Department of Financial Mathematics

Comparison of forecasting models for value at risk

Author:
Hafees Adebayo Yusuff

Supervisor:
Prof. Ralf Korn

September 7, 2021

TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

This thesis is written as a requirement for the completion of my degree of Master of Science at Technical University of Kaiserslautern.

Contents

Chapter 1

Introduction

1.1 Motivation

Basel I (Basel Accord) is the agreement reached in Basel, Switzerland, in 1988 by the Basel Committee on Bank on Bank Supervision (BCBS), involving the governors of the central banks of some European countries and the United States of America. This agreement makes recommendations on banking regulation in relation to credit, market and operational risks. It aims to ensure that financial institutions have sufficient capital to meet their obligations and absorb unexpected losses.

For a financial institution, measuring the risk to which it is exposed is an essential task. In the specific case of market risk, one possible method of measurement is to assess the losses that are likely to occur if the price of portfolio assets falls. This is the task of the Value at Risk (VaR). Value at Risk(VaR) is the most common method of measuring market risk. It determines the largest possible loss assuming a α level of significance under normal market conditions at a given point in time.

Many VaR estimation methods have been developed to reduce uncertainty. However, it is of interest to compare these methods and determine the extent to which one VaR estimation approach is preferred over others.

1.2 Literature review

Beder (1995, 1996), Hendricks (1996), and Pritsker (1997), are among the first set of papers in which comparison of value at risk methods were made. They reported that the Historical Simulation performed at least as well as the methodologies developed in the early years, the Parametric approach and the Monte Carlo simulation. These papers conclude that among earlier methods, no approach appeared to perform better than the others (see Abed et all, 2013). The evaluation and categorization of models carried out in the work by McAleer, Jimenez-Martin and Perez-Amaral(2009) and Shams and Sina (2014), among others, try to determine the conditions under which certain models predict the best. Researchers made comparison of models in the time of varying volatility-before the crisis and after the crisis (When there was no high volatility and when volatility was high, respectively). However, this confirms that some models have good predictions before the start of the crisis, but their quality reduces with increased volatility. Others are more conservative during periods of low volatility, but have relatively low amount of errors in the period of crisis (see Buczyński & Chlebus, 2018).

Bao et al.(2006), Consigli(2002) and Danielson(2002), among others, show that "in stable periods, parametric models provide satisfactory results that become less satisfactory during high volatility periods". Sarma et al. (2003), and Danielson and Vries (2000) favour Parametric methods with evidence from their comparison of Historical simulation and Parametric methods.

"Chong(2004), who uses parametric methods to estimate VaR under a Normal distribution and under a Student's t-distribution, finds a better performance under Normality" (see Abed and Benito, 2010). McAleer et al (2009) presented RiskMetrics^{TM} as the best fitted model during high volatility, while Shams and Sina(2014) recognized GARCH(1,1) and GJR-GARCH as better forecasting models. In opposition to the results obtained by McAleer et al (2009), the level of quality of forecasts generated by the RiskMetrics^{TM} model was labelled unsatisfactory by them. However, there is difference in the sample used in their respective studies as the former used that of a developed country (S&P500,USA) while the latter used that of a developing country (TSEM,Iran) (see Buczyński & Chlebus, 2018). Taylor(2020) evaluate Value at Risk models using quantile skill score and the conditional autoregressive model outperformed others.

Attempts have been made to predicts VaR with ANN. VaR estimation on the exchange rate market in the context of ANNs is dealt with in Locarek-Junge and Prinzler (1999), who illustrate how VaR estimates can be obtained by using a USD-portfolio. The empirical outcomes demonstrate an evident superiority of the neural network to other VaR models. Hamid and Iqbal(2004) compared volatility forecasts from neural networks with forecasts of implied volatility from S&P500 index futures options, using the Barone-Adesi and Whaley (BAW) American futures options pricing model. Forecasts from NN outperformed implied volatility forecasts. Similar results are put forth by He et al. (2018), who propose an innovative EMD-DBN type of ANN to estimate VaR on the USD against the AUD, CAD, CHF and the EUR. The authors find positive performance improvement in the risk estimates, and argue that the utilization of an EMD-DBN network can identify more optimal ensemble weights and is less sensitive to noise disruption compared to a FNN. Nevertheless, it is worthwhile to mention that although foreign exchange volatility forecasting through ANNs have gained some attention in the academic field, it still remains a fairly undeveloped area.

All in all, there is no full approval in the evaluation of which models should be used during periods of calm (low volatility), and which ones during crisis (High volatility).

1.3 Thesis Structure

The next chapter of discusses the properties and basic methods to estimate VaR. Subsequent chapters discuss use of Neural Network in Estimating Value at Risk and numerical comparison of the methods with examples. Findings are summarized in the last chapter.

Chapter 2

Value-at-Risk: Concept, properties and methods

2.1 Concept

"Higher volatility in exchange markets, credit defaults, even endangering countries, and the call for more regulation drastically changed the circumstances in which banks have to perform". These situations of uncertainty are called risks and managing them is of great importance to financial institutions (e.g Banks) in order to keep them afloat (see Kremer, 2013). Value at risk measures the losses which may be incurred when the price of the portfolio falls. Hence it is an important measure of risk to financial institutions.

According to Jorion (2007), "VaR measure is defined as the worst loss over a target horizon such that there is a low prespecified probability that the actual loss will be larger'. For example, if a financial institutions says that the daily VaR of its trading portfolio is $2 million at the 99% confidence level, this simply means that under normal market conditions, only 1% of the time, the daily loss will be more than $2 million (99% of the time, its loss will not be more than $2 million). As represented in the mathematical representation below, it can also be stated as the least expected return of a portfolio at time t and at a certain level of significance, α.

Assume $r_1, r_2, ..., r_n$ to be independently and identically distributed(iid) random variables representing financial log returns. Use $F(r)$ to denote the cumulative distribution function, $F(r) = Pr(r_t < r|\Omega_{t-1})$ conditional on the information set Ω_{t-1} available at time t-1. Assume that $\{r_t\}$ follows the stochastic process;

$$\begin{aligned} r_t &= \mu_t + \varepsilon_t \\ \varepsilon_t &= \sigma_t z_t \qquad z_t \sim iid(0,1) \end{aligned} \tag{2.1}$$

where $\mu_t = E[\varepsilon_t|\Omega_{t-1}]$, $\sigma_t^2 = E[\varepsilon_t^2|\Omega_{t-1}]$ and z_t has a conditional distribution function $G(z)$, $G(z) = Pr(z_t < z|\Omega_{t-1})$. The VaR with a given probability $\alpha\ \epsilon(0,1)$, denoted by VaR(α), is defined as the α quantile of the probability distribution of financial returns:
$F(\text{VaR}(\alpha)) = Pr(r_t < \text{VaR}(\alpha)) = \alpha$ or $\text{VaR}(\alpha) = \inf\{v|P(r_t \leq v) = \alpha\}$

One can estimate this quantile in two different ways: (1) inverting the distribution function of financial returns, F(r), and (2) inverting the distribution function of innovations, with regard to $G(z)$ the latter, it is also necessary to estimate σ_t^2. (see Abed and Benito, 2010)

$$\text{VaR}(\alpha) = F^{-1}(\alpha) = \mu + \sigma_t G^{-1}(\alpha) \tag{2.2}$$

Hence, a VaR model involves the specification of $F(r)$ or $G(r)$. There are several method for these estimations. Having explained the concept of Value at Risk, it is however necessary to state some of its properties or attributes.

2.2 Properties

A functional $\tau : X, Y \to \mathbb{R} \cup \{+\infty\}$ is said to be coherent risk measure for portfolios X and Y if it satisfies the following properties:

- Normalization
  $\tau[0] = 0$
  The risk when holding no assets is zero.

- Monotonicity
  if $X \leq Y$ then $\tau(X) \geq \tau(Y)$
  For financial applications, this implies that a security that always has higher return in all future states has less risk of loss.

- Translation invariance
  $\tau(X + c) = \tau(X) - c$
  In effect, if an amount of cash $\varkappa$ (or risk free asset) is added to a portfolio, then the risk is reduced by that amount.

- Positive Homogeneity
  $\tau(cX) = c\tau(X)$ if $c > 0$.
  In effect, if a portfolio or capital asset is, say, doubled, then the risk will also be doubled.

