Deep Calibration of Option Pricing Models

Sahil Dadah

Supervisor: A/Prof. Peter Ouwehand

A dissertation submitted to the Faculty of Commerce, University of Cape Town, in partial fulfilment of the requirements for the degree of Master of Philosophy.

October 7, 2022

MPhil in Mathematical Finance,
University of Cape Town.

Declaration

I declare that this dissertation is my own, unaided work. It is being submitted for the Degree of Master of Philosophy in the University of the Cape Town. It has not been submitted before for any degree or examination in any other University.

October 7, 2022

Abstract

This dissertation investigates the calibration efficiency of short rate models using deep neural networks. The main focus is on the calibration of one-and-two factor Hull-White models to caplets and swaptions data, where the inputs are interest rate derivative prices or implied volatilities, and the outputs are the model parameters. A direct and indirect neural network calibration framework is adopted. The former method involves a direct inversion of the standard option pricing function using neural network. The indirect framework uses two consecutive steps; the first step estimates the option pricing function using a neural network. This is followed by applying the pre-trained model in a calibration procedure to fit the model parameters to a set of market observables. The neural networks are trained using simulated data and an optimum set of hyperparameters is obtained via the Bayesian optimization. The best set of hyperparameters is used to train the networks and tested on out-of-sample actual market yield curves data. It is shown that the direct method has substantial improvements in time with a sacrifice in accuracy (a mean relative error of 2.88%). On the other hand, using the indirect method, it is shown that the calibrated parameters reprice the set of options to a mean relative error of less than 0.1% (similar to numerical calibration), with a significant improvement in speed whose execution is twenty-six times faster compared to the conventional calibration procedures currently used.

Acknowledgements

First and foremost, I would like to thank my supervisor A/Prof. Peter Ouwehand for his considerable efforts to make this dissertation as smooth as possible. I appreciate him for not only his continuous support and assistance for the completion of this dissertation, but also providing me with a with topic that I enjoyed exploring.

I would also like to express my sincere gratitude to the AIFMRM department for accommodating me to this masters degree and also providing me with the necessary funds to undertake this degree. I would like to thank my fellow colleagues, Jenna Goosen for her assistance to bridge my knowledge in machine learning and Darvesh Shibduth for his support throughout the masters degree and proof reading my work. Many thanks to my employers Elenjical Solutions for providing me with the time to complete my dissertation.

Finally, I would like to thank my family, my parents in particular who have always been by my side to support me in my endeavours. All other unmentioned people that supported me towards the completion of this dissertation are also thanked.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and motivation

The classical theory of option pricing models was laid out by Black and Scholes (1973) and Merton (1973) in their seminal papers. In those papers, the famous Black-Scholes formula made its debut, together with the introduction of Itô calculus to the financial world (Itô, 1944). However, this model was developed under a set of assumptions which does not accurately capture the behaviour of underlying market processes. Therefore, a variety of more complex and sophisticated models have been developed to address this concern. Most of these models have attempted to incorporate the stochastic nature of interest rates and volatilities which were assumed to stay constant in the Black-Scholes framework. This research focuses on the use of the stochastic interest rate models, also known as short rate models. In particular, the one-and-two factor Hull-White models are considered.

In order to obtain fair prices using a financial model, calibrating it to desired financial instruments is a crucial underlying process (Büchel *et al.*, 2021). These instruments, also known as *calibration instruments*, are typically chosen as highly liquid assets with easily accessible prices. Consequently, the calibrated model is used to price less liquid and more complex financial instruments. Therefore, the aim of this dissertation is to calibrate short rate models to liquid interest rate options.

Calibration is an intuitive procedure as it deals with the selection of model parameters that replicates the market observable information. This process can be time-consuming which makes it an expensive task performed by financial institutions on regular basis. Ideally, traders would prefer a calibration model which accurately captures the market movements. Accurate models are generally more complex and take longer to calibrate. As a result, these models are not of practical use during the calibration. More recently, Hernandez (2016) explored the use of deep neural networks in the calibration problem and showed that neural networks calibrate 225 times faster than conventional numerical calibration techniques. This

was achieved by performing the bulk of the calibration in an offline space away from the trading desk, allowing the online calibration to run significantly faster (equivalent to multiplying a few matrices). As a consequence, this enables the trader to select financial models purely based on the efficacy in pricing and hedging, rather than the computational time required to calibrate. Therefore, to explore this enhanced effect, this paper considers the calibration of the one-and-two factor Hull-White models to at-the-money (ATM) caplet and swaption data using deep neural networks.

1.2 Literature review

1.2.1 Short-rate models

Short rate models are used to describe the evolution of interest rates. Understanding and modelling the evolution of interest rates is a non-trivial exercise in financial research (Gibson *et al.*, 2010). However, this comes with a great deal of significance, as the interest rate models have a variety of applications, such as investments, long term debts, valuations of contingent claims, together with pricing, hedging and managing risks associated with interest rate derivatives (Canto, 2008).

In the past few decades, multiple researchers have attempted to best describe the yield curve projections using single factor models. These models assume the interest rate is driven by one source of noise for all maturities. One of the first researchers to propose a one-factor model for interest rates was Merton (1973). He developed a stochastic model which allowed negative rates (due to the normal distribution assumption) and assumed a constant volatility. However, these assumptions do not hold in reality and thus the model was seen as inaccurate. Furthermore, due to the lack of boundary conditions on the mean and variance, the model allowed negative and infinite rates, which led to model instability (Gibson *et al.*, 2010). Shortly after, Vasiček (1977) attempted to improve the Merton model, by proposing a model with a mean reverting process, also known as the Ornstein-Uhlenbeck process. In the Vasiček model, interest rates are defined as a function of time, market risk and equilibrium value, and revert to a mean of these factors over time. Despite the Vasiček model being widely applicable in future valuations and bond pricing, the main drawback of the model was allowing the interest rates to become negative, an undesirable effect for any economy in the past. The shortcomings of the Vasiček model were addressed by Dothan (1978) and Rendleman and Bartter (1980) where they proposed a log-normal distribution for the evolution of the short rate. Using this as a foundation, Cox *et al.* (1985) went on to propose a noncentral-chisquared distribution model for the instantaneous spot rate. How-

ever, all these models had one thing in common–they did not use the current term structure of interest rates as an input to the model, but rather provided it as an output. As a result, these models suffered from poor fitting of the initial term structure. This issue was first addressed by Ho and Lee (1986), and later in the Hull and White (1990) and Black and Karasinski (1991) model.

In particular, the Hull-White model was introduced as an extension to the Vasiček model. Like the Vasiček model, the Hull-White model is mean-reverting and has the possibility of producing negative interest rates. It adapted the no-arbitrage framework proposed by Ho and Lee, which allowed the model to fit the current term structure of interest rates. In addition, Björk (1998) and Brigo and Mercurio (2006) showed that closed form solutions for interest rate options can be derived under this model by considering the pricing of zero coupon bonds. As a result, this model is widely used by practitioners in the pricing of interest rate caps, floors and swaptions. Despite the good tractability of the Hull-White model, the major drawback is the inability to reproduce large volatility surfaces due to lack of free calibration parameters (Scheffer and Zacharias, 2018).

Beside one-factor models, there are various multi-factor short rate models. Longstaff and Schwartz (1992), Hull and White (1994), and Chen (1996) are the most famous multi-factor models. These models work on the assumption that more than one stochastic factor affects the evolution of the interest rates. They tend to be complex to solve, but provide a better accuracy as more factors are incorporated. Generally, multi-factor models are preferred for the evolution of interest rates over single-factor models. However, there is minimal literature supporting multi-factor models when it comes to performance in pricing interest rate derivatives. According to the theoretical arguments posed by Rebonato (1999), one-factor models can suffice for pricing of interest rate caps and floors, but are unlikely to be appropriate for swaption pricing, due to the dependence of the correlation between interest rates of different maturities. This drawback is a consequence of assuming perfect instantaneous correlation (correlation equal to one) of interest rates in one-factor models, contradicting the empirical observations (Driessen *et al.*, 2003).

Despite a variety of short rate models proposed by various researchers, we will focus on the one-factor and two-factor Hull-White models (equivalently G2++ model) for calibration. These models are analytically tractable and provide good accuracy for pricing contingent claims. Furthermore, the initial term structure of interest rates is fitted exactly, which improves model stability and calibration. Typically these interest rate models are calibrated to caps, floors or swaptions (Brigo and Mercurio, 2006).

1.2.2 The calibration problem

The calibration of a market model refers to the procedure of determining the optimal set of parameters to 'best' fit the market quotes. In financial models, this process requires inverting the derivative pricing function. Most derivatives pricing models used cannot be inverted directly, thus numerical methods are used for calibration (Cont, 2010).

The standard approach for calibration of an option pricing model given a set of observed derivative prices is through minimization of a cost function. Different cost functions are available to suit the practicality (e.g. calibration time) of different models, ranging from convex functions to more complex structures with multiple local minima, for which a global minima is required to obtain an accurate solution (Hernandez, 2016). For illustration, we shall use the common least squares method as the cost function.

Consider a model with n parameters to calibrate using N market quotes, either option prices or implied volatilities. The least-squares cost function[1] will be denoted by

$$\inf_{\theta \in S \subseteq \mathbb{R}^n} \sum_{i=1}^{N} (Q_i^{\theta} - Q_i^{mkt})^2 \tag{1.1}$$

where Q_i^{mkt} represents the observable market quotes, and Q_i^{θ} is the model quote based on parameter values θ. In the optimization process, we attempt to recover parameters that minimize the distance between the model price and observed market prices.

