ERASMUS UNIVERSITY ROTTERDAM

Erasmus School of Economics

Bachelor Thesis IBEB

Long Short-Term Memory: Can Artificial Neural Networks beat Econometric Models?

Name Student: Damla Ogus

ID: 450623do

Supervisor: Rogier Quaedvlieg

Final Version Submitted on 29/07/2019

Abstract

Long Short-Term Memory (LSTM) is a modern artificial neural network architecture for sequence learning which is capable of handling long-term dependencies and detecting hidden patterns in the data. Even though it has not been widely employed in the field of finance except for some very recent attempts, it is essentially suitable for financial time series predictions.

In this paper, LSTM is compared to a set of traditional econometric models to predict out-of-sample realized variances of the S&P 500 from 2010 until 2019. The results show that the performance of the LSTM models are very sensitive to the setting of their hyperparameters. Overall, it was obtained that ARMA and GJR-GARCH gave the best estimates in terms of mean squared error (MSE) in the two separate forecasting intervals belonging to 2010-2019 and that LSTM underperformed all models in both periods. Different combinations of hyperparameters should be experimented in order for LSTM to compete with the econometric models.

Keywords: forecasting volatility, artificial neural networks, recurrent neural networks, GARCH models, HAR-RV

Table of Contents

1. Introduction

Modeling the conditional variance of financial time series, which is viewed as a measurement of risk, is important for the pricing of derivatives, their trading, portfolio optimization and risk management (Hansen & Lunde, 2005; Tsay, 2005). However, due to the complex relationships between financial variables, predicting volatility represents a challenge for practitioners and financial institutions.

Although in the past some experts had believed that financial assets' return variances cannot be predicted for short horizons, econometric literature has found that this is actually possible with specific time series models (Franses & van Dijk, 1996). Many researchers have found Generalised Autoregressive Conditional Heteroskedasticity (GARCH) model, introduced by Bollerslev in 1987, to give very promising forecasts for the returns data (Franses & van Dijk, 1996; Brailsford & Faff, 1996). Some other researchers have also found the specifications of the GARCH model, which account for asymmetries in past returns, to be the best performing in their study samples (Glosten et al., 1993). Furthermore, some practitioners report the Heterogeneous Autoregressive Model of Realized Volatility (HAR-RV) proposed by Corsi (2009), which models daily, weekly and monthly realized variances together, to have good forecasting performance.

On the other hand, modeling with these traditional linear or nonlinear time series models require every mathematical statement of relationships to be predefined in the computer environment, plus have a set of a priori assumptions to be fulfilled. However, subtle and time-varying relationships may exist: Sometimes the connections between the variables are unknown or hard to mathematically formulate (Zhang et al., 1998).

Hence a system which works with very few assumptions, which can recognize without explicit programming the hidden patterns in a dataset, learn continuously about the time-varying relationships between variables as well as their importance towards the determination of an output of interest, referred as Neural Networks, might improve the success of the current forecasts.

By construction, these 'computerized intelligence systems' imitate the structure of a human brain (Fadlala & Lin, 2001). Shown inputs or in other words after some training, Neural Networks find the best procedure to achieve the desired outputs via developing complex algorithms in its black box (Gonzalez Miranda & Burgess, 1997). Along with many other researchers, Bildirici and Ersin suggest that NN's can capture the typical properties of financial returns, namely leptokurtosis, volatility clustering, and leverage effects better than the GARCH models (2009).

Hence, this study is an attempt to shed light on the comparative performances of the Neural Networks and some frequently used econometric models to predict return volatility. In the next sections, the daily realized variances of the S&P 500 Index are forecasted and compared to its real values. Additionally, a specification amongst the Neural Network models is used in this research: Long-Short Term Memory Recurrent Neural Network (LSTM). The reasoning of this choice is going to be provided in the upcoming chapters.

The remainder of this study is as follows: First, a detailed view on the traditional econometric models, Neural Network family and specifically LSTM will be given, together with the findings of the previous researchers. Next, the sample used in this research will be analyzed in the *Data* section. Technical details about the models and the analysis will be demonstrated in *Methodology*. Finally, the results, discussion, and conclusive remarks will follow.

2. Literature Review: Econometric Models

2.1. ARCH Family

The volatility of financial returns is not directly observable, however literature has found that it commonly exhibits some specific patterns: For example, squared returns are serially correlated and they tend to appear in clusters via large (small) positive or negative changes being followed by large (small) changes, of either sign (Jondeau et al., 2007). They also fluctuate within a fixed range and do not diverge to infinity, which means that volatility is mostly stationary. Being the first model until then which takes into account these behaviors of asset returns, Engle has proposed in 1982 the Autoregressive Conditional Heteroskedasticity (ARCH) model for variance, which was essentially an AR (Autoregressive) process. Later in 1986, Bollerslev has proposed a specification of ARCH, namely Generalized Autoregressive Conditional Heteroskedasticity (GARCH) model, which allows for a more parsimonious representation as it estimates only two parameters instead of infinity in the case of ARCH (Alberg et al., 2008).

However, both econometric models suffer from some weaknesses, such as they treat positive and negative shocks as if they have the same effect on return volatility. In fact, negative unexpected returns cause higher volatility than the positive ones (Donaldson & Kamstra, 1996). Therefore, in order to capture these asymmetric effects, several extensions of the GARCH model were introduced. For example, the conditional variance in GJR-GARCH by Glosten et al. (1993) responds differently to positive and negative innovations (Bollerslev, 2008).

There are many other separate, nested or hybrid models in the ARCH family which account for the different properties of volatility. However, in this study, only the above-mentioned asymmetric extension of the GARCH model is used, as it has been found to be among the most popular forecasting methods. 'Glossary to ARCH' by Bollerslev (2008) can be revisited for the full family tree.

Yet, the extensive literature on the performances of ARCH and its extensions show mixed results. For example, in a study on Japanese daily stock returns by Engle and Ng (1993), it was shown that among a class a of GARCH specifications where GJR-GARCH was also included, none of the models consistently outperformed the simple GARCH(1,1) model in-sample. However, they find that GJR-GARCH offered the most potential in modeling the asymmetric effects. Nevertheless, Pagan and Schwert (1990) draw attention to the fact that researchers should analyze the out-of-sample prediction power to make any conclusions about the performance of a volatility model. Donaldson and Kamstra (1997) explain that the greater in-sample predictability might result from overfitting the data, and add that out-of-sample performances had not been not widely discussed in previous research. Studying four indexes with GARCH and two asymmetrical extensions, they find that GJR-GARCH encompasses the other asymmetric GARCH model in the study for the S&P 500. For NIKKEI, they argue that GARCH significantly underperforms compared to the asymmetrical models. Another research is by Hansen and Lunde (2005), where they forecast the one-day-ahead conditional variance of IBM returns as well as the DM-$ exchange rate with 330 models from the GARCH family. They find that while the other models did not offer a significantly higher out-of-sample predictive power compared to GARCH(1,1) in the case of exchange rate, asymmetrical models which account for the leverage effect outperformed GARCH(1,1) in IBM returns.

2.2. HAR-RV

Another model which will be tested in this study is an AR-type model, called HAR-RV by Corsi (2009) which is based on the Heterogeneous Market Hypothesis: Investors with different investment period choices might react differently to volatility changes (Müller et al., 1997). Therefore, the forecasted realized variance is a function of daily, weekly and monthly realized variances. Researchers suggest that HAR-RV is able to capture some stylized features of stock volatility, of which one of the most important is *long memory*, meaning in the persistence of autocorrelations between squared returns. For Andersen et al. (2007), HAR-RV deals with it better than the standard GARCH model.

Although not many studies were done which compares its forecast performance to that of the asymmetrical ARCH specifications, Corsi (2009) has compared HAR-RV to long-term AR(I)MA models, specifically to ARFIMA (Autoregressive Fractionally Integrated Moving Average) and to the simple AR model. He has found that for three sets of realized variance series, namely the USD/CHF, S&P500, and T-Bonds, HAR-RV has a better out-of-sample performance. Nonetheless, many researchers have extended the HAR-RV to include overnight returns, trading volume, and additional leverage effects, etc. to increase the model's predictive power (Wang et al., 2015).

3. Neural Networks: Design and Concepts

As it was mentioned above, Neural Networks offer an improvement in traditional forecasting methods as they are tolerant to incomplete and noisy data, can learn and generalize when presented examples and create their own algorithms to solve a problem (Taylor, 1995; Kristjanpoller et al., 2014). Essentially, its architecture was inspired by the human brain: The latter consist of E+11 neurons which communicate with each other via electrical signals. The receptors of a neuron, so-called 'dendrites', constantly receive signals from the outer environment which reach the cell-body, the decision unit of the neuron. Some of these stimuli excite the neuron and some have the inhibitory property. If the stimuli are above a certain benchmark, the neuron sends an impulse to the other neurons with the help of ions. This ion exchange takes place in the synapses, simply in the interneuronal spaces.

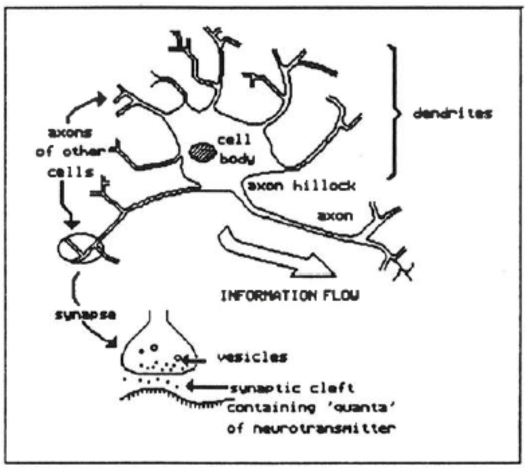


Figure 1: A biological neuron. Reprinted from *Neural Networks*, by J.G. Taylor, 1995, Henley-on-Thames: Alfred Waller in association with UNICOM.

In an artificial neuron as depicted in the below figure, the inputs can be thought as the stimuli from the outer environment or the impulses sent from other neurons. As some of them cause an excitatory effect and some inhibitory, they are multiplied by 'weights', before being summed up by the cell body equivalent. If the total passes the threshold, the node produces the binary output 1. This basic model is called the binary decision (BDN) model and is a simple representation of the biological process.

Artificial neural networks (ANN) refer to a web of artificial neurons linked to each other and they typically consist of 3 layers: First is the input layer, which is passive and which only carries the inputs entered in the system without changing them. The data

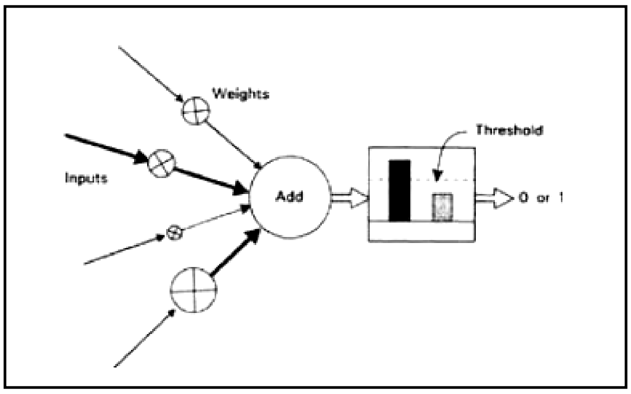


Figure 2: An artificial neuron with BDN. Reprinted from *An Introduction to Artificial Neural Networks,* by K. Gurney, 1997, London: UCL Press.

points are then transformed into the hidden layer(s) with an associated weight, depending on their relative importance in the dataset. Here, they pass through neurons which possess an *activation function (*or *unit step function)*: This function determines if and to what extent that neuron is going to be *active* towards the determination of the outputs in the next layer. These inputs are transformed in a non-linear format via some mathematical operations such as the sigmoid transformation to range values from [-∞, +∞] into [0, 1]. Without a nonlinear transformation process, Neural Networks would act like simple linear regression and would not be able to capture the non-linearity in the dataset (Taylor, 1995). Finally, the output layer produces the end output of the system with the updated weights.

Neural Networks can possess any amount of hidden layers or nodes. However, the most used structure is with one hidden-layer. Furthermore, ANN's with two or more hidden layers are called 'deep learning' systems.