- subadditivity:
  $\tau(X + Y) \leq \tau(x) + \tau(Y)$. Indeed, the risk of two portfolios together cannot get any worse than adding the two risks separately: this is the diversification principle.

Out of the above properties, all but subaddivitivity is not always satisfied by VaR. This is however a disadvantage of value at risk as a risk measure because it might discourage diversification (see for example Acerbi and Tasche, 2002).

2.3 Popular methods for estimating VaR

The estimation of these functions ($F(r)$ or $G(r)$) can be carried out using the following methods:

2.3.1 Historical simulation

The historical simulation involves using past data to predict future. First of all, we have to identify the market variables that will affect the portfolio. Then, the data will be collected on the movements in these market variables over a certain time period. This provides us the alternative scenarios for what may happen between today and tomorrow. For each scenario, we calculate the changes in the dollar value of portfolio between today and tomorrow. This defines a probability distribution for changes in the value of portfolio. For instance, VaR for a portfolio using 1-day time horizon with 99% confidence level for 500 days data is nothing but an estimate of the loss when we are at the fifth-worst daily change.

Basically, historical simulation is extremely different from other type of simulation in that estimation of a covariance matrix is avoided. Therefore, this approach has simplified the computations especially for the cases of complicated portfolio.

The core of this approach is the time series of the aggregate portfolio return. More importantly, this approach can account for fat tails and is not prone to the accuracy of the model due to being independent of model risk. As this method is very powerful and intuitive, it is then become the most widely used methods to compute VaR. However, Historical simulation requires data on all risk factors to be available over a reasonably long historical period in order

to give a good representation of what might happen in the future. As it depends on history, if we run a Historical Simulations VaR in a bull market, VaR may be underestimated. Similarly, if we run a Historical Simulations VaR just after a crash, the falling returns which the portfolio has experienced recently may distort VaR.

2.3.2 GARCH Model

The Generalized Autoregressive Conditional Heteroskedasticity(GARCH) model, proposed by Bollerslev (1986) is a generalization of the ARCH process created by Engle (1982), in which the conditional variance is not only the function of lagged random errors, but also of lagged conditional variances. The standard GARCH model (p, q) can be written as:

$$\begin{aligned} r_t &= \mu_t + \varepsilon_t \\ \varepsilon_t &= \sigma_t \xi_t \end{aligned} \tag{2.3}$$

where r_t = rate of return of the asset in the period t,
μ_t =conditional mean

ε_t = random error in the period t, which equals to the product of conditional standard deviation σ_t and the standardized random error ξ_t in the period t ($\xi_t \sim$ iid(0,1))

In turn, the equation of conditional variance, in the GARCH(p,q) model is assumed to be of the form:

$$\sigma_t^2 = \omega + \sum_{i=1}^{q} \alpha_i \varepsilon_{t-i}^2 + \sum_{i=1}^{p} \beta_i \sigma_{t-i}^2 \tag{2.4}$$

where σ^2 =conditional variance in the period t,
ω = constant ($\omega > 0$)
α_i = weight of the random squared error in the period $t-1$,
β_i = weight of the conditional variance in the period $t-1$,
ε_{t-i}^2= squared random error in the period $t-1$,
σ_{t-i}^2 =variance in the period $t-1$,
q = number of random error squares periods used in the functional form of conditional variance,
p = number of lagged conditional variances used in the functional form of conditional variance (see Buczyński & Chlebus (2018)).

Ruilova & Morettin (2020) "When we use high frequency data in conjunction with GARCH models, these need to be modified to incorporate the financial market micro structure. For example, we need to incorporate heterogeneous characteristics that appear when there are many traders working in a financial market trading with different time horizons. The HARCH(n) model was introduced by Müller et al. (1997) to try to solve this problem".

2.3.3 HAR Method

Introduced by Müller et al. (1997) to estimate the VaR for High frequency data (data that are measured in small time intervals). This type of data is essential in studying the micro structure of financial markets and increase in computational power and data storage make their usage more feasible. "In fact, this model incorporates heterogeneous characteristics of high frequency financial time series and it is given by

$$\begin{aligned} r_t &= \sigma_t \varepsilon_t \\ \sigma_t^2 &= c_0 + \sum_{j=1}^{n} c_j \left(\sum_{i=1}^{j} r_{t-i} \right)^2 \end{aligned} \tag{2.5}$$

where $c_0 > 0, c_n > 0, c_j \geq 0 \; \forall j = 1, ..., n-1$ and ε_t are identically and independent distributed (i.i.d.) random variables with zero expectation and unit variance"(see Ruilova & Morettin (2020)).

Intraday data are deemed useful in estimating features of the distribution of daily returns. For instance, in forecasting the daily volatility, the realized volatility has been widely used as basis. "The heterogeneous autoregressive (HAR) model of the realized volatility is a simple and pragmatic approach, where a volatility forecast is constructed from the realized volatility over different time horizons (Corsi, 2009)". An alternative way of capturing the intraday volatility is to use the intraday range (daily high and low prices), due to its ready availability compared to intraday data. Where Range_t is the difference between the highest and lowest log prices on day t, to predict tomorrow's range from past daily, weekly, monthly averages of Range_t, we set up the linear regression model;

$$\text{Range}_t = \beta_1 + \beta_2 \text{Range}_{t-1} + \beta_3 \text{Range}^w_{t-1} + \beta_4 \text{Range}^m_{t-1} + \varepsilon_t$$

$$\begin{aligned} \text{Range}^w_{t-1} &= \frac{1}{5} \sum_{i=1}^{5} \text{Range}_{t-i} \\ \text{Range}^m_{t-1} &= \frac{1}{22} \sum_{i=1}^{22} \text{Range}_{t-i} \end{aligned} \tag{2.6}$$

where Range^w_{t-1} and Range^m_{t-1} are averages of Range_t over a week and month, respectively; ε_t is an i.i.d. error term with zero mean; and the β_i are parameters that are estimated using least squares. The conditional variance (see Equation 2.5) is then written as a linear function of the square of Range_t , where the intercept and the coefficient are estimated using maximum likelihoods based on a Student t distribution. A variance forecast is produced with this model, and VaR forecasts are estimated by multiplying the forecast of the standard deviation by the VaR of the student t distribution (Taylor, 2020)

2.3.4 CaViaR Method

Engle and Manganelli (2004) propose a conditional autoregressive quantile specification (CAViaR) quantile estimation. Instead of modeling the whole distribution, the quantile is modelled directly. "The empirical fact that volatilities of stock market returns cluster over time may be translated in statistical words by saying that their distribution is autocorrelated". Consequently, the VaR, which is a quantile, must behave in similar way. A better way to show this feature is to use some type of autoregressive specification

Engle and Manganelli (2004) "suppose that we observe a vector of portfolio returns$\{y_t\}_{t=1}^T$. Let θ be the probability associated with VaR. Let x_t be a vector of time t observable variables, and let β_θ be a p-vector of unknown parameters. Finally, let $f_t(\beta) \equiv f_t(x_{t-1}, \beta_\theta)$ denote the time t θ-quantile of the distribution of the portfolio returns formed at $t-1$, where we suppress the θ subscript from β_θ for notational convenience. A generic CAViaR specification might be the following

$$f_t(\beta) = \beta_0 + \sum_{i=1}^{q} \beta_i f_{t-i}(\beta) + \sum_{j=1}^{r} \beta_j l(x_{t-j}) \tag{2.7}$$

where $p = q + r + 1$ is the dimension of β and l is a function of a finite number of lagged values of observables. The autoregressive terms $\beta_i f_{t-i}(\beta)$, $i = 1, ..., q$, ensure that the quantile changes "smoothly" over time. The role of $l(x_{t-j})$ is to link $f_t(\beta)$ to observable variables that belong to the information set". The parameters of CaViaR are estimated by quantile regression.

Note: In this work, value at risk will be estimated based on financial log-returns. Historical simulation, Garch(1,1) model, and long short term memory neural network will be used for our VAR estimation. The latter will be discussed in the next chapter.

Chapter 3

Estimating VaR using Neural Networks

Neural networks, also known as artificial neural networks (ANNs) are class machine learning algorithm vaguely inspired by the biological neural networks that constitute animal brains.

Neural networks contain an input layer, one or more hidden layers, and an output layer, and each of these layers has node(s). Each node are connected to eachother and has an associated weight and threshold. If the output of any individual node exceeds the specified threshold value, that node is activated, transferring data to the next layer of the network. Else, no data will be passed along to the next layer of the network (see IBM, 2020).

