Department of Financial Mathematics

Compare the quality of forecasting models for value at risk

Author:
Hafees Adebayo Yusuff

Supervisor:
Prof. Ralf Korn

August 19, 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 1988 in Basel (Switzerland) by the Basel Committee on Bank Supervision (BCBS), involving the chairmen of the central banks of some European countries and the United States of America. This accord provides recommendations on banking regulations with regard to credit, market and operational risks. It aims to ensure that financial institutions hold enough capital on account to meet obligations and absorb unexpected losses.

For a financial institution measuring the risk it faces is an essential task. In the specific case of market risk, a possible method of measurement is the evaluation of losses likely to be incurred when the price of the portfolio assets falls. This is what Value at Risk (VaR) does.

Value at Risk(VaR) is the most common way of measuring market risk. It determines the greatest possible loss, assuming an α significance level under a normal market condition at a set time period.

Many VaR estimation methods have been developed in order to reduced uncertainty. It is however of interest to compare these method and determine the prevalence of one VaR estimation approach over others.

1.2 Literature review

The first papers involving the comparison of VaR methodologies, such as those by Beder (1995, 1996), Hendricks (1996), and Pritsker (1997), 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. 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 compared models in periods 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 in the time of the crisis the number of errors made by these models is relatively low.

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. Additional studies that find evidence in favour of parametric methods are Sarma et al.(2003), who compare Historical simulation and Parametric methods, and

Danielson and Vries(2000) in a similar comparison that also includes Extreme value theory 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. McAleer et al.(2009) showed that RiskMetrics$^{\text{TM}}$ was the best fitted model during a crisis, while Shams and Sina(2014) recognized GARCH(1,1) and GJR-GARCH as well forecasting models. In contrast to the results obtained by McAleer et al.(2009), the level of quality of forecasts generated by the RiskMetrics$^{\text{TM}}$ model was considered unsatisfactory by them. However, attention needs to be drawn to one difference in the samples, on which the study was conducted, i.e. the first one comes from a developed country (USA, S&P500), and the second one from a developing country (Iran, TSEM). Taylor(2020) evaluate Value at Risk 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 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. A possible method of measurement is the evaluation of losses likely to be incurred when the price of the portfolio falls. Value at Risk (VaR) does this.

According to Jorion (2001), “VaR measure is defined as the worst expected loss over a given horizon under normal market conditions at a given level of confidence. For instance, a bank might say that the daily VaR of its trading portfolio is \$2 million at the 99% confidence level. In other words, under normal market conditions, only 1% of the time, the daily loss will exceed \$2 million (99% of the time, their 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.

Mathematically,

Let $r_1, r_2, ..., r_n$ 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;

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

where $\sigma_t^2 = E[z_t^2|\Omega_{t-1}]$ 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.

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

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.

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 can 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 estimation 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 can be written as:

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

where σ =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.

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

High frequency data are those measured in small time intervals. This kind of data is important to study the micro structure of financial markets and also because their use is becoming feasible due to the increase of computational power and data storage. The HARCH(n) model was introduced by Müller et al. (1997) to estimate the VaR for this kind of data. In fact, this model incorporates heterogeneous characteristics of high frequency financial time series and it is given by

$$r_t = \sigma_t \varepsilon_t$$

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

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 and the c_j are parameters estimated using least squares.

Intraday data have been found to be useful in estimating features of the distribution of daily returns. For example, the realized volatility has been used widely as a basis for forecasting the daily volatility. 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). However, intraday data can be expensive, and resources are required for pre-processing. Given the ready availability of the daily high and low prices, an alternative way of capturing the intraday volatility is to use the intraday range. 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;

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

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

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

where Range_t is the difference between the highest and lowest log prices on day t; 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 is then expressed 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.

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 natural way to formalize this characteristic is to use some type of autoregressive specification

let $\{y_t\}_{t=1}^{T}$ and θ be a vector of portfolio returns and the probability associated with VaR respectively. Let x_t be a vector of time t observable variables(return or any other observable variables), and 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 $t-1$, The θ subscript is however suppressed from β_θ for convenience in notation: The general specification of CaViar would be:

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

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 parameters of CaViaR are estimated by quantile regression. The role of $l(x_{t-j})$ is to connect $f_t(\beta)$ to observable variables that belong to the information set.

