Volatility Prediction Using Multivariate GRU

Shuailin He, Ruixuan Tang, Hao Wang
Carnegie Mellon University
{shuailih, ruixuant, haowang3}@andrew.cmu.edu

Abstract

Options traders rely heavily on their estimates of future volatility to guide their decisions. A successful predictive model has proven to be highly profitable. The traditional approach to volatility prediction involves an autoregressive model such as GARCH. Nevertheless, such an approach's accuracy is limited due to the inflexibility of the model to describe complex patterns in the financial market. In this research, we applied various machine learning algorithms such as XGBoost and RNN to forecast volatility. Experiment results based on historical stock data of S&P 500, Dow Jones, and Nasdaq have showcased significant improvements in model performance from using modern machine learning than the traditional econometrics approach.

1 Introduction

Volatility is often thought of as how much a stock tends to move around for a given period of time. This concept could be applied to quantify the tendency of the market. Equity derivatives such as options pricing using the Black–Scholes model rely on the traders' intuition of future volatility on underlying stocks. In modern trading firms such as Citadel Securities and Optiver, there has been a growing trend of using advanced modeling techniques to build predictive models to facilitate traders in their decision process. Nevertheless, such models are proprietary and not shared with the public.

Traditionally, volatility prediction centers on econometric models such as ARCH and GARCH. Nevertheless, more recent studies have showcased the potential of machine learning techniques in replacing traditional modeling techniques (1). Advantages include its ability to process large datasets in a shorter time period with higher accuracy.

The objective of our research is to identify the best method to predict average volatility for the next 10 days based on historical data. We plan to explore volatility prediction with deep learning algorithms such as XGBoost and LSTM in hope of achieving higher accuracy than econometric models. With the predictive model constructed, we hope to facilitate retail investors in their endeavors. The experiment results demonstrate that deep learning models have better performance than traditional econometric models.

2 Related Works

2.1 Definition of Historical Volatility

Volatility is often thought of as how much a stock tends to move around for a given period of time. Historical volatility is often used as a proxy for volatility. It is defined as a stock's volatility for a past period of time and is often denoted as the standard deviation of the stock price over time (1). Estimation formulas for historical volatility is shown below:

$$V_H = \sqrt{\frac{1}{N}\sum_{t=\tau_1}^{\tau_2}(r_t - \mu_{\tau_1,\tau_2})^2} \tag{1}$$

where $r_t = log(P_t/P_{t-1})$, $N = \tau_2 - \tau_1$, $\mu_{\tau_1,\tau_2} = \frac{1}{N}\sum_{t=\tau_1}^{\tau_2} r_t$.

2.2 Literature Review

Traditional volatility prediction depends heavily on the ARCH family of models. The ARCH model was first proposed by economist Engle (1982). (10) This pioneer algorithm models the error variance with an autoregressive structure. Then Bollerslev (1986) (9) invented the GARCH model which incorporates the conditional variance in the autoregressive structure, and thus made variance an autoregressive moving average process. Multiple variations of the GARCH model were later invented to address some potential issues with ARCH/GARCH models. Nelson (1991) (11) proposed the EGARCH model that uses logarithm to relax the positive constraints of the parameters in GARCH and captures the asymmetry and persistence of volatility shocks. The GJR-GARCH model was introduced by Glosten (1993) (13) to capture the asymmetry using an indicator term. Other ARCH family models were also developed to address multiple issues. Poon and Granger (2003) (12) examined multiple papers and made the conclusion that asymmetric variations of the GARCH model normally outperform traditional GARCH models. Schmidt (2021) (14) compared multiple ARCH family models' performance during COVID-19 and found asymmetric variations of the GARCH model outperform traditional models during the crisis.