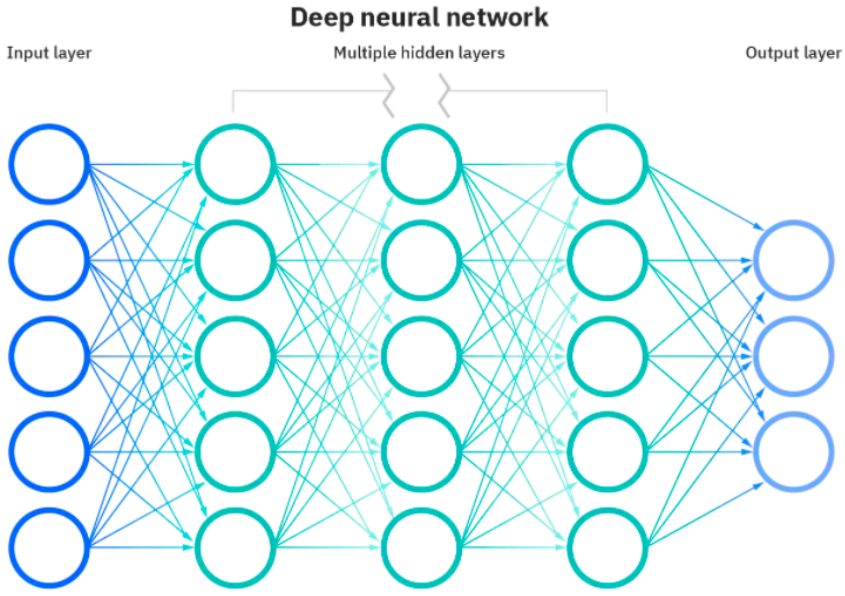

Figure 3.1: A figure showing the layers of a Neural Network (see IBM, 2020)

Neural networks depend on training data to learn and improve their accuracy over time. However, once these learning algorithms are polished for accuracy, they become powerful tools in computer science and artificial intelligence, allowing us to classify and cluster data at a high speed. "Tasks in speech recognition or image recognition can take minutes versus hours when compared to the manual identification by human experts" (see IBM, 2020).

Neural network is good for returns prediction as it can accommodate nonlinear interactions, and no distribution is assumed. However, just like historical simulation they require large data set (which is not always available) for training to perform excellently well.

3.1 Mathematics of Neural Network

The main idea of this section is gotten from the work of Chaoyi Lou, titled Artificial Neural Networks: their Training Process and Applications.

3.1.1 A single Neuron

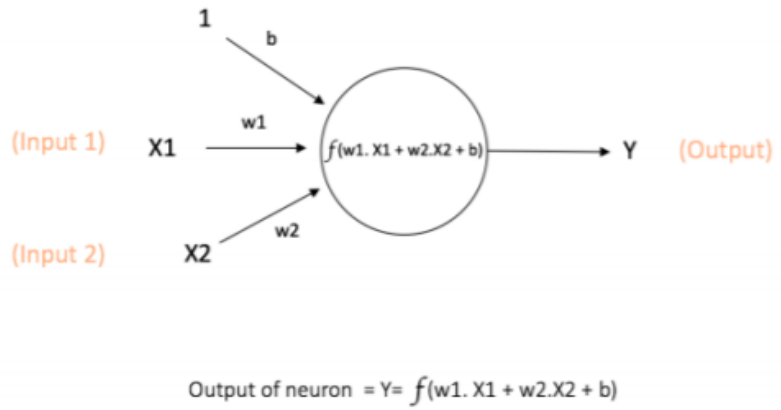

Figure 3.2: A single neuron of neural networks

Figure 3.2 shows a network with one layer containing a single neuron. This neuron receives input from the prior input layer, performs computations, and gives output. x_1 and x_2 are inputs with weights w_1 and w_2 respectively. The neuron applies a function f to the dot-product of these inputs, which is $w_1x1 + w_2x_2 + b$. Aside these two numerical input values, there is one input value 1 with weight b, called the Bias. The main function of bias is to represent unknown parameters. The dot-product of all input values and their associated weights is fed into the function f to produce the result Y. This function is known as Activation Function.

Activation functions are needed because many problems take multiple influencing factors into account and yield classifications. When faced with a binary classification problem, where the outcomes are either yes or no, activation functions are required to map the outcomes within this range. If a problem involving probability arises, one would expect the neural network's predictions to fall within the range of [0, 1]. This is what activation functions can do.

Linear and non-linear activation functions are the two types of activation functions. The most significant disadvantage of linear ones is that they cannot learn complex function mappings because they are only one-degree polynomials. As a result, non-linear activation functions are always required to produce results in desirable ranges and deliver them as inputs to the next layer. Few of the generally used non-linear activation functions will be discussed in the next section.

3.1.2 Activation Functions

An activation function takes the previously specified dot-product as an input and makes computation with it. Based on the range of the expected result, we place a certain activation function inside hidden layer neurons. The fact that activation functions should be differentiable is important because we'll use it later to train the neural network using backpropagation optimization.

Here are few commonly used activation functions:

Sigmoid: This takes a real-valued input and returns a output in the range [0,1]:

$\delta = \frac{1}{1+e^{-x}}$

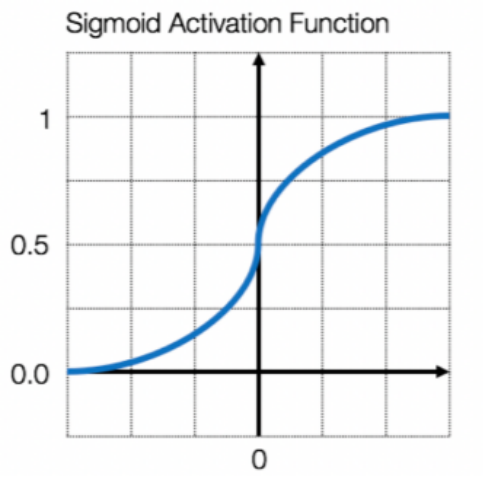

Figure 3.3: Sigmoid() Activation Function

Figure 3.3 shows an S-shaped curve and the values going through the Sigmoid function will be squeezed in the range of [0, 1]. Because the chance of anything only exists between 0 and 1, Sigmoid is a compatible probability transfer function. Despite the fact that the Sigmoid function is simple to comprehend and use, it is not widely used due to its vanishing gradient problem. The issue is that the gradient can come so close to zero in some circumstances that it fails to properly adjust the weight. In the worst-case scenario, the neural network's ability to learn will be completely disabled. Second, this function's output is not zero-centered, causing gradient updates to travel in many different directions. Furthermore, the fact that the output is limited to the range [0, 1] makes optimization more difficult. In order to compensate the deficiencies, tanh() is an alternative option because it is a stretched version of the Sigmoid function with zero-centered outputs.

tanh: This takes real-valued input and produces the results in the range [-1, 1]:

$\tanh(x) = \frac{sinh(x)}{cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

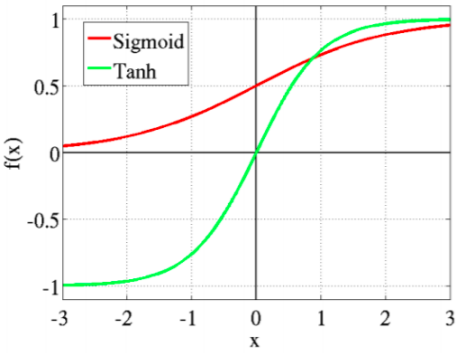

Figure 3.4: tanh() Activation Function

The benefit of this function is that negative input values will be mapped strongly negative, and extremely small values close to zero will be mapped to values close to zero. As a result, this function is helpful in doing a classification between two classes. Though in reality, this function is favoured over the Sigmoid function (because of the greater output range), the gradient vanishing problem still occurs. Using a reasonably simple formula, the following ReLU function corrects this problem.