Various kinds of optimization schemes can be used to minimize the cost function, the common ones include the Bisection Method, Newton's Method and the gradient descent. Using one of these schemes, solving for the parameter set θ yields to

$$\theta^* = \underset{\theta \in S \subseteq \mathbb{R}^n}{\operatorname{argmin}} \text{ Cost } (\theta, Q^{mkt}) = \mathbf{\Phi}(Q^{mkt})$$

where $\mathbf{\Phi}$ is the calibration function. Therefore, the calibration problem can now be summarized as

$$\Phi : \mathbb{R}^N \to \mathbb{R}^n : \bar{Q} \to \theta^*$$

where the function $\mathbf{\Phi}$ takes an N-sized vector $\bar{Q}$, which incorporates market observable quotes, and outputs an n-sized vector θ^* of calibrated parameters.

Therefore, the main focus during the calibration relies on how well the approximation of the function $\mathbf{\Phi}$ is done. When considering calibration of short rate

[1] Miller (2006) contains the statistical background for the least-squares cost function.

models, Brigo and Mercurio (2006) give details on the possible methods of calibrating several short rate models to caps and swaptions. In particular, one of the methods outlines fitting model implied volatilities to the option implied volatilities (Brigo and Mercurio, 2006). Park (2004) used this technique to calibrate the Vasiček model and the one-factor Hull-White model to historical volatilities. Meng *et al.* (2013) found that calibration of the Hull-White model to historical swaption volatilities provides better estimates of the future volatility compared to calibration using swaption prices. This dissertation will follow a similar approach, whereas, the calibration of the models will be processed using implied volatilities in preference to option prices. In addition to that, this research focuses on calibration of most liquid options, in this case ATM caplets and swaptions (Liu *et al.*, 2019).

1.2.3 Neural network calibration

Application of artificial neural networks (ANNs) in the field of derivative pricing was first explored by Haykin and Lippmann (1994). They used a non-parametric method to assess the use of ANNs in estimating the derivative pricing function. With the up-rise of artificial intelligence, the application of neural networks to financial model calibration has been closely examined by researchers.

Hernandez (2016) came up with the idea of using neural networks in estimating the pricing function, and presented a novel approach for calibration of option pricing models using neural networks. He calibrated the one-factor Hull-White model using 156 Swaption prices and 44 points on a bootstrapped yield curve. The calibration function associated was

$$\Phi : \mathbb{R}^{200} \rightarrow \mathbb{R}^2$$

where the two calibrated parameters were α, the rate of mean reversion, and σ, the volatility of the short rate. Hernandez (2016) selected the mean reversion parameter $\theta(t)$ to fit the current term structure. The performance of the neural network calibration was comparable to traditional calibration techniques for a period between six month to one year, beyond which a significant degradation in the performance of the neural network was observed. Overall, the results showed that the use of Neural Networks presents a substantial time improvement to the calibration process.

Mavuso *et al.* (2017) built on the foundation laid out by Hernandez (2016), but instead of selecting $\theta(t)$ to fit the current term structure, they adopted it as an extra parameter to calibrate. Mavuso *et al.* (2017) first validated the model using the one-factor Hull-White, and then developed a model to calibrate a mixture model consisting of two Hull-White models. For comparison purposes, they used the

same data set as Hernandez (2016), and their conclusion agreed with the findings of the latter.

Liu *et al.* (2019) approached the calibration in two steps: a neural network was trained to learn the swaption pricing function under Trolle–Schwartz model[2], followed by a backward pass of applying the resulting neural network to calibrate the model parameters to a set of market observables. This framework was found to be effective and efficient in calibration of higher-dimensional stochastic volatility models. Büchel *et al.* (2021) extended this investigation by analyzing the performance of the two-step approach by using historical market data as opposed to simulated data used by Liu *et al.* (2019). The trained model was tested to actual market yield curves unseen by the network. Büchel *et al.* (2021) concluded that this calibration framework performed well under real-time data and under different stressed market conditions. Itkin (2019) calibrated call options using market data in a no-arbitrage pricing fashion using a trained neural network. Horvath *et al.* (2021) presented a neural network method which calibrated the full implied volatility surface within milliseconds using stochastic volatility models (the rough Bergomi model, the Heston model, and SABR model). Prior to that, Bayer and Stemper (2018) presented an efficient way of calibration by combining the standard Levenberg-Marquadt calibration[3] routine with a neural network regression to replace the Monte-Carlo simulations[4] on stochastic volatility model. More recently, Alaya *et al.* (2021) used synthetic covariance matrices between forward rates of different maturities to calibrate the five parameters in the G2++ model using neural networks. The proposed calibration technique performed very quickly (less than 0.3 seconds for 2000 calibrations) and had minimal errors with a good fitting.

In order to access the speed advantage, Hernandez (2016) emphasised on splitting the processes in two distinct phases: firstly using historical and simulated data to train the model (training phase), followed by using the trained model to estimate the model parameters (calibration phase). During the training phase, the main criterion is to approximate the calibration function Φ as accurately as possible. The training of the model, which is a computationally expensive process, can be offloaded to an offline space, away from where the actual trading or hedging takes place (Hernandez, 2016). This training would need to take place infrequently (e.g. every 6 months), unlike the traditional technique where the entire calibration process is done on daily basis. Once the training is completed, the trained neural network is taken to an online space (e.g. trading desks) to be used for the calibra-

[2] Schumann (2016) discusses the Trolle-Schwartz model.
[3] Moré (1978) contains the theory behind the Levenberg-Marquadt algorithm.
[4] Mooney (1997) details the Monte-Carlo simulation approach.

tion. The online calibration is equivalent to multiplying a few-small sized matrices, making the calibration process significantly faster (Hernandez, 2016).

The primary aim of this dissertation is to develop a robust, efficient and accurate calibration framework using deep neural networks. To achieve this, this paper explores the one-factor and two-factor Hull-White models. The one-factor model is calibrated to ATM caplets data, and the latter includes the addition of ATM swaption data. Each of these model are calibrated using a direct and indirect calibration framework. The direct framework is analogous to the standard calibration, where a neural network is trained to learn the inverse pricing function. On the other hand, the indirect method uses a two-step approach initially presented by Liu *et al.* (2019). In the first step, a neural network is trained to learn the pricing function, and the latter step uses the pre-trained model in a calibration framework. The efficacy of each of these calibration frameworks are then compared to the traditional numerical calibration technique.

Following this introductory chapter, Chapter 2 provides an overview of neural networks, by covering the fundamental facets such as the architecture, hyperparameters and data preprocessing. The pricing of interest rate derivatives under the Black model and Hull-White models are presented in Chapter 3. Using the derived analytical solutions, Chapter 4 covers the generation of synthetic option prices, and establishes the neural network structure of two calibration frameworks. Chapter 5 discusses the calibration results obtained from training the neural networks, and Chapter 6 concludes this dissertation.

Chapter 2

Neural Networks

This chapter aims to provide an intuitive understanding on the fundamentals of Neural Networks. An introduction to the basic architecture and structure is initially presented followed by a detailed analysis of how information is passed through the neurons using a specific activation function. This forms the basis of the neural network computations and helps practitioners interested in neural network understand and develop architectures to suit their respective application. Consequently, optimizers that dictate the learning process of machine learning are discussed. The gradient descent technique forms the basis of the most common optimizers used, with several modifications derived from it to improve the training process. Hyperparameters are then introduced with methodologies of optimizing this set of parameters. According to the evidence gathered from the literature, the Bayesian optimization was found to be the most suitable for this dissertation. Finally, this chapter concludes by an overview of pre-processing the sample data before training the network.

2.1 Introduction to neural networks

Neural networks, also known as Artificial Neural Networks (ANNs) are computational units inspired by the neurological structure of a human brain. The basic structure of a neural network consists of three components, namely, an input layer, hidden layers and an output layer. Neurons starting from the input layer, are interconnected with the subsequent layer through trainable weights, a bias term and a non-linear activation function. The entire connection built from these neurons form a network. This network is trained to adjust the collective weights of all the layers in order to produce an output vector $y = (y_1, y_2, \ldots, y_n), n \in \mathbb{Z}$, that optimizes a pre-defined objective function given the input vector $x = (x_1, x_2, \ldots, x_p), p \in \mathbb{Z}$. In a supervised ANN, the forward pass of information trained using an optimizer is followed by a back propagation of the output error through the network, whilst

re-adjusting the weights accordingly. This cycle of forward propagation and back propagation is iteratively performed with multiple inputs until the objective function has been optimized.

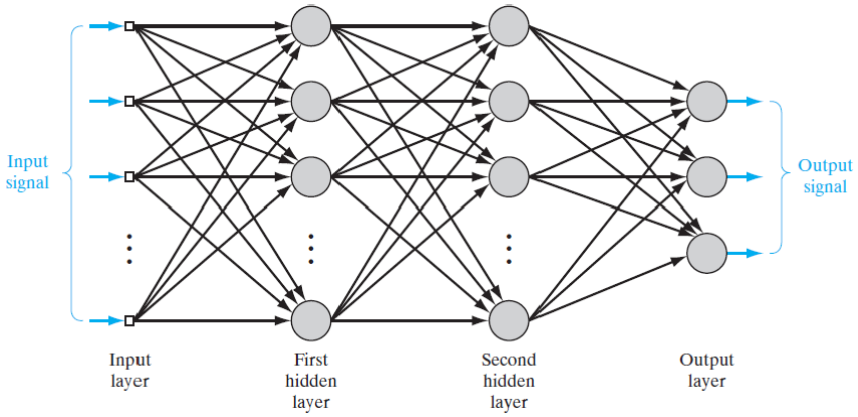


Fig. 2.1: Architecture of a feed forward deep neural network (Perez Cruz and Treisman, 2018).

