INDIAN INSTITUTE OF MANAGEMENT CALCUTTA

WORKING PAPER SERIES

WPS No.725 / April 2013

Estimating Stock Index Volatility in India

by

B.B.Chakrabarti
Professor, IIM Calcutta, Diamond Harbour Road, Joka P.O., Kolkata 700 104 India

&

Orr Ben Nathan
PGP Student, IIM Calcutta, D. H. Road, Joka P.O., Kolkata 700 104 India

Estimating Stock Index Volatility in India

B.B. Chakrabarti[1] Orr Ben Nathan[2]

Abstract

Since we find that GARCH(1,1) model fails to provide accurate one day volatility forecasts, we attempt to provide better forecasts using autoregressive neural networks. We hypothesize that due to their nonlinear nature, autoregressive neural networks are more effective in capturing the distribution of realized volatility.

We use radial-basis-function and feed-forward back-propagation autoregressive neural networks to make one day volatility forecasts, and find that an overwhelming majority of simulated networks produce forecasts that are in order of magnitude more accurate than GARCH(1,1) model is. In addition, we learn that in India, realized volatility does not have an especially long memory, and approximately one week of trading sessions holds all the information a researcher may require to generate one day volatility forecasts.

1. Introduction

The importance of estimating volatility needs no introduction. Volatility is a key input to option pricing and portfolio optimization. Moreover, volatility is linked to liquidity in financial markets and to news arrivals at the markets. Thus, estimation of volatility is essential for the pricing of assets and derivatives as well as for trading and hedging strategies. Among the most common techniques to estimate volatility are GARCH models, historical volatility, and implied volatility. Each technique has its strengths and weaknesses, which we briefly discuss below, however, none has managed to consistently exhibit a satisfactory low estimation error when is used to make forecasts. As we show below, GARCH(1,1) model generates a mean-square-error (MSE), which is in order of magnitude larger than the mean-square-errors generated by our simulated autoregressive neural networks.

We found that during the financial crisis of 2008-2009, which included extended periods in which volatility of stocks was record high, one day volatility forecasts made by GARCH(1,1) model exhibited large estimation errors. Hence, we decided to examine whether autoregressive neural networks generate better forecasts of

[1] B.B. Chakrabarti, Professor of Finance
Indian Institute of Management Calcutta
bbc@iimcal.ac.in
[2] Orr Ben Nathan, Graduate
Indian Institute of Management Calcutta
orrb2010@email.iimcal.ac.in

volatility than GARCH(1,1) model does. To make our research comparable in simplicity to GARCH(1,1) model, we use autoregressive neural networks that neither are particularly complex nor take as an input much data.

Unlike other approaches, autoregressive neural networks do not make assumptions regarding the distribution of the data, but adapt to them. We hypothesize that due to their nonlinear nature, autoregressive neural networks are more effective in capturing the distribution of stock index volatility, and as a result produce more accurate forecasts.

We also intend to test how sticky stock index volatility is in India. When a researcher uses an autoregressive model to make forecasts, he or she is required to decide how many lagged variable to include in the model. If volatility has a long memory, that is to say that relatively old trading sessions affect a trading session at the present, then the researcher is required to include more lagged variables. However, if the memory of volatility is not long, the model need not include too many lagged variables. As we do not know how sticky volatility is in India, we aim to investigate it using autoregressive neural networks.

The rest of this paper is organized as follows. In section 2, we discuss the traditional methods to estimate volatility and their limitations. In section 3, we survey several applications of neural networks in financial markets. In section 4, we discuss the methodology used in our research. The data and their characteristics are introduced in section 5. In section 6, we present out results. Our conclusions and suggestions for further research are provided in section 7.

2. Traditional Methods

2.1. GARCH Models

Perhaps the most popular volatility estimation technique is GARCH(1,1) model. GARCH models use historical data to produce estimates of current and forecast of future volatility. GARCH models recognize that volatility is not constant, and attempt to keep track of its evolution throughout time. GARCH models appear to be effective in explaining autocorrelation in squared returns time series [11]. However, volatility term structure estimated by GARCH models is not usually the same as the actual volatility term structure. To illustrate the point, when the current volatility is higher than the long term volatility, GARCH models estimate a downward sloping volatility term structure, and vice versa. As a result, GARCH models often fail to capture highly irregular phenomena such as wild market fluctuations, and other unanticipated events that can lead to significant term structural changes. In other words, GARCH models often fail to fully capture the fat tails observed in asset returns series.

2.2. Historical Volatility

Historical volatility is defined as the standard deviation of the return provided by an asset's returns. In most estimation endeavors, more data lead to more accuracy, but in the case of historical volatility, more data ignore the fact that volatility does change over time, and data that are old may not be relevant for predicting future volatility. Historical volatility is backward looking, and when a market practitioner uses it, he or she assumes that the past will repeat itself. This assumption has been proven unjustifiable time and time again, and therefore, one may conclude that the use of historical volatility is a bit naive, and does not suit financial market characteristics in effect.

2.3. Implied Volatility

Implied volatility is the volatility implied by option prices observed in the market. Unlike historical volatility, and similar to GARCH models, implied volatility is forward looking in nature. One may perceive it as the market's expectation of future volatility. However, implied volatility is based on Black-Scholes model, which assumes that asset prices follow a lognormal distribution. Two of the conditions for an asset price to follow a lognormal distribution are: (a) the volatility of the asset is constant, and (b) the price of the asset changes smoothly without discontinuities. If these conditions are not satisfied, and they rarely do, there is no theoretical basis to the assumption that the implied volatility does indeed estimate future volatility. In addition, options with different maturities produce different implied volatility estimates, and create what is known as a volatility smile. Consequently, implied volatility does not produce a unique forecast to the latent volatility in the market.

3. Literature Survey

In this section we survey previous applications of neural networks in financial markets. Our intention is not to produce a complete list, and should not be taken as such.

Nabney and Cheng [17] predicted the conditional variances of daily exchange rate data of five currencies using mixture density networks, which combine a multilayer perceptron and a mixture model. They trained the networks using a maximum likelihood approach, and compared their proposed networks with a linear ARIMA model and a conventional neural network trained with a sum-of-squares error function. Their results demonstrated that the mixture density networks method performed best in all currencies tested. They also empirically showed that early stopping had very little effect on generalization performance.

Early stopping is a popular technique in neural networks aimed at avoiding over-fitting. If a neural network over-fits a training dataset, it exhibits poor results when

simulated using different datasets. Instead of building a model that explains a general phenomenon - that is to say generalizes, the model merely 'explains' a specific dataset.

Malliaris and Salchenberger [15] developed a neural network to forecast future implied volatility of S&P 100 Index options. They tested their neural network on seven independent out-of-sample datasets, and used mean absolute deviations, mean-square-errors, and the number of correct direction forecasts to measure the accuracy of their network. They used a feed-forward back-propagation network with three layers, 13 nodes in the input layer, 7 nodes in the hidden layer, and 1 node in the output layer. Malliaris and Salchenberger found that their network provided very accurate estimates of future implied volatility.

