Department of Financial Mathematics

Comparison of forecasting models for value at risk

Author:
Hafees Adebayo Yusuff

Supervisor:
Prof. Ralf Korn

October 14, 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 predict VaR using ANN. Locarek-Junge and Prinzler (1999) illustrate how VaR estimates can be obtained based on a USD portfolio by estimating VaR using ANN. The empirical results show a clear superiority of the neural network over other VaR models. The Barone-Adesi and Whaley (BAW) American Futures Options Pricing Model was used by Hamid and Iqbal (2004) to compare volatility forecasts from neural networks with implied volatility forecasts from S&P 500 index futures options. NN's forecasts outperformed the implied volatility forecasts. A similar approach is used by He et al. (2018), who develop a novel type of ANN based on the EMD-DBN method to estimate VaRs of the USD versus the AUD, CAD, CHF, and EUR. They find that an EMD-DBN network identifies more optimal ensemble weights and is less sensitive to noise than an FNN in predicting risk. Nonetheless, it is worth noting that while the prediction of FX volatility by ANNs has attracted some attention in academia, it is still a rather underdeveloped field.

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

Increasing volatility in exchange markets, increased credit defaults, even putting countries' financial security at risk, and calls for more regulation drastically changed the circumstances in which banks operate. 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 conditionally independent 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;

$$\begin{aligned} r_t &= \mu_t + \varepsilon_t \qquad z_t \sim N(0,1) \text{ or student's t} \\ \varepsilon_t &= \sigma_t z_t \end{aligned} \tag{2.1}$$

where ε_t= random error at time t, and E$[\varepsilon_t]$=0
$\mu_t = E[\varepsilon_t|\Omega_{t-1}]$, $\sigma_t^2 = E[\varepsilon_t^2|\Omega_{t-1}]$ 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:
$F(\text{VaR}(\alpha)) = Pr(r_t < \text{VaR}(\alpha)|\Omega_{t-1}) = \alpha$ or 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.

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

Hence, a VaR model involves the specification of $F(r)$ or $G(r)$ (see Abed and Benito, 2009). 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

First, we will introduce the notion of a coherent risk measure. 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 c (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). Despite this shortcoming, the VaR is the most popular risk measure and often asked for by regulations. We will therefore in the remaining of this thesis restrict to the VaR.

2.3 Popular methods for estimating VaR

As considering and comparing all methods for estimating the VaR that are given in the literature is beyond the scope of this thesis, we will concentrate on two of the methods described below, i.e. historical simulation and GARCH-type methods. The neural network approach will be discussed in the next chapter.

2.3.1 Historical simulation

Historical simulation uses past data to predict future performance. To begin, we have to identify the market variables that will affect a portfolio. Then, we collect historical data related to these market variables over a certain timeframe. By calculating the changes in portfolio prices between today and tomorrow, we determine what might happen between today and tomorrow, along with probability distributions associated with changes in portfolio values. An example would be the VaR calculated for a portfolio invested for 1 day with 99% confidence for 700 days of data which is nothing but seventh greatest loss.

This approach has simplified the computations, especially for portfolios with complicated holdings, because no estimation of a covariance matrix is required.

Essentially, this approach accounts for fat tails and is not impacted by model risk due to being independent of it, and this is the core of the approach. Having proven to be extremely useful and intuitive, this method becomes the most popular one to calculate VaR (see Li Hui, 2006). In order to produce a good (Unbiased) simulation, historical data must be available on all risk factors over a relatively long period of time. Historical Simulations VaR may be underestimated if we run them during a bull market or distorted if we run them just after a crash or bear market due to its dependency on history.

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 ($\xi_t \sim$ N(0,1) or student's t)

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)).

GARCH models, when used with high frequency data, must be modified to incorporate the microstructure of the financial markets. For example, there are heterogeneous characteristics that appear when a market has many traders trading with various time horizons. The HARCH (n) model was developed by Müller et al. (1997) to help address this concern (see Ruilova & Morettin, 2020).

2.3.3 HAR Method

The HAR method has been introduced by Müller 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. As stated by Rilova and Morettin (2020), this model incorporates heterogeneous characteristics of high frequency financial time

series It has the defining relations:

$$r_t = \sigma_t \varepsilon_t$$

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

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 rational approach, where a volatility forecast is built from the realized volatility over distinct time horizons (see 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;

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

$$\text{Range}_{t-1}^{w} = \frac{1}{5} \sum_{i=1}^{5} \text{Range}_{t-i} \tag{2.6}$$

$$\text{Range}_{t-1}^{m} = \frac{1}{22} \sum_{i=1}^{22} \text{Range}_{t-i}$$

where Range_{t-1}^{w} and Range_{t-1}^{m} 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 (see 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

Assume that we observe a vector of portfolio returns$\{y_t\}_{t=1}^{T}$. Let θ be the probability linked with the VaR. Let x_t be a vector of time t visible variables, and let β_θ be a p-vector of unknown parameters. Lastly, 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 (see Engle and Manganelli (2004)).

Note: In this thesis, 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. CaViaR and Harch model are not commonly used in practice. Moreover, there is no intraday data to use in modelling VaR by Harch method.

Chapter 3

Estimating VaR using Neural Networks

Neural networks, also known as artificial neural networks (ANNs) are a class of machine learning algorithms 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).

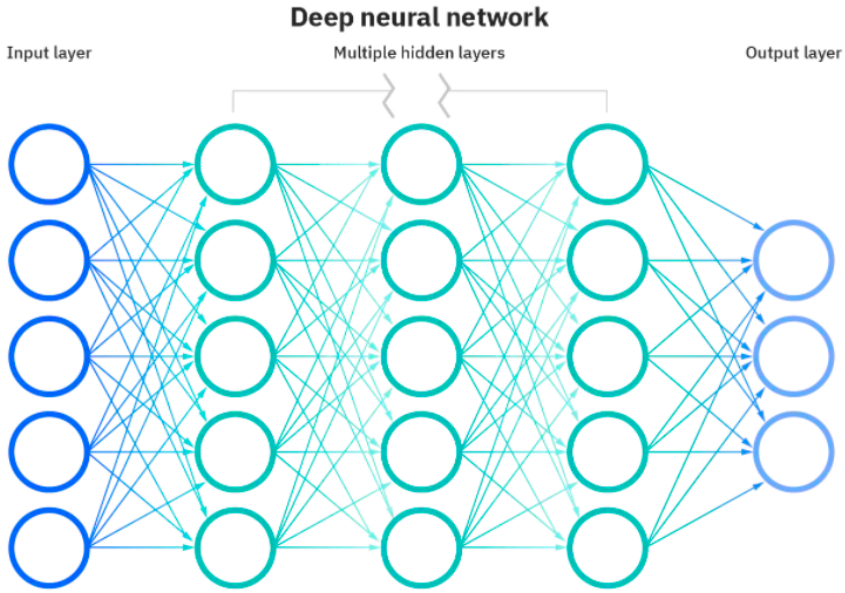

Figure 3.1: A figure showing the layers of a Neural Network (see IBM, 2020)

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. (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 thesis of Chaoyi Lou, titled Artificial Neural Networks: their Training Process and Applications.