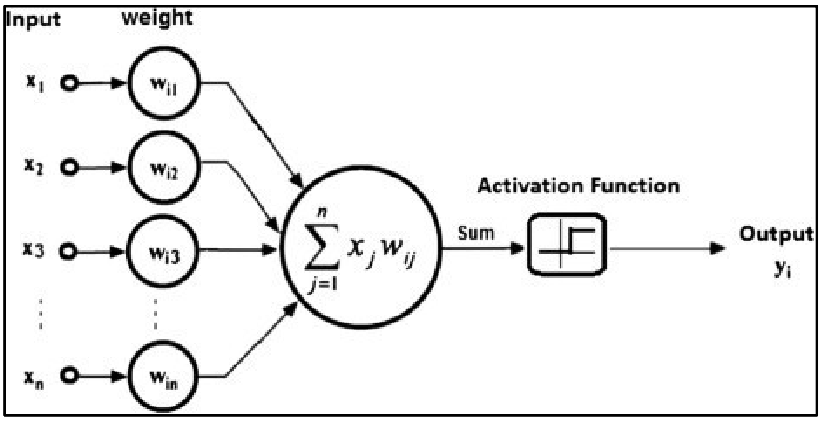


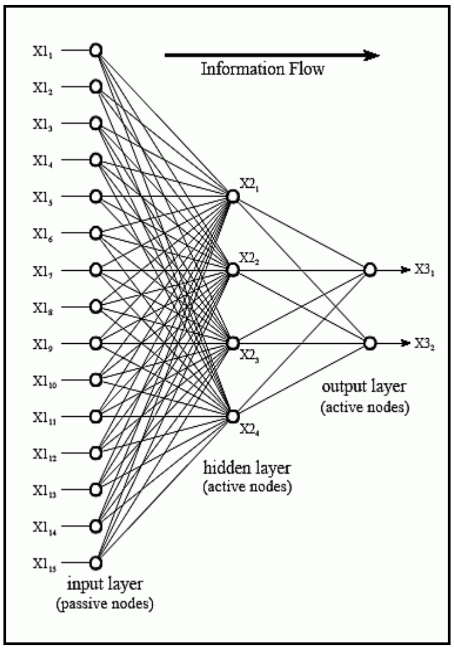
Figure 3: An artificial neuron with a more technical representation. Reprinted from *Artificial Neural Networks(Basics) | Introduction to Neural Networks*, by S. Saxena, 2017, Retrieved from https://becominghuman.ai/artificial-neuron-networks-basics-introduction-to-neural-networks-3082f1dcca8c

4. Types of Neural Nets

Neural Networks are mainly classified into two categories: Feedforward and feedback (recurrent) networks.

4.1. Feedforward Networks

In feedforward networks, information only flows forward from the input layer to the hidden layer(s) and finally to the output layer. No feedback is fed to the previous layers. The depiction to the right is an example of this type of net. These neural Networks are also called Multi-Layer Perceptron (MLP).



Information Flow
input layer (passive nodes)
hidden layer (active nodes)
output layer (active nodes)

Figure 4: A feedforward neural network. Reprinted from *The scientist and engineer's guide to digital signal processing*, by S. W. Smith, 1997, San Diego, CA: California Technical Hub.

4.2. Feedback (Recurrent) Networks

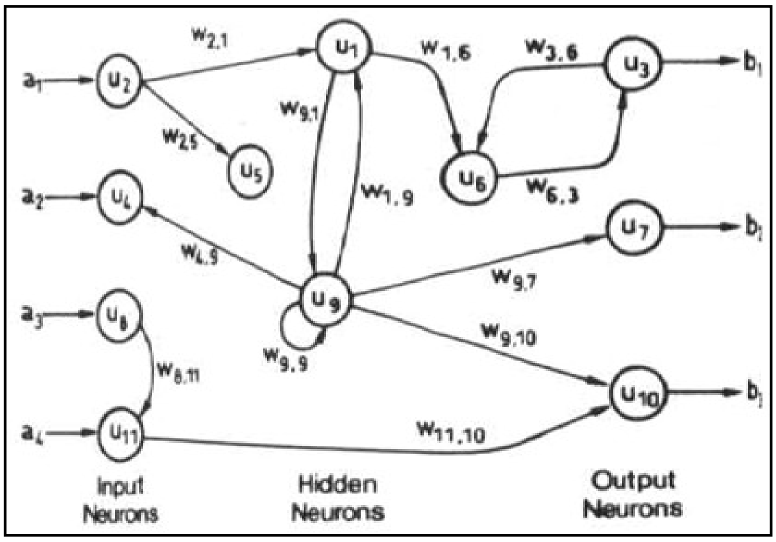


Figure 5: A recurrent neural network. Reprinted from *Neural Networks*, by J.G. Taylor, 1995, Henley-on-Thames: Alfred Waller in association with UNICOM.

Recurrent Neural Networks differ from Feedback Neural Networks as they do not distinguish between input, hidden and output layers: Neurons of any layer can be linked to each other. Computations obtained from previous layers are fed to the web back again, which creates a form of ‘memory’. As financial time series data is dependent on previous time points, RNN can offer better results than MLP (Kim & Won, 2018).

5. Literature Review: From ANN to Long Short-Term Memory

The econometric models that have been explained in the previous chapters have been frequently used in the literature and are well-grounded in the sense that they are based on a statistical formulation. However, an important point is that they are built on the assumption of explanatory variables all being stationary (Kim & Won, 2018). In contrast, Artificial Neural Networks do not require prior assumptions of data distribution, have weaker restrictions and are more noise-tolerant compared to the traditional econometric models (Haykin, 1999; Cao & Tay, 2001). They are flexible and powerful models to capture non-linearity. (Kim & Won, 2018; Haykin, 1999).

Since the mid-1990s, many researchers applied ANN to make forecasts on diverse subjects. In the context of finance, among various topics studied, exchange rates (Zhang, 2003), inflation (Moshiri & Cameron, 2000), stock prices (Oliviera et al., 2013) and GDP growth (Tkacz, 2001) can be exemplified.

Specifically for stock return volatility which is the subject of this paper, the most popular approach was the Feedforward Neural Networks (or MLP). For example, Gonzalez Maranda and Burgess (1997) forecasted IBEX 35 index options' implied volatility and found that MLP has a better out-of-sample performance than OLS and MA. Donaldson and Kamstra (1997) used a combined GARCH & Artificial Neural Network approach on 4 different stock exchanges. They have shown that their hybrid GARCH-ANN model makes better return volatility forecasts compared to GARCH and its asymmetrical extensions including GJR-GARCH. Bildirici and Ersin (2009) studied the volatility of daily returns in ISE between 1987-2008, and have found that ANN integration brings improvement in forecast power for the ARCH family models. Many studies in this field include both linear and non-linearly hybridized ANN models.

On the other hand, some studies have used the Feedback (Recurrent) Neural Networks for forecasting financial time series, which are simply MLP's including lags of dependent variable as explanatory variables (Bekiros & Georgoutsos, 2008). This inclusion creates dynamic feedback on errors of past patterns, which might result in richer modeling (Zhang, 2004). Among the first uses of RNN is the work by Kamijo and Tanigawa (1990) on stock price pattern recognition and they have found that RNN's recognized 15 out of 16 patterns present in the data accurately. Another study by Li et al. (2004) uses RNN on the prediction of short-term exchange rates and finds that it performs better than MA and exponential smoothing methods.

Although RNN's are effective in time series modeling, researchers have recognized some technical problems in its engineering: Roughly speaking, when error signals are fed back in the web (backpropagation) consisting of neurons with application functions like sigmoid, their magnitude quickly vanishes, known as the v*anishing gradient issue* (Schmidhuber et al., 1999). The detailed analysis can be found in Hochreiter (1998). This problem makes difficult for RNN to learn if the time lags between the signaling information and target event are greater than ten time steps (Schmidhuber et al., 1999). So, long-term memory is in a sense vulnerable (Kim & Won, 2018).

That is why an improved version which overcomes these architectural difficulties was proposed by Hochreiter and Schmidhuber in 1997: the Long Short-Term Memory. LSTM was used in language modeling and processing, handwriting synthesis and speech recognition. In finance, this area still remains undiscovered as too few research were found: Fischer and Krauss (2018) applied LSTM to predict the probability for each S&P 500 stock to outperform the general market (of Fama & French) between 1992 and 2015. They have found that LSTM outperforms a set of memory-free neural network models in the study, including the standard deep neural networks. Additionally, Kim and Won (2018) integrated LSTM with GARCH and its specifications. They have found that their version of the hybrid model, GEW-LSTM (GARCH, EGARCH, EWMA, and LSTM) gives the lowest prediction error. Sang and Di Pierro (2019) used LSTM to improve traditional technical analysis trading algorithms. They find that this combination of the two methods gives better prediction power in comparison to the traditional technical analysis algorithms alone.

Furthermore, most of the other studies attempt to alter the technical algorithms behind LSTM, such as in Liu (2019) and in Schwedersky et al. (2019). Apparently, the integration of finance and LSTM is a very new concept.

6. Contribution and Hypotheses

As it was shown in the previous chapters, the time-series forecast performance of LSTM models in comparison to traditional econometric models were not explicitly analyzed in the literature. Rather, it was seen that LSTM was either hybridized with econometric models or its technical algorithms were altered. On the other hand, there are too few researches concentrating on both finance and stock returns. In general, researchers have used LSTM to make predictions on topics related to linguistics.

Therefore, this research aims to shed light on the applicability of LSTM in the field of stock return volatility prediction. It is intended to offer a comparative analysis on out-of-sample forecasts of econometric models versus the long short-term memory model.

Hence, some selections of econometric models are made. AR(1), ARMA(1,1), GARCH(1,1), GJR-GARCH(1,1) and HAR-RV are chosen to be compared to LSTM as they are among the most popular and effective models in the literature.

Nonetheless, the findings of this research can be useful to practitioners, researchers and academicians as long short-term memory recurrent neural networks represent an opportunity to achieve better return volatility forecasts. This might have financial implications as volatility estimation is essential in determining investment strategies and is relevant in the calculation of important measures such as Value at Risk (VaR) or the Sharpe ratio (Kristjanpoller et al., 2014). Especially, the risk management field can benefit as the accurateness of the statistical estimates form the base of the sector (Fleming et al., 2001).

This paper tests the period of January 2010 to June 2019 to assess the models' performances. Hence, it revolves around the following research question:

"Does LSTM outperform the econometric models in terms of return volatility forecasting on S&P 500 between January 2010 and June 2019 ?"

Additionally, the testing period was chosen to be separated into two parts: 2010-2011 and 2012-2019. The reasoning of this choice is going to be explained in the next chapter.
This preference resulted in the below two hypotheses:

H_1: *The LSTM forecasts on S&P 500 between January 2010 and December 2011 outperform those of the AR, ARMA, GARCH, GJR-GARCH and HAR models.*

H_2: The *LSTM forecasts on S&P 500 between January 2012 and June 2019 outperform those of the AR, ARMA, GARCH, and GJR-GARCH models.*

7. Data

The data of this research comes from the Realized Library of the Oxford-Man Institute of Quantitative Finance. The dataset consists of daily frequencies of 5-min realized return volatility and open-to-close returns of the S&P 500 Index. The period studied is the trading days from the beginning of 2000 until the end of June 2019, making a total of 4891 observations. The interval from the beginning of 2000 until the end of 2009 will be used for the training of LSTM. Rest of the data points until the end of June 2019 will be used as the test data to compare the forecasting performances. Also, the testing period is analyzed in two parts: January 2010-December 2011 and January 2012-June 2019. Further explanation for this is provided in Descriptive Statistics.

7.1. Data Transformation

The open-to-close returns are multiplied by E+2 and realized variances by E+4 in this research for the sake of easier reading.