The asymmetric slope is a variant of CaViar model, which allows the response to positive and negative returns to be different. It is modelled as:

$f_t(\beta) = \beta_1 + \beta_2 f_{t-1}(\beta) + \beta_3 (y_{t-1})^+ + \beta_4 (y_{t-1})^-$. where, $(y_{t-1})^+ = \max(y_{t-1}, 0)$ and $(y_{t-1})^- = \min(y_{t-1}, 0)$

Note: In this work, value at risk will be estimated based on financial log-returns. Historical simulation, Garch models, 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 computing systems vaguely inspired by the biological neural networks that constitute animal brains. Their name and structure are inspired by the human brain, mimicking the way that biological neurons signal to one another.

Neural networks are made of node layers, containing an input layer, one or more hidden layers, and an output layer. Each node (or artificial neuron), connects to another and has an assigned 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 is passed along to the next layer of the network.

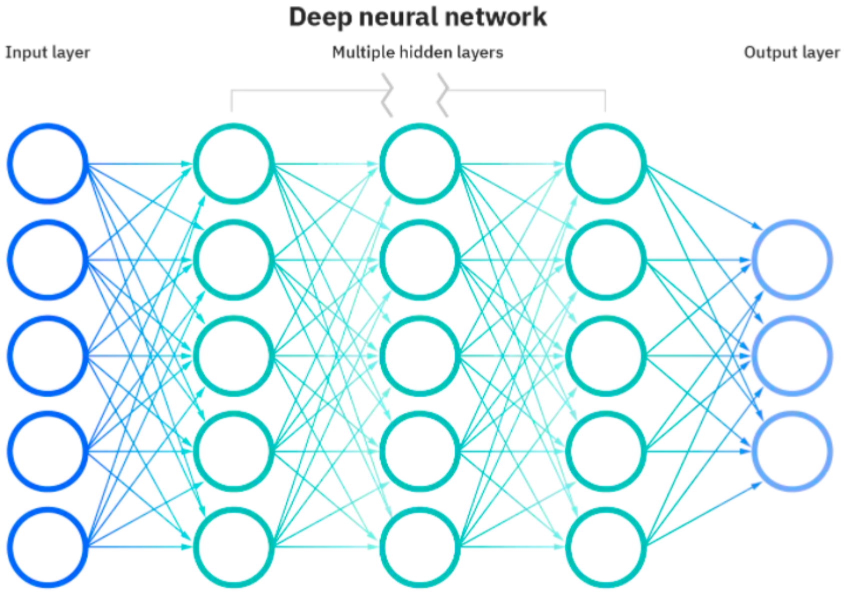

Figure 3.1: A figure showing the layers of a Neural Network (see IBM)

Neural networks rely 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.

One major advantage of Neural Networks is that they have a flexible nonlinear function mapping capability (Cybenko (1989) and Hornik et al. (1989)), this means that any model can be arbitrarily well approximated by a sufficiently large ANN. This model-free property of ANNs is an important advantage since returns do not display a specific nonlinear pattern. 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

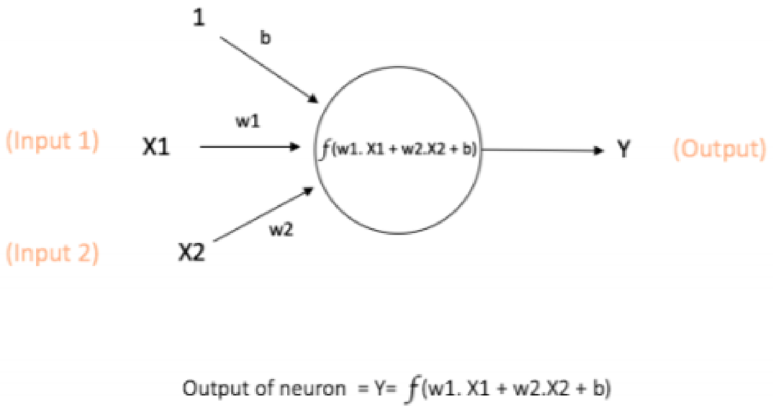

Figure 3.2: A single neuron of neural networks

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 $w_1x1 + w_2x_2 + b$. Besides the two numerical input values, there is one input value 1 with weight b, called the Bias. The main function of bias is to stand for unknown parameters or unforeseen factors. The output Y is computed by taking the dot-product of all input values and their associated weights and putting it into the function f. This function is called the Activation Function.

Activation functions are needed because many problems take multiple influencing factors into account and yield classifications. For example, if one encounters a binary classification problem, the results would be either yes or no, activation functions are needed to map the results inside this range. If one encounters a problem involving probability, then one would wish to see the predictions from the neural network being in the range of [0, 1]. This is what activation functions can do.