The structure of the neural network plays a vital role in estimating a function. A network with multiple hidden layers has a greater likelihood of learning non-linear input-output mappings. The fundamental motivation of approximating non-linear functions using neural networks stems from the Universal Approximation Theorem. Using the results of Cybenko (1989) and Haykin and Lippmann (1994), the universal theorem for arbitrary width and bounded depth is as follows:

Theorem 2.1. *Let ψ: $\mathbb{R} \to \mathbb{R}$ be a non-constant, continuous and bounded function (activation function). Then for every continuous function $f : [0,1]^n \to \mathbb{R}$, $n \in \mathbb{Z}^+$ and $\epsilon > 0$, there exists a set of real constants α_i and w_{ij} , where $i = 1, \dots n$, $j = 1, \dots n_1, n_1 \in \mathbb{Z}^+$, such that:*

$$\sup_{x \in [0,1]^n} |f(x) - f^{NN}(x)| < \epsilon,$$

where

$$f^{NN}(x) = \sum_{i=1}^{n} \alpha_i \cdot \psi \left(\sum_{j=1}^{n_1} w_{ij} x_j + w_{i0} \right).$$

This powerful theorem tells us that neural networks can have any kind of universality, such that, no matter how complex the function $f(x)$ is, there is a network that can approximate this function. With regards to that, the option prices are continuous as a function of model parameters, thus making it suitable to estimate the calibration function using neural networks.

2.2 Neuron activation

Each layer of the network consists of neurons, which are building blocks to the structure. Every neuron transmits information from the previous layer to the next layer through activation functions. To illustrate this, we consider the j^{th} neuron in the l^{th} layer that is connected to n neurons with values $\{x_k\}_{i=1}^{n}$. The input of the j^{th} neuron is given by

$$net_j^l = \sum_{k=1}^{n} w_{jk}^l x_k^{l-1} + b_j^l$$

where w_{jk}^l is the weight for the connection from the k^{th} neuron in the $(l-1)^{th}$ layer to the j^{th} neuron in the l^{th} layer, and b_j^l is the bias term on the j^{th} neuron in the l^{th} layer.

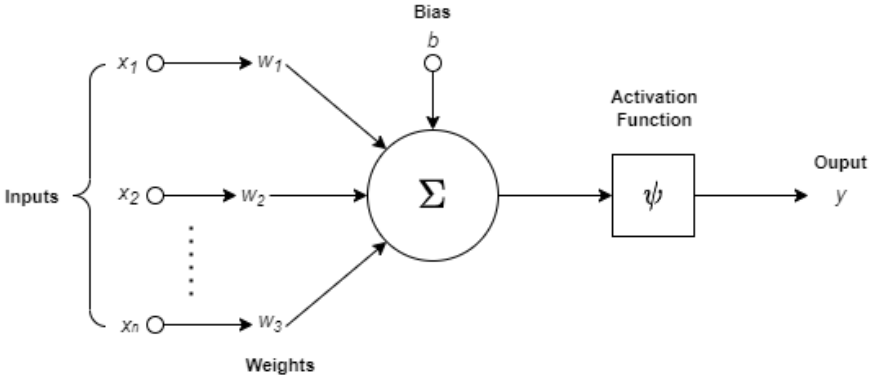


Fig. 2.2: Composition of a neuron.

The output value of the neuron is obtained after the application of the activation function ψ, to the input, which is given by

$$a_j^l = \psi\left(net_j\right). \tag{2.1}$$

It is important to note, the output value obtained following the application of the activation function is analogous to f^{NN} from Theorem 2.1.

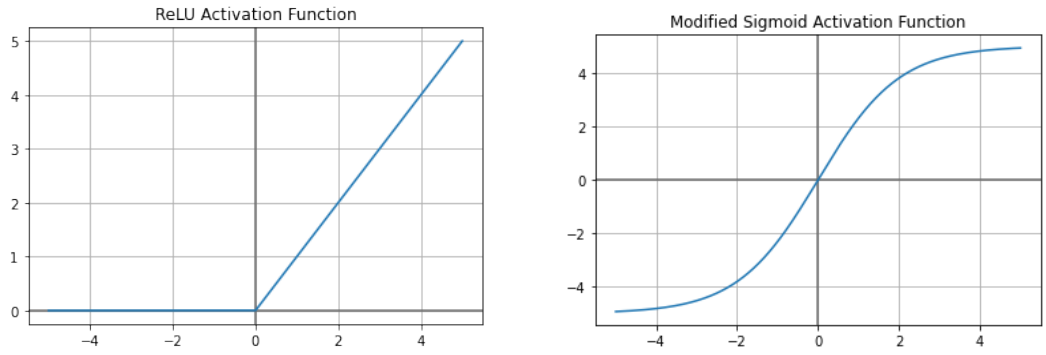
The role of the activation function is crucial in processing the information. Activation functions have the ability to decide whether a neuron should be activated or not to propagate the data (Szandała, 2021). This is achieved by comparing the output value of the node, given by Equation 2.1, to a threshold (Szandała, 2021). The neuron is activated only if the value obtained is greater than the threshold. In addition to that, activation functions control the non-linearity of the network. An

ANN with a linear activation function would be limited to learning linear mappings of the inputs and outputs. On the other hand, using a non-linear activation function that does not fit to the model data would degrade the performance of the neural network. Therefore, it is vital to select the appropriate activation function that fits the network.

The choice of activation functions in the output layer usually depends on the type of task and prediction the network has to perform. The tasks are categorized between classification or regression. For the former, it is common to use a sigmoid (binary classification) or softmax (multi-class classification) activation, while for the latter, a linear activation function is deployed.

For the hidden layers, there is a wider choice, with each function having specific properties which suit different applications. For instance, a neuron with values between zero and one will be better suited with a sigmoid or a positive part of the Rectified Linear Unit activation function (ReLU). In other instances, it might be useful to adapt a function that smoothes (changes value) slowly towards $-\infty$, for which the Exponential Linear Unit (ELU) would be applied. For networks where multiple activation functions can be selected, it is recommended to find the optimum function through hyperparameter optimization, covered in section 2.4.

In this research, the ReLU activation function is predominantly applied to the hidden layers. In cases where a customized activation function was required, the bounds of the sigmoid function were modified and applied to the applicable layers.



(a) ReLU **(b)** Modified Sigmoid

Fig. 2.3: Activation functions applied to neurons in the hidden layers.

2.3 Optimizers

Optimization algorithms are one of the important tools in deep learning. They allow continuous update of model parameters (weights) to minimize the loss function evaluated on the training set. The performance of the network is heavily influenced by the choice of the optimization algorithm. The most efficient algorithm can be chosen through the hyperparameterization. The gradient descent method is one of the most common techniques used in machine learning due to its stable and fast convergence. It is a first-order iterative algorithm that takes the partial derivative of the function to determine the gradient, and moves in the opposite direction of the gradient. This is iteratively repeated until the zero gradient is reached, the minimum cost point. Mathematically, with weights w evaluated at point k, the iterative algorithm is represented as:

$$w_{k+1} = w_k - \alpha J'(w_k)$$

where J is the cost function to be minimized.

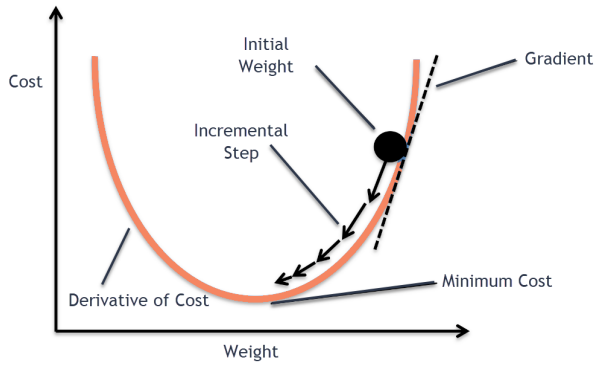


Fig. 2.4: Gradient descent algorithm (Chauhan, 2021).

Incremental steps taken towards the descent are dependent on the pre-defined value, α, also known as the learning rate. The learning rate is used to dictate the pace at which the model learns. Generally, the larger the value of α, the faster the training process. However, special care has to be taken in selecting this value, a large learning rate would result in large weights leading to oscillations of the loss function over the training epochs[1]. The oscillating performance can prevent the gradient descent algorithm from converging to the optimal solution. Moreover, if the learning rate is too small, the algorithm may not converge or may stop at a

[1] Training epoch: A term used in machine learning to indicate the number of passes of the entire training data set the machine learning algorithm has completed

sub-optimal solution (Senior *et al.*, 2013). Thus, the learning rate is the most crucial hyper parameter in machine learning.