7.2. Descriptive Statistics

Table 1: Descriptive Statistics for Realized Variance of the Full Sample

	Realized Variance*
Mean	1.071
Standard Deviation	2.428
Minimum	0.012
Maximum	77.477

**Real values multiplied by E+4.*

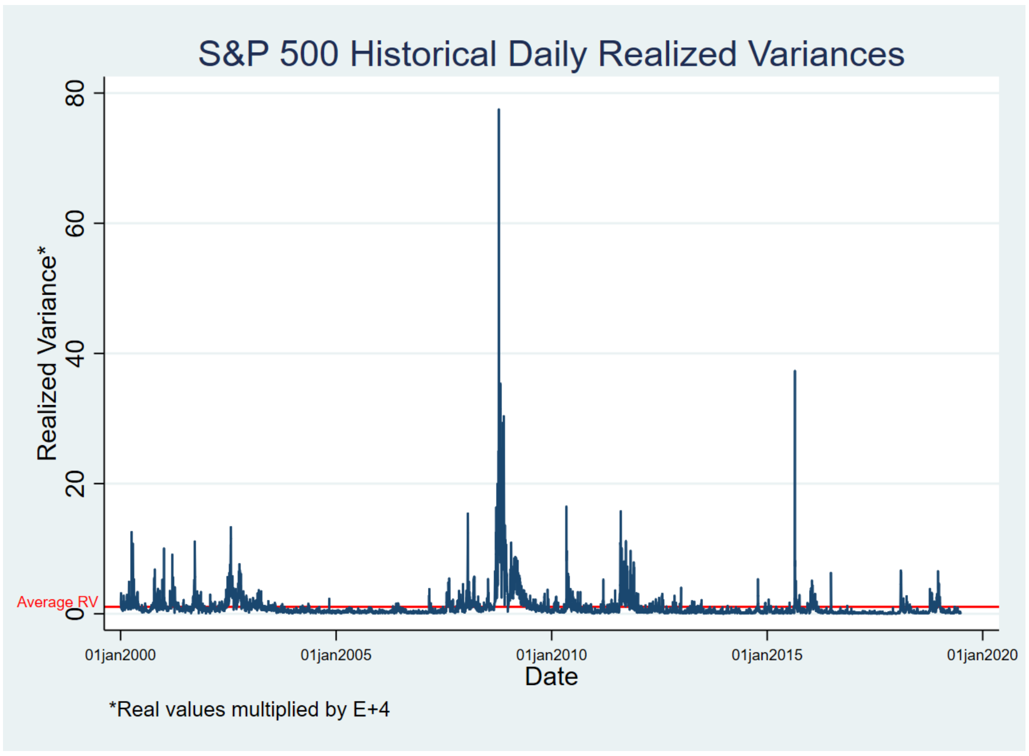


Figure 6: Historical Daily Realized Variances of the S&P 500 from January 2000 till June 2019. The red line depicts the average realized variance.

Above figure shows the historical dispersion of the 5-minute realized variances of the S&P 500. The preview of the sample shows that their average is 1.071, as represented by the red horizontal line in Figure 5. Higher-than-average volatilities were pronounced especially during three moments: From the beginning of the dataset until 2003, the fluctuations correspond to the Early 2000's Recession due to the collapse of the technology bubble and 9/11 terrorist attacks. The other period where the S&P 500 return volatility peaked was between 2007-2009, and it corresponds to the 2008 economic crisis where the housing bubble burst and US financial institutions have failed due to the mortgage crisis. Especially on November 10, 2008, the highest value in the dataset was observed: A realized variance of 77.477. Despite the decreases in volatility, the fluctuations continued in the post-crisis period, as observed between 2010 to 2012.

Furthermore, another large peak can be observed after 2015, which corresponds to August 24. Such increase in the return volatility was related to big sell-offs in Asia provoking declines in European and U.S. stock features. The sell-offs resulted from China's economic slowdown (CNBC, 2016). Later, 2017 was one of the historically least volatile periods since the '60s for the market, which explains the minimum value of 0.012 observed at the end of 2017 (Reuters, 2019). 2018 came with slight fluctuations, with high and low return volatilities dominating the year (CNBC, 2018).

Referring back to the testing period, the choice was made to analyze it in two parts as 2010-Dec 2011 and 2012-June 2019. This decision is due to their different characteristics that the post-crisis period consisted of many unexpected shocks and 2012 onwards being relatively stable with the exception of a peak in 2015.

7.3. Software and Hardware

In this research, two softwares are used: The analysis of AR, ARMA, GARCH, GJR-GARCH, and HAR-RV are studied in STATA. The LSTM model is set in Python 3.7 with the use of *numpy* and *pandas* packages as in Fischer and Krauss (2018). Additionally, the open-source library *Keras* which allows fast-experimentation with neural networks was used on top of *TensorFlow*, the latter being another open-source math library used for machine learning.

8. Definitions - Econometric Models

The definitions of the models used in this research are as follows:

AR (p):

The autoregressive process where the series stationary value y_t depends on p past values is represented by the below equation

$$y_t = \mu + u_t + \sum_{i=1}^{p} (\phi_i y_{t-i})$$

where $\phi_i \dots \phi_p$ are the autoregressive coefficients, u_t the white noise at time t and μ the constant.

ARMA (p,q):

ARMA of order p and q consists of two components: Autoregressive (AR) and Moving Average (MA) processes.

$$y_t = \mu + u_t + \sum_{i=1}^{p} (\phi_i y_{t-i}) + \sum_{i=1}^{q} (\theta_i u_{t-i})$$

y_t in the equation above represents the series stationary value, ϕ and θ the parameters of the model, u the residuals and μ the constant.

The model is based on the idea that the value of a current observation in a time series depends both on the value of its lagged observations and previous shocks.

GARCH (p,q):

GARCH model is built upon the theory that the conditional variance depends on p lags of its values σ^2 and on q lags of the squared error, as noted by u^2. The constant is represented by μ.

$$\sigma_t^2 = \mu + \sum_{i=1}^{q} (\alpha_i u_{t-i}^2) + \sum_{j=1}^{p} (\beta_j \sigma_{t-j}^2)$$

From the above volatility equation, it can be seen that the model accounts for volatility clustering, as periods with high volatility tend to result in the next periods having higher volatility via the last term of the equation.

GJR-GARCH:

One of the GARCH extensions which captures the asymmetrical effects is the GJR-GARCH model. The general equation below has an indicator function noted by I, which takes the value of 1 if the lagged unconditional standard deviation u_{t-i} is smaller than 0. Indicator function helps at modeling the asymmetry caused by positive and negative shocks: While a positive shock has an effect of α, negative news has $\alpha+\gamma$ via the indicator function. Conditional variance is always positive through some restrictions.

$$\sigma_t^2 = \mu + \sum_{i=1}^{q} (\alpha_i u_{t-i}^2) + \sum_{j=1}^{p} (\beta_j \sigma_{t-j}^2) + \sum_{i=1}^{q} (\gamma_i u_{t-i}^2 I_{t-i})$$

$$where\ I_{t-1} = 1\ if\ u_{t-1} < 0$$

$$otherwise\ = 0.$$

HAR-RV:

HAR model makes use of different time horizons to predict variance. In the below equation, RV_t^d, RV_t^w and RV_t^m represent the daily, weekly and monthly realized volatilities respectively.

$$RV_{t+1d}^{(d)} = \mu + \beta^{(d)} RV_t^d + \beta^{(w)} RV_t^w + \beta^{(m)} RV_t^m + u_{t+1d}$$

The weekly and monthly components of the equation are defined as the averaged realized variances of the last 5 and 22 days (denoted by d), respectively.

$$RV_t^w = \frac{1}{5}\sum_{i=0}^{4} \mathrm{RV}_{\mathrm{t-id}}^{\mathrm{d}}, \qquad RV_t^m = \frac{1}{22}\sum_{j=0}^{21} \mathrm{RV}_{\mathrm{t-jd}}^{\mathrm{d}}$$

$$where\ RV_t^w = \frac{1}{5}(\mathrm{RV}_{\mathrm{t}}^{\mathrm{d}} + \mathrm{RV}_{\mathrm{t-1d}}^{\mathrm{d}} + \mathrm{RV}_{\mathrm{t-2d}}^{\mathrm{d}} + \mathrm{RV}_{\mathrm{t-3d}}^{\mathrm{d}} + \mathrm{RV}_{\mathrm{t-4d}}^{\mathrm{d}})$$

All the above models will be used to make one-step-ahead forecasts of S&P 500 return volatility for the period between January 2010 and June 2019.

9. Long Short-Term Memory: Architecture

LSTM models are created to overcome the technical problems seen in the Recurrent Neural Networks by Hochreiter & Schmidhuber (1997). As the latter systems were not able to remember correlations between events separated by more than 10 steps, LSTM models were built with memory blocks instead of neurons to overcome these technical difficulties (Schmidhuber et al., 2007).

The LSTM networks consist of input, output and hidden layer(s), just as in the basic ANN architecture. Input layer only serves to transfer the inputs to the hidden layer(s) without any change. The power of the LSTM's comes from the hidden layer(s) as the latter contains *memory blocks* for each timestep t. These memory cells have three *gates* that determine the *cell state* s_t, which is essentially the memory of the cell of the optimal weights for all the connections in the LSTM model to transform the inputs to the desired outputs.

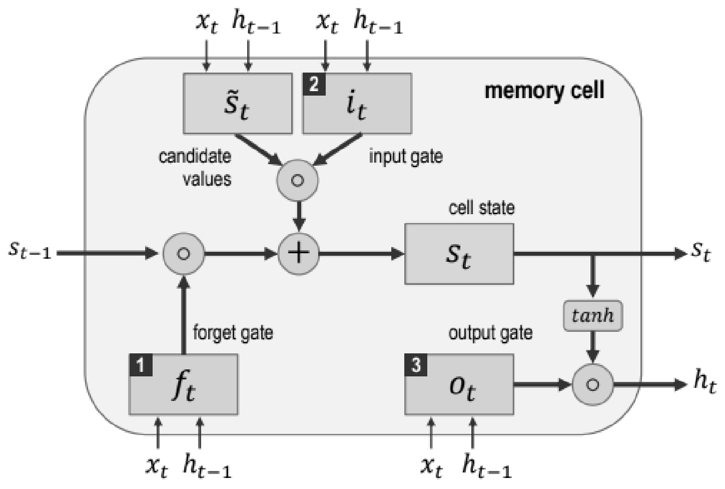


Figure 7: The structure of an LSTM memory cell. Reprinted from "Deep learning with long short-term memory networks for financial market predictions" by T. Fischer, C. Krauss, 2018, *European Journal of Operational Research, 270*(2), p. 657.

The three gates are named as input i_t, output g_t and forget f_g. At each time step t, they receive the same information: The current inputs x_t and the output of the memory cell belonging to the previous time step, h_{t-1}. However, each of them has a different purpose:

-The forget gate determines which information to delete from the cell state s_t.

-The input gate determines which information to add to the cell state s_t.

-The output gate determines which information to send as an output to the next timestep from cell state s_t.

The following variables, also present in the above figure's notations, need to be defined as they are the basis of the LSTM equations. Below-mentioned details are extracted from Fischer and Krauss (2018).

- x_t: The input vector at time t
- s_t *and* $\tilde{s}_t$: The cell state (memory of the weights at time t) and candidate value vectors (inputs to add to the cell state at time t) respectively.
- $W_{f,x}, W_{f,h}, W_{\tilde{s},x}, W_{\tilde{s},h}, W_{i,x}, W_{i,h}, W_{o,x}$ *and* $W_{o,h}$: The weight matrices of the inputs for three gates forget f, input i, output o, and the candidate values $\tilde{s}$. The

second lower index represents the source of the information: x refers to current inputs and h to the previous memory block's output.

- $b_f, b_{\tilde{s}}, b_i$ *and* b_o: the bias vectors. The term *bias* stands for a constant added to the activation function, with the latter being a general name given to all functions which transform an input to an output. Therefore, adding constant results in the shifting of the activation function. Bias term can be useful if the predictions are systematically far away from the real values: The addition of a constant term to a function can shift the prediction line closer to the benchmark line.
- f_t, i_t *and* o_t: The activation value vectors for their corresponding gates at time t. The activation value can be thought of as a benchmark value for information flow at different gates. This is analog to the level of stimuli in a biological neural network setting after which the neuron starts to send impulses to the other neurons.
- h_t: The LSTM output vector which is sent to the memory block of the next time step.
- Sigmoid function: A function which scales down values between 0 and 1 on the basis of the following equation, where x represents the input to the function.

$$sigmoid(x) = \frac{1}{1+e^{-x}} \qquad (1)$$

Values of 0 and 1 have different meanings for the different gates, which is explained below.

- Hyperbolic tangent function (tanh): A function to regulate the network which scales down values between -1 and 1 on the basis of the following equation, where x represents the input to the function.

$$tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)} \qquad (2)$$

The learning begins in the forget gate, where it is decided which information to remove from the memory of the previous memory block, denoted as s_{t-1}. For this, the activation value, in other words, the weighted sum of the forget gate inputs needs to be calculated: It is a function of the current inputs x_t, the outputs coming from the memory block of the previous time step h_{t-1} and the bias term b_f. The output of this function passes through sigmoid function which squashes the values between [0,1], determining the extent it is going to be forgotten. 0 stands for completely forgetting the information and 1 for completely remembering.

$$f_t = sigmoid(W_{f,t}x_t + W_{f,h}h_{t-1} + b_f) \quad (3)$$

Second, the input gate decides on which new combination of relationships (information) to add in the cell state s_t. This is done in two steps: Initially, the candidate values $\widetilde{s_t}$ which bring new information are calculated (4). Then, the activation value for the input gate i_t is determined (5).

$$\tilde{s}_t = \tanh(W_{\tilde{s},x}x_t + W_{\tilde{s},h}h_{t-1} + b_{\tilde{s}}) \, (4)$$
$$i_t = sigmoid(W_{i,x}x_t + W_{i,h}h_{t-1} + b_i) \quad (5)$$

Tanh in the candidate values equation (4) stands for the hyperbolic tangent function. It serves as a network regulator by ranging values between [-1, 1]. For the sigmoid function in the input gate activation equation (5), value 0 is interpreted as the information not adding value, and 1 as bringing new insights.