Bartlmae and Rauscher [8] forecasted one day DAX Index volatility using autoregressive neural networks. The neural networks used were feed-forward back-propagation single hidden-layer networks with hyperbolic tangent as a transfer function. The networks are very similar to the networks we construct using feed-forward back-propagation architecture. The forecasts generated by the autoregressive neural networks where compared with GARCH(1,1), and were found more accurate for risk management purposes, but not for option pricing. Bartlmae and Rauscher did not try to make point estimation of volatility, and then to use it for option pricing. Instead, to test the effectiveness of autoregressive neural networks, they set value-at-risk limits with volatility estimates generated by autoregressive neural networks and GARCH(1,1), and then compared the number of breaches of each limit.

Kim et. al. [13] argued that over-fitting was useful in complex financial series analysis by predicting the movement of KOSPI Index. They proposed three autoregressive feed-forward back-propagation neural networks with number of nodes in hidden layer equal to number of autoregressive variables, activation function was chosen to be the logistic function. They trained their networks twice; first, with a validation dataset, and second, without a validation dataset. Then Kim et. al. used the errors generated by each trained network to plot its autocorrelation function. They showed that the errors of the models with a validation dataset suffered from autocorrelation more than the models without a validation dataset. Consequently, they concluded that autocorrelation imposed a bigger threat on neural networks which use early stopping to avoid over-fitting than on networks that do not.

Samur and Temur [19] used multi-layers perceptron neural networks to predict the price of call and put options on the S&P 100 Index. They trained their neural networks with 1 to 5 hidden layers, and 1 to 15 neurons in each hidden layer. Then, Samur and Temur evaluated the accuracy of their neural networks using mean-square-errors. Their research found that all neural networks performed best, both for call and put options, with 2 hidden layers. However, the number of neurons in each hidden layer was not fixed, but varied from 2 to 9. They also found that their proposed neural networks performed better when they made forecasts of call options

price than of put options price. A very interesting finding was that volatility of the S&P 100 Index should be omitted as an input to the neural networks. Neural networks without volatility of the S&P100 Index as an input performed better than neural networks with volatility as an input. They hypothesized that since volatility was not observable, but estimated, their estimation of volatility using historical volatility was not fit to the model. Instead, they reckoned, the neural networks learned the impact of volatility on option prices through the effect of other variables.

Malliaris and Salchenberger [16] estimated the market price of options using neural networks, which took as input the inputs of Black-Scholes model. They found that in about half of the cases they examined, the mean-square-error generated by their neural networks was lower than than that generated by Black-Scholes model. They also pointed out that unlike Black-Scholes model, neural networks do not make assumptions regarding the distribution of the returns, and therefore, constitute a more resilient model.

Dutta and Shekhar [9] predicted the rating of corporate bonds using feed-forward back-propagation neural networks with two and three layers. They tested neural networks with both 10 and 6 financial ratios as inputs, and bonds' ratings as targets. Dutta and Shekhar empirically found that their neural networks predicted an issue's rating much better than a comparable regression did.

Salchenberger, Cinar, and Lash [18] used feed-forward back-propagation neural networks to predicted the probability of failure for saving and loan associations. Their neural networks inputs included 5 financial variables that signaled a deterioration in a saving and loan association financial condition. Each variable represented a CAMEL category. Salchenberger et. al. compared their results with results of logit model, and empirically found that their neural networks were at least as good as or better than logit model in predicting saving and loan associations failures.

4. Methodology

We model volatility with autoregressive neural networks. The structure of the autoregressive neural networks is not predetermined. To find the optimal designs, we loop through a wide range of structures. Then, to estimate their efficiency, we make one day volatility forecasts, and calculate the mean-square-errors they generate. We also use the mean-square-errors to compare the performance of the suggested autoregressive neural networks with GARCH(1,1) model as a benchmark model.

In order to generate estimates using autoregressive neural networks, we have to measure the observed volatility in the market. Next, we use the observed volatility to train and parameterize the autoregressive neural networks. Unfortunately, volatility is not observable, but latent, and usually is measured by techniques as mentioned in

section 2. As discussed, all of these approaches, valuable as they may be, have significant weaknesses.
We calculate realized volatility by summing intraday squared returns. In theory, by sampling intraday returns frequently enough, the summation of intraday squared returns approximates the underlying integrated volatility. Differently put, the integral of instantaneous volatility is the volatility measure itself. As a result, for all purposes we can treat volatility as observed [7]. Thus, we use the observed volatility both as input to our autoregressive neural networks, and as their target.

Calculating realized volatility is not as simple as it may first seem. High sampling frequencies introduce a couple of difficulties. First, tick-by-tick quotes are available in unevenly spaced time points. Solving this issue requires interpolation, which by itself is an estimation procedure that introduces noise into the estimation process. Second, it is a well known fact that high frequency quotes suffer from autocorrelation [6].

A good choice of sampling frequency must balance between the aforementioned competing forces. Anderson et al. [1, 2, 3, 5, 6, 7] found that a 5 minutes sampling frequency was high enough to eliminate measurement errors, yet low enough to avoid microstructure biases. Accordingly, we use an approximate 5 minutes sampling frequency to calculate realized volatility. Since our data does not contain equally spaced quotes, if a quote is not available exactly 5 minutes after the last quote, we take the quote immediately after the 5 minutes mark. Consequently our sample of returns is not equally spaced, but contains small deviations of a couple of seconds that we assume to be insignificant. We are reluctant to use interpolation to avoid estimation errors that result from considering unobservable factors. The realized volatility found is used to train the autoregressive neural networks both as input in the form of lagged variables and as target, and again to calculate the error of the estimation using the test dataset.

We break down the dataset to three independent subsets: training, validation, and test datasets. The training dataset includes 365 observations, and the validation and test datasets include 45 observations each. To render the datasets independent, we keep a buffer of 10 trading sessions. Since we do not consider, for complexity reasons, autoregressive neural networks with more than 10 lagged variables, the training, validation, and test datasets we create do not overlap, and therefore, are independent.

To build our proposed autoregressive neural networks, we use two different architectures: radial-basis-function networks, and feed-forward back-propagation networks. In the next two sections we explain the methodology used in each architecture.

4.1. Radial-Basis-Function

We build a radial-basis-function network with number of mean-square-errors failures, number of autoregressive variables, size of spread of the radial-basis-function neurons, and number of radial-basis-function neurons variable. We use MATLAB's NEWRB function to build our radial-basis-function networks. NEWRB adds radial-basis-function neurons, with e^{-x^2} as a transfer function, to the network until the network achieves a target accuracy using mean-square-error. We control the number of neurons in each network by setting the target mean-square-error to zero and limiting the number of radial-basis-function neurons NEWRB can add.

Our algorithm contains four loops:

The first, and outer-most, loop controls the maximum number of mean-square-error failures in the validation dataset allowed during the training process. When we train a specific structure, we optimize the network using the training dataset, and then simulate it using the validation dataset. We calculate and register the mean-square-error generated by the simulation using the validation dataset, and compare it with the next step's mean-square-error. The next step's network has the same structure, but with an additional radial-basis-function neuron. If the mean-square-error of a step is not smaller than the mean-square-error of the previous step, we call it "mean-square-error failure." For complexity reasons, we allow mean-square-error failures to vary from 1 to 10 in our search for optimal network structures.