ReLU (Rectified Linear Unit): It takes a real-valued input and replaces the negative values with zero:

$R(x) = \max(0, x)$

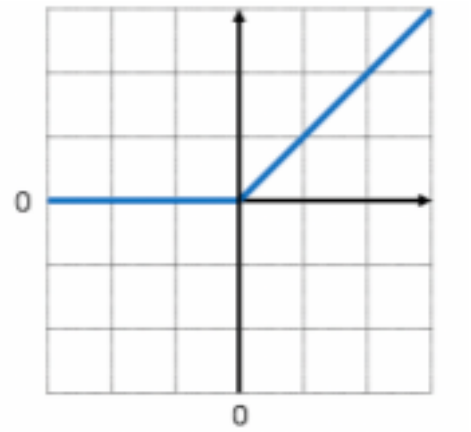

Figure 3.5: ReLU() Activation Function

As it is a very simple and efficient function that avoids and corrects the gradient vanishing problem, it is employed in practically all convolutional neural networks or deep learning. The difficulty with this activation function is that after it is activated, all negative values become zeros, which has an impact on the outcomes because negative values are not taken into account.

When we know what qualities of outcomes we want to observe, we apply different activation functions.

3.2 General Model Building

Having discussed the mathematics behind neural network, it is however in important to talk about the neural network architectures and other components

3.2.1 Neural network Architures

Neural Networks are complex structures made of artificial neurons that can accommodate several inputs to produce output(s). As stated earlier, a Neural Network consists of an input layer, one or multiple hidden layers and output layer(s). In a Neural Network, all the neurons(contained in each layers) affect each other, and hence, they are all connected. How the input neurons produce a certain output is depends on the structure of the neural network. The two main classes of network architectures are discussed below

Feed-forward Neural Network

(see Bijelic & Ouijjane, 2019) In feed-forward neural network(FFN), each neuron in a particular layer is connected with all neurons in a subsequent layer. The information flow in the network is of feedforward type (i.e the connections can never skip a layer, or form any loops backwards).

As shown in the above figure, the values of the input are transported to the hidden layer through connections, each being characterised by certain weight coefficient, $W_{i,k}$. The degree of connection between the input node and a hidden node is reflected by these weight coefficients. Defining $[x_{1,t}; x_{2,t}; ...; x_{n,t}]$ as the vector of the input signals and $[h_{1,t}; h_{2,t}; ...; h_{m,t}]$, the

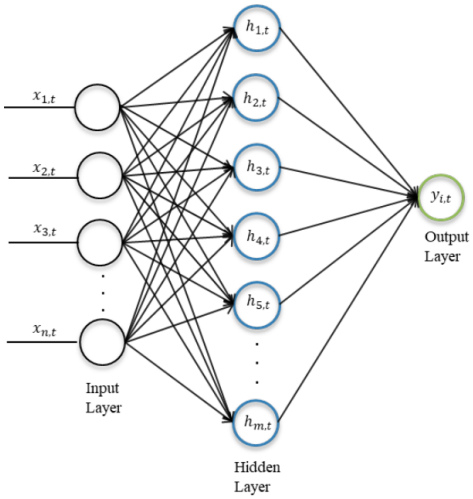

Figure 3.6: A fully connected FFN with a single hidden layer (see Bijelic & Ouijjane, 2019)

propagation of the input nodes to one hidden node can mathematically be described by:

$$h_{k,t} = \sum_{i=1}^{n} W_{i,k} \cdot x_{i,t} \text{ for } k = 1, 2, ..., m$$

An undesirable property of the formula is its linear representation, which, if applied, would suggest that the output prediction would be a linear function, which is not always the case. In order to deal with this, a non-linear activation function, $\Phi(\cdot)$ is applied to the weighted sum of inputs into a hidden node. This activation function, which in the majority of applications takes the form of a sigmoid function or a ReLu function, makes the neural network a universal approximator. However, before applying the activation function, a bias vector $[b_1; b_2; ...; b_m]$ is added, which essentially indicates whether a neuron tends to be active or inactive in the prediction process. The propagation from the input layer to the hidden layer in a feed-forward neural network may now be reformulated to:

$$h_{k,t} = \Phi(b_{k,0} + \sum_{i=1}^{n} W_{i,k} \cdot x_{i,t}) \text{ for } k = 1, 2, ..., m$$

The feedforward neural network has the major disadvantage that it cannot model temporal dependencies in the data. However, this shortcoming of not being able to account for correlations between inputs is overcome in the recurrent neural network, which is able to selectively feed forward information over sequences of elements by generating cycles in the network.

Recurrent Neural Network

(see Bijelic & Ouijjane, 2019) Recurrent Neural Networks (RNN) can handle sequential data due to the capability of each neuron to maintain information about previous inputs, contrary feedforward neural networks. This implies that the prediction a recurrent neural network node made at previous time step $t-1$ affects the prediction it will make one moment later, at time step t. RNN nodes can be thought of as having memory as it takes inputs not only the current signal, but also what has been perceived previously in time.

RNNs contain feedback loops from the so-called hidden states, and this allows preservation of information from one node to another while reading in inputs. "This feedback loop mechanism occurs at each time step in the data series, which results in each hidden state containing traces not only of the previous hidden state, but also of all the preceding ones, for as long as the memory of the network persists (Skymind, 2019)".

"The unrolled RNN illustrates how the network allows the hidden neurons to see their own previous output, so that their subsequent behavior can be shaped by previous responses

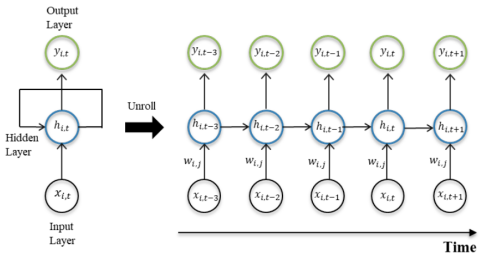

Figure 3.7: Representation of an unrolled plain vanilla recurrent neural network. (see Bijelic & Ouijjane, 2019)

(Tenti, 1996)". In addition, utilization of a RNN is particularly desired when there are time dependencies in the data series, this is evident when we introduce time-lagged model components. using the initial notation, and suppose that the hidden states are the ones looped back, the output from a hidden node in the RNN model relies not only on the input values at time t, but also on its own lagged values at order p as represented below:

$$h_{k,t} = \Phi(b_{k,0} + \sum_{i=1}^{n} W_{i,k} \cdot x_{i,t}) + \sum_{k=1}^{m} \gamma_k \cdot h_{k,t-p} \text{for } k = 1, 2, ..., m$$

where $h_{k,t-p}$ represents the lagged hidden state values at order p, and γ_k a coefficient. Another outstanding feature of recurrent networks is that recurrent neural networks share the same weight parameter within each layer of the network, unlike feedforward networks that have different weights across each node. Through the processes of backpropagation and gradient descent these weights are adjusted to enable reinforcement learning. Backpropagation through time (BPTT) algorithm is exploited by Recurrent neural networks to determine the gradients, which is a bit different from traditional backpropagation as it is specific to sequence data. In BPTT errors are summed up at each time step whereas feedforward networks do not have to sum errors because it shares no parameter across each layer.

In this process, RNNs tend to experience two issues, known as exploding gradients and vanishing gradients. These issues are defined by the gradient size, which is the slope of the loss function along the error curve. If the gradient is too small, the weight parameters will be updated until they become insignificant - i.e. 0, and the algorithm will no longer learn. Gradients explode when they are too large, creating an unstable model. In this case, the model weights will grow too large and will eventually be labelled NaN values. In order to reduce these issues, it is possible to reduce the number of hidden layers within the neural network, thereby reducing its complexity (see IBM 2020).

Long Short-Term Memory Recurrent Neural Network

LSTM is a class of recurrent neural networks and its main feature is its purpose-built memory cells, which allows it to capture long range dependencies in the data. Unlike other neural network designs, LSTMs are applied recurrently.

A previous sequence element and the output from the network function serve as input for the next sequence element in the network function. As such, the LSTM can be compared to a HMM (Hidden Markov Model), in which there is a hidden state which conditions the output distribution. Furthermore, LSTM hidden state not only depends on its previous states but also reflects long-term sequence dependence since it is recurrent. In particular, the receptive field size of an LSTM (i.e. the size of the input region that generates the feature) is unbounded architecture-wise, unlike simple feed forward networks and CNNs. Receptive field of the LSTM is solely dependent on the ability of the LSTM to remember previous inputs (see Arimond et al, 2020).

Due to the attractiveness of the LSTM, it will be used in this work to forecast volatility of returns of stock markets.