All these above calculations on forgotten and added inputs should be fed into the current memory of the cell s_t. Let Θ denote the element-wise multiplication. The current cell state is computed as follows:

$$s_t = f_t \, \Theta \, \mathrm{s}_{t-1} + \mathrm{i}_t \, \Theta \, \tilde{s}_t \qquad (6)$$

Hereby, the f_t can be interpreted as the ratio of past information reaching the current state, and i_t as the ratio of candidate values that are added to the current cell memory. The gates, therefore, prevent irrelevant inputs from entering the memory and make long-term memory storage possible (Malhotra et al., 2015).

Finally, the combined information which needs to be sent to the memory block in the next time-step, namely h_t, is computed via the below two equations.

$$o_t = sigmoid(W_{o,x}x_t + W_{o,h}h_{t-1} + b_o) \quad (7)$$

$$h_t = o_t \, \Theta \, \tanh(s_t) \quad (8)$$

It is also important to note that there exist different functions instead of sigmoid and tanh. However, these are the standard and most used configurations of LSTM, therefore they are chosen to be applied in this study. The logic of this standard is provided in the *Long Short-Term Memory: Configuration* chapter.

10. Long Short-Term Memory: Terminology

In the training of neural networks, the important terminologies are cost function, gradient descent, backpropagation, epochs, batch size, and dropout regularization. By the term *training*, we refer to finding the best combination of weights and biases to be assigned to the neural connections in a network which gives the closest values to a targeted output. To train a network, we need a training set which is essentially a part of our dataset and a cost function. With the latter, we refer to a function which measures the differences between the network output and the desired target, namely the network error (Gurney, 1997). Cost functions have many types, such as mean squared error, mean absolute error and mean error.

In order to obtain the most accurate predictions, the network error needs to be minimized. Therefore, the set of weights and biases which result in so should be found. The methodology to achieve the lowest network error is called *gradient descent optimization.*

It is important to note that we cannot simply take the minimum of the cost function and find the corresponding weights, because in artificial neural networks we are dealing with non-linear optimization surfaces as a function of thousands of weights and biases. Therefore, minimizing the network error represents navigating on a 3D surface and gradually approximating towards the local minimum as depicted by the lines in the below figure. However, it is not guaranteed the local minimum reached is the global minimum of the cost function as there might exist many local minima in the error surface. Notice that the below figure corresponding to the error surface of a neural network model with two weights has more than one minimum error points, colored with purple.

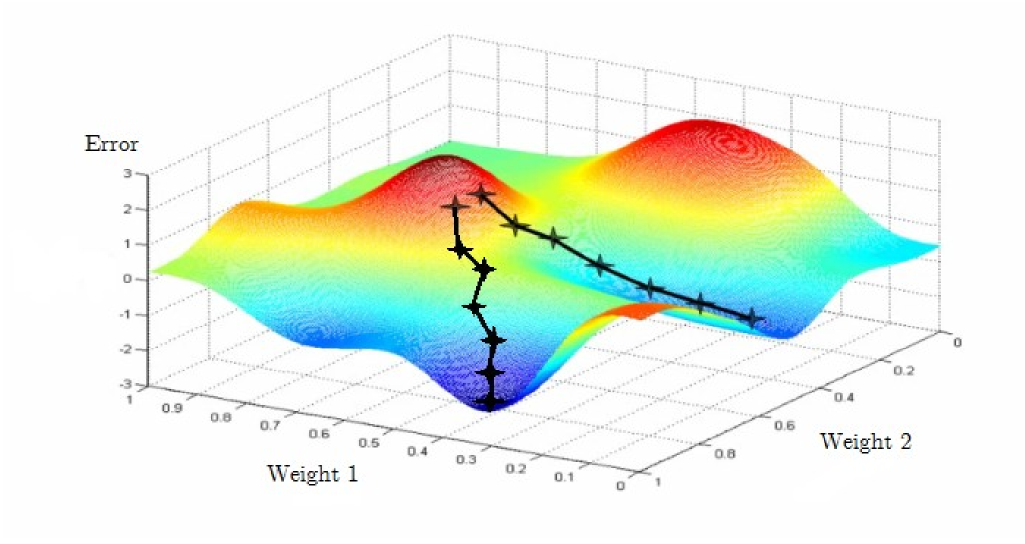


Figure 8: An exemplary error surface with 2 weights. Reprinted from *01 and 02: Introduction, Regression Analaysis and Gradient Descent* by A. Ng, ?, Retrieved from http://www.holehouse.org/mlclass/01_02_Introduction_regression_analysis_and_gr.html.

Two terms are essential in gradient descent optimization, first being *gradient.* This term stands for the direction of the steepest increase of the error, given a starting point on the error surface. The opposite of this vector naturally tells the directions of the steepest ascent in the surrounding error surface, meaning the closest way towards the local minimum.

Another important term is the learning rate, which is the measure of how quickly the model changes its weights to reach the local cost (error) minimum, or how big its steps are while changing the weights and biases. If its value is too large, the model might miss the local minimum. If it is too small, the algorithm will take a long time to converge.

The weight updates via the gradient optimization are based on the below equations, where w_{n+1} and w_n are the new and old weights assigned to the connections between neurons respectively, γ the learning rate and ∇w_n a vector being the gradient of the cost function C. Bias updates b are deduced in the same fashion.

$$w_{n+1} = w_n - \gamma . \nabla w_n \ where \ \nabla w_n = \frac{\partial C(w_n)}{\partial (w_n)} \quad (9)$$

$$b_{n+1} = b_n - \gamma . \nabla b_n \ where \ \nabla b_n = \frac{\partial C(b_n)}{\partial (b_n)} \quad (10)$$

It is important to note that when the training data is fed into the system, random weights and biases are assigned to all the connections in the network. The network reaches an output, with some deviation from the desired values known as the errors. The term *back-propagation* stands for these errors being distributed back from the output to through the layers of the neural network, so as to tune all the weights and biases via the *gradient descent* method to reach a minimized network error.

However, this does not happen at a time: The algorithm does not immediately come to a solution as the weight and bias changes happen in small steps. It means that it does not fully learn every information in the data at one go, and an exposure of the training set of multiple times is needed for it to converge towards the local minima (we use the plural form here, as in each exposure we start with random weights and biases which brings us to different minima). Therefore, we refer here to the term *epoch,* representing the number of complete presentations of the training set to the algorithm.

Also, as the number of training data that the computer should analyze is usually quite large, it was found that it is faster to analyze it in pieces: Hereby, we refer to the term *batch*es, the number of groups in which we pack the observations of the training data.

The final term to be explained is the *dropout ratio*, which is actually related to the concept of *overfitting*: When the neural networks make predictions, they base their estimates on their learnings from the training data. However, if the model fits these limited data points too closely, it might not be able to generalise well when it is given a deviating set. This is when an *overfitting* happens: The model performs too well on the training set but produces high errors on the unseen data. The avoid this situation, a technique called *dropout regularization* can be used. Via randomly dropping a specified amount of neurons in the hidden layer(s) and their connections in each epoch, we can remove the complexities in the model's configuration which might be the reason why the model is not generalizing well. This method helps to reduce the dependency of neurons to each other, meaning the fact that them detecting the same features from the inputs. This dependency worsens the model's performance as errors would be repeated by the dependent neurons. Randomly dropping some hidden units from the model each time would, therefore, give a set of more independently working neurons, decreasing the model's error.

11. Long Short-Term Memory: Configuration

The LSTM model implemented in this study to predict t+1 return volatility has the below technical specifications:

LSTM Model	
Architecture	Recurrent Neural Network
Type	Long Short-Term Memory
Number of Hidden Layers	Single LSTM layer
Number of LSTM Units in Each Layer	50
Inputs	RV_{t-21} till RV_t
Activation Functions	Sigmoid and tanh
Number of epochs during training	1000
Batch Size	32
Loss Function	Mean Squared Error
Dropout Ratio	0.2
Training Set as of Total Data	50%
Optimization Algorithm	Adam

As explained in the LSTM Architecture section, the general configuration for the LSTM models is to have sigmoid and hyperbolic tangent (tanh) as activation functions. The sigmoid function is used in the three gates as it ranges information between no flow and complete flow. On the other hand, we use tanh in the calculation of the vector output of the memory block: LeCun et al. (1998) have argued that due to the shape of the tanh function, its derivatives are larger than that of the sigmoid which results in achieving the local minima faster (referring back to the gradient issue, the steepest ascent). Therefore, these two standard functions are used in our study.

As inputs, the realized variances of the S&P 500 from t-21 until t were used to make t+1 forecasts. This choice was inspired by the HAR model's inclusion of the monthly component of the volatility process and being very effective at capturing its stylized facts. A similar choice is also present in Kim and Won (2018).

For the other factors such as the number of hidden layers, LSTM units in each layer, epochs, batch size, dropout, and training set as of total data, it should be noted that there is no definitive rule to find their optimal numbers. They are actually hyperparameters which require trials with a set of different combinations so that the lowest cost is achieved. However, as we are working with a very large dataset which requires Python to run for hours, we selected to use the values commonly found in the literature: The final selection was one single hidden layer with 50 LSTM units, 1000 epochs, batch size of 32 data points (it is a default value in the LSTM literature as it yields faster and accurate computation of the local minima), 0.2 dropout rate (20% of the hidden LSTM units are dropped randomly in each epoch) and 50% as the ratio of training set to the total data.

For the optimization algorithm, we have used an alternative of the previously mentioned classical gradient descent model (also known as stochastic gradient descent, SGD). 'Adam', in other words 'Adaptive Moment Estimation' is another very popular machine learning algorithm. It is based on the idea of learning the learning rates throughout training, as opposed to the classical model having a fixed pre-determined learning rate. However, Adam's most important feature is that its algorithm takes much less time to converge to the local minima compared to many other optimization algorithms including SGD, therefore providing relatively faster results (Sang & Pierro, 2019). That is why this optimizer was selected in our study instead of the classical SGD model. The weight assignment equations for Adam will not be provided as they are very complex to derive and that these derivations are out of the scope of this analysis.

12. Performance Measure

Performances of the above models are going to be compared with Mean Squared Error (MSE), which was prevalently used in literature (Kim & Won, 2018; Kristjanpoller et al., 2014). MSE is defined by the below equation, where n is the number of observations in the forecasting period, RV_i and $\widehat{RV}_i$ the observed and predicted realized variances respectively.

$$MSE = \frac{1}{n} * \sum_{i=1}^{n} (RV_i - \widehat{RV}_i)^2$$

13. Results

Below table reports the MSE's obtained of all models and their rankings in this study.

Table 2: Mean Squared Error of the Models and Their Rankings

Period 2010-2011			Period 2012-2019	
Model	MSE*	Ranking	Model	MSE*
ARMA(1,1)	2.453	1	GJR-GARCH(1,1)	0.903
HAR	2.457	2	GARCH(1,1)	0.957
AR(1)	2.612	3	HAR	0.996
GJR-GARCH(1,1)	2.812	4	ARMA(1,1)	1.010
GARCH(1,1)	2.957	5	AR(1)	1.122
LSTM	4.175	6	LSTM	3.112

*Real MSE's multiplied by E+8

- Period 2010-2011

The graphs for the models of this period are attached under Appendix 1. The outcome of the analysis is that ARMA(1,1) produced the best forecasts for this period characterized by many fluctuations with an MSE of 2.453. The model is followed by HAR with an MSE of 2.457. Interestingly, both models produced a very similar pattern for this period, as exhibited by the figures 1A and 1B in the Appendix.

The third best forecasting model was the AR(1) model with an MSE of 2.612 (Figure 1C in the Appendix). It is important to note that even though AR(1) was more successful at predicting especially the fluctuations after July 2011 very well and high realized variances corresponding to the peaks in general, it has its lower bound of predictions lying much above than that of the realized values. This should be the reason why it has a higher MSE than the first two models.

Furthermore, the worst three processes were GJR-GARCH(1,1), GARCH(1,1) and LSTM with out-of-sample error values of 2.812, 2.957 and 4.175 respectively. In figure 1D and 1E in the Appendix, it can be seen that GJR-GARCH produced closer estimates to the realized values for the peaks than the GARCH model. It could also predict the downward moves better than the latter: Overall, we have smaller distances between the two lines during the period of study, pronounced for around July 2010. Thus, it is expected that the GJR-GARCH(1,1) would give a better MSE value than the GARCH(1,1) model.