3.1.1 A single Neuron

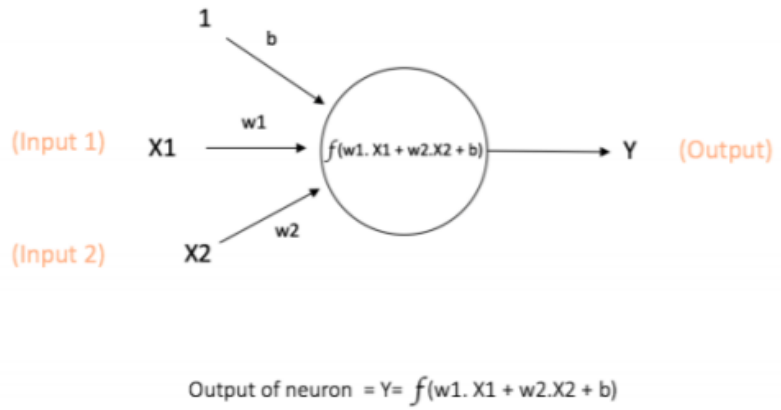

Figure 3.2: A single neuron of neural networks

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]:

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

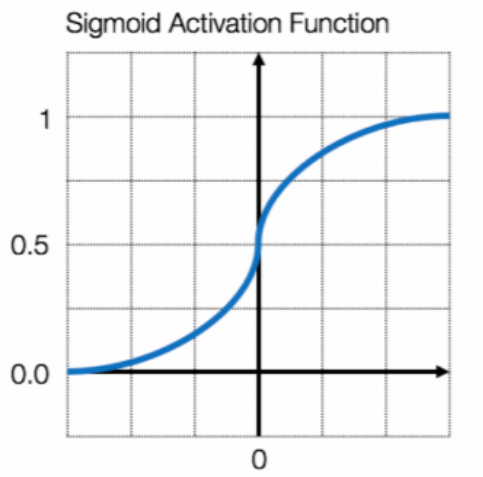

Figure 3.3: Sigmoid() Activation Function

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]. As the sigmoid function attains all values between 0 and 1 (with 0 and 1 attained in the limitting cases), the sigmoid function (also called logistic function) 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]:

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

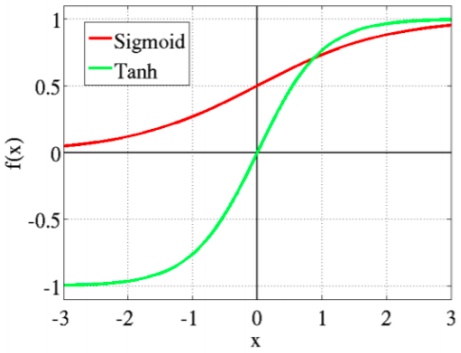

Figure 3.4: tanh() Activation Function

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): ReLU is just another name for the positive part of the argument, i.e it takes a real-valued input and replaces the negative values with zero:

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

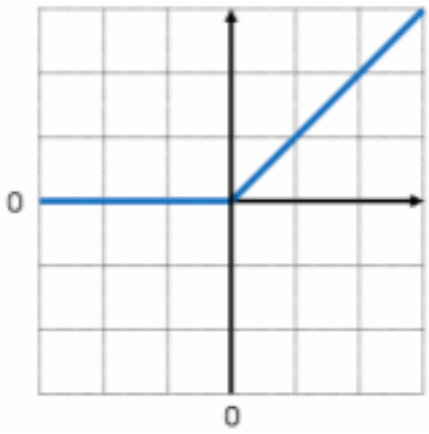

Figure 3.5: ReLU() Activation Function

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 dense 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).

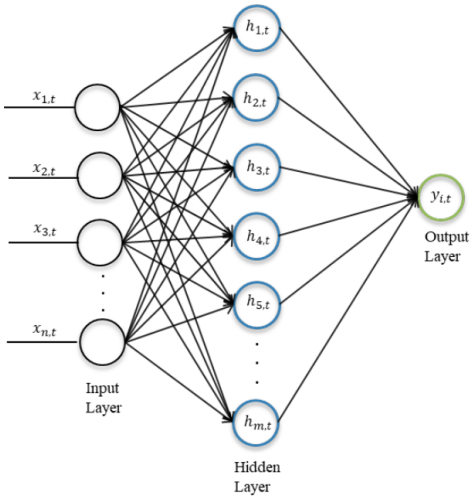

Figure 3.6: A fully connected FFN with a single hidden layer (see Bijelic & Ouijjane, 2019)

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 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 i.e neural networks have the capability of approximating any measurable function to any desired degree of accuracy, in a very specific and satisfying sense (see Hornik et al (1989) for details). 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. At each time step in the data series, the feedback loop mechanism occurs, which causes each hidden state to contain traces not only of the respective hidden state before it, but also of all those preceding it, for as long as the memory of the network lasts.

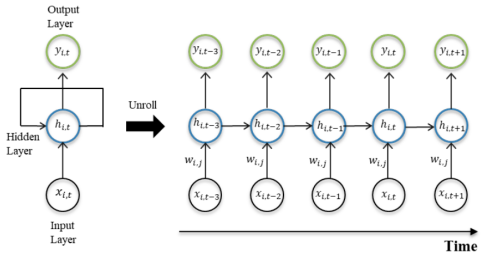

Figure 3.7: Representation of an unrolled plain vanilla recurrent neural network. (see Bijelic & Ouijjane, 2019)

The unrolled RNN shows how the network enables the hidden neurons to see their own previous output, so that their subsequent behavior can be shaped by their responses in the past (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 (see IBM 2020 for details).

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.

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 (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.

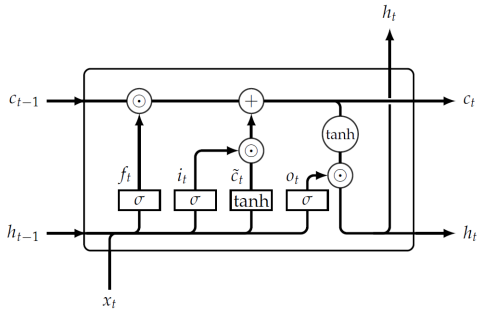

Figure 3.8: Diagrammatic representation of an LSTM cell (see KRETSCHMER, 2019)

LSTM has basically the same setting as the standard RNN. In each time step t, the input x_t and the past state information, for an LSTM c_{t-1} and h_{t-1}, are taken into the hidden mapping. However, in the LSTM the hidden mapping is not just a linear layer followed by an activation function. It rather uses a group of gates and activations to determine the flow of information. A representation of the architecture is shown in the above figure where the rectangular blocks represent layers and nodes pointwise operations.

A gate is a simple sigmoid layer which may at its extremes be either open or closed if its outputs 1 or 0 respectively, depending on the input. For this reason, if another variable gets multiplied by such a gate, the model may either pass the variables information (gate open) or ends the flow (gate closed). The forget, the input and the output gate exist in LSTM

In contradiction to the basic RNN, the LSTM has two state variables: the cell state and the hidden state. The cell state may be seen as a summary statistic of past and current information and the hidden state may be viewed as a polished version of the cell state capturing enough information for the output.