This method is called "early-stopping," and is used to prevent over-fitting of networks. However, we use early-stopping in an unconventional way. Usually, early-stopping is used to stop the training process itself. We use it to stop increasing the number of radial-basis-function neurons to the network structures.

The second loop controls the number of lagged variables in the autoregressive vector that is input into a network. We allow the number of lagged variables to vary from 1 to 10. Normally, 10 trading sessions represent two full weeks of trading, and we assume, for complexity reasons, that all the information in historical data that is required to make a one day volatility forecast can be extracted from the last 10 trading sessions.

The third loop controls the spread, which is also known as the radius, of the radial-basis-function neurons. The use of large spreads smoothes the estimated function. However, if the spread is too large, the network requires many neurons to fit the estimated function to the dataset. In a like manner, if we use spreads, which are too small, the networks require many neurons to fit the estimated function to the dataset, and in addition, the network may not generalize well.

In search of optimal network structures, we allow the spread of radial-basis-function neurons to vary from 0.01 to 1.14 in steps of 0.01. We choose 1.14 as an upper limit, because the range of the volatility series used in the dataset is 1.1354, and hence, a radial-basis-function neuron with a spread of 1.14 covers the whole range.

The fourth, and inner-most, loop controls the number of radial-basis-function neurons in a network structure. For a given number of mean-square-error failures allowed, lagged variables in a autoregressive vector, and spread size, we first train the network using the training dataset with one radial-basis-function neuron. Then, we simulate the trained network using the validation dataset, and calculate its mean-square-error. We add a radial-basis-function neuron, and repeat the training exercise. If a new mean-square-error decreases, we continue to add neurons. On the other hand, if a new mean-square-error fails to decrease, we check how many mean-square-error failures have occurred, and according to the number of mean-square-error failures allowed, we decide whether to add another radial-basis-function neuron, or to stop adding radial-basis-function neurons. Having stopped the training of the network, we simulate the trained network using the test dataset and calculate its mean-square-error.

In search of the optimal network structures, we cap the number of radial-basis-function neurons at 45. We choose 45 since the strength of both the validation dataset and the test dataset is 45, and a network with 45 radial-basis-function neurons can perfectly fit a dataset of 45 observations. Hence, we do not find a justification to fit a radial-basis-function network with more than 45 neurons to a 45 observation dataset.

4.2. Feed-Forward Backpropagation

To find the most efficient feed-forward back-propagation network structures, we use a similar technique to the one we use to find the most efficient redial-basis-function network structures. We first build a single hidden layer network with tan-sigmoid as a transfer function. The networks use a gradient descent with momentum weight and bias learning function. We use MATLAB's default setting of learning rate at 0.01 and of momentum constant at 0.9.

The number of the neurons in the hidden layer is variable, and so is the number of variables in the autoregressive vector. For the same reason we mention in section 4.1., we cap the number of neurons in the hidden layer at 45.

In the case of feed-forward back-propagation networks, the outer loop controls the number of lagged variables in the autoregressive vector. Since we do not know the exact specification of the model, we allow the number of lagged variables to vary from 1 to 10. As explained in the section 4.1., an autoregressive vector with 10 lagged variables encompasses two full weeks of trading sessions. For computational reasons, we assume that volatility does not have a memory of more than two trading weeks. Fortunately, the results support our presumption.

The inner loop controls the number of neurons in the hidden layer. We start with a single neuron in the hidden layer. Using the training dataset, we train the network with MATLAB's TRAINBR function. TRAINBR applies the Levenberg-Marquardt

optimization algorithm to update the weights and biases. The algorithm puts in use a Bayesian regularization that produces networks that generalize well. We allow the training process to run until it converges to a solution. Consequently, the training process results in solutions that are in order of magnitude unique.

To evaluate the feed-forward back-propagation network structures, we simulate the trained networks using the validation dataset and the test dataset to calculate their mean-square-errors. Having trained all 450 possible structures (10 lags times 45 neurons), we use the series of validation mean-square-errors to determine the number of mean-square-error failures as defined earlier.

5. Data

A vector of realized volatilities of the SENSEX Index[3] is used both as input vector to the autoregressive neural networks, and as target vector for training, validating, and testing purposes. The realized volatility vector includes 485 trading sessions from January 1st, 2008 to December 31st, 2009. We estimate realized volatility of a trading session by sampling intraday returns of SENSEX Index once in approximately 5 minutes from high frequency data provided by Bombay Stock Exchange[4] and Bloomberg. Then, to arrive at the realized volatility estimate, we sum the squares of the returns as suggested by Anderson et al. [1, 2, 3, 5, 6, 7]. As we mentioned in Section 4, the summation of intraday squared returns approximates the underlying integrated volatility.

As can be seen in Table 1, realized volatility varies significantly from 8.81% p.a. to 122.35% p.a., with a mean of 27.15% p.a. and a median of 24.92% p.a. The fat right tail seen in Figure 2 results in realized volatility having a mean that is greater than its median. In addition, although the standard deviation may seem tamed at 13.36%, the dataset contains a 7.1 sigma event (October 27th, 2008) that according to a normal distribution, is supposed to occur once in almost 7.6 Billions years, a 6.3 sigma event (January 22nd, 2008) that according to a normal distribution, is supposed to occur once in almost 26 Millions years, two 5 sigma events, and other events that according to a normal distribution, are supposed to occur very rarely. These events raise our suspicion that it is naive to assume that realized volatility of stock index in India follows a normal distribution.

Table 1: Data Statistics of Realized Volatility Dataset

[3] SENSEX Index is a value-weighted index composed of 30 largest and most actively traded stocks on the Bombay Stock Exchange representing 12 major sectors.

[4] Bombay Stock Exchange is the oldest stock exchange in Asia. Today, BSE is number 1 in the world in the number of listed companies and number 5 in the world in handling of transactions through its electronic trading system. As of July, 2009, the total market capitalization of companies listed on BSE is more than USD Trillion 1. On July 31, 2009, BSE had almost 5000 listed companies.

Min	0.0881	Median	0.2492
Max	1.2235	Standard Deviation	0.1336
Mean	0.2715	Range	1.1354

Figure 1 exhibits that realized volatility spikes at few turbulent periods such as the announcement of the general election results in India on May 16th, 2009 (3rd spike), and is significantly lower at most of the other periods. Figure 1 may suggest that realized volatility is non-stationary throughout time. To test for non-sationarity, we run an augmented Dickey-Fuller test, and get a test statistic of -3.7733 while the critical value is -1.9411. Hence, the p-value clocks at 0.001, and consequently, we reject the null hypothesis that the realized volatility series is non-stationary.

Irrespective of the Dickey-Fuller test, we do not perceive non-stationarity as a major concern. Supposedly, non-stationarity hinders a model's capability to accurately make forecasts as the parameters of the underlying distribution change throughout time. The accuracy of the forecasts we seek to make is captured by the mean-square-error used to measure the accuracy of our autoregressive neural networks. If, in fact, non-stationarity does prevent us from making accurate forecasts, the mean-square-errors of our autoregressive neural networks is expected to be higher than the mean-square-error of a benchmark model such as GARCH(1,1). On the other hand, if the mean-square-error generated by our autoregressive neural networks is found to be lower than the benchmark model's mean-square-error, then non-stationarity is, in practice, irrelevant.