3.3 The LSTM Architecture

Our LSTM has a single hidden layer, with 'tanh' as the activation function (because it produces result that ranges from -1 to 1). The following are the hyper parameter used in within the LSTM neural network:

- Optimizers are algorithms used to change the properties of the neural network such as weights and learning rate to minimize the losses. Adam is the chosen optimizer in our LSTM model. Some of its advantages are that the magnitudes of parameter updates are invariant to rescaling of the gradient, its stepsizes are approximately bounded by the stepsize hyperparameter, it does not require a stationary objective, it works with sparse gradients, and it naturally performs a form of step size annealing (Kingma and Ba, 2015).

- The batch size is the number of inputs that will be propagated in the LSTM neural network during the training process. In our LSTM model 128 is the chosen batch size, and this means that the volatility inputs are fed in the network in batches, each containing 128 inputs. After the propagation of a batch, the network is trained before receiving another batch of 128 inputs. This operation continues until all inputs are propagated.

- The look ahead is the amount of time steps, i.e. the lagged inputs the RNN should use to forecast the desired outputs. For all trials, the look back is set to 90 lagged inputs, which corresponds to about 3-month period in the data sample.

- The dropout function is a regularization method used to prevent overfitting by allowing the LSTM neural network to drop a random set of neurons while training the network. Ignoring several neurons for each iteration during the training process is necessary, because if the network is fully connected, neurons will become interdependent, leading to overfitting of the training data. For example, if the dropout function is set to 0.25, this means that 25% of the existing neurons within the network will be ignored during the training process.

- The number of epochs can also influence the accuracy of a neural network. It refers to the number of times all the training and validation datasets are propagated through the LSTM neural network. The standard procedure is to increase the number of epochs until the chosen metric – in this case the MSE – decreases for the validation set, while it continues to increase for the training set, i.e. when the training set shows signs of overfitting.

- The Mean Square Error (MSE): This is the average squared difference between the estimated values and the actual value.

$$L_{MSE} = 1/N \sum_{k=1}^{N} (\hat{y}_k - y_k)^2$$

  It is also chosen as the loss function (between the predicted outputs and the actual outputs) and the performance measure(to assess the model fit while training and validating the network) of the LSTM neural network

The number of neurons, dropout function, activation functions and epochs are changed in each LSTM models to choose the one that performs best. That is, the LSTM with the lowest MSE value. This will be discussed in details in the next chapter.

Chapter 4

Numerical comparisons

For the empirical study, the day-ahead forecasting of the 1% and 5% VAR for daily log-returns(natural log of the new value divided by the initial value) of the following stock markets: NIKKEI 225, FTSE 100 and S&P 500 is considered. The data is downloaded from DataStream. Each series (NIKKEI 225, FTSE 100 and S&P 500) consist of 8477 daily price indices (measure of how prices change over a period of time), the start date and end date are 04/01/1988 and 30/06/2020 respectively. Upon calculating the log-returns, which is given as

$$R_{log} = ln(\frac{R_f}{R_i}), \text{ where } R_f \text{ and } R_i \text{ are current return and initial return respectively}$$

the data in the first row of the series vanishes leaving us with 8476 daily log-returns and 05/01/1988 as starting date. Basically, this means we use 8476 daily log-returns for our VaR estimations. This longer sample is desirable for our models, most especially the historical simulation and neural network as they work best with large data. Moreover, data contains periods of low and high volatilities, which mitigates the probability of the historical simulation being bias (underestimation or overestimation) in the return estimation.

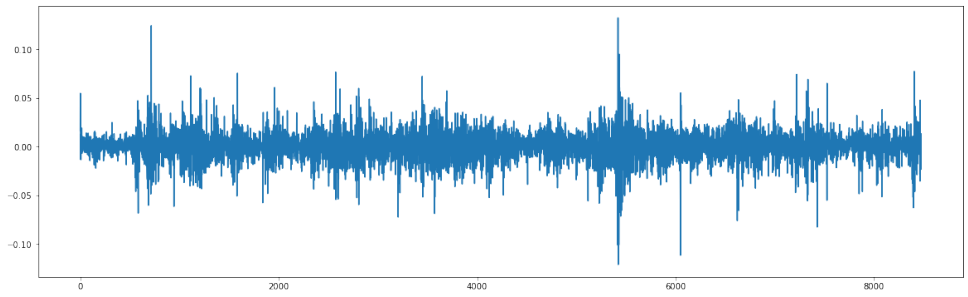

Figure 4.1: The series of log-returns of Nikkei 225

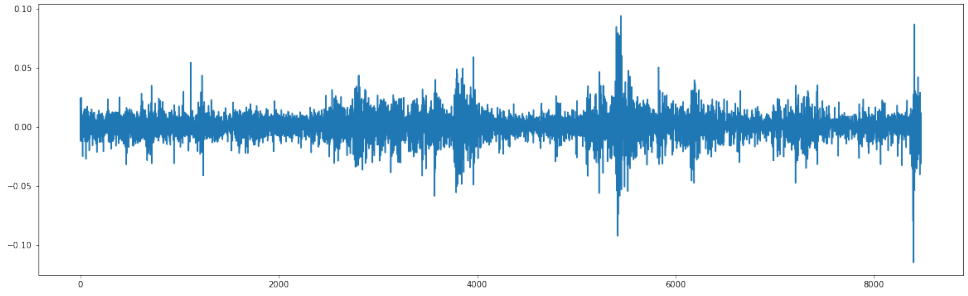

Figure 4.2: The series of log-returns of FTSE 100

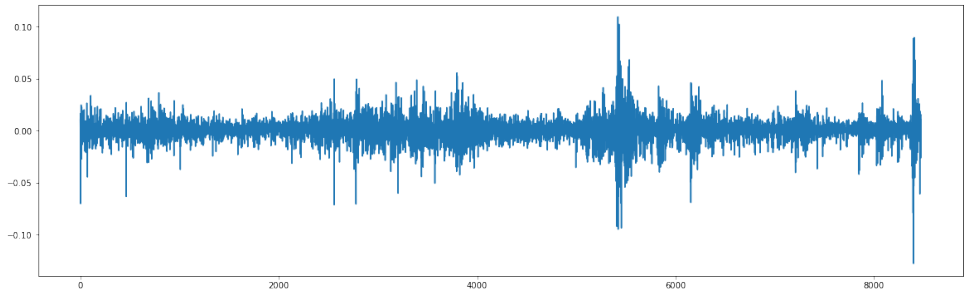

Figure 4.3: The series of log-returns of S&P 500

4.1 Partitioning the Dataset

4.1.1 LSTM Neural Network Model

Generally, data is divided into two main parts in neural network models: training set and test set. However, an additional intermediate set called validation set (sometimes modelled as part of training), is sometimes employed in order to avoid overfitting. The training and validation data can be jointly referred to as In-sample data, while the test data is sometimes referred to as Out-of-sample data. In most literatures, the common choice for training set between 70% to 90% of the original dataset, and 10% to 20% of the training are used as validation dataset. The rest are, of course the testing dataset.

In this study, the first 7000 daily log-returns of each series (each series contains 8473 daily log-returns) are used as training dataset, which is around 83% of each of the series. The last 700 (10%) values (daily log-returns) of the training dataset are used for validation. The remaining 1476 daily log-returns are used for testing. However, the first 90 values in our test dataset vanishes due to use of lookahead (timesteps) of 90 days. Apparently, our resultant forecasts (out-of-sample) are 1386 daily log-returns. The dates for the data split are reported in the table below.

In-sample	out-of-sample
Training set: 05/01/1988 – 27/02/2012 Validation set: 28/02/2012– 03/11/2014	Test set : 10/03/2015 – 30/06/2020

Table 4.1: Data splits

An important pre-processing step is input normalization, as it is considered good practice for neural network training, data scaling helps neural networks train and converge faster. We use the z-score (StandardScaler):

$$X_{new} = \frac{X_i - \mu}{\sigma}$$

where X_{new} is the standardized data point, X_i is the initial data point, μ is the sample mean and σ is the sample standard deviation.

4.1.2 Historical simulation and GARCH(1,1) Volatility model

For congruency with the LSTM model (in regards to number of predictions), we use a rolling window of 7090 for Historical simulation and GARCH(1,1) Volatility model, which stands as our in-sample data and we have 1386 out-of-sample data (predictions).

4.1.3 Trial Results of the LSTM Neural Network

As discussed in the previous chapter, the performance of our LSTM model is based on MSE. In this paper, we follow a best-out-of-5 approach, that means for each stock market, we train our model five times with different values of the hyperparameters and the best one (in each of the model training for the three stock market) is selected for VAR estimation. Tanh is the activation function in all models.