Several extensions have been made to the gradient descent algorithm to facilitate faster convergence and avoid reaching a local minima. Stochastic gradient descent is one of the variant which updates the model parameters more frequently. The mini-batch gradient descent, the best among the gradient descent variations, updates the model parameters after every batch. More recent methods, such as the Adam optimizer, uses the momentum approach which accelerates the convergence where the gradient is consistent and reduces the fluctuations in irrelevant directions (Mavuso *et al.*, 2017). According to the recent studies, the Adam algorithm is generally the most efficient and effective optimizer in machine learning. Therefore, this study will make use of this optimizer in the deep network.

2.4 Hyperparameter optimization

In neural network calibration, two sets of parameters are involved, the calibration parameters related to the financial model, and the hyperparameters of the neural network (Hernandez, 2016). The latter set of parameters controls the learning process, which are selected before the training of the network begins. They typically determine the architecture of the neural network, which is composed of the number of layers, number of neurons per layer and the activation functions. In addition to that, the learning rate is also a very important hyperparameter to be considered despite not impacting the architecture directly. For an efficient neural network, an optimal set of hyperparameters needs to be found for a particular calibration problem (Hernandez, 2016).

The process of inferring the optimal set of hyperparameters is known as hyperparameter tuning or optimization. Searching for the optimum set of hyperparameters manually can be tedious, hence search algorithms like the random search and grid search are often used. Both these methods use a multidimensional grid of parameters to obtain the hyperparameter combinations with the best test results. The grid search set up relies on intuition, whereby the grid is setup with a specified minimum and maximum value for each parameter. On the other hand, the random search arbitrarily values a random sample of hyperparameters from the specified bounds. These methods require a dense search space to obtain the best performing parameters. This can be time consuming and computationally expensive. An alternative method to these is the Bayesian Optimization. The application of Bayesian methods in global optimization was initiated in a paper by Betrò and Schoen (1991). Its application was extended by Snoek *et al.* (2012) to deep learning, where it was

found to be a versatile method to optimize back-box derivative functions. A more recent study from Wu *et al.* (2019) compared different hyperparameter optimization techniques and concluded that Bayesian optimization was the most efficient and reliable technique to improve the efficiency of the network, whereby, the optimum set of hyperparameters can be found in lesser time and fewer iteration using Bayesian search compared to the grid and random search.

Bayesian optimization is a sequential model based optimization technique that uses results from the previous iteration to obtain the next set of hyperparameters. Unlike the grid and random search, the hyperparameters are selected strategically based on a statistical modelling. The use of the Bayesian search is more suitable to an expensive black box function $f(x)$, with no closed form solutions and possibility of noisy observations. These properties directly apply to the option pricing models considered. The objective of the Bayesian optimization model is therefore to minimize the function f bounded on a domain $X \subseteq \mathcal{X}$, represented by

$$x^* = \underset{x \in X}{\operatorname{argmin}} f(x).$$

The application of the Bayesian search relies on Gaussian processes to model the objective function. This model is famously known as the surrogate model (posterior distribution), it is generated by approximating the objective function using a bounded number of sample points and a probabilistic model (Gaussian process). The probabilistic model creates a joint probability distribution for the parameters of interest. The function is fitted with a range of hyperparameters x, on the objective function $f(x)$ is shown in Figure 2.5 below. To reiterate, the true function is evaluated at a few points (in black) with a confidence interval (in light blue). The resulting best estimate function is then represented by the dark blue line. This function is set as a prior and trains the model using the hyperparameters x.

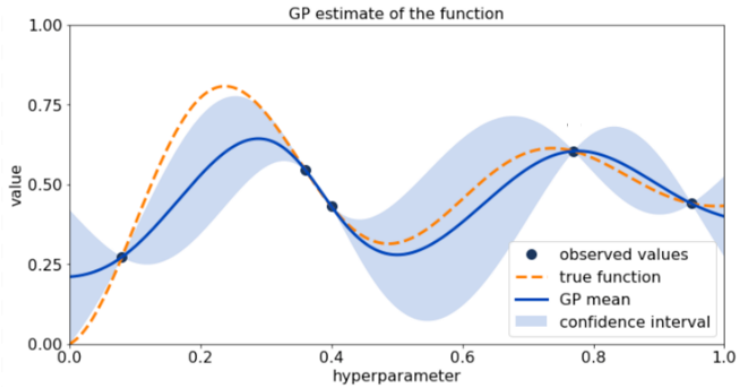


Fig. 2.5: Bayesian optimization framework (Ravikumar *et al.*, 2020).

After approximating the surrogate function, an acquisition function, also known as a selection function, is used to select the next hyperparameters of choice by using information from the posterior distribution. The mean and variance over the domain of $f(x)$ are used to calculate a value to determine how promising each location is, if it were to be evaluated next. The evaluation of the function at each of these points is computationally inexpensive, which makes the Bayesian optimization process an efficient method. A wide range of acquisition functions are available with the most common ones being the, upper bound confidence, probability of improvement, and expected improvement. More details regarding these functions can be found in Snoek *et al.* (2012). Minimizing one of these acquisition function yields to the next best point, which is used to update the surrogate model. This process of obtaining the minimum point of the acquisition function and updating the surrogate model is repeated iteratively until the points are sufficiently close to the true function.

For this study, the Keras tuner library is used to model the Bayesian optimization (O'Malley and Bursztein, 2019). The built-in Keras tuner takes in an objective function with minimum and maximum bounds of each hyperparameter to be optimized and returns the validation loss of the network. The acquisition function of upper confidence is set and the function is called n-many times for the optimization. This means, the function will iterate over n-many distinct hyperparameters sets to find the best set of parameters.

2.5 Handling the sample data

Neural networks are used to solve for the mapping function between the input and output variables. A pair of input and output forms an observation pair that will be used in the construction of the neural network. The number of observation pairs required for the construction is dependent on the complexity of the function trying to be estimated. In machine learning, it is common to split the data sets into three subsets, namely; training, validation and test data sets. The training data is the initial data set used to fit the model, it offers supervised examples to tune the network weights. The validation data set is used to provide unbiased evaluation of a model fit on the training set while tuning the model hyperparameters, it aids in preventing over-fitting of the model. Finally, the test data set is used to provide an unbiased evaluation of the final model fit on the training data set. Typically, the training and testing data sets are disjoint sets, this ensures the test data sets consists of observations which are unseen by the model during the training process, this is commonly referred to as out-of-sample testing. On the other hand, the training and

validation data sets are used together while training the model . In general, most of the sample data is used in the training set (i.e. 80%), with almost an equivalent split between the validation and testing set (i.e. 10% - 10%).

Data processing is the next vital step in machine learning, the quality of the data and the useful information it contains directly impacts the learning ability of the model. Therefore it is essential to pre-process the data before feeding it to the model. A series of pre-processing steps need to be followed, such as cleaning, integrating, reducing, and transforming the data set. In this research, we generate synthetic data sets, thus the main concern is to transform the data. Several methods of transforming the data are available, in the case of regression models, it is common to normalize or scale the data between a specified range. The normalization of the data is achieved using the Z-score Patro and Sahu (2015) given by,

$$Z = \frac{x - \mu}{\sigma}$$

where x is the observed value, μ is the mean, and σ is the standard deviation of the training samples.

The other possible technique is to scale the data in a pre-specified range. The transformation of scaling the sample data to a range of $[X_a, X_b]$ is simply

$$X_{scaled} = \frac{X - X_{min}}{X_{max} - X_{min}}(X_b - X_a) + X_a.$$

Both of these methods will be explored in this paper. The method with the lowest validation loss will be selected to scale the data.

Chapter 3

Interest Rate Modelling

In this chapter, the pricing of interest rate derivatives (caplets and swaptions) under the respective models are reviewed. The interest rates used to price these instruments are modelled in a pre-2007 crisis, single curve framework. The instruments are first priced using the well-known industry standard Black model (Cohen *et al.*, 2021). Closed form solutions for caplets are then stated under the one-and-two factor Hull-White models, while the analytical expression for swaptions is provided under the two-factor model only. Consequently, these closed form solutions are used in conjunction with the Black model to estimate the implied volatilities required for calibration. A numerical analysis is then performed to observe the effect of individual parameters on implied volatility surface. This study will assist in selection of parameter bounds for simulating prices that replicate the market observables.

3.1 Interest rate options

Interest rate options are over-the-counter traded derivatives whose value is derived from an underlying interest rate. They are typically used by corporations or financial institutions to manage interest rate risk by providing the holder protection against rising/falling floating interest rate movements. The most common type of interest options are caps, floors and swaptions, where each of these provides unique protections to interest rate moves. Our interest for this research is in the pricing of interest rate caplets and swaptions.

Interest rate caplets are a type of call option based on an interest rate, where the buyer receives a payment at maturity when the forward interest rate exceeds the strike rate. At the time of the purchase, the two parties agree on an underlying interest rate, typically the LIBOR rate, the start and expiry of the option, and the strike rate. The purchaser of the caplet will be entitled to receive a payoff at time

T_n given by:

$$[L(T_{n-1}, T_n) - R]^+ \times \delta_n \tag{3.1}$$

where δ_n is the year fraction from T_{n-1} to T_n, R is the strike rate, and $L(T_{n-1}, T_n)$ is the spot LIBOR rate for the period $[T_{n-1}, T_n]$. The term inside the parentheses, shortened as $[x]^+$, in the above equation is a maximum of 0 and x.

