A hybrid artificial neural network-GJR modeling approach to forecasting currency exchange rate volatility," *Neurocomputing*, vol. 365, pp. 285–301, 2019.

[16] S. Lahmiri, "Modeling and predicting historical volatility in exchange rate markets," *Physica A: Statistical Mechanics and Its Applications*, vol. 471, pp. 387–395, 2017.

[17] Y. Peng, P. H. M. Albuquerque, J. M. Camboim de Sá, A. J. A. Padula, and M. R. Montenegro, "The best of two worlds: forecasting high frequency volatility for cryptocurrencies and traditional currencies with support vector regression," *Expert Systems with Applications*, vol. 97, pp. 177–192, 2018.

[18] E. Ramos-Pérez, P. J. Alonso-González, and J. J. Núñez-Velázquez, "Forecasting volatility with a stacked model based on a hybridized Artificial Neural Network," *Expert Systems with Applications*, vol. 129, pp. 1–9, 2019.

[19] H. Y. Kim and C. H. Won, "Forecasting the volatility of stock price index: a hybrid model integrating LSTM with multiple GARCH-type models," *Expert Systems with Applications*, vol. 103, pp. 25–37, 2018.

[20] Y. Liu, "Novel volatility forecasting using deep learning-long short term memory recurrent neural networks," *Expert Systems with Applications*, vol. 132, pp. 99–109, 2019.

[21] Y. Hu, J. Ni, and L. Wen, "A hybrid deep learning approach by integrating LSTM-ANN networks with GARCH model for copper price volatility prediction," *Physica A: Statistical Mechanics and Its Applications*, vol. 557, pp. 1–14, 2020.

[22] F. Z. Xing, E. Cambria, and Y. Zhang, "Sentiment-aware volatility forecasting," *Knowledge-Based Systems*, vol. 176, pp. 68–76, 2019.

[23] A. Vidal and W. Kristjanpoller, "Gold volatility prediction using a CNN-LSTM approach," *Expert Systems with Applications*, vol. 157, pp. 1–9, 2020.

[24] S. Yu and Z. Li, "Forecasting stock price index volatility with LSTM deep neural network," in *Recent Developments in Data Science and Business Analytics*, M. Tavana et al., Ed., pp. 265–272, Springer, 2018.

[25] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.