NIKKEI

The training and validation loss continues to decrease as the number of epochs increases until 200 epochs, after which the model starts to overfit. This was made evident as a result of a slow and steady rise in the validation loss after 200 epochs, while the training loss keeps reducing. It was also observed that the most effective size of the LSTM network is 350, and the higher the intensity of regularization applied, the better the model was able to perform.

Trials	Epochs	Dropout	Hidden Neurons	Validation result
1	500	0.5	256	0.00036662753
2	300	0.75	256	0.0001696553
3	300	0.75	350	0.00017357965
4	200	0.5	256	0.0002184841
5	200	0.75	350	0.00016831204

In the first trial, the model is trained for 500 epochs, which is more than the desired 200 epochs. The intensity of regularization is also low at 0.5, the LSTM network size is also low. As a result of the high number of epochs and low regularization effect, it suffers from the highest variance and produced the worst result.

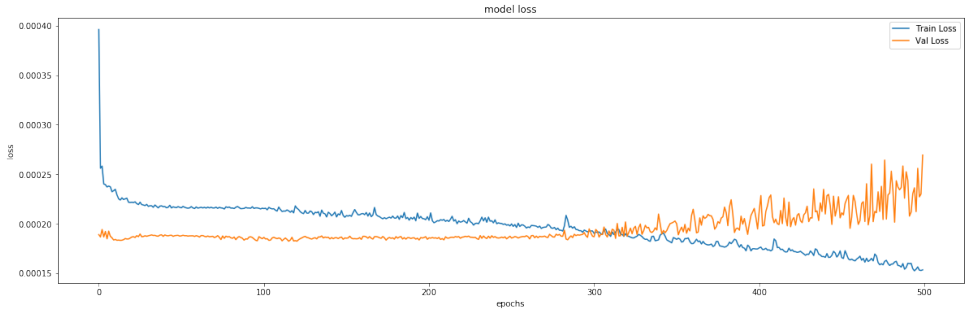

Figure 4.4: Training and Validation loss functions under Trial 1 (Nikkei 225))

The second gives the second best result,as it is trained for more than 200 epochs, and also the LSTM network at 256 fell short of 350. As a result of this, the model suffers from a higher bias as compared to the last trial, because the model is not able to learn well as a result of the inadequate LSTM network.

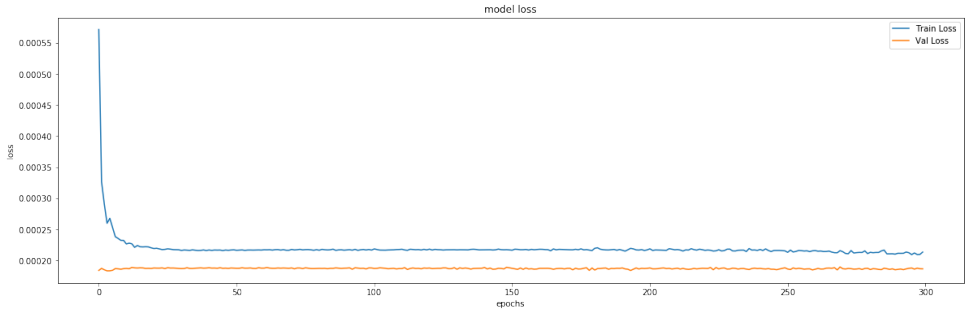

Figure 4.5: Training and Validation loss functions under Trial 2 (Nikkei 225))

Trial 3 is the third in line, the model is trained for 300 epochs which is longer than the required 200 epochs, this made the model to suffer from high variance. Since the LSTM is large although the right size, an increase above the required number of epochs will lead the model to start to overfit rapidly, that was why this trial performed worse than the second trial.

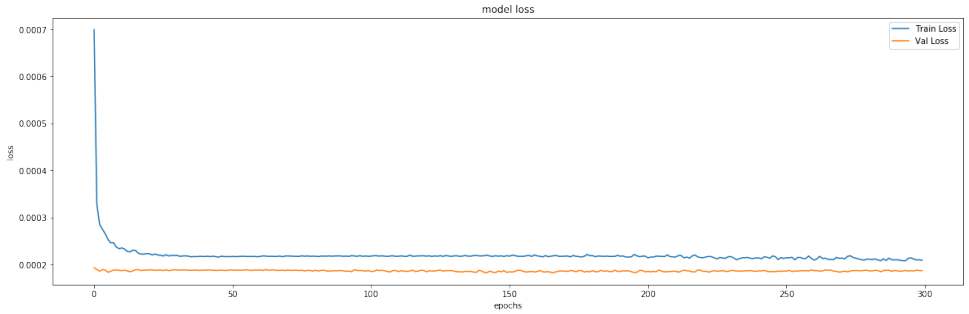

Figure 4.6: Training and Validation loss functions under Trial 3 (Nikkei 225))

Trained with the desired 200 epochs, with a low dropout of 0.5, and a low LSTM size of 256. The forth trial has the lower regularization effect of using a dropout of 0.5 and an insuffcient LSTM network made the model suffer from higher bias.

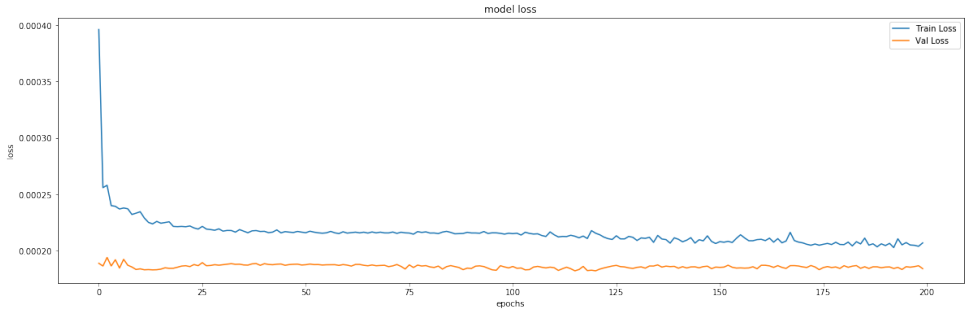

Figure 4.7: Training and Validation loss functions under Trial 4 (Nikkei 225))

The best performed trial is the fifth trial. As a result of the 200 epochs the model trained for being within the range of avoiding high variance (overfitting) and high bias (underfitting). The dropout regularization intensity is also high enough at 0.75, and the size of the Long Short Term Memory (LSTM) network at 350 is large enough.

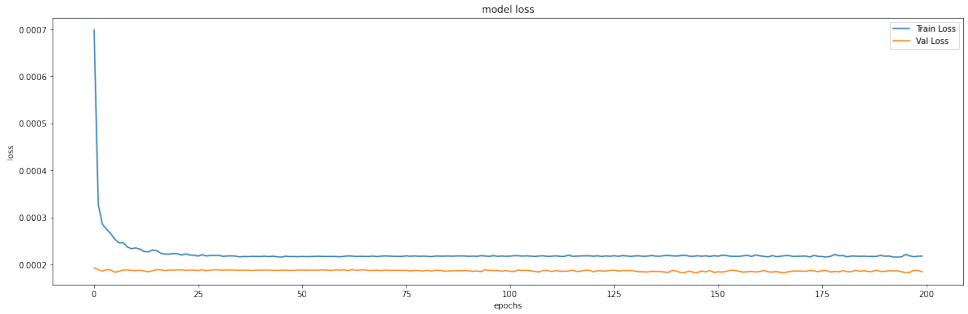

Figure 4.8: Training and Validation loss functions under Trial 5 (Nikkei 225))

FTSE 100

A decrease in the training and validation loss continues to occur as the number of epochs increases until 205 epochs, after which the model starts to overfit. This is made evident as a result of a slow and steady rise in the validation loss after 205 epochs, while the training loss keeps reducing. It was also observed that the most effective size of the LSTM network is 350, and the higher the intensity of regularization applied, the better the model was able to perform.

Trials	Epochs	Dropout	Hidden Neurons	Validation result
1	500	0.5	250	0.00013862312
2	300	0.75	250	0.00012870964
3	300	0.75	350	0.000117358715
4	200	0.5	250	0.00012332022
5	200	0.75	350	0.00011699893

The last trial produced the best result, as a result of the 200 epochs used in the training, being within the range of avoiding high variance and high bias. The dropout regularization intensity is also high enough at 0.75, and the size of the Long Short Term Memory (LSTM) network at 350 is large enough.