An interest rate swaption is an option that provides the holder the right but not obligation to buy an interest rate swap agreement at a pre-determined swap rate on a specified future date. Similar to interest rate swaps, swaptions are categorized into payer swaptions and receiver swaptions, where in a payer swaption, the buyer receives floating interest rates and pays a fixed rate. At expiry of the option, the purchaser would only exercise the option if the contract swap rate of the swaption is higher than the prevailing swap rate. The contrary applies to the receiver swaption. At the time of purchase, the two parties agree on underlying interest rate, start and expiry of the options, the number of payment dates, the tenor between the payment dates and the strike rate. The purchase of the swaption will be entitled to receive a payoff at the first reset date T_n (time point at which the floating interest rate is observed) given by

$$\left(\sum_{i=n+1}^{\beta} P(T_n, T_i)\delta_i(L(T_n; T_{i-1}, T_i) - R) \right)^+ \tag{3.2}$$

where T_β is the last payment date, and $P(t, T)$ is the zero coupon bond price at time t maturing at T.

To price contingent claims as such, it is important to first assume a unique equivalent martingale measure (EMM). This dictates the assumption that the market is complete and free of arbitrage opportunities. This assumption has been widely used in the field of financial theory and forms a fundamental part of the Black-Scholes formula. Since the Black-Scholes formula incorporates pricing of vanilla call and puts only, Black (1976) devised a model to extend this framework with a minor variation to accommodate the pricing of future, bond and interest rate options.

3.2 Black's model

Black's model has been an industry standard model in pricing options on interest rate futures, bonds, caps, floors and swaptions (Cohen *et al.*, 2021). It is categorized into a class of log-normal forward models, also referred to as the LIBOR market model. As the name of the class suggests, Black (1976) assumed that price of the future underlying observed at T_{n-1} is log-normally distributed in the risk-neural

world, and the expected changes of the future price is zero. The formulation developed under the Black model uses the the definition of zero-coupon bond $P(t,T)$ to aid in the pricing of bond options. The time-t price of a zero-coupon bond, maturing at T under the EMM is formally defined as

$$P(t,T) = \mathbb{E}_t^{\mathbb{Q}}\left[\exp\left\{-\int_t^T r_s ds\right\}\middle|\mathcal{F}_t\right] \tag{3.3}$$

where $\mathcal{F}_t$ is the filtration containing information up to time t.

3.2.1 Caplet pricing

Consider the payoff of the caplet from Equation 3.1 under the risk-neural measure $\mathbb{Q}$. The time T_0 value of the caplet C_0 can be expressed as

$$\mathbb{E}_{\mathbb{Q}}\left[e^{-\int_0^{T_n} r_t dt}\left(L(T_{n-1},T_n)-R\right)^+\delta_n\right] \approx P(T_0,T_n)\mathbb{E}_{\mathbb{Q}}\left[\left(L(T_{n-1},T_n)-R\right)^+\right]\delta_n$$

where by the approximation in the above formulation comes as a result of discounting outside the risk neutral expectation.

We define T_x-forward measures associated with numéraire $P(t,T_x)$ and risk-adjusted measure $\mathbb{Q}^{T_x}$, where $x \in (T_{n-1},T_n)$. Using the assumption that prices are log-normally distributed and the forward rate process $L(T_{n-1};T_{n-1},T_n)$ is a $\mathbb{Q}^{T_{n-1}}$-martingale, we can show that

$$C_0 \approx P(T_0,T_n)\left(\mathbb{E}_{\mathbb{Q}^{T_{n-1}}}\left[\left(L(T_{n-1};T_{n-1},T_n)\right]\Phi(d_+)-R\Phi(d_-)\right)\right)\delta_n$$

$$C_0 \approx P(T_0,T_n)\Big(L(T_0;T_{n-1},T_n)\Phi(d_+)-R\Phi(d_-)\Big)\delta_n$$

where

$$d_\pm = \frac{\ln\frac{L(T_0;T_{n-1},T_n)}{R} \pm \frac{1}{2}\sigma^2 T}{\sigma\sqrt{T}}$$

given σ is the volatility of the forward rate $L(T_{n-1},T_n)$ referred to as Black implied volatility (discussed in section 3.3) and $\phi(\cdot)$ is the cumulative normal distribution function.

The price of the caplet shown above is an approximation, however, for it to be exact, it is necessary to assume $L(T_{n-1},T_n)$ is log-normal under the T_n-forward measure $\mathbb{Q}^{T_n}$. To justify the assumptions theoretically is difficult, although, one can use the LIBOR market model framework to prove these assumption do hold, and thus showing Black's model is arbitrage free (Björk, 1998). Using this assumption

together with the change of numéraire invariance, the exact price is given by

$$C_0 = P(T_0, T_n)\mathbb{E}_{\mathbb{Q}^{T_n}}\left[\frac{(L(T_{n-1}; T_{n-1}, T_n - R)^+ \delta_n}{P(T_n, T_n)}\right]$$
$$C_0 = P(T_0, T_n)\Big(L(T_0; T_{n-1}, T_n)\Phi(d_+) - R\Phi(d_-)\Big)\delta_n.$$

More complex models, such as the Hull-White and G2++ models, rely on pricing zero-coupon bonds to retrieve the analytical solution for the caplets. The caplet pricing formula devised above can be shown to be equivalent to decomposing the caplet in a portfolio of put options on a zero coupon bond. This can easily be shown by considering the payoff of the caplet at T_{n-1}, where the forward LIBOR rate is observed. The equivalent payoff at T_{n-1} is:

$$\frac{[L(T_{n-1}, T_n) - R]^+ \times \delta_n}{1 + \delta_n L(T_{n-1}, T_n)} = (1 + \delta_n R)\left[\frac{1}{1 + \delta_n R} - P(T_{n-1}, T_n)\right]^+.$$

This payoff can be seen as a portfolio of $(1 + \delta_n R)$-many put options on $P(t, T_n)$, with strike $\frac{1}{1+\delta_n R}$ expiring at T_{n-1}. Since the payoff of the caplet can be represented using this portfolio at time T_{n-1}, by the law of one price, the value of the caplet would have the same value as a portfolio of put options at an earlier time (Björk, 1998). Therefore, the C_0 value of the caplet would be equivalent to the portfolio of put options at T_0.

3.2.2 Swaption pricing

Now, we consider a forward payer swaption initiated at time t, starting at time $T \geq t$, with tenor structure $T = T_0, T_1, \ldots T_\beta$. In valuation of the swaption, it be can shown that the initial value of the forward swaption is equal to zero if the T-forward swap rate $S_{t,T}$ is given by:

$$S_{t,T} = \frac{P(t, T) - P(t, T_\beta)}{\sum_{n=1}^{\beta} \delta_n P(t, T_n)}. \tag{3.4}$$

At time T, the holder of the swaption would exercise if and only if the prevailing swap rate $S_{T,T} \geq R$ the agreed swap rate, at time t (Brigo and Mercurio, 2006). The exercise of this option, would give a series of payments at times T_n as

$$\delta_n(S_{T,T} - R)^+.$$

Each of these payments is equivalent to δ_n many call options with strike rate R, and underlying $S_{T,T}$. Using the general version of Black's model (applied above), assuming $S_{T,T}$ is log-normal under the risk neutral measure, we obtain the value

of each of these payments at time $t = 0$. Summing each of these payments gives the swaption value C_0 as

$$C_0 \approx \sum_{n=1}^{\beta} \delta_n P(0, T_n)\big[S_{0,T} N(d_+) - RN(d_-)\big] \tag{3.5}$$

where

$$d_\pm = \frac{\ln \frac{S_{0,T}}{R} \pm \frac{1}{2}\sigma^2 T}{\sigma\sqrt{T}}$$

and σ is the volatility of the future spot swap rate $S_{T,T}$ referred to as Black implied volatility (discussed in section 3.3).

As for the case of the caplet pricing, the above price is an approximation. It can be proven that, in order for the price to be exact, we assume the swap rates are log-normally distributed under the EMM $(\mathbb{Q}_X)$ associated with an annuity process $X_t = \sum_{n=1}^{N} \delta_n P(t, T_n)$ (Björk, 1998). Using this as the numéraire, the exact price is obtained by

$$C_0 = \sum_{n-1} \delta_n P(0, T_n) \mathbb{E}_{\mathbb{Q}_X}[(S_{T,T} - R)^+]$$

where

$$\mathbb{E}_{\mathbb{Q}_X}[S_{T,T}] = S_{0,T}.$$

Therefore, it can be concluded that

$$C_0 = \sum_{n=1}^{\beta} \delta_n P(0, T_n)\big[S_{0,T} N(d_+) - RN(d_-)\big].$$

3.3 Stripping implied volatility

It is common practice to infer the volatility from quoted option prices based on the Black-Scholes model, this is regarded as the implied volatility. It is now a market standard to quote over-the-counter traded instruments in terms of their implied volatilities, where the volatility surface of interest rate options can directly be observed from the market. Practitioners find quoting implied volatilities beneficial over the option prices due to the ease in calibration of vanilla options. Calibrating to an implied volatility surface helps to specify the prices regardless of the type of option (e.g. caplet or floorlet) under consideration (Liu *et al.*, 2019). The computation of implied volatilities relies on a bootstrapping procedure to option prices. The value of the σ_{imp} is stripped by equating the option price under a particular

model given a set of parameters θ with the Black-76 model. Since option pricing models are not invertible, root-finding algorithms[1] are often employed to invert the model. Root finding algorithms such as Brent's method, Bisection method or the Nelder–Mead can be applied to the stripping process with the objective function given by

$$f^{Model}(\theta) - f^{Black}(\sigma_{imp}) = 0.$$

The implied volatility recovered from this model will be used in the calibration process, as detailed in section 4.1.3.