Modern volatility prediction relies on various kinds of machine learning techniques. Various types of Neural Networks are applied to forecasting volatility and hybrids of different models are also invented. Roh (2007) (15) combined Neural Networks with variations of GARCH models and found that such a hybrid of NN and time-series model could enhance the predictive power. As different Neural Networks are developed and applied, Recurrent Neural Networks are favored by researchers to predict volatility, due to their natural compatibility with time-series financial data. Xing (2019) (16) applied RNN to analyzing social media texts and forecasting volatility. Zhang (2020) (18) built the LSTM+Realized GARCH+RVaR model and achieved better predictions than individual models. In prior works, researchers tended to use the Long Short-Term Memory model, which was first proposed by Sepp Hochreiter in 1997, to overcome the issue of exploding/vanishing gradients in RNN (3). This trend was carried from research done in NLP where data are abundant (see Librispeech dataset). However, in the field of finance, datasets were generated from real-life trading. Data of daily frequency, which are of interest to our paper, are scarce. Most stocks only extend back to year 2000, with around 5000 days worth of datapoints. Therefore, to adapt to this challenge, we decide to use GRU, a new type of RNN proposed by Dzmitry Bahdanau et al. in 2014, which also have the gate mechanism that handles exploding/vanishing gradients but requires less data to train due to fewer number of weights and parameters to update (6).

3 Data

Yahoo Finance is an open-sourced finance time-series data source that provides quotes on a wide range of assets from equities to crypto. We can retrieve historical stock data using Yahoo finance API. By inputting the stock's ticker, Yahoo finance API returns a pandas data frame consisting of Open, High, Low, Close, Adj Close, and Volume of the stock for a pre-specified period.

For daily frequency, Yahoo Finance can retrieve data as far back as to 1970. However, for high-frequency data, such as 1-minute level or tick level, is only saved for the past 7 days. We are not able to locate any open-source tick-level data and have to suffice with daily-level data. No bias is found with the data source.

For this project, we first proceeded with S&P 500 index data from the date 1990.1.2 to the date 2022.12.12. Then, to train the deep learning models, we splited the data in a ratio of train:validation:test = 50:20:30, each portion being used for training, validation, and testing. Noted, our split is sequential, meaning the initial 11 years of stock data (50%) is used for training. The purpose is to test if our model can generate robust predictions generalizable to 10 years after. After the model is trained, we proceeded with a robustness test to see if the model's architecture can be generalized to predict securities other than S&P 500. This will give us a more comprehensive view of the model's performance.

4 Method

4.1 Feature Engineering

The raw data is a set of `Open, Close, Adj Close, High, Low, Volume` data of S&P 500(or VIX). We processed the raw data on this dataset to transfer it into usable features, which would be fed into various kinds of models.

- `Log Return` captures the daily price fluctuation. It is calculated by taking log on the ratio of Adjusted Close prices of two consecutive days: $r_t = \log\left(P_{Adj\_Close,t}/P_{Adj\_Close,t-1}\right)$
- `10-day-vol` captures the volatility in the past 10 days. It is approximated by the standard deviation of the `Log Returns` in the past 10 days: $vol_t = \sqrt{\frac{1}{10}\sum_{\tau=t-9}^{t}(r_\tau - \mu_{t-9,t})^2}$
- `30-day-vol` captures the volatility in the past 10 days. It is approximated by the standard deviation of the `Log Returns` in the past 30 days: $vol_t = \sqrt{\frac{1}{30}\sum_{\tau=t-29}^{t}(r_\tau - \mu_{t-29,t})^2}$
- `Log Volume Chg` captures the daily trading volume change on S&P. It is calculated by taking log on the ratio of `Volumes` of two consecutive days: $vlmchg_t = \log\left(Volume_t/Volume_{t-1}\right)$
- `Log Range` captures the intraday price fluctuation. It is calculated by taking log on the ratio of High and Low prices of a trading date: $range_t = \log\left(High_t/Low_t\right)$
- `VIX` is the `Adj Close` price of the VIX index. The VIX index is an indicator of markets' expectation on the volatility of the coming 30 days.

Note that these features are all continuous scalar values. For a trading date, we would have a 6-by-1 feature vector. The correlation heatmap is below. Notice that the correlation between `10-day-vol`, `30-day-vol`, `VIX`, `Log Range` are higher than other correlations as these features are directly associated to future 10-day average volatility.