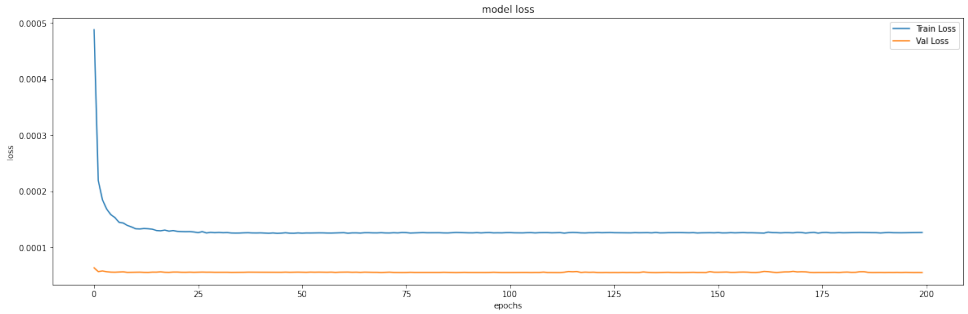

Figure 4.9: Training and Validation loss functions under Trial 5 (FTSE 100))

The third attempt has the second lowest MSE, it fell short because it is trained with more than 200 epochs. As a result of this, the model suffers from a higher variance as compared to trial 5.

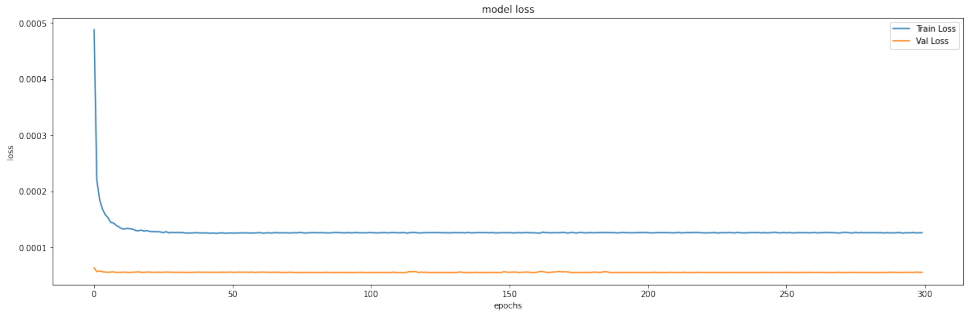

Figure 4.10: Training and Validation loss functions under Trial 3 (FTSE 100))

Although, the forth trial is trained with the required 200 epochs but fall short, as a result of the low dropout of 0.5 and inadequate LSTM network size. The low regularization intensity had the greatest effect, leading to high variance.

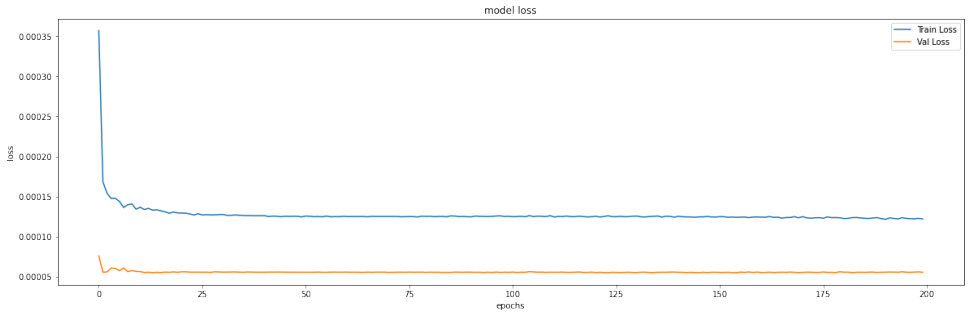

Figure 4.11: Training and Validation loss functions under Trial 4 (FTSE 100))

The second trial is overtrained for 300 epochs, the dropout at 0.5 is low, and the LSTM network at 250, are what led to a less impressive result compared to trial 5,3, and 4

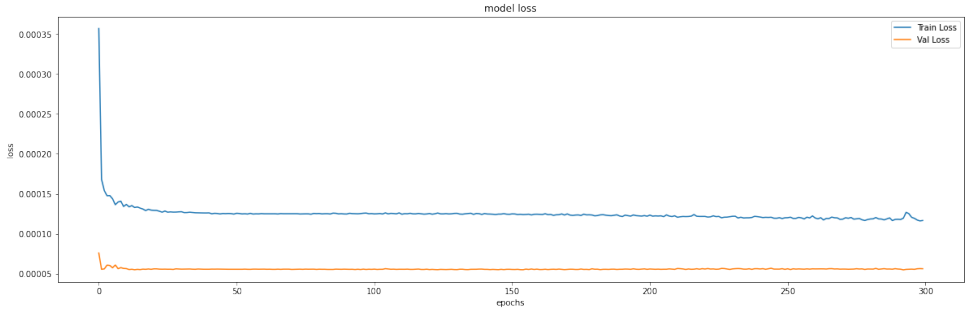

Figure 4.12: Training and Validation loss functions under Trial 2 (FTSE 100))

Suffers the same set back as the second trial, the first trial is trained with an even larger number of epochs at 500. This made it perform the worst in comparison with the rest of the trails.

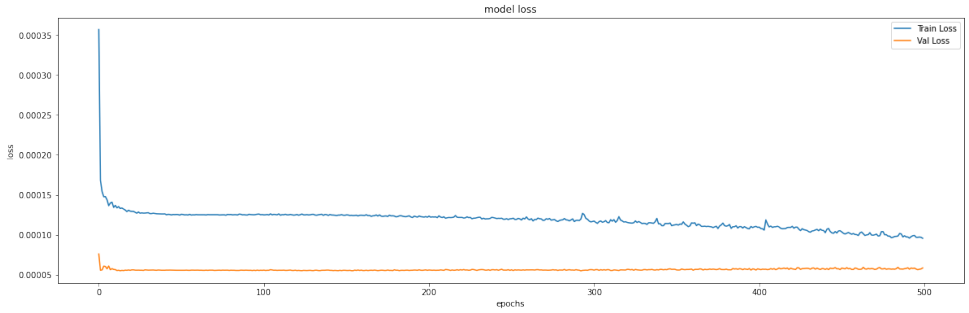

Figure 4.13: Training and Validation loss functions under Trial 1 (FTSE 100))

S&P500

A decrease in the training and validation loss continues to occur as the number of epochs increases until 270 epochs, after which the model starts to overfit. This was made evident as a result of a slow and steady rise in the validation loss after 270 epochs, while the training loss keeps reducing.

Trials	Epochs	Dropout	Hidden Neurons	Validation result
1	500	0.5	250	0.00020557812
2	300	0.75	250	0.00017026176
3	300	0.75	350	0.00013683487
4	270	0.5	256	0.00015936441
5	270	0.8	356	0.00013847416

With the largest number of epochs of 500. The first trial suffers from severe overfitting, and made it perform the worst compared to the rest of the trials.

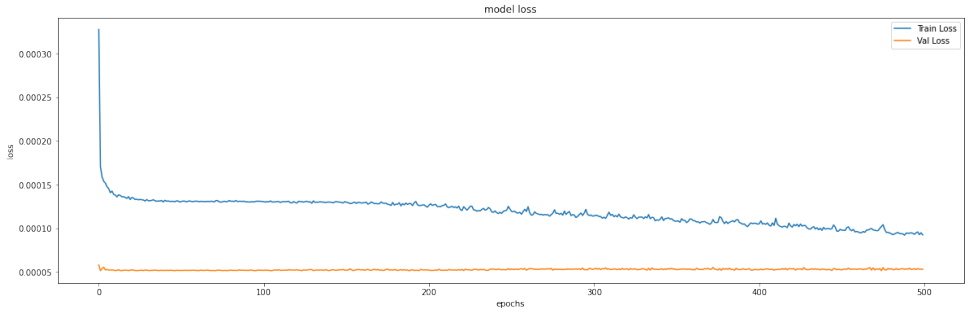

Figure 4.14: Training and Validation loss functions under Trial 1 (S&P 500)

The second trial is modelled with 300 epochs, the dropout at 0.5 is low, and the LSTM network at 250 neurons, are what led to a less impressive result compared to the subsequent trials.