3.4 One-factor Hull-White model

The one-factor Hull-White model is a short rate model belonging to the class of no-arbitrage models with the ability to exactly fit the current term structure. It is predominately used to price interest-rate derivatives, such as caps and floors. Hull and White (1990) assumed a normally distributed short rate r_t, whose risk neutral dynamics are given by:

$$dr_t = (\theta(t) - \alpha r_t)dt + \sigma dW_t$$

where $\theta(t)$ is the mean reversion level, α is the mean reversion rate, σ is the annual standard deviation of the short rate (volatility), and W_t is the standard Brownian motion.

The drift term incorporates mean reversion, whereas, the interest rate converges to a mean reversion level of $\theta(t)$ at a speed of α. The parameter $\theta(t)$ is regarded as a deterministic function of time which determines the average direction r_t moves. When the short rate drifts above the reversion level, it is pulled back down and similarly when the short rate drifts below the reversion level is pushed back up Burgess (2014). On the other hand, α and σ are constant model parameters, therefore, when calibrating the model, these parameters will be estimated using the neural networks.

To reiterate, pricing of caplets using the Hull-White model is dependent on pricing a portfolio of European put options on zero coupon bonds. Using the results from Björk (1998), the closed form solution for the European put option on zero coupon bonds is given by

$$\textbf{ZBP}(t, T_{n-1}, T_n, X) = XP(t, T_{n-1})\Phi(-d_-) - P(t, T_n)\Phi(-d_+)$$

[1] See Amat *et al.* (2004) for information about root-finding algorithms.

where

$$d_{\pm} = \frac{\ln \frac{P(t,T_n)}{XP(t,T_{n-1})} \pm \frac{1}{2}\sigma_{av}^2 T_{n-1}}{\sigma_{av}\sqrt{T_{n-1}}}$$

and

$$\sigma_{av}^2 T_{n-1} = \frac{\sigma^2}{2\alpha^3}\left(1 - e^{-\alpha(T_n - T_{n-1})}\right)^2 \left(1 - e^{-2\alpha T_{n-1}}\right).$$

Proof: *See Björk (1998) page 335 to 337.*

To apply this formulation on caplet pricing, we recall the pay-off of the caplet with a reset date T_{n-1}, payment date T_n, and caplet rate R, is equivalent to the pay-off of $(1 + R\delta_n)$-many European put options with strike $\frac{1}{1+R\delta_n}$ with maturity T_{n-1} written on a zero coupon bond $P(t, T_n)$. The caplet price can therefore be written as

$$\textbf{Cplt}(t, R; T_{n-1}, T_n) = (1 + \delta_n R)\textbf{ZBP}(t, T_{n-1}, T_n, \frac{1}{1 + \delta_n R}). \tag{3.6}$$

A numerical study is performed to analyze the effect of the parameters on caplet implied volatility. A reference set of parameters is selected based on the study done by Falcó *et al.* (2009). The individual parameters are then varied whilst keeping the others fixed. The closed form solution above is used to determine caplet prices, and in turn, the Black formula is used to infer the implied volatilities. The reference set of parameters used for this study are

$$\alpha = 0.15 \qquad \sigma = 0.06.$$

The volatility surface is generated for an ATM caplet with time-to-maturity (TTM) varying from 0.25 years to 5 years as shown in Figure 3.1 below.

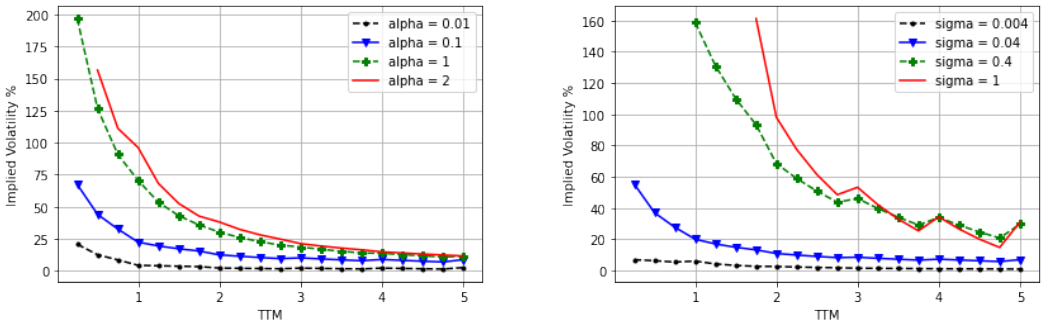


Fig. 3.1: Implied volatility sensitivity to one-factor model parameters.

As it can be seen from the plots above, both parameters have a similar effect, where an increase in the model parameters leads to an increase in the implied volatility. It can also be observed, when the model parameters are lower than the

reference parameters set, we obtain very small changes in the implied volatility. On the other hand, when the model parameters are higher, the implied volatility substantially increases (more than 100%). In addition to that, it can be observed that the model tends towards instability for caplets with short tenors for the set of parameters higher than the reference set. This is noted by observing the green/red line in the plot above does not start from the 0.25 years vertex, as the Black model failed to retrieve an implied volatility value for the respective time point.

The focus of this study is to simulate realistic market observables whilst ensuring model stability. From this numerical investigation, it can be seen that model parameters greater than the reference set are undesirable.

3.5 Two-additive-factor Gaussian G2++ model

The G2++ model is a two-factor additive Gaussian short rate model which also belongs to the class of no-arbitrage models with the ability to fit the initial term structure of interest rates. It was introduced as an equivalent model to the Hull-White two-factor model, with less complex formulation and easier implementation achieved by the addition of additive factors. The model dynamics are driven by two stochastic factors that follow the mean reversion Ornstein-Uhlenbeck process, where the stochastic components are correlated. The additional stochastic factor provides greater flexibility to capture the actual variation in market rates by incorporating a correlation term ρ, between interest rates of different maturities. Furthermore, the short-rate dynamics are assumed to be Gaussian (similar to Hull-White), this leads to good analytical tractability for which explicit formulas for discount bonds and instruments such as European swaption, caps and floors can be derived. Under the risk-neural dynamics $\mathbb{Q}$, the short-rate r_t process is defined by

$$r(t) = x(t) + y(t) + \psi(t), \;\; r(0) = r_0,$$

where $\psi(t)$ is a deterministic function, $x(t)$ and $y(t)$ are stochastic processes driven by:

$$\begin{aligned} dx(t) &= -ax(t)dt + \sigma dW_1(t), \;\; x(0) = 0, \\ dy(y) &= -by(t)dt + \eta dW_2(t), \;\; y(0) = 0 \end{aligned}$$

with r_0, a, b, σ, η as positive constants, and $(W_1, W_2)_t$ a two-dimensional Brownian motion with instantaneous correlation structure, $\rho \in [-1, 1]$, such that:

$$dW_1(t)dW_2(t) = \rho dt,$$

The pair of stochastic differential equations above can also be represented in-terms of independent standard Brownian motions $\tilde{W}_1(t)$ and $\tilde{W}_2(t)$, by using the Cholesky decomposition of the correlation matrix of $(W_1, W_2)_t$,

$$dx(t) = -ax(t)dt + \sigma d\tilde{W}_1(t), \;\; x(0) = 0,$$
$$dy(t) = -by(t)dt + \rho\eta d\tilde{W}_1(t) + \eta\sqrt{1-\rho^2}d\tilde{W}_2(t), \;\; y(0) = 0$$

where

$$dW_1(t) = d\tilde{W}_1(t), \;\; dW_2(t) = \rho d\tilde{W}_1(t) + \sqrt{1-\rho^2}\, d\tilde{W}_2(t).$$

As before, we use the fact that the price of a caplet is equivalent to a portfolio on zero coupon bonds. We state the closed form solution without proof, for the time-t price of a European put option, with strike X and expiry T_{n-1}, on a zero-coupon bond of maturity T_n as

$$\begin{aligned}\textbf{ZBP}(t, T_{n-1}, T_n, X) =& XP(t,T_{n-1})\Phi\left(\ln \frac{\frac{XP(t,T_{n-1})}{P(t,T_n)}}{\Sigma(t,T_{n-1},T_n)} + \frac{1}{2}\Sigma(t,T_{n-1},T_n)\right) \\ & -P(t,T_{n-1})\left(\ln \frac{\frac{XP(t,T_{n-1})}{P(t,T_n)}}{\Sigma(t,T_{n-1},T_n)} - \frac{1}{2}\Sigma(t,T_{n-1},T_n)\right)\end{aligned}$$

where

$$\begin{aligned}\Sigma(t,T_{n-1},T_n)^2 =& \frac{\sigma^2}{2a^3}\left[1-e^{-a(T_n-T_{n-1})}\right]^2\left[1-e^{-2a(T_{n-1}-t)}\right] \\ & + \frac{\eta^2}{2b^3}\left[1-e^{-b(T_n-T_{n-1})}\right]^2\left[1-e^{-2b(T_{n-1}-t)}\right] \\ & + 2\rho\frac{\sigma\eta}{ab(a+b)}\left[1-e^{-a(T_n-T_{n-1})}\right]\left[1-e^{-b(T_n-T_{n-1})}\right]\left[1-e^{-(a+b)(T_{n-1}-t)}\right].\end{aligned}$$

Proof: *See Brigo and Mercurio (2006) page 153 to 156.*

Therefore, caplet prices under this framework can be obtained using Equation 3.6.