There are two types of activation functions: linear activation functions and non-linear ones. The biggest limitation of linear ones is that they cannot learn complex function mappings because they are just polynomials of one degree. Therefore, we always need non-linear activation functions to produce results in desired ranges and to send them as inputs to the next layer. The following subsection will introduce few generally used non-linear activation functions.

3.1.2 Activation Functions

An activation function takes the dot-product mentioned before as a input and performs a certain computation on it. We put a certain activation function inside of neurons of hidden layers based on the range of the result we expect to see. A notable property of activation functions is that they should be differentiable, because later we need this property 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]:

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

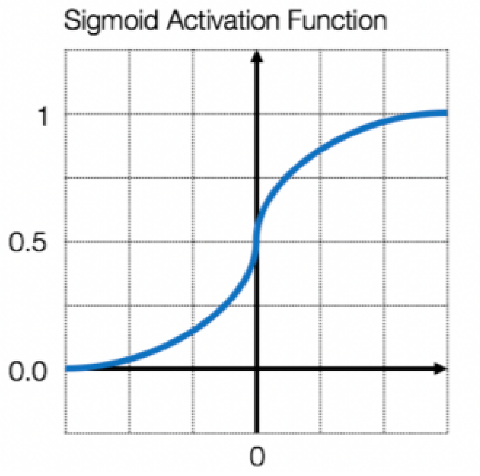

Figure 3.3: Sigmoid() Activation Function

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]. Since the probability of anything exists only between the range of 0 and 1, Sigmoid is a compatible transfer function for probability. Although the Sigmoid function is easy to understand and ready to use, it is not frequently used because it has vanishing gradient problem. This problem is that, in some cases, the gradient gets so close to zero that it does not effectively apply change to the weight. In the worst case, this may completely stop the neural network from further training. Second, the output of this function is not zero-centered, which makes the gradient updates go far in different directions. Besides, the fact that output is in the narrow range [0, 1] makes optimization harder. In order to compensate the shortcomings, tanh() is an alternative option because it is a stretched version of the Sigmoid function, in which its output is zero-centered.

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

tanh(x) $= \frac{sinh(x)}{cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
The advantage is that the negative input values will be mapped strongly negative and the zeros will be mapped near zero through this function. Therefore, this function is useful in the performance of a classification between two distinct classes. Though, this function is preferred over the Sigmoid function in practice (output in wider range), but the gradient vanishing problem still exists. The following ReLU function rectifies this problem using a relatively simple formula.

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

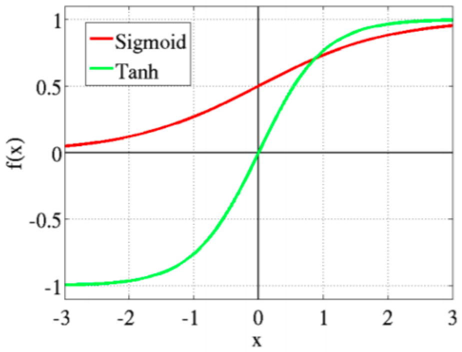

Figure 3.4: tanh() Activation Function

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

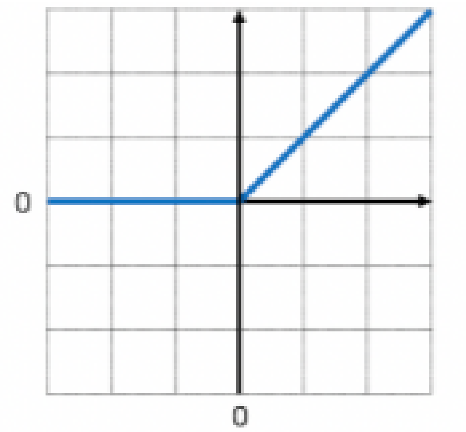

Figure 3.5: ReLU() Activation Function

It is used in almost all the convolutional neural networks or deep learning because it is a relatively simple and efficient function which avoids and rectifies the gradient vanishing problem. The problem of this activation function is that all the negative values become zeros after this activation, which in turns affects the results by not taking negative values into account.

We use different activation functions when we know what characteristics of results we expect to see.