Let us have a look at each of the steps within an LSTM cell

- Forget

  First, the LSTM determines which part of the past information in the cell state c_{t-1} can be removed. This may be useful if context changes and previous information is no longer a valid predictor for future tasks. Hence, the forget gate f_t is calculated based on the past hidden state h_{t-1} and the current input x_t

$$f_t = \sigma(W_f x_t + V_f h_{t-1} + b_f) \in (0,1)^D$$

- Input

  Next, the LSTM recognizes new information which should be incorporated in the cell state c_t. Like the forget gate, the input gate i_t gets calculated

$$i_t = \sigma(W_i x_t + V_i h_{t-1} + b_i) \in (0,1)^D$$

to decide which part of the cell state to add the new information $\tilde{c}_t$ given by

$$\tilde{c}_t = \tanh(W_c x_t + V_c h_{t-1} + b_c) \in (-1,1)^D$$

- Update cell state
  Combine Step 1 and Step 2 to obtain the new cell state

$$c_t = f_t \odot + c_{t-1} + i_t \odot \tilde{c}_t \in \mathbb{R}^D$$

  giving a summary statistics of relevant past and current information.

- Update hidden state
  Determine the parts of the cell state that are enough for the output by calculating the output gate

$$o_t = \sigma(W_o x_t + V_o h_{t-1} + b_o) \in (0,1)^D$$

  and use this to determine the hidden state h_t as a polished version of the cell state.

$$h_t = o_t \odot \tanh(c_t) \in (-1,1)^D$$

(see KRETSCHMER, 2019)

3.3 The LSTM Architecture

Our LSTM has a single hidden layer, with 'tanh' as the activation function. The following are the hyper parameter (all but MSE has to be determined before you can actually start training, validating and testing your LSTM net) used in within the LSTM neural network:

- To determine the best weights of the neural network, Adam (Adaptive Moment Estimation) is the chosen optimizer in our LSTM model. "Some of its advantages are that the sizes of parameter updates are invariant to rescaling of the gradient, its stepsizes are approximately bounded by the stepsize hyperparameter, stationary objective is not required, it works with sparse gradients, and it naturally performs a form of step size annealing (see 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 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 ahead 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 (in different training steps one drops different neurons as otherwise you will not train the full network).

- 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, 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

$$R_t = ln(\frac{S_t}{S_{t-1}}), \text{ where } S_t \text{ and } S_{t-1} \text{ are current return and initial return respectively}$$

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.

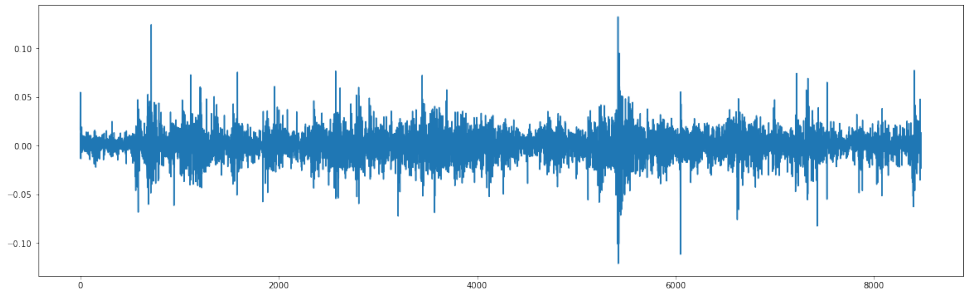

Figure 4.1: The series of log-returns of Nikkei 225

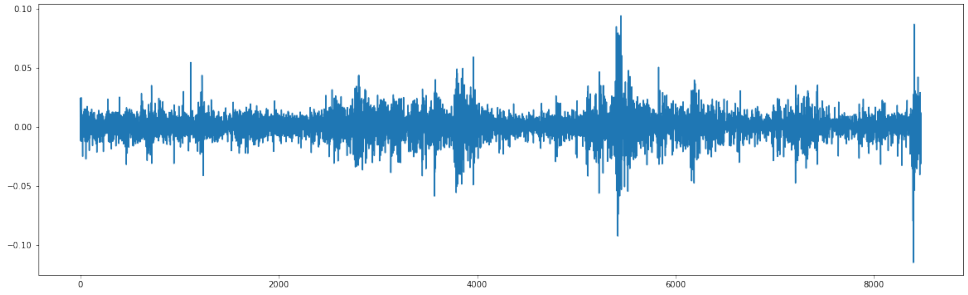

Figure 4.2: The series of log-returns of FTSE 100

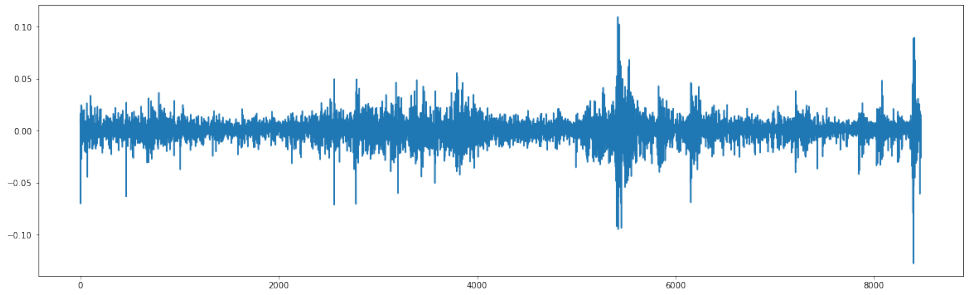

Figure 4.3: The series of log-returns of S&P 500

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, we have a lookahead (timestep) of 90 days, and in turn, we are left with 8386 days for training and testing. The first 7000 daily log-returns of each series are used as training dataset, which is around 83% of each of the series. The last 1400 (20%) values (daily log-returns) of the training dataset are used for validation. The remaining 1386 daily log-returns are used for testing. The dates for the data split are reported in the table below.

In-sample	out-of-sample
Training set: 10/05/1988 – 26/10/2009 Validation set: 27/10/2009– 09/03/2015	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

Trials	Epochs	Dropout	Hidden Neurons	Validation result
1	500	0.1	100	0.0002802554
2	190	0.1	100	0.00020421705
3	111	0.2	100	0.0001751952
4	111	0.2	150	0.0001786659
5	111	0.3	150	0.0001772641

In the first trial, the model is trained for 500 epochs. Both training and validation losses keep reducing until around 120 epochs where model begins to show overfitting as validation loss starts increasing. Overfitting becomes obvious after 190 epochs.

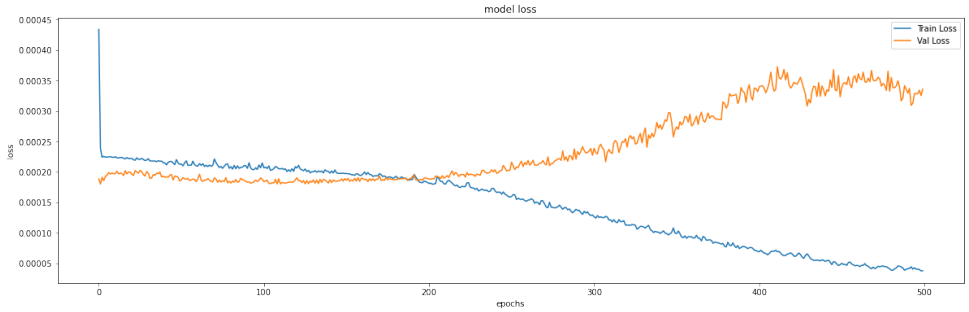

Figure 4.4: Training and Validation loss functions under Trial 1 (Nikkei 225))

The second test is run with 190 epochs to discard the utmost overfitting and to confirm the intuition that the lowest loss on the validation set exists before the 120th epoch. Here the validation loss has its lowest value at 111th epoch.