Now, consider a European swaption with strike X, maturity T, with payment times $\mathcal{T} = \{t_1, \ldots, t_n\}$, $t_1 > T$. Each of the fixed payments are denoted by $c_i := X\tau_i$ for $i = 1, \ldots, n-1$ and $c_n := 1 + X\tau_n$, where τ_i is the year fraction between two payment dates. The pricing of this swaption under the two-factor model requires additional tedious derivations. Omitting the proofs the final "analytical" solution proposed by Brigo and Mercurio (2006) for the time $t = 0$ price of the European swaption is given by

$$\begin{aligned}&ES(0,T,\mathcal{T},X,\omega) = \\ &\quad \omega P(0,T)\int_{-\infty}^{\infty}\frac{e^{-\frac{1}{2}(\frac{x-\mu_x}{\sigma_x})^2}}{\sigma_x\sqrt{2\pi}}\left[\Phi\big(-\omega h_1(x)\big) - \sum_{i=1}^{n}\lambda_i(x)e^{\kappa_i(x)}\Phi\big(-\omega h_2(x)\big)\right]dx\end{aligned}$$

where $\omega = 1(-1)$ for a payer (receiver) swaption, and

$$\begin{aligned}
h_1(x) &:= \frac{\bar{y}-\mu_y}{\sigma_y\sqrt{1-\rho_{xy}^2}} - \frac{\rho_{xy}(x-\mu_x)}{\sigma_x\sqrt{1-\rho_{xy}^2}} \\
h_2(x) &:= h_1(x) + B(b,T,t_i)\sigma_y\sqrt{1-\rho_{xy}^2} \\
\lambda_i(x) &:= c_i A(T,t_i)e^{-B(a,T,t_i)x} \\
\kappa_i(x) &:= -B(b,T,t_i)\left[\mu_y - \frac{1}{2}(1-\rho_{xy}^2)\sigma_y^2 B(b,T,t_i) + \rho_{xy}\sigma_y\frac{x-\mu_x}{\sigma_x}\right]
\end{aligned}$$

$y = \bar{y}(x)$ is the unique solution of the following equation

$$\sum_{i=1}^{n} c_i A(T,t_i)e^{-B(a,T,t_i)x - B(b,T,t_i)\bar{y}} = 1,$$

and

$$\begin{aligned}
\mu_x &:= -M_x^T(0,T), \\
\mu_y &:= -M_y^T(0,T), \\
\sigma_x &:= \sigma\sqrt{\frac{1-e^{-2aT}}{2a}}, \\
\sigma_y &:= \eta\sqrt{\frac{1-e^{-2bT}}{2b}}, \\
\rho_{xy} &:= \frac{\rho\sigma\eta}{(a+b)\sigma_x,\sigma_y}\left[1-e^{-(a+b)T}\right]. \\
A(t,T) &:= \frac{P^M(0,T)}{P^M(0,t)}\exp\left\{\frac{1}{2}[V(t,T)+V(0,t)-V(0,T)]\right\} \\
B(z,t,T) &:= \frac{1-e^{-z(T-t)}}{z}
\end{aligned}$$

given $P^M(0,T)$ is the market observed zero-coupon bond price, and

$$\begin{aligned}
V(t,T) =& \frac{\sigma^2}{a^2}\left[T-t+\frac{2}{a}e^{-a(T-t)} - \frac{1}{2a}e^{-2a(T-t)} - \frac{3}{2a}\right] + \\
& \frac{\eta^2}{b^2}\left[T-t+\frac{2}{b}e^{-b(T-t)} - \frac{1}{2b}e^{-2b(T-t)} - \frac{3}{2b}\right] + \\
& 2\rho\frac{\sigma\eta}{ab}\left[T-t+\frac{e^{-a(T-t)}-1}{a} + \frac{e^{-b(T-t)}-1}{b} - \frac{e^{-(a+b)(T-t)}}{a+b}\right].
\end{aligned}$$

Proof: *See Brigo and Mercurio (2006) page 158, and page 173 to 174.*

The above formula is not truly analytical as the integral does not have clear bounds. Therefore, it is necessary to truncate the region to obtain the swaption prices. To achieve this, the integration region is limited between

$$[\mu_x - 10\sigma_x, \mu_x + 10\sigma_x]$$

given values outside these bounds have negligible changes in swaption prices (Brigo and Mercurio, 2006). It would have been more suitable to run a `while` loop to change the boundary according to changes in swaption prices, however, this would slow down the calibration procedures for minor changes in swaption prices and implied volatility. The integral was then solved using the trapezoidal rule approximation which was implemented by the `trap` function in `matlab`.

Similar to the one-factor model, the parameter sensitivity analysis is performed on ATM implied caplet volatilities. The analysis on caplets was preferred since the investigation for swaption is tedious due to the generation of the volatility surface for a range of option expiry and tenors. It is more practical to observe the effect on different level of moneyness for swaption contracts. The main focus of the numerical study is to obtain realistic market observables whilst ensuring the model stability. The set of parameters obtained from this numerical study will be tested on ATM swaptions to ensure the criterion is satisfied.

The reference set of parameters used for the G2++ model are:

$$a = 0.0093 \qquad b = 0.013 \qquad \sigma = 0.0087 \qquad \eta = 0.014 \qquad \rho = -0.64.$$

The same steps as for the one-factor model are then followed using the reference parameters set above, the sensitivity plots are shown in Figure 3.2 below.

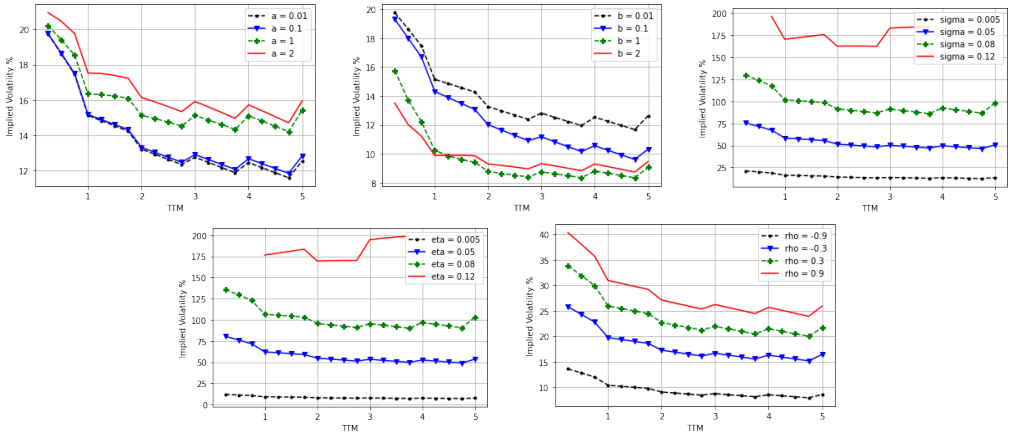


Fig. 3.2: Implied volatility sensitivity to two-factor model parameters.

From the plots above, it can be observed that parameters a and b have inverse effects, an increase in a leads to an increase in the implied volatility, whereas, the implied volatility decreases with an increase in b. It is noted, these two parameters do not greatly affect the implied volatility and model stability. When observing the effects of σ and η, both have similar impacts, where a rise in the parameters also

yields to a rise in the volatility. It is observed that the model tends toward instability when these parameters are greater than 0.1. Finally, it can be observed when ρ increases, the implied volatility surface also shifts up. As multiple parameters are involved in this model, it is not immediately obvious when the model tends towards instability. As a result, to obtain realistic simulated data, this numerical study, literature recommendation, and sufficient testing on caplets and swaptions, will be used to obtain defined parameter bounds.

Chapter 4

Neural Network Calibration Methodology

The aim of this chapter is to use the theoretical and mathematical tools developed in the previous chapters to construct and train neural networks. To reiterate, the option pricing models are calibrated to synthetically generated data. Financial instruments trading in the South African market are used to bootstrap the initial yield curve. In turn, statistical modelling methods are employed to generate a larger sample of yield curves from the actual market curves. The statistically simulated yield curves and uniformly drawn model parameters from pre-specified bounds are used to simulate option prices under the respective models. The obtained prices are then used in the two distinct calibration frameworks presented. Accordingly, the neural network architectures for each of these frameworks are discussed and methodology followed to retrieve the optimum set of hyperparameters. Furthermore, the respective methods and metrics used in testing the neural network are presented in this chapter.

4.1 Data analysis

The entire process from training to testing the neural network requires a large data set. The requirement is dependent on the function being estimated. The complexity of the function varies greatly for option pricing models. In the simplest case, using the Black-Scholes model, a data set of 10 000 observation pairs may suffice for accurate and efficient calibration, whereas this can be deemed to be insufficient when training more complex models, such as the multi-factor short rate models. Nevertheless, having a data set with even 10 000 observation pairs readily available from the market is implausible. Therefore it is necessary to simulate data to facilitate the neural network calibration process. In this study, we generate synthetic data of 100 000 observation pairs based on observable market information and relevant

literature publications. The simulated data is comprised of synthetic yield curves and model parameters which are in turn used to price the respective option pricing contracts. In particular, we will model caplet prices with option expiry up to 5 years with a tenor of 3 months, and swaption prices with expiry and tenor combinations of $\{1, 2, 3, 4, 5\} \times \{1, 2, 3, 4, 5\}$ (in years).