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 take in several inputs to produce a single(or more) output(s). Normally, a Neural Network consists of an input and output layer with one or multiple hidden layers within. In a Neural Network, all the neurons(contained in each layers) influence each other, and hence, they are all connected. The manner in which the input neurons produce a certain output is intimately linked to the structure of the neural network. The two main classes of network architectures are discussed below

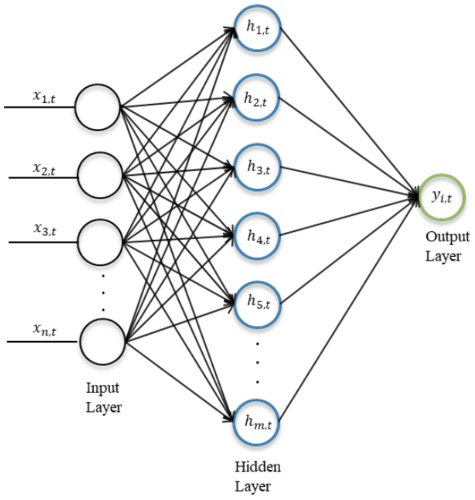

Figure 3.6: A fully connected FFN with a single hidden layer (see Anna & Bijelic, 2019)

Feed-forward Neural Network

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 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 capable of approximating virtually any function. 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 can then be reformulated to:

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

Although the model-free assumption underlying the feedforward neural network theoretically suggests that it should far better than the conventional GARCH(1,1) in predicting volatility, the feedforward neural network is subject to a major concern in that it precludes modelling time-dependencies in the data. This deficiency of not being able to take correlations between inputs into account is however resolved in the recurrent neural network, which is able to selectively pass information across sequences of elements by creating cycles in the network.

Recurrent Neural Network

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 means that the prediction a recurrent neural network node made at previous time step

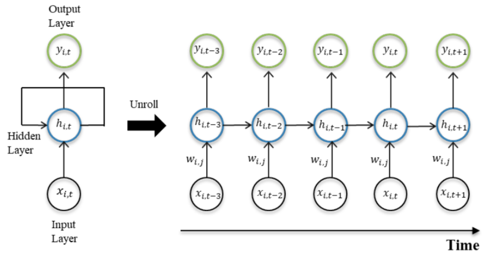

Figure 3.7: Representation of an unrolled plain vanilla recurrent neural network. (see Anna & Bijelic, 2019)

$t-1$ 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 (Tenti, 1996). Furthermore, by introducing time-lagged model components, it becomes evident that the utilization of a RNN is particularly desired when there are time dependencies in the data series at hand. Using the previous notation and assuming that the hidden states are the ones looped back, the output from a hidden node in the RNN model depends on the input values at time t, but also on its own lagged values at order p as shown below:

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

where $h_{k,t-p}$ represents the lagged hidden state values at order p, and γ_k a coefficient. Another distinguishing characteristic of recurrent networks is that they share parameters across each layer of the network. While feedforward networks have different weights across each node, recurrent neural networks share the same weight parameter within each layer of the network. That said, these weights are still adjusted in the through the processes of backpropagation and gradient descent to facilitate reinforcement learning. Recurrent neural networks leverage backpropagation through time (BPTT) algorithm to determine the gradients, which is slightly different from traditional backpropagation as it is specific to sequence data. The principles of BPTT are the same as traditional backpropagation, where the model trains itself by calculating errors from its output layer to its input layer. These calculations allow us to adjust and fit the parameters of the model appropriately. BPTT differs from the traditional approach in that BPTT sums errors at each time step whereas feedforward networks do not need to sum errors as they do not share parameters across each layer.

Through this process, RNNs tend to run into two problems, known as exploding gradients and vanishing gradients. These issues are defined by the size of the gradient, which is the slope of the loss function along the error curve. When the gradient is too small, it continues to become smaller, updating the weight parameters until they become insignificant—i.e. 0. When that occurs, the algorithm is no longer learning. Exploding gradients occur when the gradient is too large, creating an unstable model. In this case, the model weights will grow too large, and they will eventually be represented as NaN. One solution to these issues is to reduce the number of hidden layers within the neural network, eliminating some of the complexity in the RNN model(see, IBM 2020).

Long Short-Term Memory Recurrent Neural Network

Originally introduced by Hochreiter and Schmidhuber (1997), a main characteristic of LSTMs – which are a sub class of recurrent neural networks - is its purpose-built memory cells, which allows it to capture long range dependencies in the data. From a model perspective, LSTMs differ from other neural network architectures in that they are applied recurrently

The output from a previous sequence of the network function serves – in combination with the next sequence element - as input for the next application of the network function. In this sense, the LSTM can be interpreted as being similar to an HMM (Hidden Markov Model), in that there is a hidden state which conditions the output distribution. However, the LSTM hidden state not only depends on its previous states, but it also captures long term sequence dependencies through its recurrent nature. Maybe most notably, the receptive field size (i.e the size of the region in the input that produces the feature) of an LSTM is not bound architecture wise as in case of simple feed forward network and CNN. Instead, the LSTM's receptive field depends solely on the LSTMs ability to memorize the past input. 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. It has 90 nodes in the input layer and 150 nodes in hidden layer, with 'tanh' as the activation function. The following are the hyper parameter used in within the LSTM neural network:

- The optimizer mirrors the use of the gradient descent method, as it controls the magnitude of changes to the weights within the network, with regard to the loss gradient. The optimization algorithm Adam is chosen as the optimizer in the LSTM neural network, because in contrast to the stochastic gradient descent, which maintains one single learning rate for all weight updates, Adam separately adapts a learning rate for each network weight. Some of Adam's 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, and in this case the batch size is set to 128. As such, volatility inputs are fed in the network through numerous batches, each containing 128 inputs. After the propagation of a batch, the network is trained before receiving another batch of 128 inputs. This operation is repeated until the end of the training and validation processes, i.e. when all inputs are propagated.

- The look ahead specifies the number of time steps, i.e. the lagged inputs the RNN should use to forecast the desired outputs. To enforce the ability of the RNN to find repeating temporal patterns, the look ahead technique is used with the LSTM neural network. However, the sliding window (look ahead) should not be too small or the time dependencies will not be well captured, nor too large to avoid feeding excessively the same inputs, as this could lead to overfitting problems. 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 GRU 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) 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-return, which is given as

$$R_{log} = ln(\frac{R_f}{R_i})$$

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.

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):

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

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 each training is done five times with varying initialization and the best one is selected for VAR estimation.

NIKKEI

The training and validation loss continues to decrease as the number of epochs increases until 200 epochs, after which the model started 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 kept reducing. It was also observed that the most effective size of the LSTM network was 350, and the higher the intensity of regularization applied, the better the model was able to perform.

Trials	Epochs	Dropout	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 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 was also low. As a result of the high number of epochs and low regularization effect, it suffered from the highest variance and produced the worst result.

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

Trial 3 is the third in line, the model trained for 300 epochs which is longer than the required 200 epochs, this led 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 over t rapidly, that was why this trial performed worse than the second trial.

Trained with the desired 200 epochs, with a low dropout of 0.5, and a low LSTM size of 256. This 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.

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 and high bias. The dropout regularization intensity was also high enough at 0.75, and the size of the Long Short Term Memory (LSTM) network at 350 was large enough.

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 started to over
t. This was made evident as a result of a slow and steady rise in the validation loss after 205 epochs, while the training loss kept reducing. It was also observed that the most effective size of the LSTM network was 350, and the higher the intensity of regularization applied, the better the model was able to perform.

Trials	Epochs	Dropout	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 the model trained for being within the range of avoiding high variance and high bias. The dropout regularization intensity was also high enough at 0.75, and the size of the Long Short Term Memory (LSTM) network at 350 was large enough.

The third attempt produced the next best result, it fell short of case 10 because it trained more than 200 epochs. As a result of this, the model suffered from a higher variance as compared to case 10.

Although, the forth trial was trained for the required 200 epochs but fell 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.

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

Suffered the same set back as the second trial, the first trial was trained with an even larger number of epochs at 500. This made it perform the worst compared to the rest of the trails.