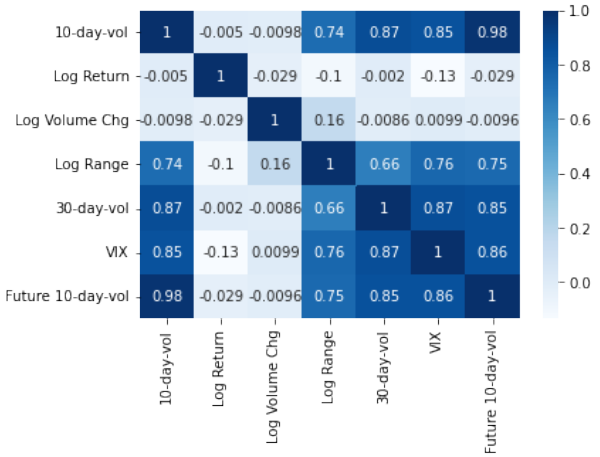


Figure 1: Feature Correlation Heatmap

4.2 Baseline Econometric Method: GARCH

In the GARCH(p, q) model, volatility is modeled by the formulas

$$X_t = e_t\sigma_t \tag{2}$$

$$\sigma_t^2 = \omega + \alpha_1 X_{t-1}^2 + ... + \alpha_p X_{t-p}^2 + \beta_1\sigma_{t-1}^2 + ... + \beta_q\sigma_{t-q}^2 \tag{3}$$

where e_t (the 'innovations') are i.i.d. with expectation 0 and variance 1 and are assumed independent from σ_k for all k $\leq$ t. Our GARCH model would apply in the `Log Return` time series and predict volatility within the next 10-day horizon. Model parameters p and q (numbers of lagging log returns

and lagging variance to incorporate) are selected by picking the pair in [10] x [10] that grants the least Bayesian information criterion (BIC). Here, $BIC = k\log(n) - 2\log(L)$ where L is the maximized value of the likelihood function of the model; n is the number of data points, and k is the number of free parameters to be estimated.

4.3 Boosting Method: XGBoost

XGBoost, the Extreme Gradient method, is based on boosting tree models. Unlike other boosting tree models, XGBoost applies a second-order Taylor Expansion to its loss function, which implements distributed training by automatically using the multi-threading of the GPU for parallel computing. The loss function for XGBoost is defined as follows:

$$l(y, \hat{y}) = l(y, a + \Delta\hat{y}) \tag{4}$$

$$l(y, \hat{y}) = l(a) + l'(a)(\hat{y} - a) + \frac{1}{2}l''(a)(\hat{y} - a)^2 \tag{5}$$

Suppose, $a = \hat{y}_{t-1}$ and $\Delta\hat{y} = R_t(X)$, we have

$$l(y, \hat{y}_{t-1} + R_t(X)) = l(y, \hat{y}_{t-1}) + l'(\hat{y}_{t-1})R_t(X) + \frac{1}{2}l''(\hat{y}_{t-1})(R_t(X))^2 \tag{6}$$

The XGBoost model takes in the 6 by 1 feature vectors (as described in the `Feature Engineering` session) of all the trading dates we have in the training dataset as well as the corresponding labels (10-day-vol within the next 10-day window) of these dates. It outputs the predicted 10-day-vol. The hyper-parameter space has dimension 5. It consists of the following parameters:

- `n estimators`: the number of trees in the ensemble. Value range: [400, 800, 1200, 1600]
- `max depth`: the maximum depth of each tree. Value range: [3, 6, 9]
- `eta`: the learning rate used to weight each model. Value range: [0.05, 0.1, 0.2]
- `subsample`: the number of samples (rows) used in each tree. Value range: [0.3, 0.6, 0.9]
- `colsample bytree`: the number of features (columns) used in each tree. Value range: [0.3, 0.6, 0.9]

We pick the best set of hyper-parameters by performing a 10-split K-Fold validation against the training dataset. The cross-validation-score metric is the negative mean squared error. We should pick the hyperparameter set with the highest cross-validation score.

4.4 Recurrent Neural Network : LSTM & GRU

Our experiment on Recurrent Neural Networks follows a sequential process. We started off with a baseline LSTM model and experiment with what improvements we can make to the training method, model architecture, input features, and hyperparameters.

Baseline LSTM model Our baseline model is based on the baseline provided in Jason C. Sullivan's paper (8). Our LSTM_base takes in only the past 20 days' 10-day average volatility and its objective is to predict average volatility for the next 10 days. The baseline model consists of one LSTM layer with 50 hidden units, followed by a 1-unit output layer on the final hidden output. We use SGD as our optimizer.