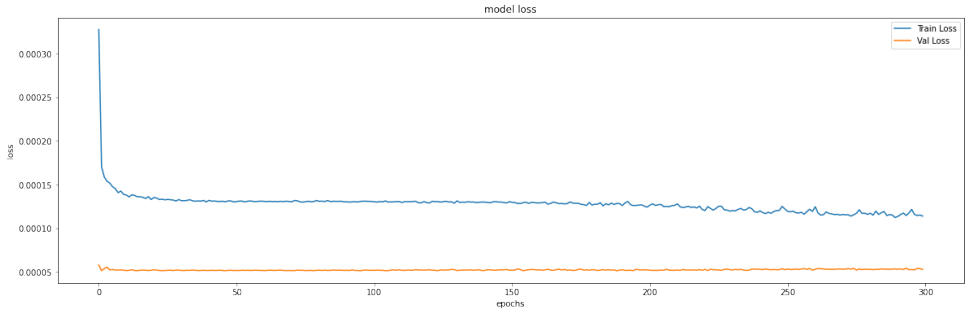

Figure 4.15: Training and Validation loss functions under Trial 2 (S&P 500)

The third attempt has the same number of epochs as second trial, produced the least MSE value, but still suffers from much overfitting despite increasing the dropout and the number of neurons.

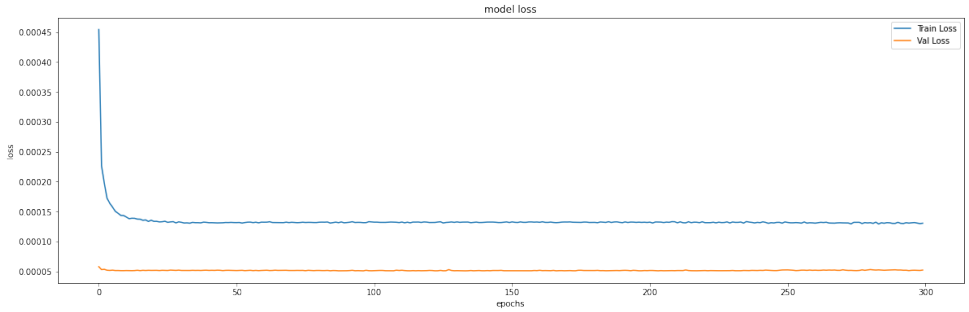

Figure 4.16: Training and Validation loss functions under Trial 3 (S&P 500)

The penultimate trial is trained with the sufficient 270 epochs, but as a result of the low dropout of 0.5 and inadequate LSTM network size. The low regularization intensity has the greatest effect, leading to high variance.

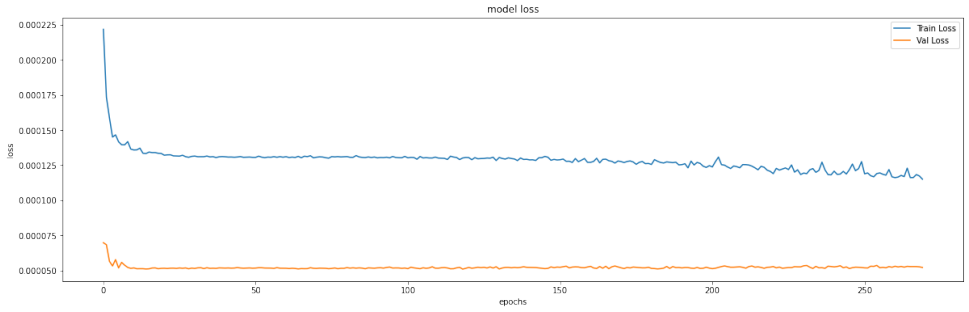

Figure 4.17: Training and Validation loss functions under Trial 4 (S&P 500)

The final trial is trained with 270 epochs and the increased number of hidden neurons and dropout made it outperform the fourth trial and this is considered the best model.

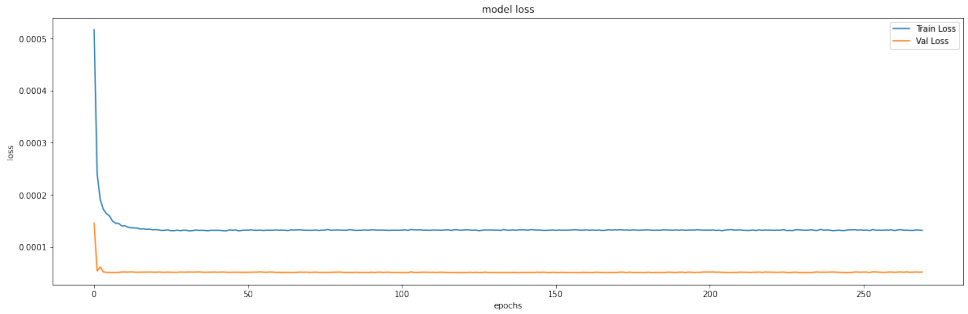

Figure 4.18: Training and Validation loss functions under Trial 5 (S&P 500)

4.2 VAR Estimation

In this section, we discuss the estimation of one step-ahead value at risk.

4.2.1 GARCH (1,1) model

For our VaR estimation, we use the conditional variance given by GARCH(1,1) model. We assume the random error to have a student's t-distribution. Our Value at Risk is given by:

$$\text{VaR}_{t+1|t} = \mu_{t+1|t} + \sigma_{t+1|t} * q_\alpha$$

where μ is the conditional mean, σ is the conditional volatility, and q_α is the α quantile of the standardized residuals

4.2.2 The LSTM model

We determine our value at risk by calculating the 0.05 quantile (95% Var) and 0.01 (99% Var) of our predicted values. Our LSTM model equation is assumed to be of the form:

$$y_t = tanh(\sum_{i=1}^{i=90} W_i u_i + \beta) + \varepsilon_t$$

where u_i are the inputs(initial timesteps), W_i are the weights, β is the bias and ε_t is the error term which has a standard normal distribution, tanh is the activation function

4.3 VAR Backtesting

One easy way to test the efficiency of a VaR model is to count the number of violation (number of days when portfolio returns are less than VaR model estimates). A VaR model performs overestimation of risk if the number of exceptions is less than the selected confidence level, and underestimation if there are too much violation. It is nearly impossible to have the exact amount of violation specified by the confidence level.

Suppose we use a 99% confidence for our VaR model, we have K as the number of violations and N observations. The ratio k/N gives the failure rate. Our null hypothesis is that the frequency of tail loss is p= 0.01. Assuming that the model is accurate, the observed failure rate k/N should act as an unbiased measure of p, and thus converge to 1% as sample size is increased. (Jorion, 2007)

"The setup for this test is the classic testing framework for a sequence of success and failures, also called Bernoull trials". Under the null hypothesis, the number of violations(breaches) k follows a binomial probability distribution:

$$f(k) = \binom{N}{k} p^k (1-p)^{N-k}$$

"When T is large, we can use the central limit theorem and approximate the binomial distribution by the normal distribution"

$$z = \frac{k - pN}{\sqrt{p(1-p)T}} \sim N(0,1)$$

(Jorion, 2007)

4.3.1 Kupiec POF-Test

Kupiec POF-test (proportion of failures) is based on failure rate and was propose by Kupiec (1995). Under null hypothesis that the model is correct, the number of violations follows the binomial distribution. According to Kupiec (1995), the POF-test is best conducted as a likelihood-ratio (LR) test. The test statistic takes the form

$$LR_{pof} = -2 \ln \left(\frac{(1-p)^{N-k} p^k}{[1-(k/N)]^{N-k} (k/N)^k} \right)$$

LR_{pof} (when N is large) is asymptotically χ^2 distributed with one degree of freedom under the null hypothesis that our model is correct. Thus our null hypothesis will be rejected if LR_{pof} is greater than the critical value of the χ^2 significant level (Jorion, 2007).

4.3.2 Results

For congruency with the VaR calculated from our LSTM forecasts, we calculate our actual VaR as the average of daily VaRs stimulated in the Garch and historical simulation model, and this actual VaR is used for comparison with our out-of sample data.

Appendix A

Graphs of log-return series of each stock market

Appendix B

In all the graphs below, number of epochs is represented by the horizontal axis (X-axis), while the loss functions are represented by the vertical axis (Y-axis).

Appendix C

Another example

C.1 More stuff

Bla bla.

Bibliography

List of Figures

List of Tables