Appendix A

Graphs of log-return series of each stock market

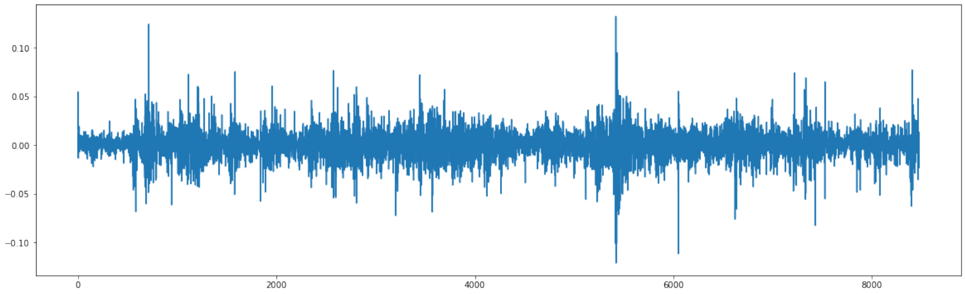

Figure A.1: log-returns of Nikkei 225

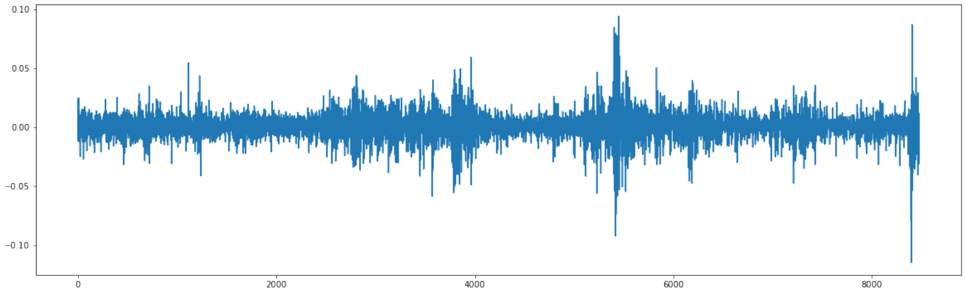

Figure A.2: log-returns of FTSE 100

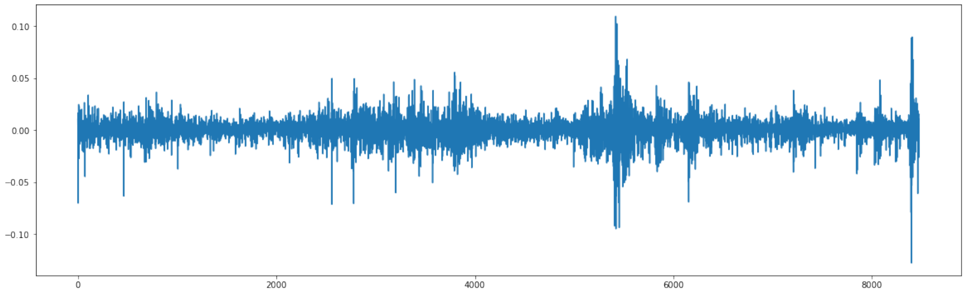

Figure A.3: log-returns of S&P 500

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).

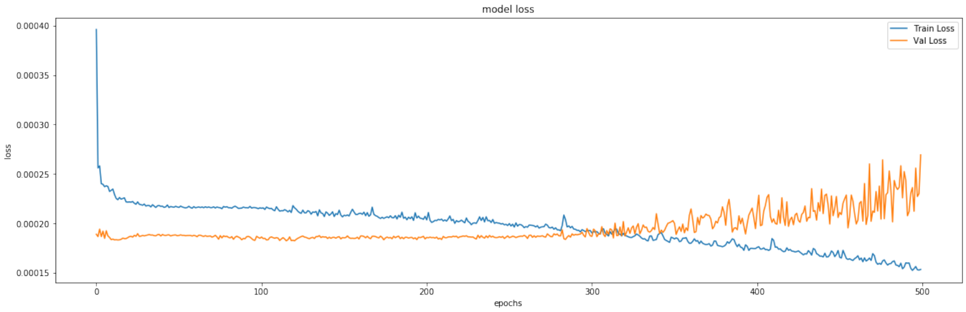

Figure B.1: Training and Validation loss functions under Trial 1 (Nikkei 225))

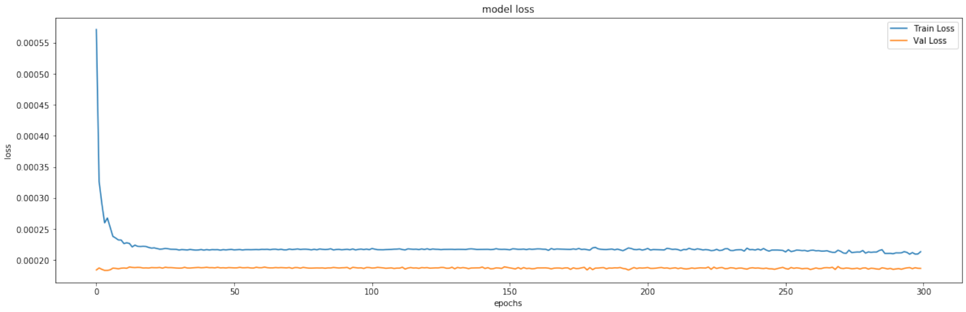

Figure B.2: Training and Validation loss functions under Trial 2 (Nikkei 225))