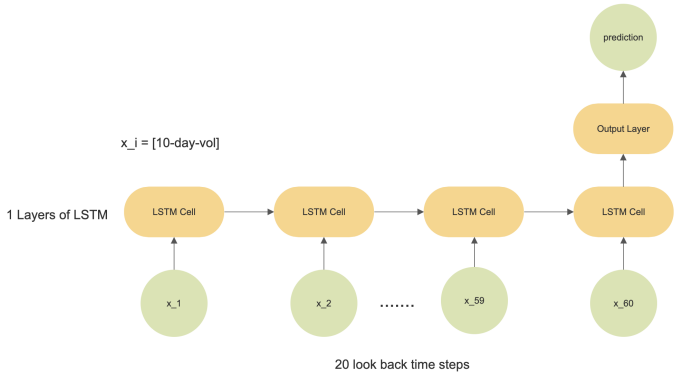


Figure 2: LSTM_base Architecture

Improvement on Training Method According to the experiment results, our baseline model did not perform well. Thus, we applied other training methods to improve its performance. For example, since SGD may optimize the model with the incorrect gradient, we selected AdamW optimizer. AdamW, Adam algorithm with weight decay, unlike Adam, the weight decay is applied only to control the parameter-wise step size, as shown below,

$$x_t = x_{t-1} - \eta_t(\alpha \hat{m}_t / \sqrt{\hat{v}_t + \epsilon} + w x_{t-1}) \tag{7}$$

It means that the weight decay would not stop updating at the averages. Compared AdamW to SGD and general Adam, it can extremely save computation power on gradient optimization and model generalization. See Appendix B for specific update rule. We applied a learning rate scheduler, ReduceLROnPlateau with factor 0.75 and patience 2 on validation loss, that a decrease in the learning rate every time the model is stuck which can output hyperparameter pair that makes the model with the best performance.

Improvement on Input Features Our baseline model uses 10-day average volatility for prediction. However, this data feature may be singular, which may impede the model's performance. Thus, we included additional features, including 10-day-vol, Log Return, Log Volume Chg, Log Range, 30-day-vol, and VIX. Applying the new model LSTM_mult_feats_base, it can reduce MSE from 0.222 to 0.077 verses LSTM_3layers. We conducted another experiment on whether including only highly correlated features as predictive variables may increase the model's performance, thus removing Log Return and Log Volume Chg from our data. We are able to see a slight decrement in MSE from 0.077 to 0.073, especially in more long-term predictions. Increasing the length of data input from 20 days to 60 days also proved to be an improvement. However, increasing the number of timesteps further turns out to deteriorate the model's performance from the empirical result.

Improvement on Model Architecture We applied four changes to the model's architecture: a CNN layer for feature extraction, an increment to LSTM layers, replacing LSTM with GRU, and additional activation before the output layer. Our dataset has 6 features after applying feature engineering. More latent features in the data can be extracted with a convolution neural network. We added a 1-D CNN layer with a kernel size of 3 and a padding size of 1, extending the feature size from 6 to 64. But, this modification seems not to increase the model's performance obviously. Because our baseline model consists of one layer of LSTM, which may not be able to capture the long-term trend. Thus, we increased LSTM layers to three with a dropout of 0.2 for regularization. In addition, we applied a ReLU activation layer for better prediction. These two changes together are proven to be worthwhile as the MSE for our new model LSTM_3layers is decreased from 0.318 to 0.222 verses LSTM_base. Replacing LSTM with GRU proved to be yet another useful technique. Since we don't have access to tick-level data, we only have roughly 5000 data points to train our model. GRU, with fewer parameters to train, while providing the same memory retention mechanism as LSTM. This modification brings about a quite significant reduction in MSE from 0.077 to 0.034 verses LSTM_mult_feats_base.