When the two models are compared to the best performing ARMA(1,1), it can be observed that they were not as successful as the latter due to the fact that ARMA(1,1) predictions were almost parallel to the realized line with just small differences. The pattern is more irregular for GARCH and its asymmetric counterpart GJR-GARCH, which were not able to capture the direction and the magnitude of the fluctuations very well from around June 2010 until July 2011.

Finally, the worst predictive model is LSTM for this study period. It has an MSE of 4.175, equaling almost 1.7 times the error for the best model ARMA(1,1). Figure 1F in the Appendix shows that it tends to produce much higher estimates than the realized values after a peak (marked with circles) and much lower values after a drop (marked with diamonds) compared to the other models in this study. Therefore, it resulted in having the lowest MSE. This model is apparently not suitable for predicting time series with fluctuations.

- Period 2012-2019

The graphs for this period are attached in Appendix 2. The outcome was that the best performing model was instead GJR-GARCH(1,1) with an MSE of .903 followed by GARCH(1,1) with .957 (Appendix Figure 2A and 2B respectively). Two models especially differed in high fluctuation moments of around January 2016 and January 2018 onwards: From the end of 2015 until the second half of 2016, GJR-GARCH produced much closer estimates to the realized line. However, from the beginning of January 2018, it overpredicted some points as opposed to GARCH giving closer values. But overall, GJR-GARCH performed very well at capturing the up-and-downs in this period and achieved the lowest MSE among all the models.

Moreover, the next best performing models were HAR and ARMA(1,1) with very close mean squared error values of .996 and 1.010 (Appendix 2C and 2D). It was surprising to see that those two models performed at a similar level again as they did in the 2010-2012 period. Overall, a very similar pattern of forecasting is detected, with slight differences at the high fluctuation moments where HAR RV produced closer estimates to the realized line than the ARMA(1,1) model. These periods correspond to around January 2016 and January 2018 onwards.

Finally, the worst two forecasting models were AR(1) and LSTM (Appendix 2E and 2F). While the first model performed slightly worse than the ARMA(1,1) with a value of 1.122, LSTM gave an MSE of 3.122 which is almost 3.5 times that of the best performing GJR-GARCH(1,1). For the AR(1) model, it was noticed that even though the predictions were mostly parallel to the realized line, the lower bound of the estimates lie much above the red line. This feature was also detected in the 2010-2011 period. Additionally, until the end of 2015, the model had its upper bound of predictions lie much closer to the blue line than those of GJR-GARCH(1,1).

For LSTM, it was observed again that it predicted much higher variances after a peak, but this time it did not estimate much lower values after a drop. Yet, it can be said that this LSTM configuration is especially unsuccessful when there are fluctuations and large shocks, as exemplified by the beginning and the end of 2015.

14. Discussion and Limitations

Due to the high fluctuations between 2010-2011, it was chosen to analyze the this period separately from the rest of the data points until June 2019. The outcome of the study was that ARMA(1,1) had the best predictive power between 2010-2011 while GJR-GARCH(1,1) performed the best between 2012-2019 in terms of the performance metric MSE.

- 2010-2011

For this period, it was unexpected that ARMA provided better return volatility forecasts than all the other models. In the literature, most of the findings are in favor of the GARCH family models. However, in this sample, ARMA could predict the ups and downs much more accurately and gave a much parallel look to the realized values compared to both of the GARCH class models. This was probably due to the nature of this period: The realized variance line depicts multiple shocks followed by decreases in volatility which has a tendency to turn back to its average level of around 1. This is what the ARMA process is actually aiming at: By a merger of the autoregressive and the moving average components, it captures the effects of mean reversion and of unexpected shocks. In that sense, it is reasonable that it represented the movements in this period better than the others.

The models which followed ARMA were HAR, AR, GJR-GARCH, GARCH, and LSTM respectively. The HAR process almost had the same value of the error metric as the ARMA model, and surprisingly it has given a very similar forecasting pattern. In that sense, this result is supportive of the prior research concluding the effectiveness of using daily, weekly and monthly components to forecast realized variances.

About the AR process which was the third best performing model on this period, even though it has given an almost parallel line when compared to the real values, it was observed that it over-predicted the mean. This was probably due to the peaks throughout the dataset, inflating AR's estimates. On the other hand, the fact that AR performed better than the GARCH family models was surprising as the latter is known to capture the stylized facts of asset volatility such as volatility clustering. However, this outcome is most probably due to the choice of the time frame and not valid for the whole testing period.

Another result from 2010-2011 was that GJR-GARCH had a lower MSE than GARCH. This finding is also present in Awartani and Corradi (2005), where the researchers conclude that GJR-GARCH predicts one-step-ahead better than GARCH on S&P 500 between 1990 and 2001.

Finally, the LSTM model showed the worst performance in this period. This configuration was not able to predict the fluctuations and it has produced a 'delayed look': After an upward directioned shock at time t, normally the real values tend to go down in the next period t+1. However, instead of predicting a drop at t+1, the model generally predicts very high estimates for t+1 than drops sharply at t+2, replicating the shock and the volatility drop that happened in the previous time step. It is obvious that these settings are not ideal to predict a period with many unexpected shocks. Yet, as the machine learning models develop and update their own algorithms continuously in their Black Box, it is not possible for us to know what exactly caused this delayed look. We will refer to the possible treatments after studying the model's performance in the second period.

- 2012-2019

The second period of the study corresponds to a relatively stable time frame except for one peak. The outcome was that GJR-GARCH provided the best forecasts, followed by GARCH, HAR, ARMA, AR, and LSTM respectively. Therefore, we conclude that in both of the study periods, the asymmetric version of the GARCH model has beaten its parent GARCH. This is also in line with Hansen and Lunde (2005), who have found that GARCH models which account for leverage effects such as GJR-GARCH outperform GARCH(1,1) in their study on IBM returns.

On the other hand, these two models resulting in better forecasts was in our opinion due to the fact that this period was not characterized by many unexpected shocks but rather it was stable as opposed to the first period. Therefore, a model which accounts for the general characteristics of the stock volatilities fitting the sample better than AR or ARMA models is intuitive.

The next best performing models after the GARCH type processes are HAR and ARMA. In this study period, the latter two also had very similar patterns with this time HAR predicting the peaks better than ARMA.

The worst two models were AR and LSTM. Also in this period, the first model overpredicted the mean, and LSTM showed a delayed look. Even though not many studies were done on index return volatility with LSTM, the model and its extensions were effective at forecasting financial variables such as forecasting price movement in Nelson et al. (2017). In that sense, our finding that LSTM is not successful at forecasting return volatility is unexpected. However, it might be due to the hyperparameter specifications which represent a challenge in the neural networks setting as finding their optimal level are mostly based on trial and error: We have detected that automated hyperparameter tuning services which test different configurations to find the optimum values exist, however we have not observed any researcher integrating the method into their studies in the field of finance. This is due to the fact that the computational requirements and the costs are high (Domhan et al., 2015): In order to process such large datasets with different sets of hyperparameters, a lot of time and a very high performance computer is needed.

Therefore, in a setting where the tuning automation is not applied, one of the most important treatments can be to add more data points to the training set so that the model can practice on more relationships, as adviced by Williams and Zipser (1995). Similarly, a training set to total data ratio of 70% can be employed as in Kristjanpoller et al. (2014) or of 75% as in Fischer and Krauss (2018).

Moreover, trials with more epochs can be made as the latter represents the number of steps the algorithm can take to reach a local minima from a point on the error surface. More representations of the training set to the optimization algorithm might bring lower losses since going more steps on the error surface could bring us to a local minimum with a lower error value. However, due to time constraints, different trials were not possible as just this configuration took 6 hours to converge. Additionally, especially testing with different dropout ratios might enhance the model's quality (Gal & Ghahramani, 2016): Maybe a model with much less connections better suits this dataset. Last, it is also possible to try with different numbers of layers, LSTM units, and batch sizes. The fact that there exists no predefined optimal value for these hyperparameters is a challenge for the application of the neural networks.

Overall, we reject both of our hypotheses: This configuration of LSTM has actually underperformed all the econometric models in this study in both periods.

15. Conclusion and Suggestions

In this study, a set of econometric models namely AR, ARMA, GARCH, GJR-GARCH, and HAR-RV were compared to a specific type of Neural Network model: Long Short-Term Memory, so as to analyze if NN's ability to capture non-linearities in datasets can enhance index return volatility predictions. Two periods characterized by fluctuations and stability were studied: Overall, it was seen that ARMA and the asymmetric version of the GARCH model, namely GJR-GARCH, gave the best forecasts in terms of the performance metric MSE for these periods respectively. In both periods, LSTM underperformed all the other models on predicting daily return volatilities of the S&P 500 Index. These results do not necessarily mean that LSTM models are unsuccessful at predicting return volatility, but that the hyperparameter selections are crucial for achieving a good model performance. Further trials should be done with their different combinations to find a better forecasting LSTM model. Furthermore, the inclusion of other types of inputs which can be signaling for return volatility in extra to realized variances is advised to the future researchers. Nesting GARCH family models, ARMA, HAR or their specifications into LSTM is also recommended as it still remains as a field to be enlightened for forecasting return volatility.

Bibliography

Alberg, D., Shalit, H., & Yosef, R. (2008). Estimating stock market volatility using asymmetric GARCH models. *Applied Financial Economics*, *18*(15), 1201-1208.

Andersen, T. G., Bollerslev, T., &Diebold, F. X. (2007). Roughing it up: Including jump components in the measurement, modeling, and forecasting of return volatility. *The review of economics and statistics*, *89*(4), 701-720.

Awartani, B. M., &Corradi, V. (2005). Predicting the volatility of the S&P-500 stock index via GARCH models: the role of asymmetries. *International Journal of Forecasting*, *21*(1), 167-183.

Bekiros, S. D., &Georgoutsos, D. A. (2008). Direction-of-change forecasting using a volatility-based recurrent neural network. *Journal of Forecasting*, *27*(5), 407-417.

Bildirici, M., &Ersin, Ö. Ö. (2009). 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*, *36*(4), 7355-7362.

Bollerslev, T. (2008). Glossary to arch (garch). *CREATES Research paper*, *49*.

Brailsford, T. J., & Faff, R. W. (1996). An evaluation of volatility forecasting techniques. *Journal of Banking & Finance*, *20*(3), 419-438.

Corsi, F. (2009). A simple approximate long-memory model of realized volatility. *Journal of Financial Econometrics*, *7*(2), 174-196.

Domhan, T., Springenberg, J. T., & Hutter, F. (2015, June). Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

Donaldson, R. G., &Kamstra, M. (1996). Forecast combining with neural networks. *Journal of Forecasting*, *15*(1), 49-61.

Donaldson, R. G., &Kamstra, M. (1997). An artificial neural network-GARCH model for international stock return volatility. *Journal of Empirical Finance*, *4*(1), 17-46.

Engle, R. F., & Ng, V. K. (1993). Measuring and testing the impact of news on volatility. *The journal of finance*, *48*(5), 1749-1778.

Fadlalla, A., & Lin, C. H. (2001). An analysis of the applications of neural networks in finance. *Interfaces*, *31*(4), 112-122.

Fischer, T., & Krauss, C. (2018). Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, *270*(2), 654-669.

Foimbert. (2018, July 12). Just try to find a stock market year like 2018, because you won't. Retrieved from https://www.cnbc.com/2018/07/12/bespoke-just-try-to-find-a-stock-market-year-like-2018-because-you-wo.html

Franses, P. H., & Van Dijk, D. (1996). Forecasting stock market volatility using (non-linear) Garch models. *Journal of Forecasting*, *15*(3), 229-235.

Gal, Y., &Ghahramani, Z. (2016, June). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning* (pp. 1050-1059).

Gers, F. A., Schmidhuber, J., & Cummins, F. (1999). Learning to forget: Continual prediction with LSTM.

Glosten, L. R., Jagannathan, R., &Runkle, D. E. (1993). On the relation between the expected value and the volatility of the nominal excess return on stocks. *The journal of finance*, *48*(5), 1779-1801.

Gonzalez Miranda, F., & Burgess, N. (1997). Modelling market volatilities: the neural network perspective. *The European Journal of Finance*, *3*(2), 137-157.

Gurney, K. (1997). An introduction to neural networks, 1997. London: UCL Press.

Hansen, P. R., &Lunde, A. (2005). A forecast comparison of volatility models: does anything beat a GARCH (1, 1)?. *Journal of applied econometrics*, *20*(7), 873-889.

Haykin, S. (1999). *Neural Networks and Learning Machines.* New York: Prentice Hall.

Hochreiter, S., &Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735-1780.

Hochreiter, S. (1998). The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, *6*(02), 107-116.