The reader can see in Figure 1 that there are 3 major spikes in realized volatility in the timeframe examined. The first occurred on January 21st, 2008 and January 22nd, 2008. On January 21st, 2008 the SENSEX Index experienced its highest ever one day loss of 1,408 points. Media outlets explained that *"investors panicked following weak global cues amid fears of a recession in the US."*[5] On January 22nd, 2008, the SENSEX Index hit its lower circuit breaker in less than a minute after the markets had opened at 10:00 AM. Consequently, trading was suspended for an hour. After the market had reopened, it experienced its biggest intra-day fall, only to recover later that day on reassuring announcement made by the Finance Minister of India. The second spike occurred on October 27th, 2008 as SENSEX Index hit a new three years low. At one point SENSEX Index was down 11.5% on the day, but eventually the index closed a mere 2.2% down. The explanation given to the intraday crash was that Foreign Institutional Investors were fleeing the market to limit their exposure to risky assets. The third and last spike, occurred on May 19th, 2009 on the wake of the announcement of the general election results. The surprising win of the Indian Congress Party was perceived by investors as promising political stability and economic reforms that would benefit the stock market.

[5] http://en.wikipedia.org/wiki/BSE_Sensex

Figure 1: Evolution of realized volatility throughout time from January 1st, 2008 to December 31st, 2009

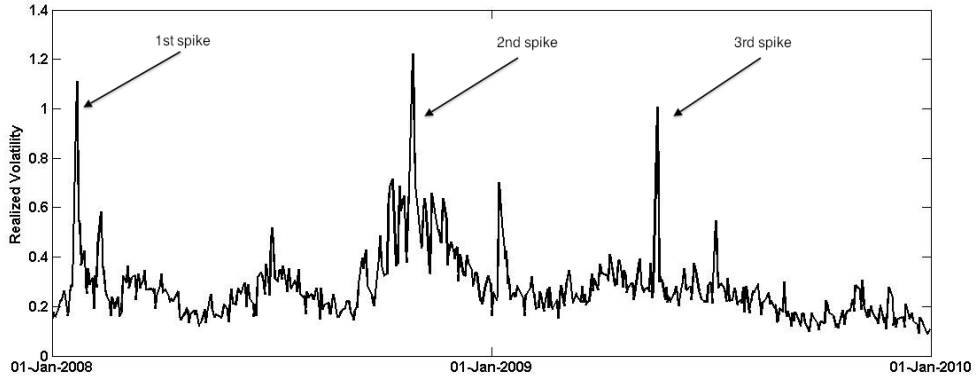
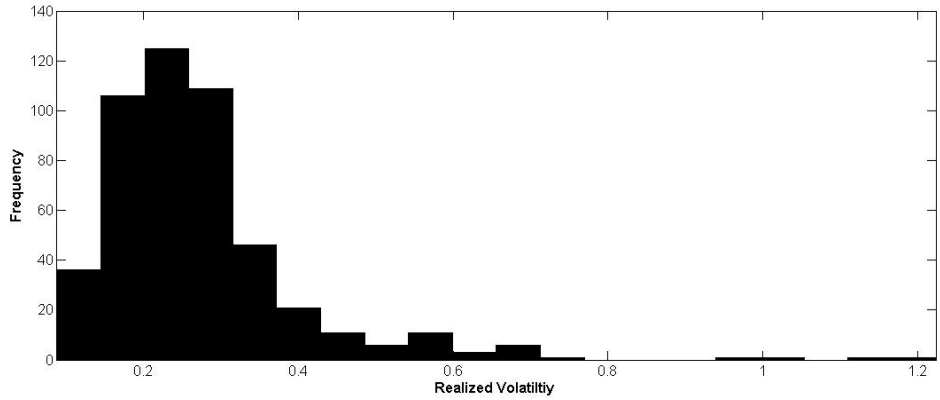


Figure 2 clearly shows that our observations are not sampled from a normally distributed population. To confirm the claim, we conduct a Jarque-Bera test. We find that the test statistic is 3,347.4, while the critical value is 5.85. Hence, the p-value is equal to 0.001, and consequently, we reject the null hypothesis that the realized volatility series is sampled from a normally distributed population. It is seen from Figure 2 that the realized volatility series suffers from a fat right tail. The observed fat right tail inhibits the ability of standard methods to make accurate forecasts of volatility, as simple distribution assumptions are rejected, and necessitate the use of alternative methods such as autoregressive neural networks, which, in their nature, do not assume a probability distribution of the subject matter.

Figure 2: Histogram of annualized daily realized volatility from January 1st, 2008 to December 31st, 2009



As can be seen in Figure 2, the most frequent bin is the 3rd bin with 125 observations, or 25.8% of all the observations. The first 5 bins are almost symmetrical around the 3rd bin, and include 422 observation, or 87% of all the observations. The fat right tale starts to take significant effect from the 6th bin onwards, as a left tail to balance it does not exist.

6. Results

6.1. GARCH(1,1)

We parametrize GARCH(1,1) model using the observations in the training and validation datasets, and with these parameters make one day forecasts. Then, we use the one day forecasts made and the realized volatility in the test dataset to calculate the mean-square-error of GARCH(1,1) model.

GARCH(1,1) model generates a mean-square-error of 0.0092, and a root-mean-square-error of 0.096. The root-mean-square-error found amounts to 57.2% of the average realized volatility in the test dataset. An error in this magnitude indicates that GARCH(1,1) model is ineffective in making one day volatility forecasts.

This result is generated in a test dataset, which is characterized by relatively calm stock market. In the test dataset, realized volatility ranges from 17.3% to 34.9%, while in the whole dataset, encompassing two full years, realized volatility ranges from 8.8% to 122.3%. We suspect that under extreme stock market conditions the error of GARCH(1,1) model may increase further in absolute terms, since, as we discussed in section 2, GARCH models, in their nature, regress to the mean, and do not to capture extreme market events.

6.2. Radial-Basis-Function Networks

Table 2 presents the 10 most accurate radial-basis-function network structures. Columns 1 and 6 consist of the number of mean-square-error failures allowed before the training process is stopped in a specific network structure. Columns 2 and 7 consist of the number of lagged variables in a specific network structure. Columns 3 and 8 consist of the width, or the spread, in percentage points of the radial-basis-function neurons in a specific network structure. Columns 4 and 9 consist of the number of radial-basis-function neurons in a specific network structure. Columns 5 and 10 consist of the mean-square-error generated by a specific network structure.

We find that radial-basis-function networks outperform GARCH(1,1) model significantly as a method to forecast one day volatility. The most accurate radial-basis-function network generated a mean-square-error that is 4.73 times smaller than the mean-square-error generated by GARCH(1,1) model when applied on the test dataset. In addition, among 10,291[6] different radial-basis-function network structures we tested, 9,759 network structures, or 94.8% of all network structures, performed better than GARCH(1,1). As a result, we find very conclusive evidence that radial-basis-function networks provide a very effective tool to make one day volatility forecasts.