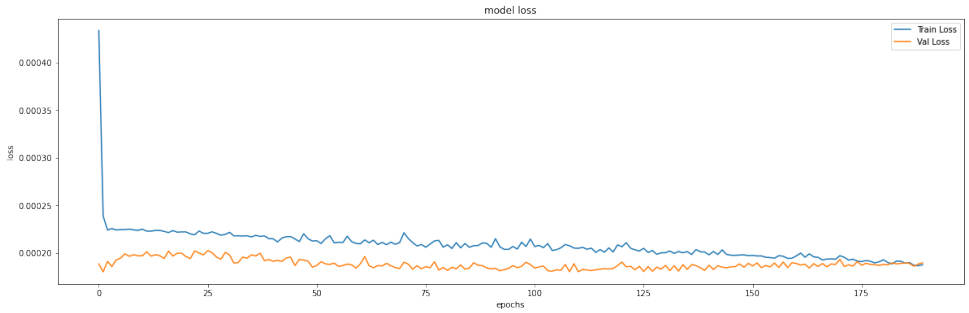

Figure 4.5: Training and Validation loss functions under Trial 2 (Nikkei 225))

As the perfect amount of epochs has been known, we run our third trial with 111 epochs and we increase our dropout. This model has the least MSE and it is chosen for our VaR estimation.

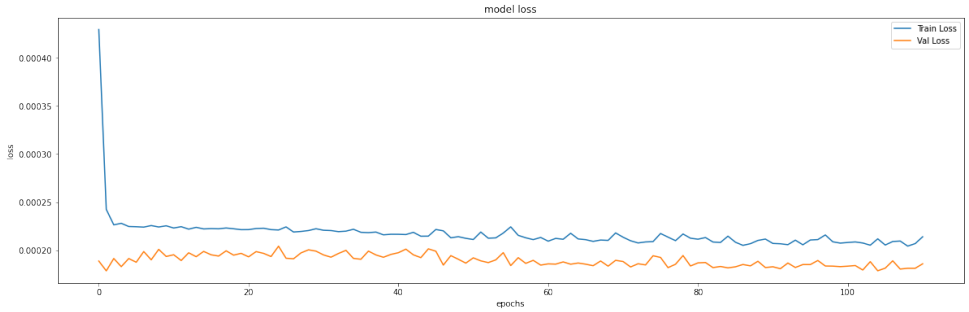

Figure 4.6: Training and Validation loss functions under Trial 3 (Nikkei 225))

We perform our fourth model with an increase in the amount of neurons in the hidden layer to see if our model will perform better, but this is not the case has it has a higher MSE than the third model.

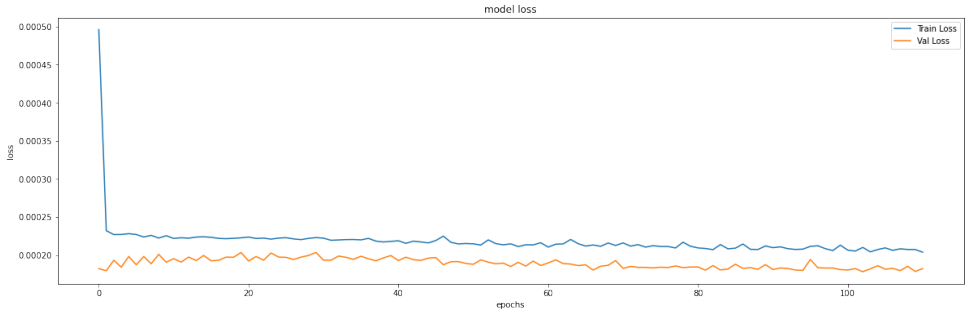

Figure 4.7: Training and Validation loss functions under Trial 4 (Nikkei 225))

The second best performed model is the fifth trial. We maintain 150 neurons in the hidden layer, however we increase the amount of dropout to 0.3.

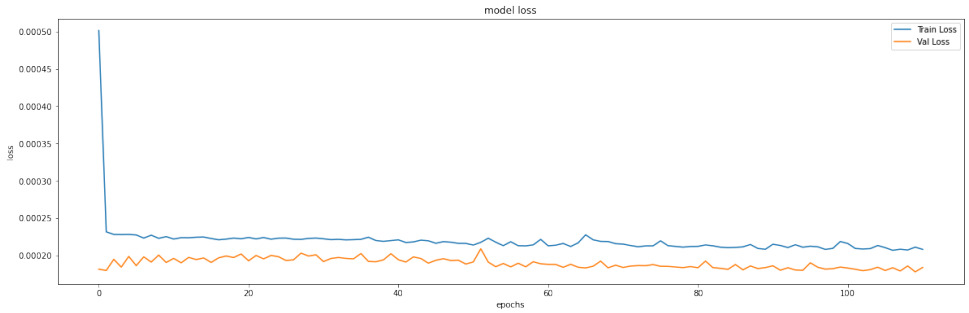

Figure 4.8: Training and Validation loss functions under Trial 5 (Nikkei 225))

FTSE 100

Trials	Epochs	Dropout	Hidden Neurons	Validation result
1	500	0.1	100	0.00018413635
2	200	0.1	100	0.00013991451
3	147	0.1	120	0.00013827156
4	147	0.2	120	0.00013800853
5	147	0.3	100	0.00012749673

In the first trial, the model is trained with 500 epochs. Both training and validation losses keep reducing until around 175 epochs where model begins to show overfitting as validation loss starts increasing. Overfitting becomes obvious after 200 epochs.

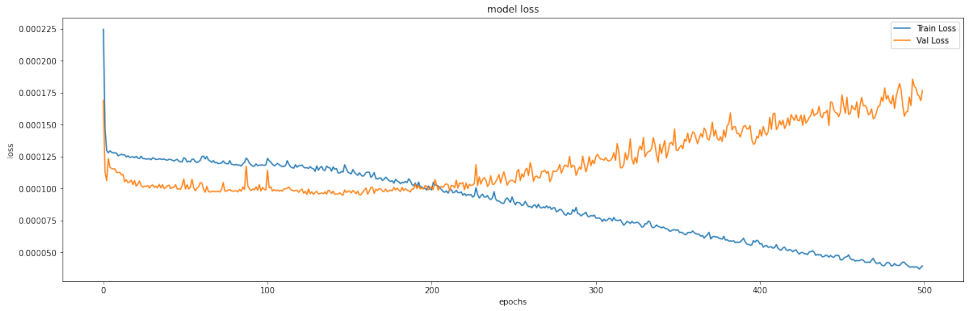

Figure 4.9: Training and Validation loss functions under Trial 1 (FTSE 100))

The second trial of our lstm model is trained with 200 epochs to remove the obvious overfitting. Here it becomes clear that our model performs well till 147 epochs after which validation losses continue to increase.