Is the Stock Market More Volatile Now Than Ever Before? (2019, April 30). Retrieved from https://www.reuters.com/article/idUSWAOA9NUAIRCF192L

Jondeau, E., Poon, S. H., &Rockinger, M. (2007). *Financial modeling under non-Gaussian distributions*. Springer Science & Business Media.

Kamijo, K. I., &Tanigawa, T. (1990, June). Stock price pattern recognition-a recurrent neural network approach. In *1990 IJCNN International Joint Conference on Neural Networks* (pp. 215-221). IEEE.

Kim, H. Y., & Won, C. H. (2018). Forecasting the volatility of stock price index: A hybrid model integrating LSTM with multiple GARCH-type models. *Expert Systems with Applications*, *103*, 25-37.

Kristjanpoller, W., Fadic, A., & Minutolo, M. C. (2014). Volatility forecast using hybrid neural network models. *Expert Systems with Applications*, *41*(5), 2437-2442.

LeCun, Y., Bottou, L., Bengio, Y., &Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278-2324.

Li, L. K., Pang, W. K., Yu, W. T., &Troutt, M. D. (2004). Forecasting short-term exchange rates: A recurrent neural network approach. In *Neural Networks in Business Forecasting*(pp. 195-212). IGI Global.

Liu, Y. (2019). Novel volatility forecasting using deep learning–Long Short Term Memory Recurrent Neural Networks. *Expert Systems with Applications*, *132*, 99-109.

Malhotra, P., Vig, L., Shroff, G., & Agarwal, P. (2015, April). Long short term memory networks for anomaly detection in time series. In *Proceedings* (p. 89). Presses universitaires de Louvain.

Moshiri, S., & Cameron, N. (2000). Neural network versus econometric models in forecasting inflation. *Journal of forecasting*, *19*(3), 201-217.

Müller, U. A., Dacorogna, M. M., Davé, R. D., Olsen, R. B., Pictet, O. V., & Von Weizsäcker, J. E. (1997). Volatilities of different time resolutions—analyzing the dynamics of market components. *Journal of Empirical Finance*, *4*(2-3), 213-239.

Nelson, D. M., Pereira, A. C., & de Oliveira, R. A. (2017, May). Stock market's price movement prediction with LSTM neural networks. In *2017 International Joint Conference on Neural Networks (IJCNN)* (pp. 1419-1426). IEEE.

Ng, A. (?). [The error surface of a neural network model]. *Introduction, Regression Analysis, and Gradient Descent.* Retrieved from http://www.holehouse.org/mlclass/01_02_Introduction_regression_analysis_and_gr.html

de Oliveira, F. A., Nobre, C. N., & Zarate, L. E. (2013). Applying Artificial Neural Networks to prediction of stock price and improvement of the directional prediction index–Case study of PETR4, Petrobras, Brazil. *Expert Systems with Applications*, *40*(18), 7596-7606.

Pagan, A. R., &Schwert, G. W. (1990). Alternative models for conditional stock volatility. *Journal of econometrics*, *45*(1-2), 267-290.

Sang, C., & Di Pierro, M. (2019). Improving trading technical analysis with TensorFlow Long Short-Term Memory (LSTM) Neural Network. *The Journal of Finance and Data Science*, *5*(1), 1-11.

Saxena, S. (2017). Figure depicting the Activation function for ANN. *Artificial Neural Networks(Basics) / Introduction to Neural Networks*. Retrieved from https://becominghuman.ai/artificial-neuron-networks-basics-introduction-to-neural-networks-3082f1dcca8c

Schmidhuber, J., Wierstra, D., Gagliolo, M., & Gomez, F. (2007). Training recurrent networks by evolino. *Neural computation*, *19*(3), 757-779.

Schwedersky, B. B., Flesch, R. C., &Dangui, H. A. (2019). Practical Nonlinear Model Predictive Control Algorithm for Long Short-Term Memory Networks. *IFAC-PapersOnLine*, *52*(1), 468-473.

Smith, S. W. (1997). The scientist and engineer's guide to digital signal processing. San Diego, CA: California Technical Hub.

Tay, F. E., & Cao, L. (2001). Application of support vector machines in financial time series forecasting. *omega*, *29*(4), 309-317.

Taylor, J. C. (1995). Chapter 1: The promise of neural networks. *Neural Networks*. Henley-on-Thames: Alfred Waller in association with UNICOM.

Tkacz, G. (2001). Neural network forecasting of Canadian GDP growth. *International Journal of Forecasting*, *17*(1), 57-69.

Tsay, R. S. (2005). Analysis of Financial Time Series Second Edition. New York: A John Wiley & Sons.

Wang, X., Wu, C., &Xu, W. (2015). Volatility forecasting: The role of lunch-break returns, overnight returns, trading volume and leverage effects. *International Journal of Forecasting*, *31*(3), 609-619.

Wells, N., &Chemi, E. (2016, August 24). A short history of stock market 'flash crashes' and 'freezes'. Retrieved from https://www.cnbc.com/2016/08/24/a-short-history-of-stock-market-crashes.html

Williams, R. J., &Zipser, D. (1995). Gradient-based learning algorithms for recurrent. *Backpropagation: Theory, architectures, and applications*, *433*.

Zhang, G., Patuwo, B. E., & Hu, M. Y. (1998). Forecasting with artificial neural networks:: The state of the art. *International journal of forecasting*, *14*(1), 35-62.

Zhang, G. P. (2003). Time series forecasting using a hybrid ARIMA and neural network model. *Neurocomputing*, *50*, 159-175.

Zhang, G. P. (Ed.). (2004). *Neural networks in business forecasting*. IGI global.

Appendix 1

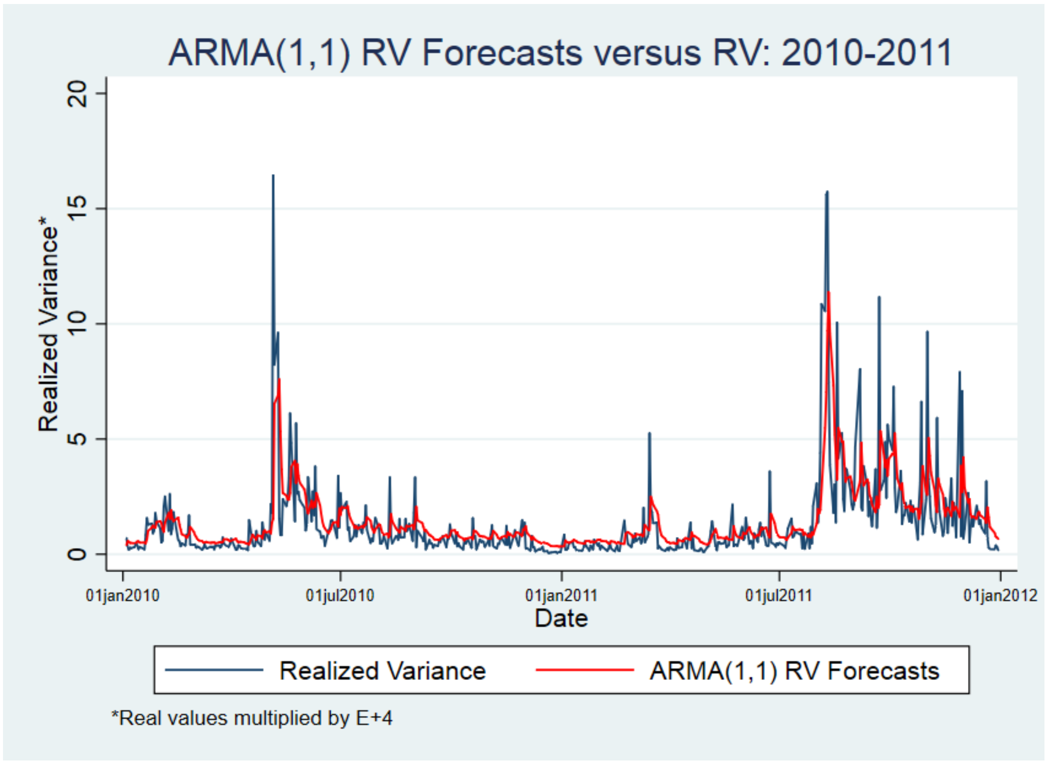


Figure 1A: ARMA(1,1) RV Forecasts versus RV: 2010-2011.

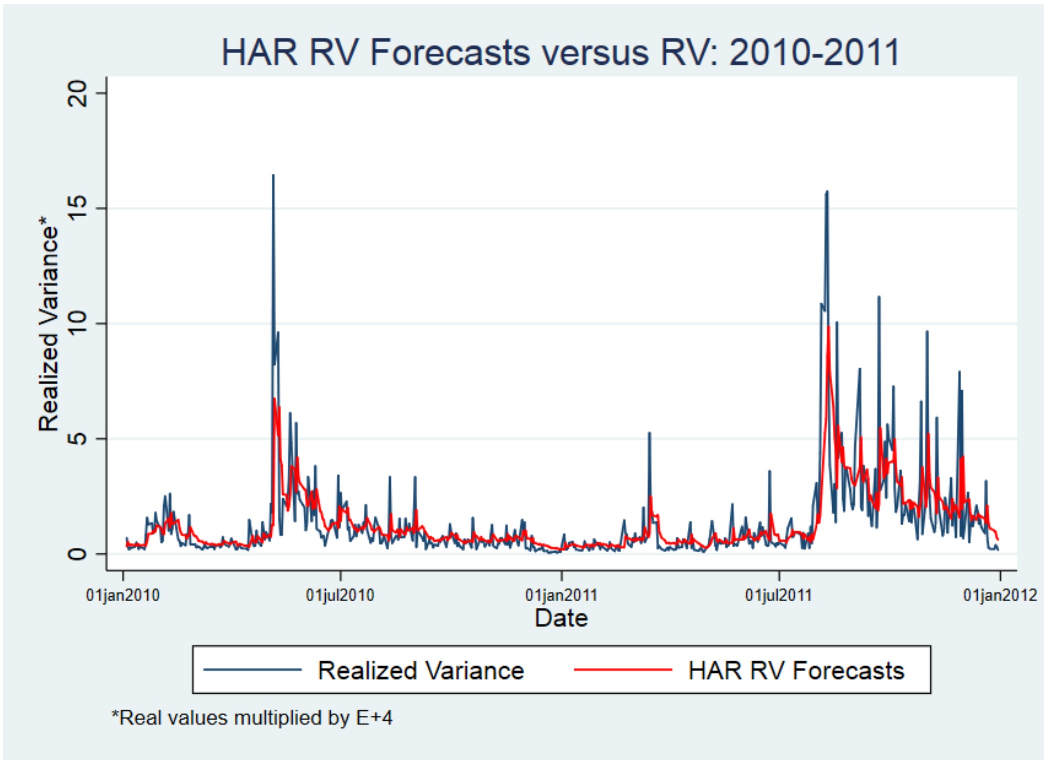


Figure 1B: HAR RV Forecasts versus RV: 2010-2011.

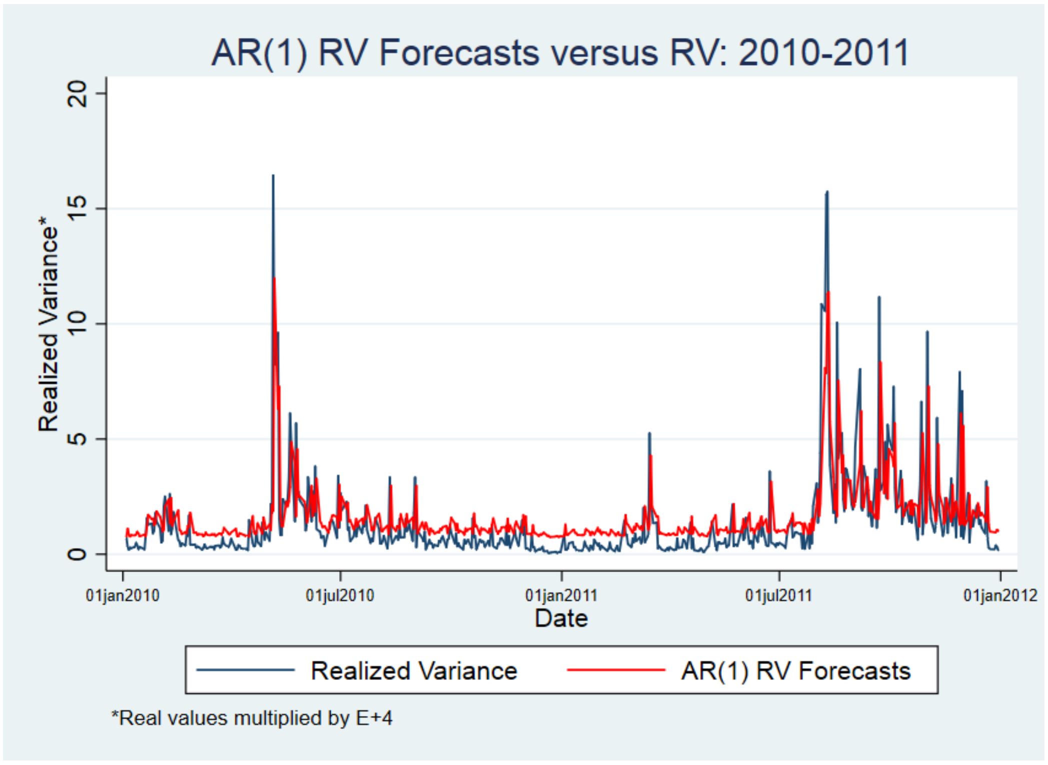


Figure 1C: AR(1) RV Forecasts versus RV: 2010-2011.