Table 2: Most accurate radial-basis-function networks

Failures	Lags	Spread	Neurons	MSE	Failures	Lags	Spread	Neurons	MSE
3	5	25	32	0.00195	9	6	32	34	0.00203
10	5	27	32	0.00196	10	6	26	41	0.00203
9	5	27	31	0.00198	8	6	31	32	0.00204
9	6	31	34	0.00200	1	6	114	1	0.00204
4	6	27	31	0.00203	9	6	26	40	0.00204
GARCH(1,1)									0.00922

Another interesting finding is that the first 76 radial-basis-function networks sorted by their mean-square-error use an autoregressive vector with 5 or 6 lagged variables. An autoregressive vector with 5 lagged variables, usually, represents a full business week. This finding indicates that, in India, volatility has a memory of about one business week, and older data do not hold more information regarding future volatility.

Out of the 532 radial-basis-function networks that we find less accurate than GARCH(1,1) model, 125 are very trivial radial-basis-function networks, which use an autoregressive vector with 1 lagged variable. Their inaccuracy does not surprise us, as they use very little information to make one day volatility forecasts.

[6] Please contact the authors for full results.

Figure 3: Performance surface of radial-basis-function networks with 5 lagged variables[7]

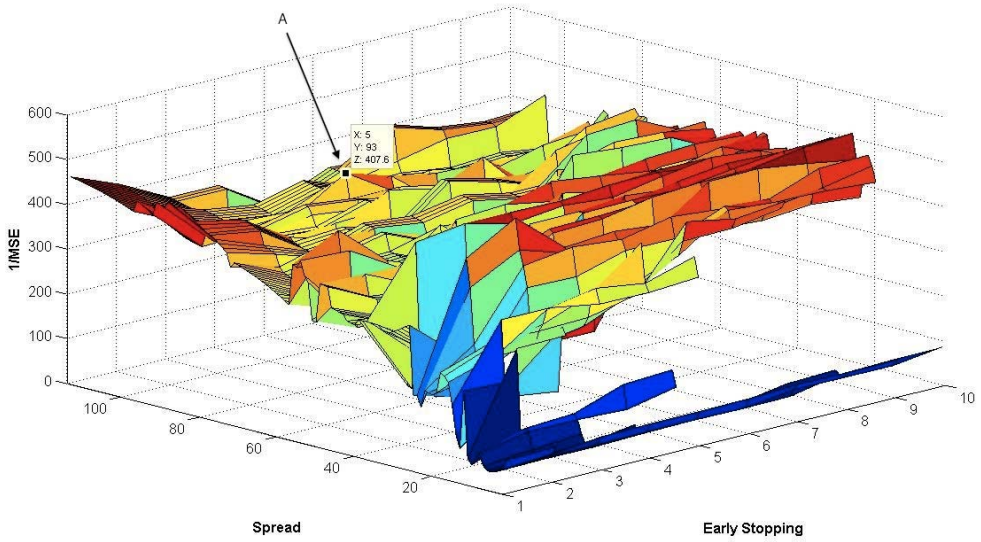
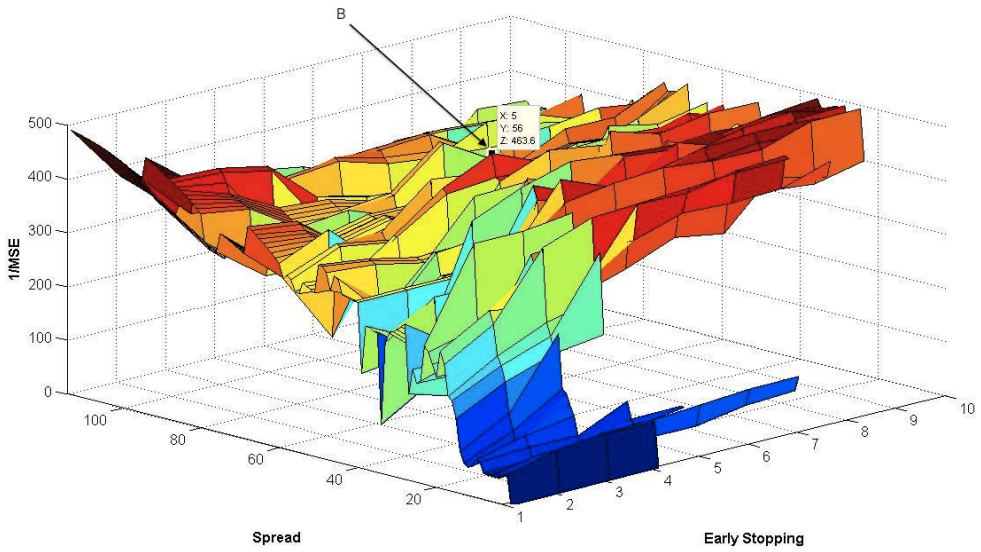


Figure 4: Performance surface of radial-basis-function networks with 6 lagged variables

[7] Figures 3 and 4 are better understood when viewed in colors. Blueish colors signify low 1/MSE and Reddish colors signify high 1/MSE



Since the results indicate that autoregressive neural networks with 5 and 6 lagged variables are the most efficient networks to forecast volatility, we further investigate the evolution of performance as a function of the spread of the radial-basis-function neurons and the number of mean-square-error failures allowed before the algorithm stops the training process. Figures 3 and 4 are a performance surface of all radial-basis-function networks with 5 and 6 lagged variables respectively. On the z-axis we measure 1/MSE of the radial-basis-function network structures. Low mean-square-error entails high 1/MSE, and therefore, relatively more accurate network structures are awarded relatively higher points in the figure. The x-axis marks the number of mean-square-errors failures in the network structure, and the y-axis marks the width of the radial-basis-function neuron of the network structures.

For illustration, in Figure 3, point 'A' represents a network structure with 5 lagged variables, 5 mean-square-error failures, radial-basis-function neuron spread of 0.93, and 1/MSE of 407.6. In Figure 4, point 'B' represents a network structure with 6 lagged variables, 5 mean-square-error failures, radial-basis-function neuron spread of 0.56, and 1/MSE of 463.6.

In both cases of 5 and 6 lagged variables, it is clear, from Figures 3 and 4, that too few failures weigh on performance as well as too small spreads. In the case of spreads, too large spreads also hurt the performance of the networks. The findings regarding the spread are expected due to its functionality in the radial-basis-function neurons.

Identifying a pattern in the results is important, because a trend indicates that the results of our research are not arbitrary, and are likely to repeat when simulated using a different and independent dataset altogether. If, on the other hand, the results exhibit no pattern, and the distribution of performance on the spread and

mean-square-error failures plane is arbitrary, we would fret that the results of our research are spurious, and would not repeat when simulated using a different dataset.

Figures 5 and 6 provide charts of summation of 1/MSE across radial-basis-function neuron spreads of network structures with 5 and 6 lagged variables respectively. That is to say, for a given radial-basis-function neuron spread, we sum 1/MSE of all network structure with number of mean-square-error failures from 1 to 10.