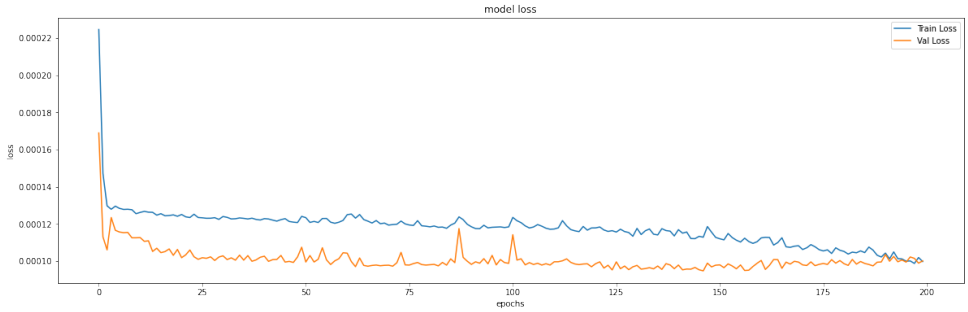

Figure 4.10: Training and Validation loss functions under Trial 2 (FTSE 100))

We perform the third trial with the required 147 epochs and the amount of neurons in the hidden layers is increased to 120. This model gives the third lowest MSE.

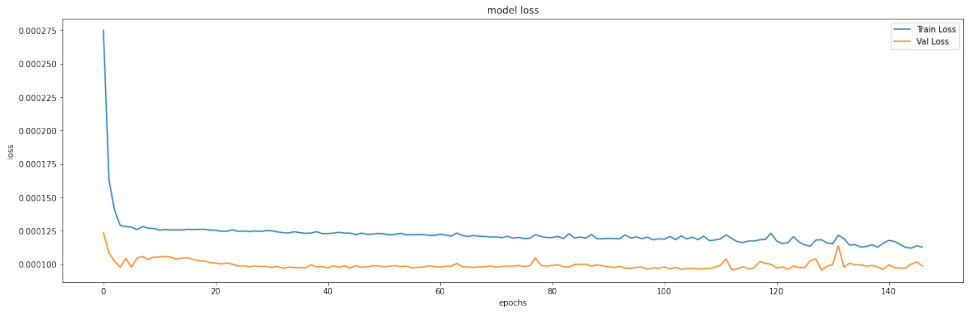

Figure 4.11: Training and Validation loss functions under Trial 3 (FTSE 100))

The fourth trial is trained with 147 epochs, the increased amount of dropout leads to a lower MSE result compared to the third trial.

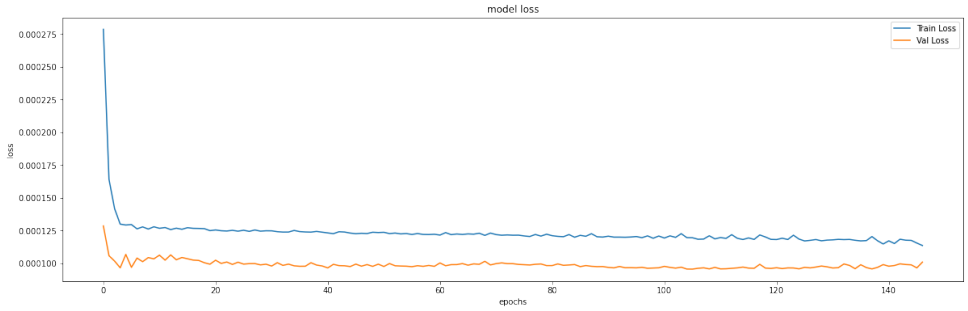

Figure 4.12: Training and Validation loss functions under Trial 4 (FTSE 100))

The last trial is the chosen model for our VaR estimation as it performs best. Dropout is increased while we reduce the amount of neurons in the hidden layer.

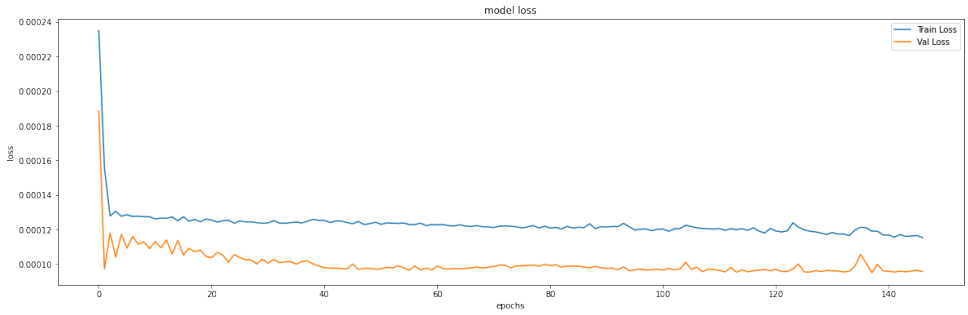

Figure 4.13: Training and Validation loss functions under Trial 5 (FTSE 100))

S&P500

Trials	Epochs	Dropout	Hidden Neurons	Validation result
1	500	0.1	100	0.00023379210
2	195	0.2	100	0.00015249624
3	124	0.2	120	0.00014183349
4	124	0.2	180	0.00014573926
5	124	0.3	180	0.00015285780

In the first trial, the model is trained with 500 epochs. Both training and validation losses keep reducing until around 150 epochs where model begins to show overfitting as validation loss starts increasing. Overfitting becomes obvious after 195 epochs.

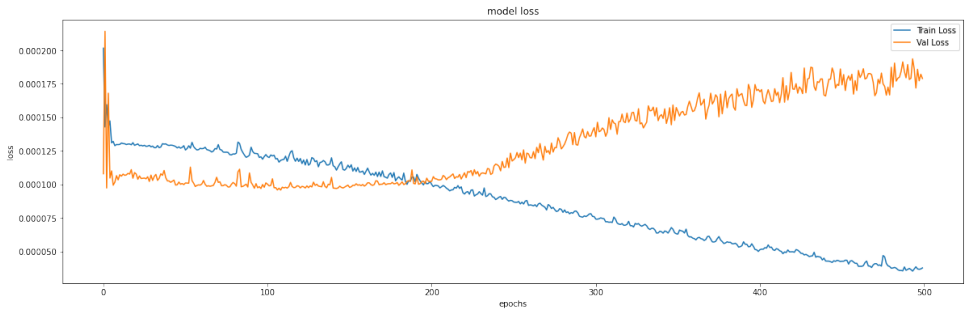

Figure 4.14: Training and Validation loss functions under Trial 1 (S&P 500)

The second test is run with 195 epochs to discard the utmost overfitting and to confirm the intuition that the lowest loss on the validation set exists before the 150th epoch. Here it

becomes clear that our model performs well till 147 epochs after which validation losses continue to increase.

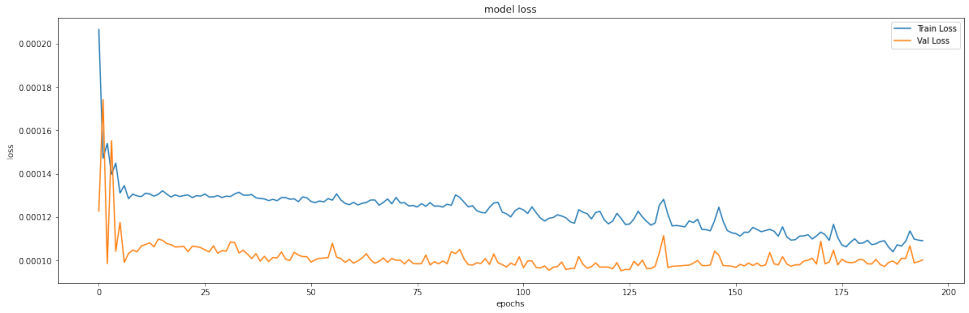

Figure 4.15: Training and Validation loss functions under Trial 2 (S&P 500)

We perform the third trial with the required 124 epochs and the amount of neurons in the hidden layers is increased to 120. This model gives the lowest MSE and will be used for VaR estimation.