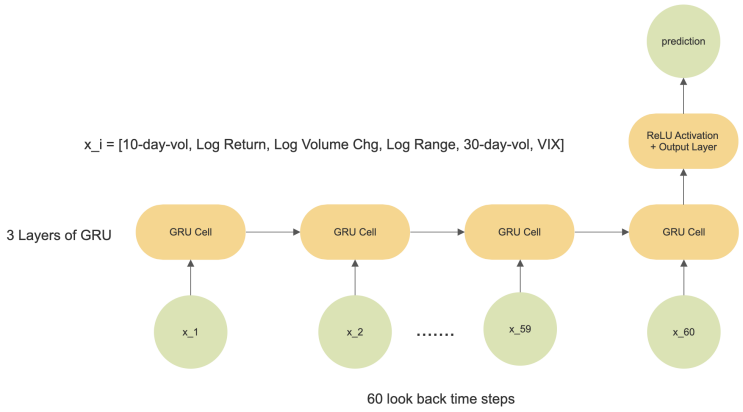


Figure 3: GRU_mult_feats_base Architecture

5 Results

5.1 Metrics

Since we are predicting financial time series data, mean square error should suffice as our objective function. In addition, to compare predictions of data of different scale, we introduced another metric called Mean absolute percentage error (MAPE).

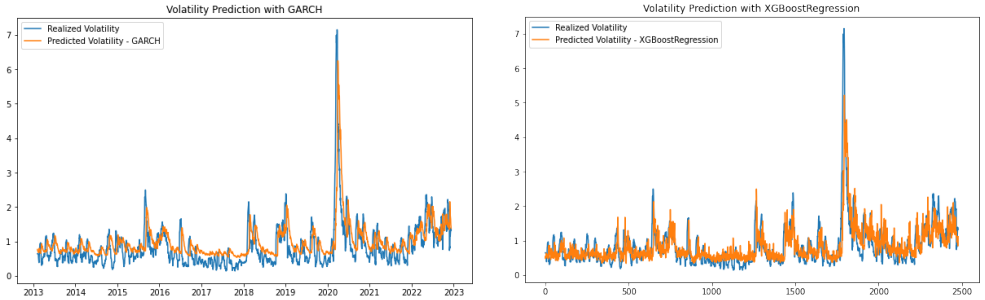
$$MSE = \frac{1}{m} * \sum_{t=1}^{m} (Vol_t - \hat{Vol_t})^2 \tag{8}$$

$$MAPE = \frac{1}{m} * \sum_{t=1}^{m} \frac{|Vol_t - \hat{Vol_t}|}{Vol_t} \tag{9}$$

where m is the number of periods, Vol_t is the observed volatility of period t, $\hat{Vol_t}$ is the predicted volatility of period t.

5.2 GARCH & XGBoost Result

GARCH model makes a decent prediction on the volatility due to the autoregressive nature of the model. The MSE of the prediction made by GARCH is 0.2780. We will use this value as a baseline to determine the performance of other machine learning models.



(a) GARCH (b) XGBoost

Figure 4: Predicted Result for Test Data From 2013 to 2022

Compared to the base GARCH model, XGBoost show a slight improvement. The MSE of the prediction made by XGBoost is 0.228. This fits our expectation of the boosting model. Different from the GARCH model which only takes in `Log Returns`, the XGBoost model is trained within higher dimensions. It also has a more complicated structure, which consists of decision trees built recursively for learning similarities between residuals.

5.3 Recurrent Neural Network Result

Table 1 contains the experiment result from our training.

Model	Eval MSE	Test MSE	Weighted Eval + Test MSE
GARCH	/	/	0.278
XGBoost	/	/	0.228
LSTM_base	0.166	0.419	0.318
LSTM_3layers	0.042	0.342	0.222
LSTM_mult_feats_base	0.056	0.091	0.077
LSTM_mult_feats_high_corr	0.060	0.083	0.074
LSTM_mult_feats_CNN	0.106	0.095	0.100
GRU_mult_feats_base	**0.026**	**0.039**	**0.034**
GRU_mult_feats_high_corr	0.031	0.058	0.047
GRU_mult_feats_lookback_30	0.039	0.072	0.059
GRU_mult_feats_lookback_126	0.261	0.261	0.261