Figure 5: Summation of 1/MSE across spreads of radial-basis-function neurons of network structures with 5 lagged variables

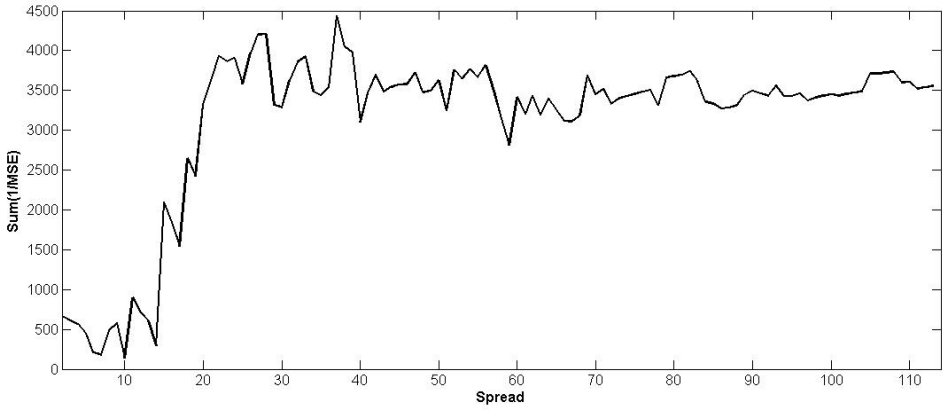
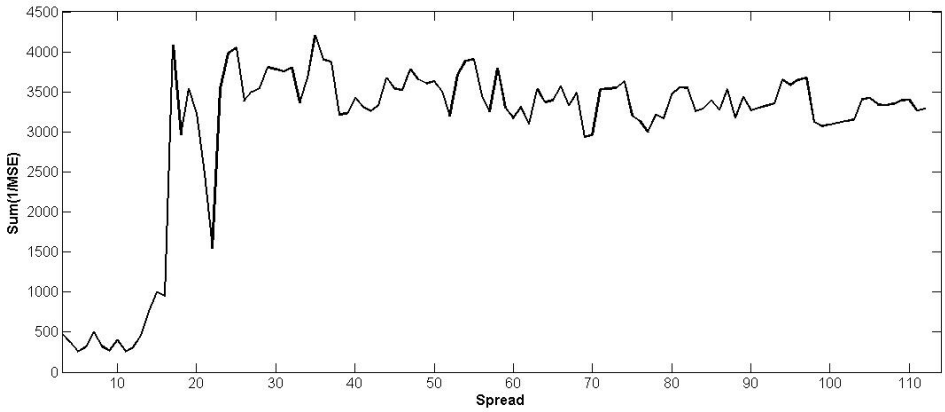


Figure 6: Summation of 1/MSE across spreads of radial-basis-function neurons of network structures with 6 lagged variables



From Figures 5 and 6, a further analysis of performance across the spread axis shows that increasing the spread affects the networks very rapidly, then stabilizes, and even develops a moderate negative trend. This finding reinforces the indication we see in Figures 3 and 4 that increasing the spread beyond a point has negative effects on performance. In addition, the reader can see, in Table 1, that 9 out of the best 10 radial-basis-function networks have a spread in the range of 0.25 to 0.32.

Figures 7 and 8 provide charts of summation of 1/MSE across number of mean-square-error failures allowed of network structures with 5 and 6 lagged variables respectively. That is to say, for a given number of mean-square-error failures allowed, we sum 1/MSE of all network structure with radial-basis-function neuron spreads from 0.01 to 1.14.

Figure 7: Summation of 1/MSE across mean-square-error failures of radial-basis-function network structures with 5 lagged variables

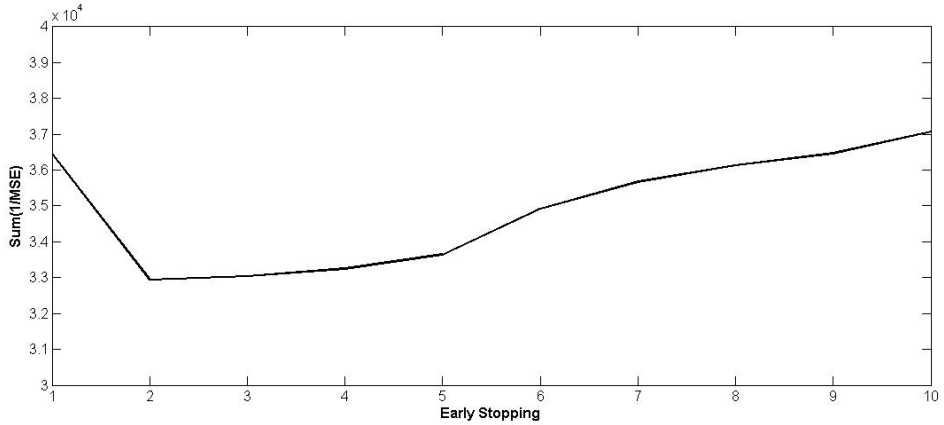
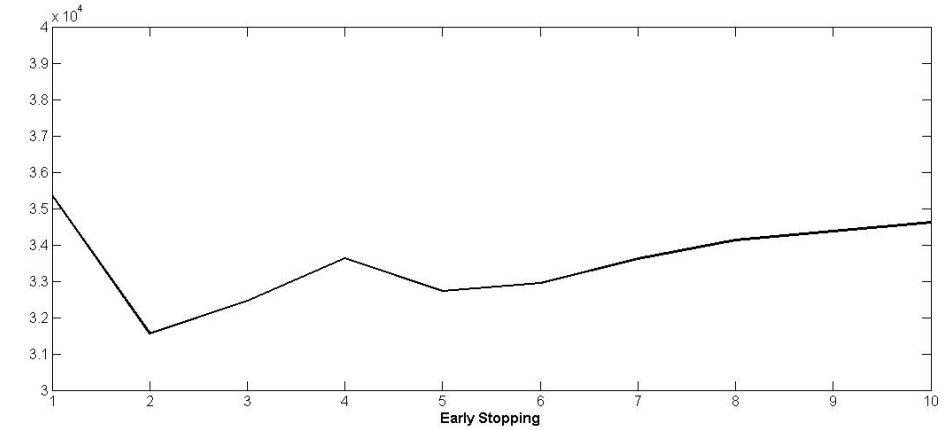


Figure 8: Summation of 1/MSE across mean-square-error failures of radial-basis-function network structures with 6 lagged variables



From Figures 7 and 8, we see that contrary to our finding in the case of radial-basis-function neuron spreads, the performance of network structures first falls, and then starts to improve. Our results show a clear positive trend in performance when we allow the number of mean-square-error failures to exceed 5. Given that in 7 out of our 10 most efficient radial-basis-function network structures, the number of mean-square-error failures allowed is in the range of 8 to 10 (see Table 1), we suggest that the training of radial-basis-function networks is not to be stopped too early. If the number of mean-square-error failures allowed is below 5, the algorithm stops the training too early, and performance achieved is below optimal.

6.3. Feed-Forward Back-Propagation Networks

Table 3 presents the 10 most accurate feed-forwards back-propagation network structures. Columns 1 and 5 consist of the number of lagged variables in a specific network structure. Columns 2 and 6 consist of the number of neurons in the hidden

layer in a specific network structure. Columns 3 and 7 consist of the number of mean-square-error failures allowed in a specific network structure. Columns 4 and 8 consist of the mean-square-error generated by a relevant network structure.