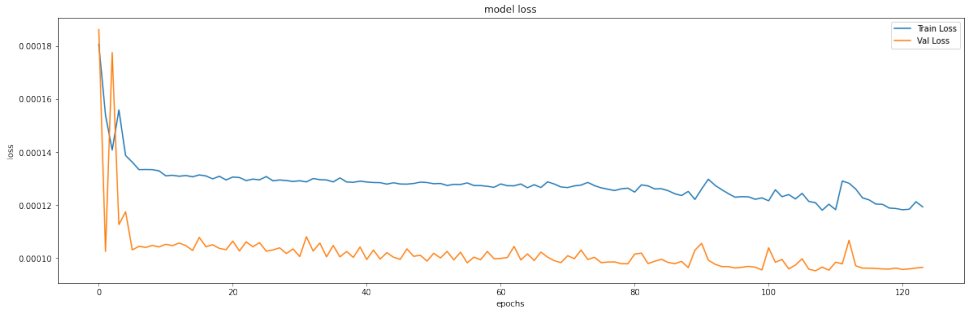

Figure 4.16: Training and Validation loss functions under Trial 3 (S&P 500)

The penultimate trial is trained with the sufficient 124 epochs. The increased amount of neurons without an increase in the dropout seems to alter its performance

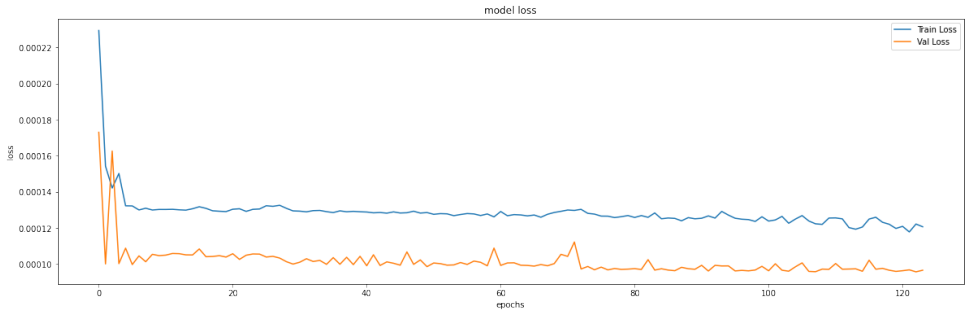

Figure 4.17: Training and Validation loss functions under Trial 4 (S&P 500)

The last trial is trained with the sufficient 124 epochs. Here we increase our dropout to 0.3 and this model takes the fourth position in terms of performance in respect to its MSE.

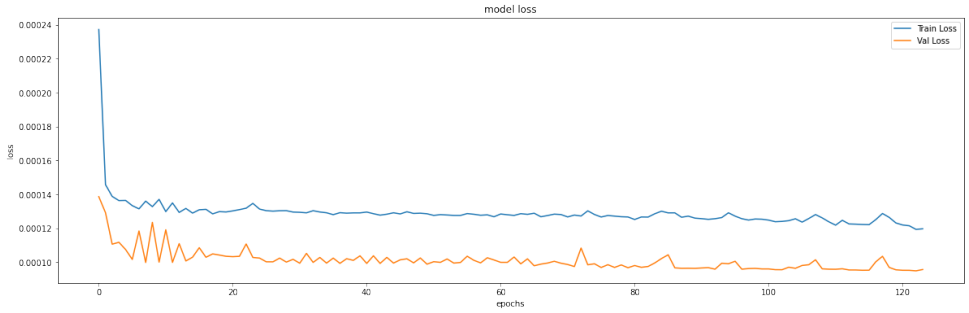

Figure 4.18: Training and Validation loss functions under Trial 5 (S&P 500)

4.2 VAR Estimation

In this section, we discuss the estimation of our value at risk.

4.2.1 Historical Simulation

As a nonparametric method, we use historical simulation with rolling window of 7090 observations to estimate the value at risk for the next 1386 days.

4.2.2 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:

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

where μ is the conditional mean, σ is the conditional volatility, and q_α is the α quantile of the student's t-distribution.

4.2.3 The LSTM model

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

$$y_1 = tanh(\sum_{i=1}^{i=90} W_i u_i + \beta) + \varepsilon_t$$

where u_i are the inputs, W_i are the weights, β is the bias and ε_t = random error in the period t, which equals to the product of standard deviation σ_t of the calibration set and the standardized random error ξ_t in the period t ($\xi_t \sim$ N(0,1)), tanh is the activation function. As the output from the network function serves as input for the next sequence element in the network function, the general equation would be of the form:

$$y_n = tanh(\sum_{i=1}^{i=90} W_i u_i + \sum_{i=1}^{n-1} y_i K_i + \beta) + \varepsilon_t$$

where K_i are weights, y_i are initial outputs.

How we estimate our value at risk from our LSTM model

Here we discuss the two approaches we use in estimating our VAR with LSTM model.

- We are at time t and have given a trained NN that uses the (log-)returns of the last 90 days (up to today) as input to predict the log return r_{t+1} from time t to time $t+1$.

- Now simulate 1000 predictions $r(1), ..., r(1000)$ of this log return from t to $t+1$ by generating a standard normal distribuion with a sample size of 1000, then scale these samples with the standard deviation of the historical returns (the first 7090 days of our original data that make up the timesteps and training). Then add each scaled samples to our predicted return r_{t+1}. Call the ordered predictions $p(1), p(2), ..., p(1000)$.
- Estimate the VaR95 = 0.5*$(p(950) + p(951))$, VaR99 = 0.5*$(p(990) + p(991))$
- Compare the actual log return r_{t+1} in the test data set with VaR95 and VaR99 and increase the number of breaches for VaR95 by 1 if you have VaR95 $< r_{t+1}$ and do the same for VaR99.
- Then update your trained NN by moving one time step forward, i.e. by including the predicted $r_t + 1$ as input.
- Go back to the first step where you replace t by $t+1$. Repeat till you get to the end of the test data.

Another alternative is to perform step 1 to 3 for each predicted log returns. Then estimate the breaches by comparing each results in step 3 with the test data.

Both methods give similar results and we evaluate our value at risk based on the first approach

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)

It is based on the classic testing framework for a series of successes and failures, also known as Bernoulli tests. Under the null hypothesis, the number of violations(breaches) k follows a binomial probability distribution:

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

By applying the central limit theorem, we can approximate the binomial distribution by the normal distribution when T is large

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

(see 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 (see Jorion, 2007). It is common to choose a 95% confidence level for backtesting and apply this level to different VaR models regardless of the VaR confidence level chosen.

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.