Table 1: Ablation Study

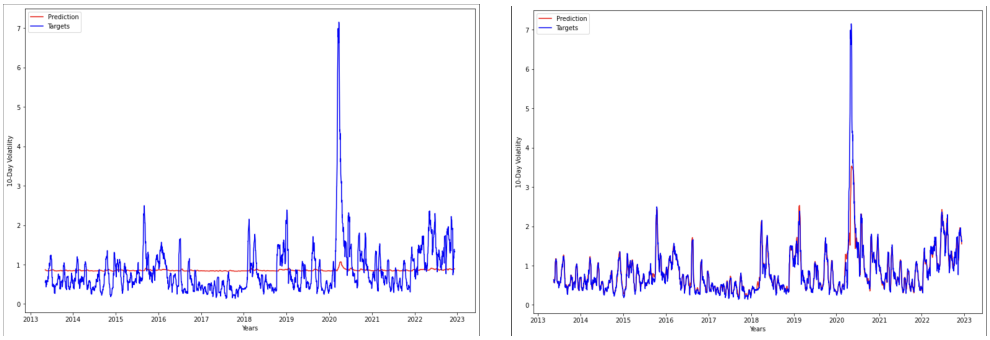
Observing the result from table 1, we can see quite a significant increase in model's performance across different modifications. Our final model, GRU network with multiple input features has an MSE of 0.034, far surpassing the baseline LSTM's performance of 0.318 and GARCH model's performance of 0.27. From 10 experiments, MSE from GRU_mult_feats_base's prediction has achieved a mean of 0.042 and a standard error of 0.00087.

Comparing to the baseline model, we have made changes to input features, model architecture, and training method. Among which, we see a 43% improvements in model accuracy between LSTM_base and LSTM_3layers. This is within expectation since three layers of LSTM provide better memory retention capacity and a better AdamW optimizer helps the model converge to more optimal local minimum. Nevertheless, the most significant improvement is seen from including more features in our dataset. Relative to LSTM_3layers, LSTM_mult_feat_base is able to achieve a reduction in MSE by 65% to 0.077. Through including features such as past VIX index and 30-days volatility, which have a correlation with future 10-days average volatility of more than 0.75, it is within the expectation that our model is able to learn more information and draw better predictions. Nevertheless, little difference in model performance is seen from excluding less correlated features from our dataset. Lastly, to improve model's performance further, we tried replacing LSTM cell with GRU cell due to the limited size of our trainig data. The improvement in model performance exceeded our expectations as we see MSE decreased by a further 56% to 0.034.

Nevertheless, we did face certain drawbacks in model performance when attempting certain modifications. First, we tried to add an additional CNN layer before going into GRU layer. Even after attempting different kernel sizes and hidden dimensions, however, our model performance is not able to improve. A potential explanation is that because of convolution, we disrupted the frequency of input data with predicted data (daily). Therefore, we dropped CNN as a method for feature extraction. Secondly, when tuning hyperparameters, we found the number of look back period have a significant impact on model's performance. We had the assumption that with further lookback periods, the model is able to capture more long term trend. However, experiment result show otherwise, increasing look back period by one fold to 126 days reduces model performance by 6.67 times. Figure 5 provides visual presentation of model predictions with LSTM_base and GRU_mult_feats_base with test data from 2013 to 2022. Significant improvements are seen through changes in model architecture and hyperparameter tuning.

5.4 Robustness Test

Index/Equity	Eval MSE	Test MSE	Weighted Eval + Test MSE
S&P 500	0.026	0.039	0.034
NASDAQ	0.012	0.021	0.017
Dow Jones	0.010	0.088	0.056
Russell 2000	0.085	0.111	0.101
Microsoft	0.021	0.044	0.035



(a) LSTM_base (b) GRU_mult_feats_base

Figure 5: Predicted Result for Test Data From 2013 to 2022

Table 2: Robustness Test

We ran the model against other indexes and stocks. Based on the result shown in table 2, the model is capable of maintaining its current performance across different securities. This further consolidated the validity of our model and confirmed its robustness for financial time series prediction.

5.5 Comparison with Current Literature

Based on a paper published in 2020 on ICPHDS on topics of volatility forecasting, their prediction based on a hybrid model of LSTM and RGARCH is able to predict a MAPE of 0.252. Our multivariate GRU model, on the other hand, is able to beat the model's performance with a MAPE of 0.153.