From Table 3 we find that similarly to the case of radial-basis-function networks, feed-forward back-propagation networks outperform GARCH(1,1) model significantly, although their accuracy is lower than the accuracy of radial-basis-function networks. In addition, all 450 feed-forward back-propagation networks that we test are more accurate than GARCH(1,1) model. For illustration, the least accurate feed-forward back-propagation network generates a mean-square-error of 0.0048, almost twice as accurate as GARCH(1,1) model that generates a mean-square-error of 0.0092.

Table 3: 10 most accurate feed-forward back-propagation networks

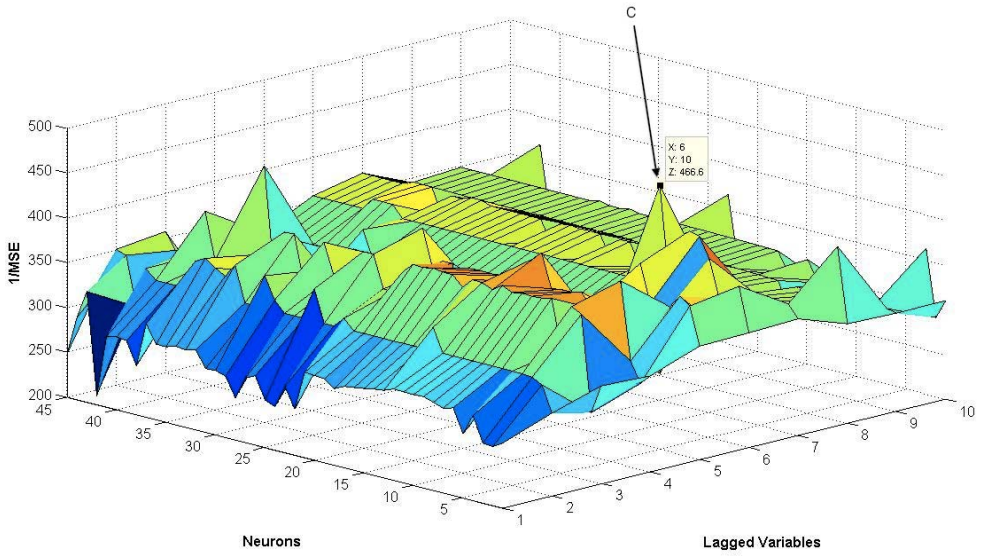
Lag	Neurons	Failures	MSE	Lag	Neurons	Failures	MSE
4	12	7	0.00242	3	11	6	0.00255
4	4	2	0.00247	3	6	3	0.00255
4	13	8	0.00248	3	18	9	0.00255
6	5	3	0.00251	3	20	10	0.00256
8	2	1	0.00254	3	16	8	0.00256
GARCH(1,1)							0.00922

We also find that, once more, autoregressive neural networks perform best without many autoregressive variables, as out of the first 21st networks sorted by mean-square-error, only one uses more than 6 lagged variables (the 5th most accurate feed-forward back-propagation network uses 8 lagged variables). This finding confirms our conclusion that, in India, volatility has a memory of about one week of trading sessions, and when one construct an autoregressive neural network to estimate volatility, one can limit oneself to about a week of lagged information.

Figure 9[8] is a performance surface of all feed-forward back-propagation network structures we examine. On the z-axis we measure 1/MSE of the radial-basis-function network structures. The x-axis marks the number lagged variable in a network structure, and the y-axis marks the number of neurons in the hidden layer of a network structure. For illustration, in Figure 9, point 'C' represents a network structure with 6 lagged variables, 10 neurons in the hidden layer, and 1/MSE of 466.6.

Figure 9: Performance surface of feed-forward back-propagation networks

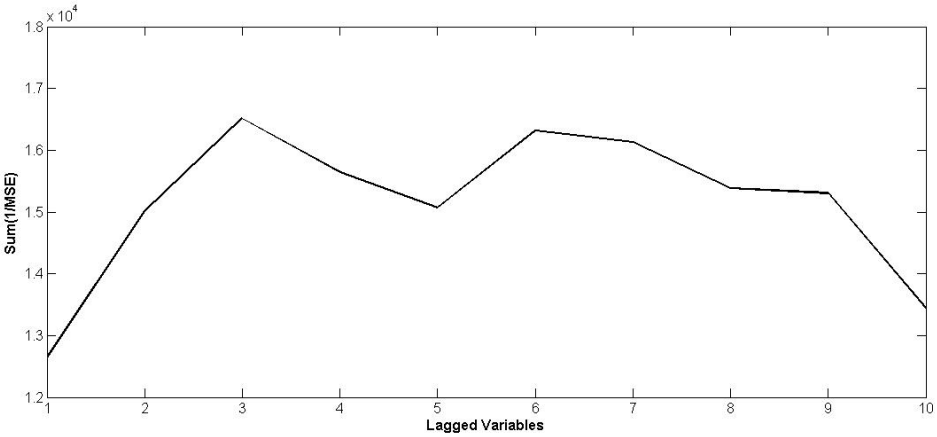
[8] Figure 9 is better understood when viewed in colors. Blueish colors signify low 1/MSE and Reddish colors signify high 1/MSE



As evident by Figure 9, most of the capacity to make one day volatility forecasts using feed-forward back-propagation networks is gained quickly in terms of autoregressive variables and neurons in the hidden layer. Moreover, once the ability to forecast is achieved, it stays stable across lags and neurons. Simply put, highly complex feed-forward back-propagation networks do not add accuracy to the forecasts. Since the additional complexity in terms of computation is not compensated with better accuracy, it is not justified.

Figure 10 provides a chart of summation of 1/MSE across number of lagged variables in a feed-forward back-propagation network structure. That is to say, for a given number of lagged variables, we sum 1/MSE of all feed-forward back-propagation network structures with neurons in the hidden layer from 1 to 45.

Figure 10: Summation of 1/MSE across lagged variables of feed-forward back-propagation network structures



In Figure 10, we see that the accuracy of feed-forward back-propagation networks improves dramatically with the addition of the first 2 lagged variables to the autoregressive vector. However, if we include more than 6 lagged variables, the performance of the networks begins to deteriorate. Figure 10 reinforces our earlier assertion that about a week of trading session contains all the information an autoregressive neural network requires to make an accurate one day volatility forecast. Moreover, if one insists on including excess lagged variables in the autoregressive vector, one's network performance deteriorates. Excess lagged variables do not have a neutral effect on performance, but a negative one.

Figure 11 provides a chart of summation of 1/MSE across number of neurons in the hidden layer of a feed-forward back-propagation network structure. That is to say, for a given number of neurons in the hidden layer, we sum 1/MSE of all feed-forward back-propagation network structures with lagged variables from 1 to 10.