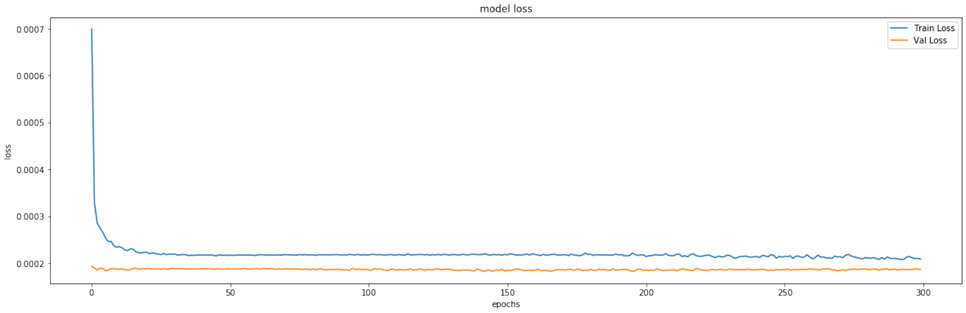

Figure B.3: Training and Validation loss functions under Trial 3 (Nikkei 225))

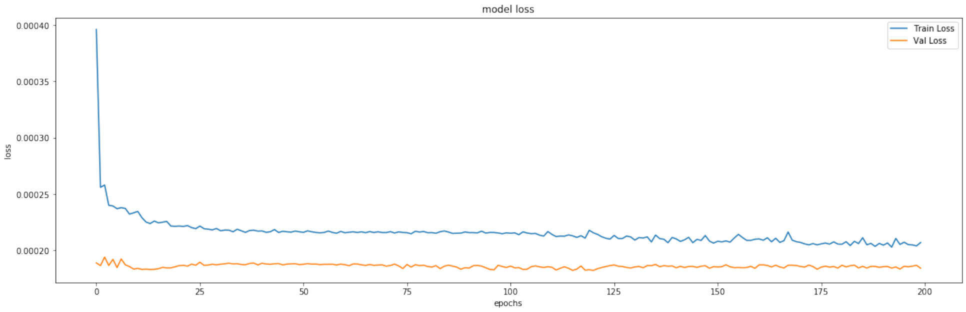

Figure B.4: Training and Validation loss functions under Trial 4 (Nikkei 225))

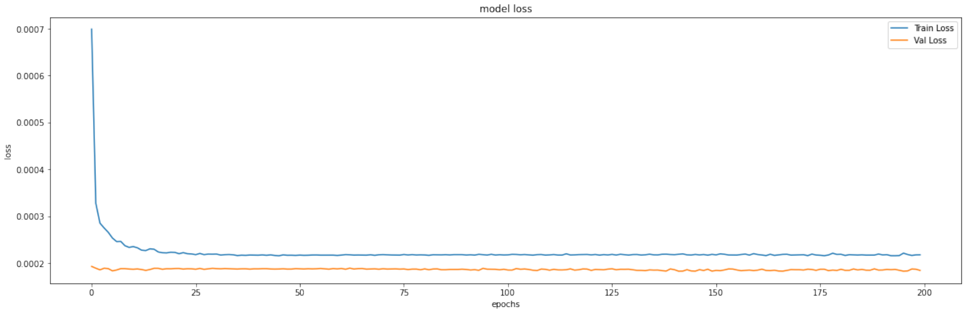

Figure B.5: Training and Validation loss functions under Trial 5 (Nikkei 225))

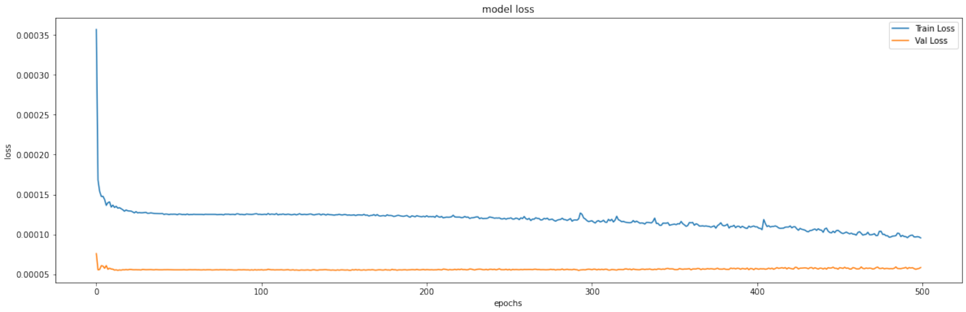

Figure B.6: Training and Validation loss functions under Trial 1 (FTSE 100))