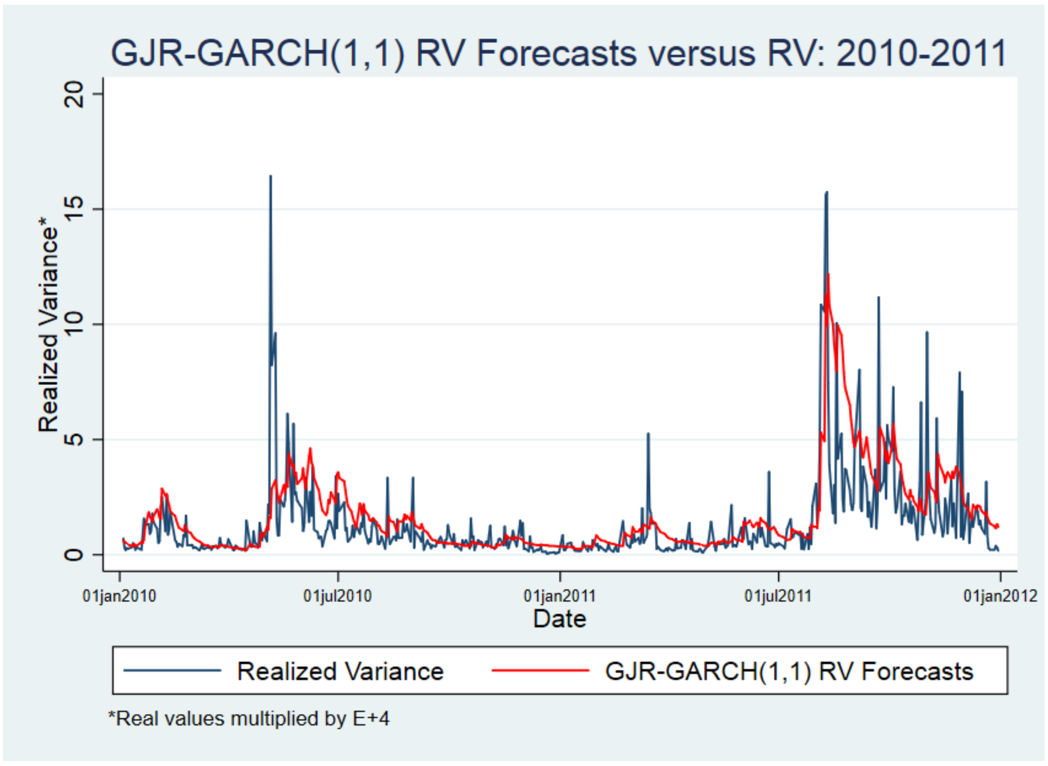


Figure 1D: GJR-GARCH(1,1) RV Forecasts versus RV: 2010-2011.

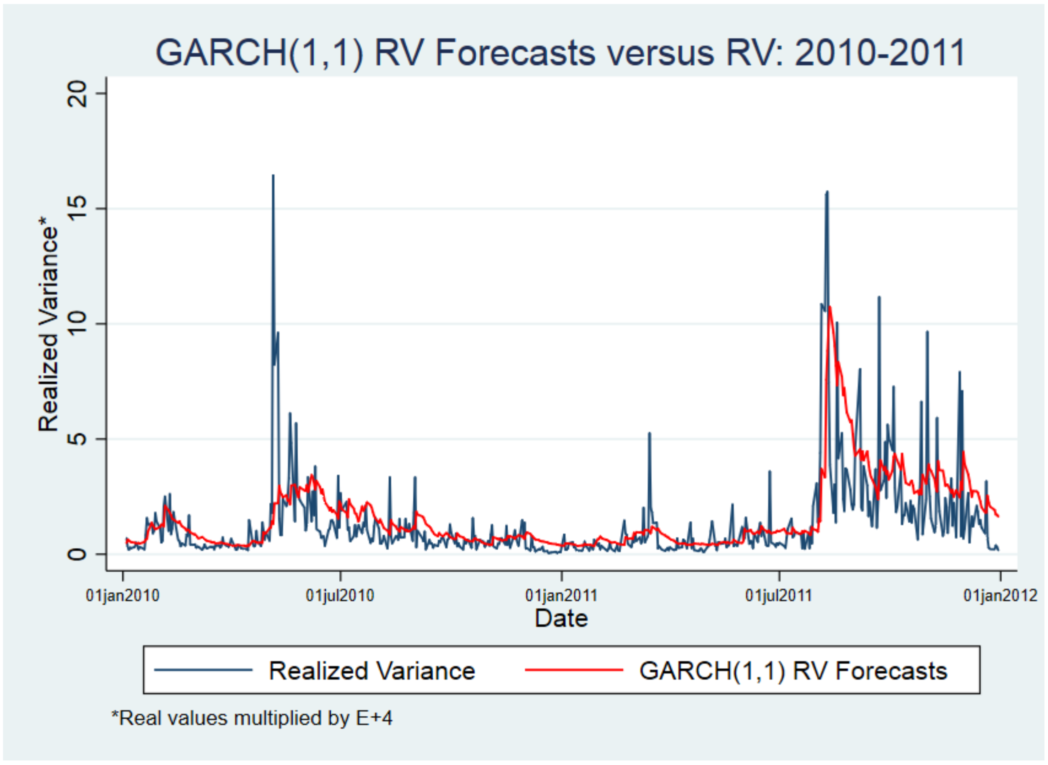


Figure 1E: GARCH(1) RV Forecasts versus RV: 2010-2011.

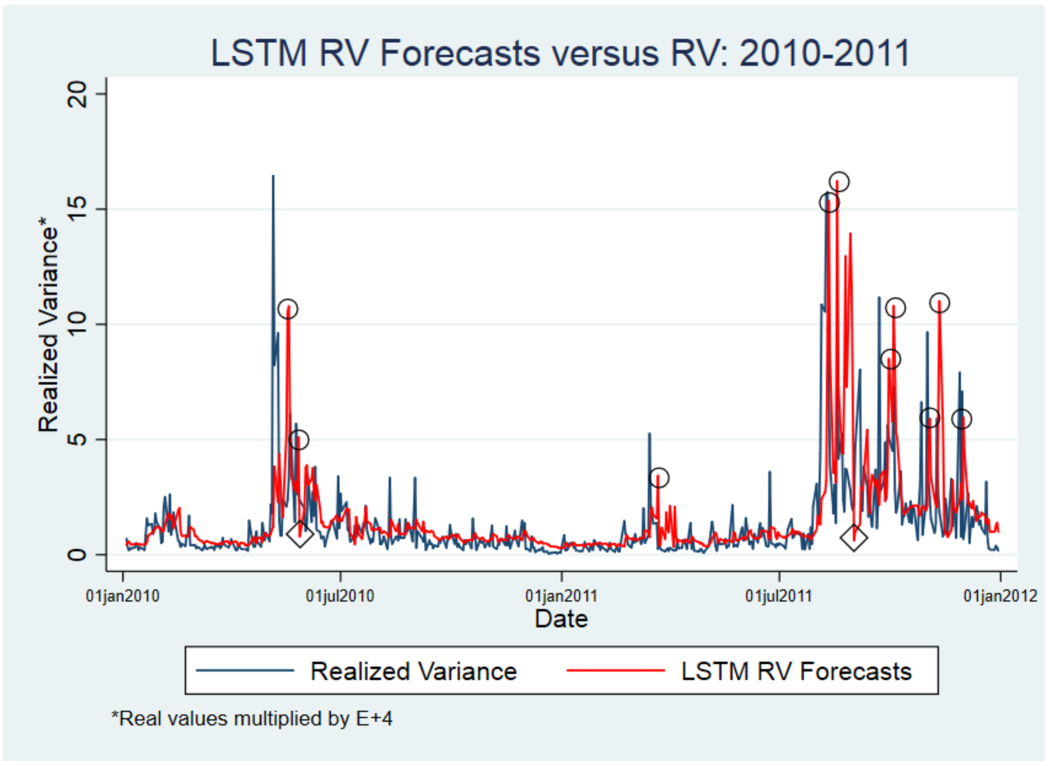


Figure 1F: LSTM RV Forecasts versus RV: 2010-2011. The circles and the diamonds represent the 'delayed look' of the model.

Appendix 2

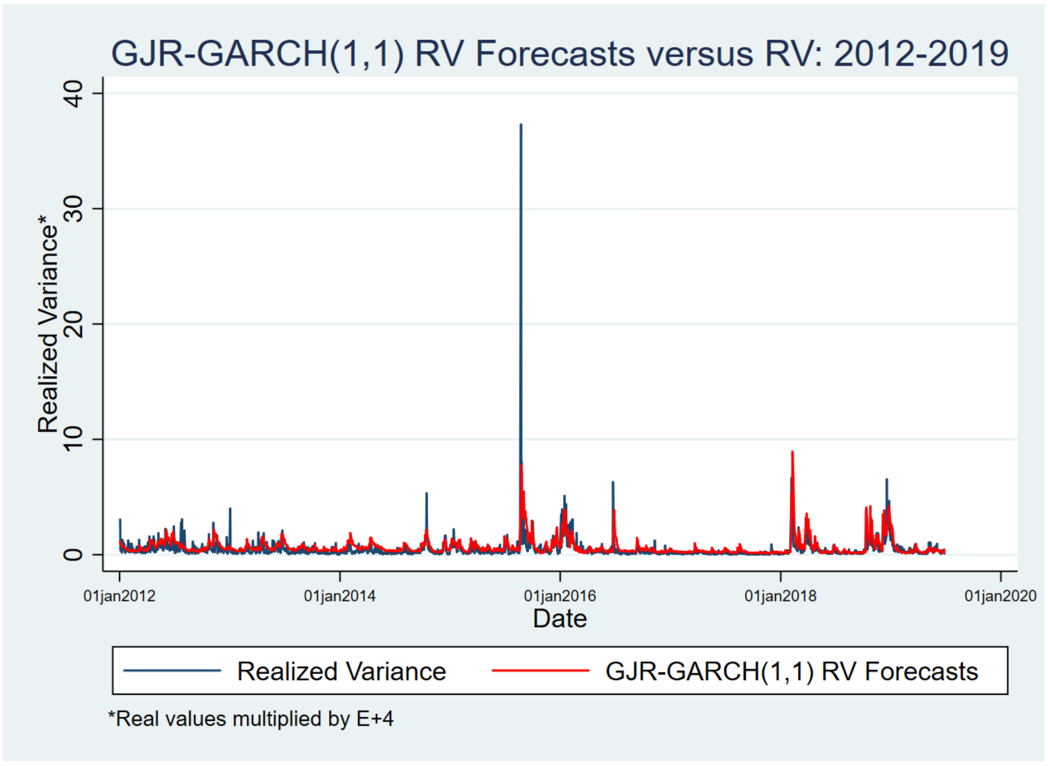


Figure 2A: GJR-GARCH(1,1) RV Forecasts versus RV: 2012-2019.