Figure 11: Summation of 1/MSE across neurons of feed-forward back-propagation network structures

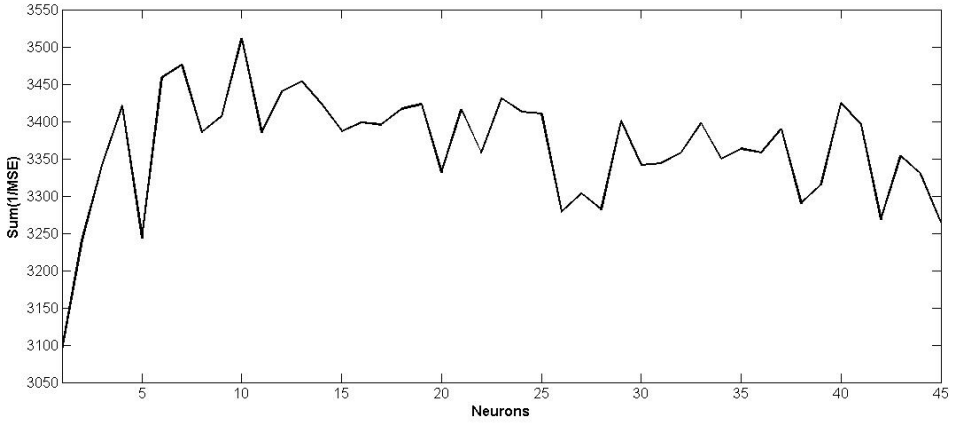


Figure 11 shows that autoregressive neural networks need not be complex in terms of neurons in the hidden layer as well. The addition of redundant neurons not only hurts the performance of feed-forward back-propagation networks, but also consumes computational resources and time. We see, in Figure 11, that the accuracy gained by rendering a network more complex peaks with approximately 10 neurons. The performance of feed-forward back-propagation networks as a function of neurons in the hidden layer shows a negative trend beyond 10 neurons, and hence, is unwarranted.

6. Conclusions and further research

We find strong evidence that autoregressive neural networks make one day volatility forecasts that are significantly more accurate than GARCH(1,1) model does. The superiority of autoregressive neural networks to GARCH(1,1) model is so profound that an overwhelming majority of the configuration we test, both of radial-basis-function networks and feed-forward back-propagation networks, are found to be more efficient than GARCH(1,1) model is.

Our research also suggests that, in India, the number of lagged variables one is advised to use in order to make one day volatility forecast using an autoregressive neural network is in the range of 5-6 trading sessions. If the researcher extends the autoregressive vector beyond the suggested range, the performance of the autoregressive neural network tends to deteriorate.

The dataset used in our research includes very extreme events such as the announcement of the general election results in India on May 16th, 2009 - an event that triggered a leap of more than 17% in SENSEX Index - that significantly affected volatility. In these circumstances traditional models to forecast volatility usually fail. Our success to make accurate one day volatility forecasts under these circumstances proves the resilience and effectiveness of autoregressive neural networks. Many market participants are reluctant to adopt neural networks as a

statistical tool. The main reason being the fact that neural networks do not provide an easy to interpret causal relationship between independent and dependent variables. The absence of an intuitive interpretation also means that market participants do not know under which circumstances neural networks may fail, and therefore, they are deterred from using them.

In this paper, we limit our investigation to autoregressive neural networks. However, one may choose to test the information regarding volatility that other variables, besides lagged variables, may hold. If other variables do hold more information that is not contained in lagged variables, then we expect the accuracy of neural networks, which take these additional variables as inputs, to better the performance we achieve.

Another avenue of further research is to test longer term volatility forecasts. When we test and evaluate our autoregressive neural networks, we make use of merely one day volatility forecasts. However, an intriguing avenue to explore is the efficacy of autoregressive neural networks in building a volatility term structure. Our results convince us that autoregressive neural networks have a high ability to predict the very near future, but how farsighted autoregressive neural networks are, we do not investigate in this paper.

7. References

(1) Andersen, Torben G. and Luca Benzoni. “Realized Volatility.” *Federal Reserve Bank of Chicago* Working Paper No. 2008-14 (2008).

(2) Andersen, Torben G. and Tim Bollerslev. “Answering the skeptics: Yes, standard volatility models do provide accurate forecasts.” *International Economic Review* 39 (1998): 885-905.
(3) Andersen, Torben G., Tim Bollerslev, and Francis X. Diebold. “Parametric and Nonparametric Volatility Measurement.” *NBER Technical Working Papers* (2002).
(4) Andersen Torben G., Tim Bollerslev, Francis X. Diebold, and Paul Labys. “Modeling and Forecasting Realized Volatility.” *Econometrica* 71.2 (2003): 579-625.
(5) Andersen, Torben G., Tim Bollerslev, Francis X. Diebold, and Heiko Ebens. "The Distribution of Realized Stock Return Volatility." *Journal of Financial Economics*, 61 (2001): 43-76.
(6) Andersen Torben G., Tim Bollerslev, Francis X. Diebold, and Paul Labys. "(Understanding, Optimizing, Using and Forecasting) Realized Volatility and Correlation." *New York University, Leonard N. Stern School Finance Department* Working Paper Series 99-061 (1999).
(7) Andersen, Torben G., Tim Bollerslev, Francis X. Diebold, and Paul Labys. “The Distribution of Realized Exchange Rate Volatility.” *Journal of the American Statistical Association*, 96 (2001): 42-55.
(8) Bartlmae, Kai and Folke A. Rauscher. “Measuring DAX Market Risk: A Neural Network Volatility Mixture Approach.” *FFM2000 Conference* (2000).
(9) Dutta, Soumitra and ShashiShekhar. “Bond Rating: A Non-Conservative Application of Neural Network.” *IEEE International Conference on Neural Networks,* 2 (1988): 443-450.
(10) Gonçalves S. and Nour Meddahi. “Bootstrapping Realized Volatility.” *Econometrica* 77.1 (2009): 283-306.
(11) Hull....
(12) “Introduction.” Bombay Stock Exchange Limited, 2010. April 5, 2010 <http://www.bseindia.com/about/introbse.asp>.
(13) Kim, Tae Yoon, Kyong Joo Oh, Chiho Kim, and Jong Doo Do. “Artificial neural networks for non-stationary time series.” *Neurocomputing* 61 (2004): 439-447.
(14) Lehar, Alfred, Martin Scheicher, and Christian Schittenkopf. “GARCH vs. stochastic volatility: Option pricing and risk management.” *Journal of Banking & Finance,* 26.2-3 (2002): 323-345.
(15) Malliaris, Mary and Linda Salchenberger. “Using neural networks to forecast the S&P 100 implied volatility.” *Neurocomputing,* 10 (1996): 183-195.
(16) Malliaris, Mary and Linda Salchenberger. “A neural network model for estimating option prices.” *Applied Intelligence,* 3.3 (2004): 193-206.
(17) Nabney Ian T. and H. W. Cheng. “Estimating conditional volatility with neural networks.” *4th International Conference on Forecasting Financial Markets*, London, May 1997. *Decision Sciences*, 23.4 (1992): 899-916.
(18) Salchenberger Linda M., E. Mine Cinar, Nicholas A. Lash. “Neural Networks: A New Tool for Predicting Thrift Failures.”
(19) Samur, Zeynep ltüzer and Gül Tekin Temur. “The Use of Artificial Neural Network in Option Pricing: The Case of S&P 100 Index Options.” *World Academy of Science, Engineering and Technology,* 54 (2009): 326-331.

(20) Shaikh A. Hamid. “Primer on using neural networks for forecasting market variables.” *Southern New Hampshire University* Working Paper No. 2004-03 (2004).