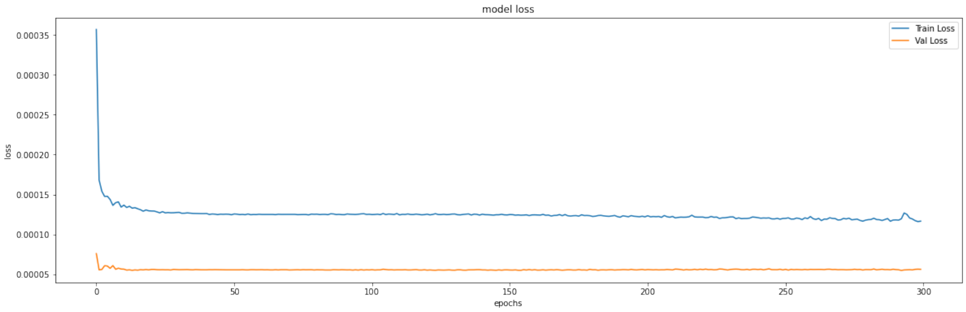

Figure B.7: Training and Validation loss functions under Trial 2 (FTSE 100))

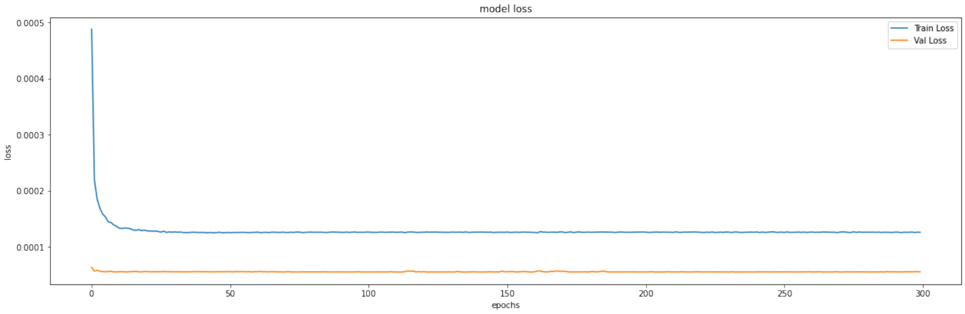

Figure B.8: Training and Validation loss functions under Trial 3 (FTSE 100))

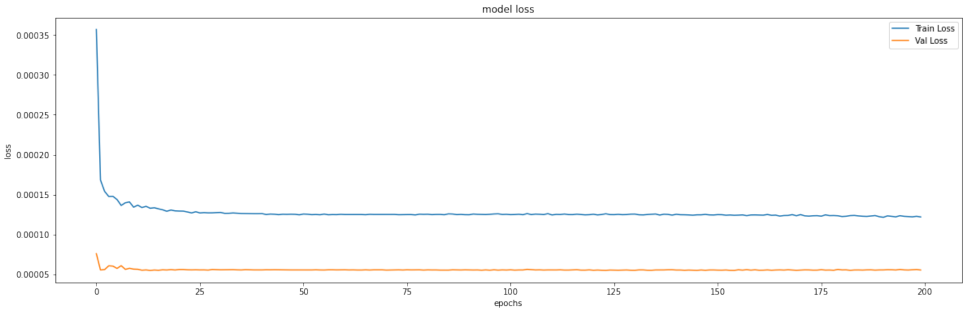

Figure B.9: Training and Validation loss functions under Trial 4 (FTSE 100))

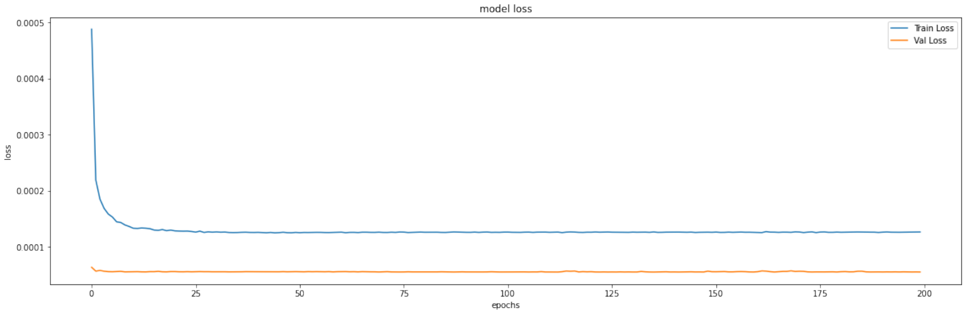

Figure B.10: Training and Validation loss functions under Trial 5 (FTSE 100))

Appendix C

Another example

C.1 More stuff

Bla bla.

Bibliography

R. Hill. The elastic behaviour of a crystalline aggregate. *Proceedings of the Physical Society. Section A*, 65:349–354, 1952.

E. Kröner. Bounds for effective elastic moduli of disordered materials. *Journal of the Mechanics and Physics of Solids*, 25(3):137–155, 1977.

List of Figures

List of Tables

Index