Table 4.2: VaR results for Nikkei 225

model	VaR		No. of VAR breaches		kupiec 95% test	
	95%	99%	95%	99%	95%	99%
Historical Simulation	-0.023307	-0.039578	57	16	2.44	0.318
Garch(1,1) model	-0.019251	-0.031422	82	29	2.32	12.71
NN model	-0.026480	-0.036325	43	21	12.08	3.21

Table 4.3: VaR results for FTSE 100

model	VaR		No. of VAR breaches		kupiec 95% test	
	95%	99%	95%	99%	95%	99%
Historical Simulation	-0.016711	-0.030739	61	21	1.09	3.21
Garch(1,1) model	-0.014778	-0.022997	73	40	0.20	33.01
NN model	-0.018395	-0.025791	55	28	3.33	11.25

Table 4.4: VaR results for S&P 500

model	VaR		No. of VAR breaches		kupiec 95% test	
	95%	99%	95%	99%	95%	99%
Historical Simulation	-0.017029	-0.031004	70	21	0.007	3.21
Garch(1,1) model	-0.014281	-0.023931	91	39	6.54	30.88
NN model	-0.018968	-0.028131	56	28	2.87	11.25

4.4 Graphs

In this section, graphical representation of our results are shown

NIKKEI 225

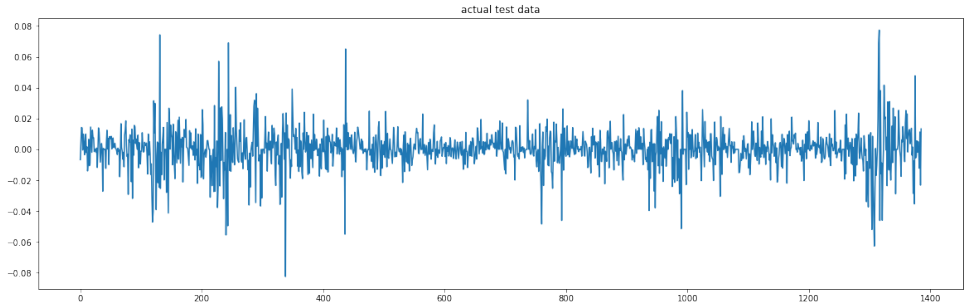

Figure 4.19: Graphical display of NIKKEI 225 test data

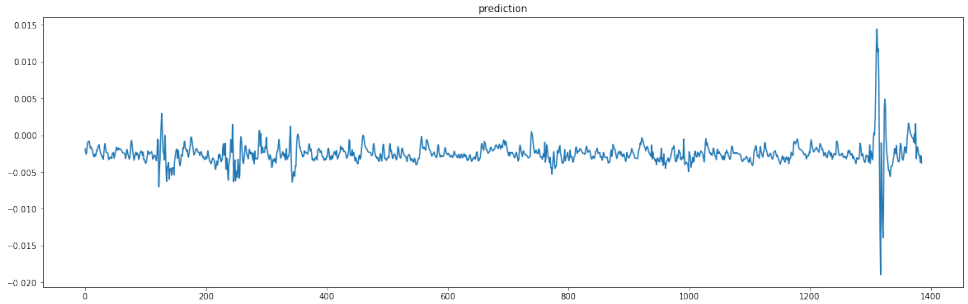

Figure 4.20: Graphical display of NIKKEI 225 out-of sample prediction

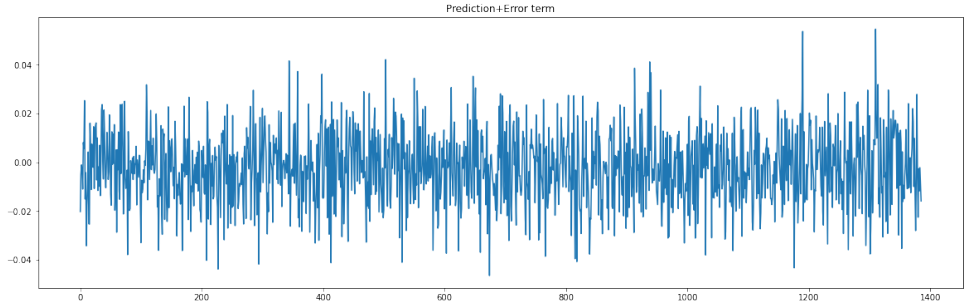

Figure 4.21: Graphical display of NIKKEI 225 out-of-sample prediction+Error term

FTSE 100

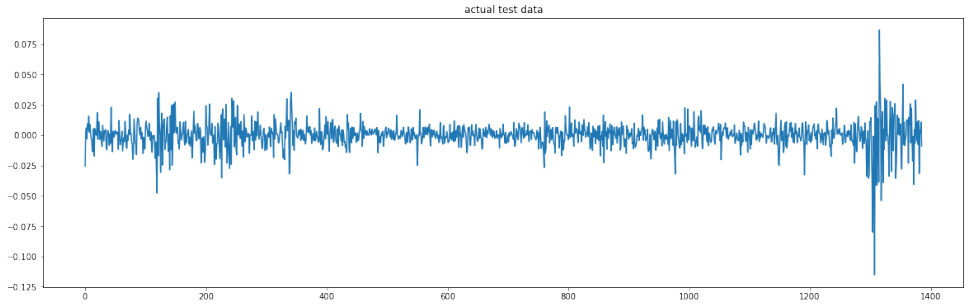

Figure 4.22: Graphical display of FTSE 100 test data

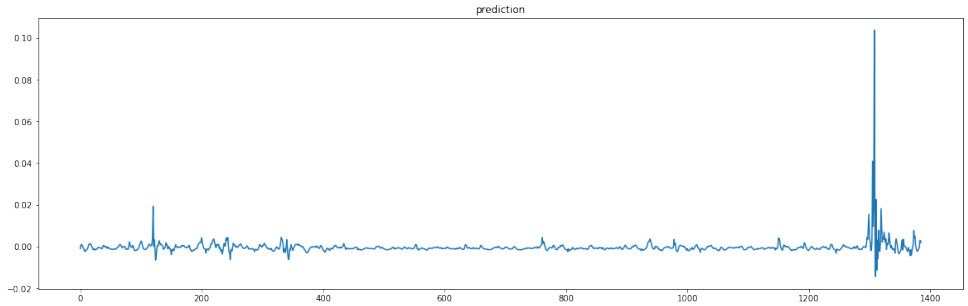

Figure 4.23: Graphical display of FTSE 100 out-of sample prediction

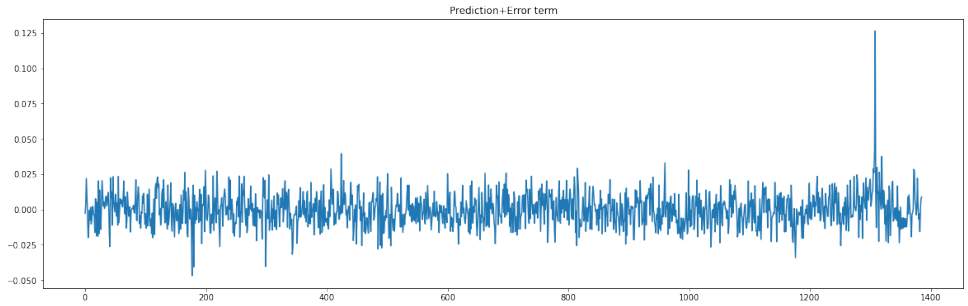

Figure 4.24: Graphical display of FTSE 100 out-of-sample prediction+Error term

S&P 500

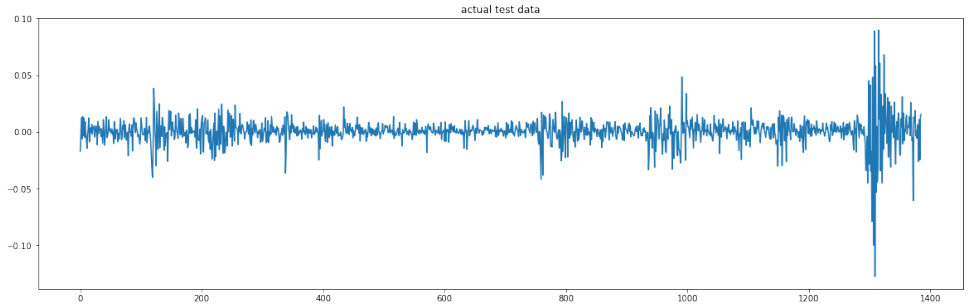

Figure 4.25: Graphical display of S&P 500 test data

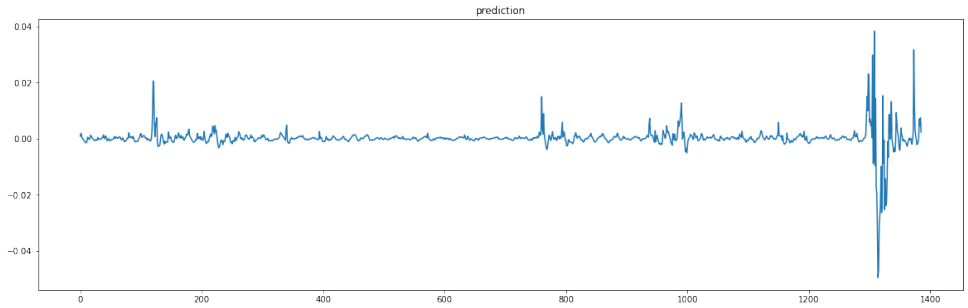

Figure 4.26: Graphical display of S&P 500 out-of sample prediction

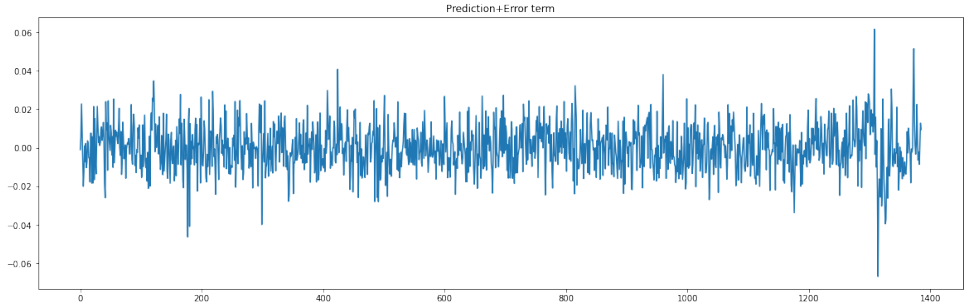

Figure 4.27: Graphical display of S&P 500 out-of-sample prediction+Error term

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

Acerbi, C. and Tasche, D. On the coherence of expected shortfall, *Journal of Banking and Finance*, Vol. 26, 2002, pp. 1496-1500.

Jorion, P. Value at Risk: The New Benchmark for Managing Financial Risk, McGraw-Hill, 2007.

Sun, W., Rachev, S., Chen, Y and Fabozzi, F., (2008). Measuring Intra-Day Market Risk: A Neural Network Approach.

James W. Taylor (2019). Forecasting Value at Risk and Expected Shortfall Using a Semiparametric Approach Based on the Asymmetric Laplace Distribution, *Journal of Business & Economic Statistics*, 37:1, 121-133.

Patton, A.J., Ziegel, J.F., Chen, R. (2019). Dynamic semiparametric models for expected shortfall (and Value-at-Risk), *Journal of Econometrics*, 211, 388-413.

Arimond A., et al (2020). Neural Networks and Value at Risk. arXiv:2005.01686

Nagai M. (2016) Estimation of Extreme Value at Risks Using CAViaR Models, *Graduate School of Economics, Hitotsubashi University.*