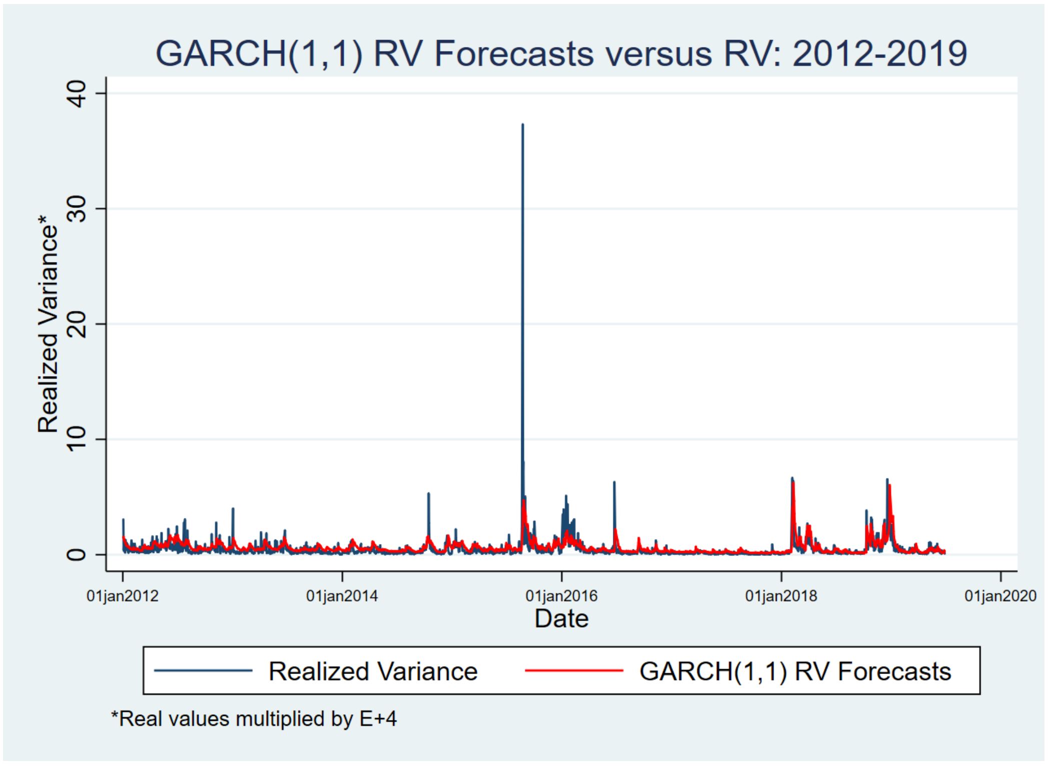


Figure 2B: GARCH(1,1) RV Forecasts versus RV: 2012-2019.

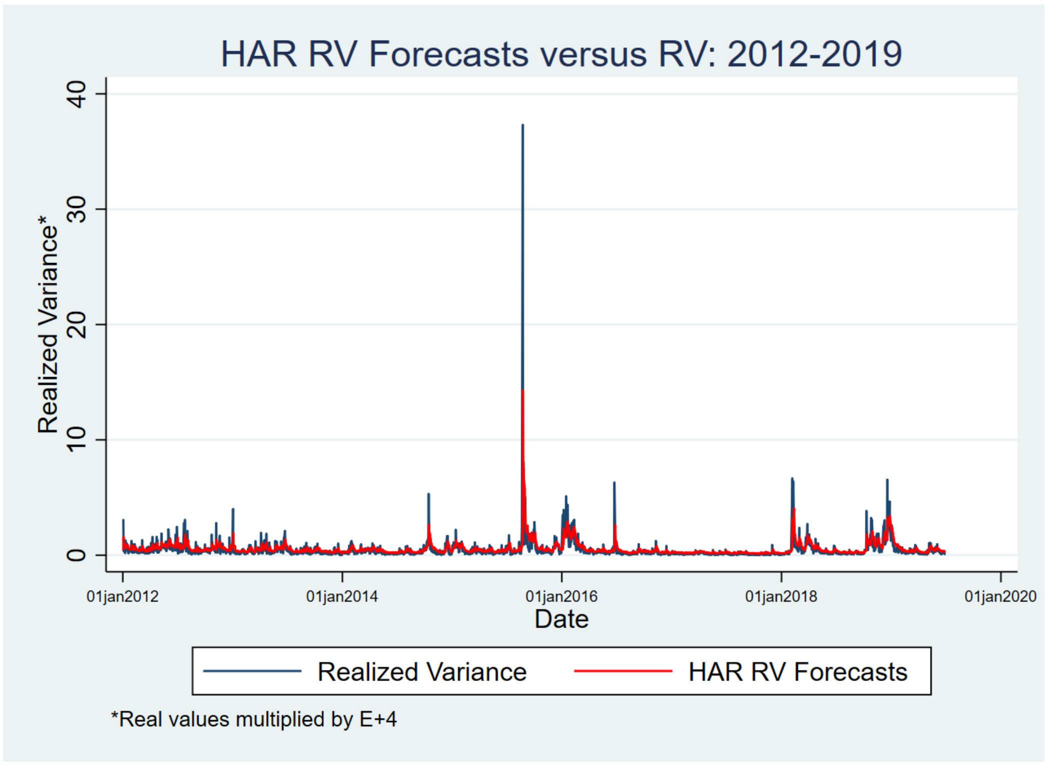


Figure 2C: HAR RV Forecasts versus RV: 2012-2019.

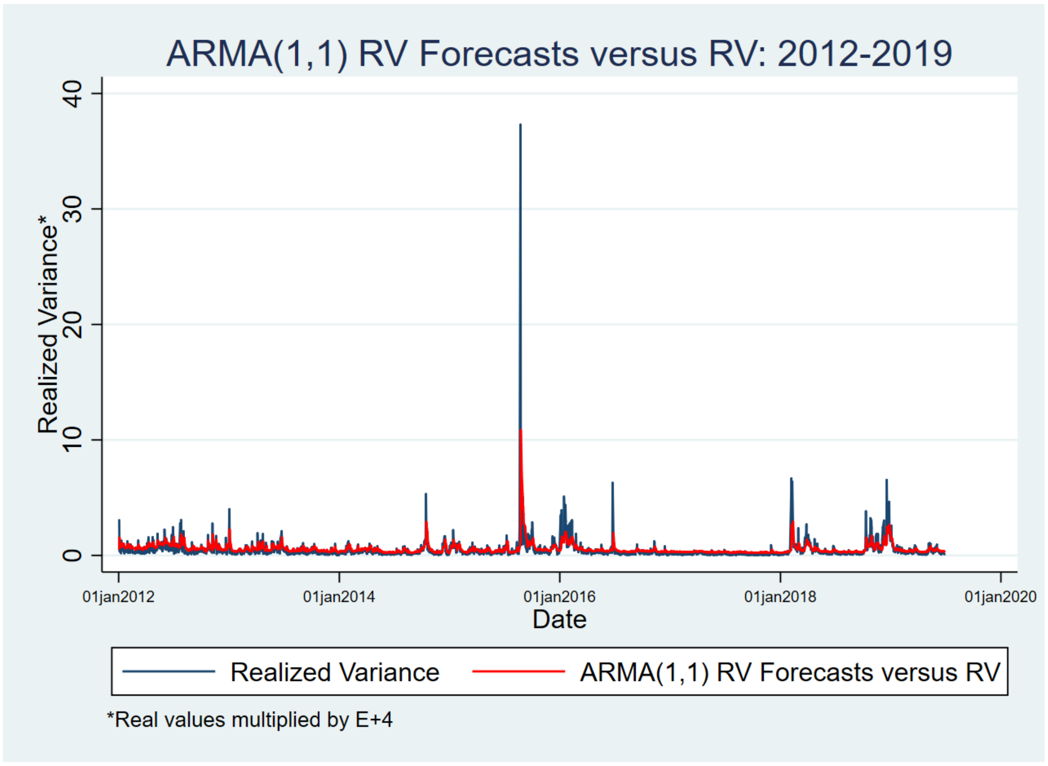


Figure 2D: ARMA(1,1) RV Forecasts versus RV: 2012-2019.

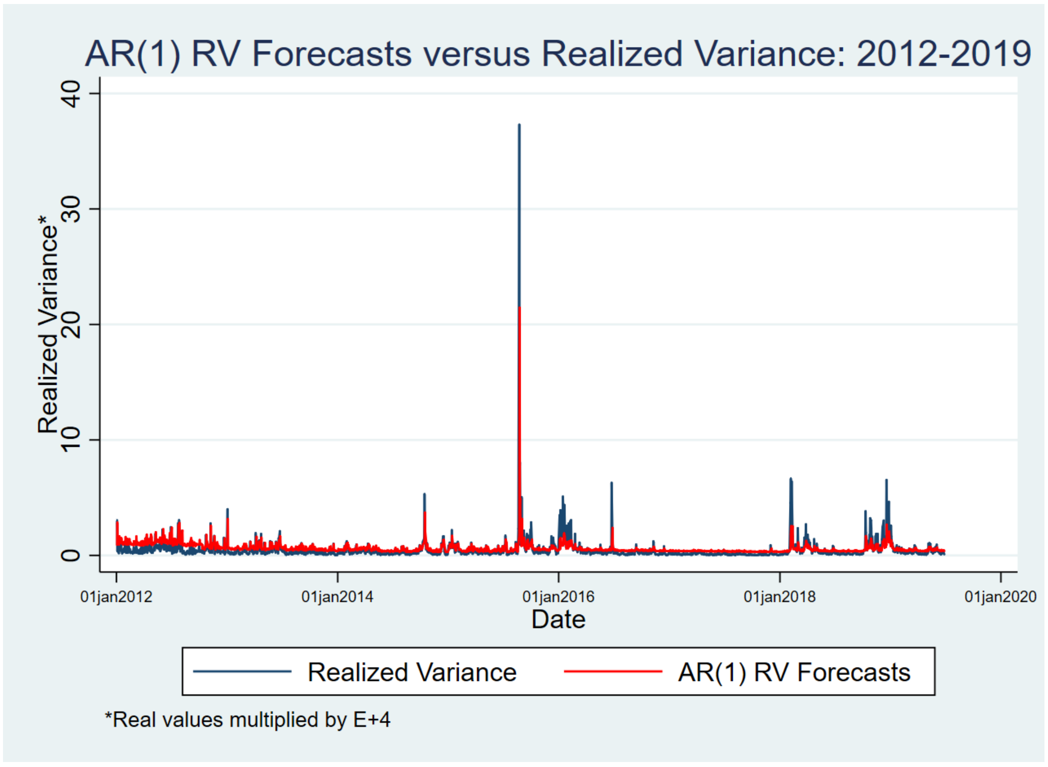


Figure 2E: AR(1) RV Forecasts versus RV: 2012-2019.

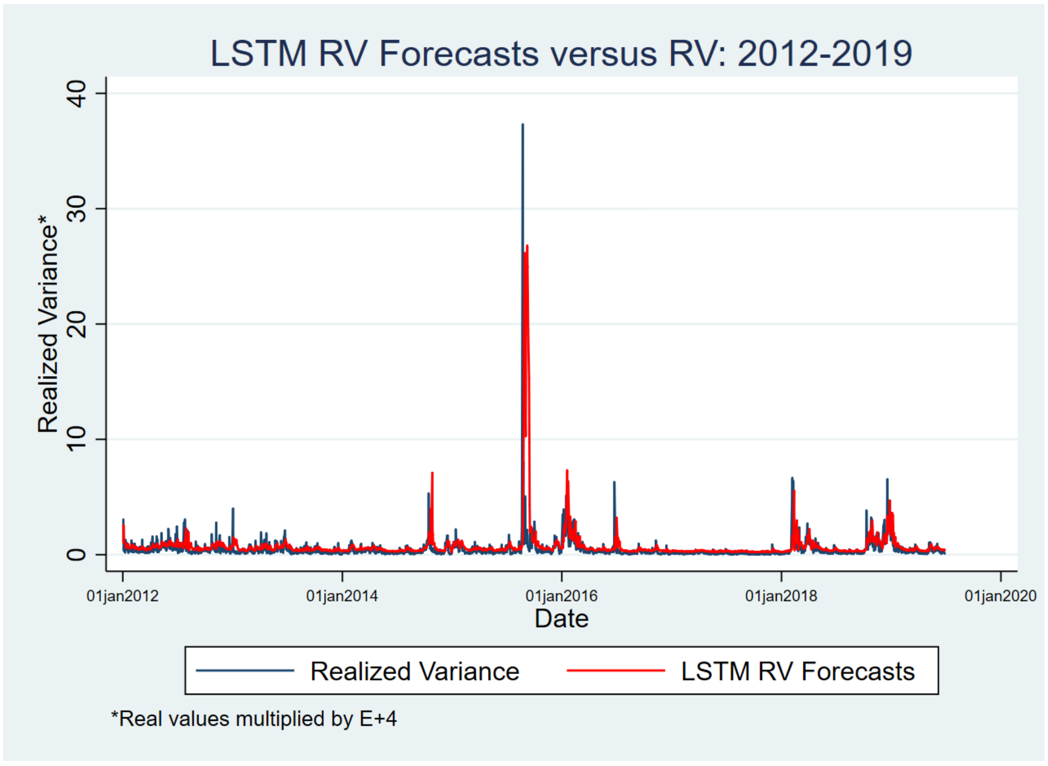


Figure 2F: LSTM RV Forecasts versus RV: 2012-2019.