6 Discussion and Analysis

In the research paper, we experimented with various methods to make predictions on future volatility. We began by using traditional models such as GARCH and more modern architecture such as XGBoost. Both models are able to achieve decent performance, with MSE of 0.278 and 0.221 respectively. Then, we applied recurrent neural networks such as LSTM and GRU to the task due to their ability to retain long-term memories. Through a series of experiments on model architecture, input features, and training methods, we are able to improve the model's performance and reduce MSE to 0.034 with GRU_mult_feats_base model. Beating model results in current literature. The complexity of financial time series data makes it very hard to model. Even with advanced modeling techniques such as GARCH, we are not able to capture all the information necessary to make accurate predictions. Nevertheless, with the introduction of deep neural networks in the field of finance, we are able to build complex networks with tens of thousands of parameters. With the correction selection of architecture and parameter combinations, we are able to draw more accurate predictions based on existing data. Reflecting on our experiment, there are a couple of limitations to our approach. The first is data limitation. With public resources, we can only get access to data of daily frequencies which limited our research topic to predicting average volatility for the next 10 days and undermines the potential of our model. If high-frequency data is available, we may adjust our model to predict volatility for the next 30 minutes, which will be much more substantial in real-life trading scenarios. The second limitation is the black box issue related to the deep neural network. Our model implemented using GRU yields much better performance than the GARCH model. However, due to our inability to explain how RNN truly functions internally beyond a general oversight, machine learning tools are not well-received among investors. Nevertheless, researchers nowadays are working on the explainability of machine learning models to add reliability to their usage in more delicate fields such as finance. Lastly, the model architecture we used is proven to be effective but the models are slightly out-of-date. More modern machine learning techniques such as self-attention with positional embedding may prove more effective when handling a large amount of high-frequency data since it allows concurrent training.

References

[1] Wenbo Ge The Australian National University, et al. “Neural Network–Based Financial Volatility Forecasting: A Systematic Review.” ACM Computing Surveys, 1 Jan. 2023, dl.acm.org/doi/10.1145/3483596.

[2] Jia, Fang, and Boli Yang. “Forecasting Volatility of Stock Index: Deep Learning Model with Likelihood-Based Loss Function.” Complexity, Hindawi, 25 Feb. 2021, www.hindawi.com/journals/complexity/2021/5511802/.

[3] Hochreiter, Sepp & Schmidhuber, Jürgen. (1997). Long Short-term Memory. Neural computation. 9. 1735-80. 10.1162/neco.1997.9.8.1735.

[4] Kang, Eugine. “Long Short-Term Memory (LSTM): Concept.” Medium, Medium, 1 Sept. 2017, medium.com/@kangeugine/long-short-term-memory-lstm-concept-cb3283934359.

[5] Olah, Christopher. “Understanding LSTM Networks.” Understanding LSTM Networks – Colah’s Blog, colah.github.io/posts/2015-08-Understanding-LSTMs/.

[6] Bahdanau, Dzmitry, et al. “Neural Machine Translation by Jointly Learning to Align and Translate.” ArXiv.org, 19 May 2016, arxiv.org/abs/1409.0473.

[7] Lendave, Vijaysinh. “LSTM vs Gru in Recurrent Neural Network: A Comparative Study.” Analytics India Magazine, 14 Dec. 2021, analyticsindiamag.com/lstm-vs-gru-in-recurrent-neural-network-a-comparative-study/.

[8] Sullivan, Jason. Stock Price Volatility Prediction with Long Short Term Memory Neural Networks. cs230.stanford.edu/projects_fall_2019/reports/26254244.pdf.

[9] Generalized Autoregressive Conditional Heteroskedasticity - Duke University. https://public.econ.duke.edu/ boller/Published_Papers/joe_86.pdf.

[10] Engle, R. F. (1982). Autoregressive Conditional Heteroskedasticity with Estimates of Variance of UK Inflation. Econometrica, 50, 987-1008. doi10.2307/1912773 - References - Scientific Research Publishing, https://www.scirp.org/(S(351jmbntvnsjt1aadkposzje))/reference/referencespapers.aspx?referenceid=534502.

[11] Nelson, Daniel. “Conditional Heteroskedasticity in Asset Returns: A New Approach.” Conditional Heteroskedasticity in Asset Returns: A New Approach on JSTOR, https://doi.org/10.2307/2938260.