Abed, P., Benito, S., (2013). A detailed comparison of value at risk estimates, *Mathematics and Computers in Simulation*, 94, 258–276.

Abed, P., Benito, S., (2009). A Detailed Comparison of Value at Risk in International Stock Exchanges

Graves, A. (2013). Generating sequences with recurrent neural networks. arXiv preprint arXiv:1308.0850.

Abed, P., Benito, S., Lopez, C. A comprehensive review of Value at Risk methodologies, *Spanish Review of Financial Economics* 12(1), 2013.

Ruilova, J.C., Morettin P. A., (2020). Parsimonious Heterogeneous ARCH Models for High Frequency Modeling, *Journal of Risk and Financial Management.*

Yu, P., Yan, X. Stock price prediction based on deep neural networks, *Journal of Neural Computing and Applications* 32(5), 2020.

Buczyński, Mateusz; Chlebus, Marcin (2018) : Comparison of semiparametric and benchmark value-at-risk models in several time periods with different volatility levels, *e-Finanse: Financial Internet Quarterly, ISSN 1734-039X, University of Information Technology and Management, Rzeszów, Vol. 14, Iss. 2, pp. 67-82, http://dx.doi.org/10.2478/fiqf-2018-0013*

Bijelic & Ouijjane (2019) : Predicting Exchange Rate Value-at-Risk and Expected Shortfall:

A Neural Network Approach. *Master thesis, School of Economics and Management, Lund University.*

Kingma, D. and Ba, J. (2015) Adam: A Method for Stochastic Optimization. *Proceedings of the 3rd International Conference on Learning Representations (ICLR 2015).*

Teo Li Hui (2006) : Comparison of Value-At-Risk (VAR) Using Delta-Gamma Approximation with Higher Order Approach. *Master thesis, Department of Mathematics, National University of Singapore.*

James W. Taylor (2020). Forecast combinations for value at risk and expected shortfall, *International Journal of Forecasting*, 36, 428-441.

Chaoyi Lou (2019). Artificial Neural Networks: their Training Process and Applications *Department of Mathematics, Whitman College.*

IBM (2020). Neural Networks, https://www.ibm.com/cloud/learn/neural-networks

IBM (2020). Recurrent Neural Networks, https://www.ibm.com/cloud/learn/recurrent-neural-networks

Brownlee, J., (2016). Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras, Deep Learning for time series.

Locarek-Junge, H. and Prinzler, R. (1999). Using ANN to Estimate VaR, working paper.

S. A. Hamid and Z. Iqbal. Using neural networks for forecasting volatility of S&P 500 Index futures prices *Journal of Business Research 57 (2004) 1116–1125.*

He, K., Ji, L., Tso, G. K. F., Zhu, B., & Zou, Y. (2018). Forecasting Exchange Rate Value at Risk using Deep Belief Network Ensemble based Approach. *Procedia Computer Science*, 139, 25-32. https://doi.org/10.1016/j.procs.2018.10.213.

K. Hornik, M. Stinchcombe, H. White. Multilayer Feedforward Networks are Universal Approximators , Vol. 2, pp. 35Y-366, 1989

List of Figures

List of Tables