[12] Poon, Ser-Huang, and Clive W.J. Granger. “Forecasting Volatility in Financial Markets: A Review.” Journal of Economic Literature, https://www.aeaweb.org/articles?id=10.1257

[13] On the Relation between the Expected Value and the Volatility of the ... https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1540-6261.1993.tb05128.x.

[14] Schmidt, Ludwig. “Volatility Forecasting Performance of GARCH Models: A Study on Nordic Indices during COVID-19.” DIVA, 15 June 2021, http://umu.diva-portal.org/smash/record.jsf?pid=diva2

[15] Roh, Tae. Forecasting the Volatility of Stock Price Index. https://doi.org/10.1016/j.eswa.2006.08.001.

[16] Xing, Frank. Sentiment-Aware Volatility Forecasting. https://doi.org/10.1016/j.knosys.2019.03.029.

[17] Y. Zhang, P. Huang, P. Zhou and Y. Wu, "Forecast Volatility based on Realized GARCH and deep LSTM Neural Network," 2020 International Conference on Public Health and Data Science (ICPHDS), 2020, pp. 99-103, doi: 10.1109/ICPHDS51617.2020.00028.

[18] Graetz, Fabio M. “Why Adamw Matters.” Medium, Towards Data Science, 12 Feb. 2020, towardsdatascience.com/why-adamw-matters-736223f31b5d.

A Code and Video Link

The project related presentation video can be watched or downloaded here: Video Link

The code can be viewed or downloaded here: Code Link

B Validation Loss Evolution

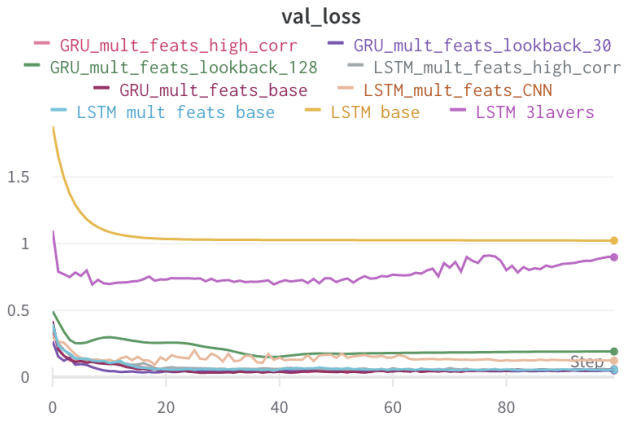


Figure 6: Validation Loss for Tested Models

C AdamW Optimizer

Algorithm 2 Adam with L_2 regularization and Adam with weight decay (AdamW)

given $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, w \in \mathbb{R}$
initialize time step $t \leftarrow 0$, parameter vector $\boldsymbol{x}_{t=0} \in \mathbb{R}^n$, first moment vector $\boldsymbol{m}_{t=0} \leftarrow \boldsymbol{0}$, second moment vector $\boldsymbol{v}_{t=0} \leftarrow \boldsymbol{0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
repeat
  $t \leftarrow t + 1$
  $\nabla f_t(\boldsymbol{x}_{t-1}) \leftarrow \text{SelectBatch}(\boldsymbol{x}_{t-1})$ ▷ select batch and return the corresponding gradient
  $\boldsymbol{g}_t \leftarrow \nabla f_t(\boldsymbol{x}_{t-1}) + w\boldsymbol{x}_{t-1}$
  $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t$ ▷ here and below all operations are element-wise
  $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)\boldsymbol{g}_t^2$
  $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t / (1 - \beta_1^t)$ ▷ β_1 is taken to the power of t
  $\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t / (1 - \beta_2^t)$ ▷ β_2 is taken to the power of t
  $\eta_t \leftarrow \text{SetScheduleMultiplier}(t)$ ▷ can be fixed, decay, or also be used for warm restarts
  $\boldsymbol{x}_t \leftarrow \boldsymbol{x}_{t-1} - \eta_t \left(\alpha \hat{\boldsymbol{m}}_t / (\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon) + w\boldsymbol{x}_{t-1} \right)$
until *stopping criterion is met*
return optimized parameters $\boldsymbol{x}_t$

Figure 7: AdamW Optimizer