4.1.1 Yield curve generation

As we saw from Chapter 3, pricing of interest rate options is significantly dependent on the current term structure of interest rates. In particular, we require market observable zero-coupon prices that can also be deciphered from an observable zero curve. However, the data available from the market observable curves is not sufficient to train the neural network. Therefore, with the aid of historical market data and statistical algorithms, a larger data set will be produced from a much smaller set.

To generate the initial term structure, we will make use of the smoothed bootstrap method which was initiated by Fama and Bliss (1987). This involves bootstrapping discrete spot rates from the market data and fitting it with a smooth and continuous curve. This process requires a specific selection of financial instruments that is dominated by the liquidity of the particular instruments trading in the market. For this study, the data made available[1] consisted of fixed deposits, negotiable certificate of deposits (NCD), foward rate agreements (FRA) and interest rate swaps (IRS) for various maturities trading in the South African market on a horizon of March 2011 to October 2018. Typically the short-end of the curve is constructed using deposits, FRA's, and interest rates futures, while the long end is constructed using IRS, which are liquidly traded out to a maturity of 30 years. Based on the options analyzed, we require a yield curve with a tenor of up to 10 years with intervals of 3 months for the first 5 years for caplets, and an interval of 1 year for swaptions. Therefore, the following instruments are used for bootstrapping the yield curve.

IRS in the above table have a payment schedule of 3 months, where the missing tenors discount factors are linearly interpolated. The matrix method discussed by Glova (2010) was used to construct the yield curves. This method is widely used in the financial industry as it simplifies the numerically severe process into a matrix multiplication. The completion of the bootstrapping using this method generated 1980 yield curves (March 2011 to October 2018) with 40 maturity vertices.

[1] The data was obtained from a private source in the South African financial industry, on the condition that they are used for non-commercial purposes. South African rates are also available via Bloomberg, although availability was dependent on one's specific degree of access. Where available, Bloomberg sourced rates are almost perfectly consistent with the privately obtained rates.

Tab. 4.1: Bootstrapping Instruments.

Instruments	Tenor
Deposit	3m
Forward Rate Agreement	3mx6m
Forward Rate Agreement	6mx9m
Interest Rate Swaps	1y to 10y

Having constructed the yield curve, we now need to generate a larger data set from this sample. This is achieved by the use of the co-variance matrix structure, by sampling at each of the discrete time vertices. The following steps summarize the generation process

- Re-scale the sample data set to have a mean of zero and variance of one.
- Calculate the sample covariance (Σ) matrix of the scaled data set.
- Compute the Cholesky decomposition of the co-variance matrix $(\Sigma = LL^T)$.
- Generate random, standard normally distributed vectors (Z) consistent with the covariance matrix with vector length representing the target data size N.
- Generate random normally distributed vectors consistent with the covariance matrix $X = LZ$.
- Apply an inverse transformation to scale back to the original mean and variance.

The application of this algorithm generates N yield curve values which are statistically similar to the initial yield curves. The efficacy of this method was validated by observing a close correlation in the mean,variance and covariance of the generated data set with the actual data set. To observe this correlation, Figure 4.1 below shows five randomly selected yield curves from the actual data set and the generated data set.

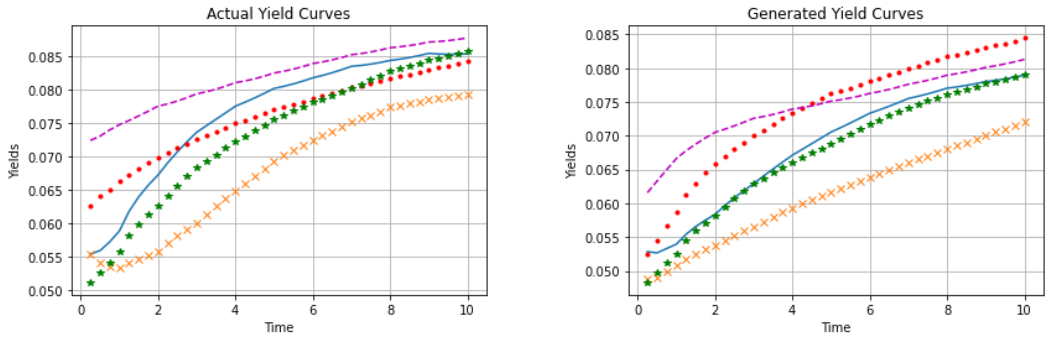


Fig. 4.1: Comparison of actual and synthetically generated yield curves.

4.1.2 Simulated model parameters

Model parameters that will eventually be learnt during the training process are generated from a uniform distribution with a specified bound. It is important to select reasonable bounds for each parameter to simulate realistic price estimations and ensure numerical stability of the model. From the numerical study performed in Chapter 3, it was shown the one-factor model tends towards instability as the parameter exceeded the reference parameter set. As a result, the parameter bounds for the one-factor model were selected in the region of the reference set as seen in Table 4.2. In the case of the G2++ model, it was not immediately obvious what bounds would lead to instability for the parameters (a, b, ρ). Therefore, to determine appropriate bounds, reference bounds from Alaya *et al.* (2021) for caplets were used, and further tested for swaptions. The parameter bounds for the G2++ model are shown in Table 4.2.

Tab. 4.2: Bounds for uniformly drawn parameters.

Model	Parameters	Bounds
One-Factor	α	[0.1,0.2]
	σ	[0.04,0.08]
Two-Factor	a	[0.001,0.015]
	b	[0.008,0.018]
	σ	[0.008,0.017]
	η	[0.008,0.017]
	ρ	[-0.870,-0.57]

4.1.3 Synthetic option prices

Closed form solutions presented in Chapter 3 alongside the simulated yield and model parameters are used to obtain option prices for caplets and swaptions under the respective models. Consequently, these prices are used to obtain implied volatilities from the Black model via a bisection root finding algorithm, as explained in section 3.3. We are interested in generating implied volatility surfaces for caplets and swaptions, which will be used in the calibration framework.

Caplet prices

The steps for generating the sample data for the ATM Caplets under the Hull-White models are as follows:

- Randomly select one of each of the model parameters by drawing from a uniform distribution within the corresponding intervals shown in Table 4.2.
- Create a grid of 20 different time points, starting from the 0.25 years mark with a tenor of three months, the associated time vector is given by

$$T = [0.25, 0.5, 0.75, \ldots, 4.5, 4.75, 5].^{\top}$$

- Generate a sample of the yield curve from the algorithm presented in section 4.1.1.
- Determine the ATM strike rate for the caplet by setting it equal to the forward rate $L(t;T,S)$ at each of these time points using

$$L(t;T,S) = \frac{1}{\delta_n}\left(\frac{P(t,T)}{P(t,S)} - 1\right)$$

  where δ_n is the tenor and $P(t,X)$ are the respective zero-coupon bond prices.
- Use the closed form solutions from Chapter 3 for the respective model to obtain the prices, and thus implied volatilities.
- Repeat the above steps to obtain N-many sample data sets.

Using the above steps, the implied volatility surface generated for caplets using one sample for one-and-two factor models are shown in Figure 4.2 respectively.

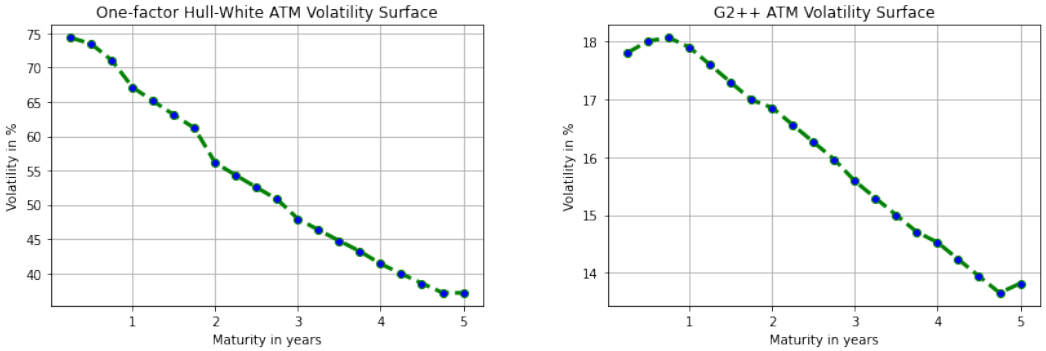


Fig. 4.2: Simulated ATM caplet volatility surface for one and two factor models.

It can be observed the implied volatility of the one-factor model is higher than the two-factor model. This is due to the two-factor model parameters bounds are chosen to ensure that both caplet and swaptions replicate market observables without losing model stability.

To price these caplets, the option pricing functions were created in `Python`. A total simulation time of 6 and 8 hours were needed to generate a sample of 100 000 volatilities in the one-and-two factor models respectively.

Swaption prices

The steps for generating the sample data for ATM swaptions under the two-factor Hull-White model is analogous to the ones described above. The ATM strike rate was determined by Equation 3.4 at $t = 0$. Additionally, for swaptions, we consider a grid of option tenor and expiry times. In particular, we select a grid of swaption prices with expiry and tenors ranging from one to five years. A combination of these two gives a swaption matrix of 25 swaption volatilities. Applying this together with the steps above, the implied volatility surface generated from one sample is shown in Figure 4.3 below.

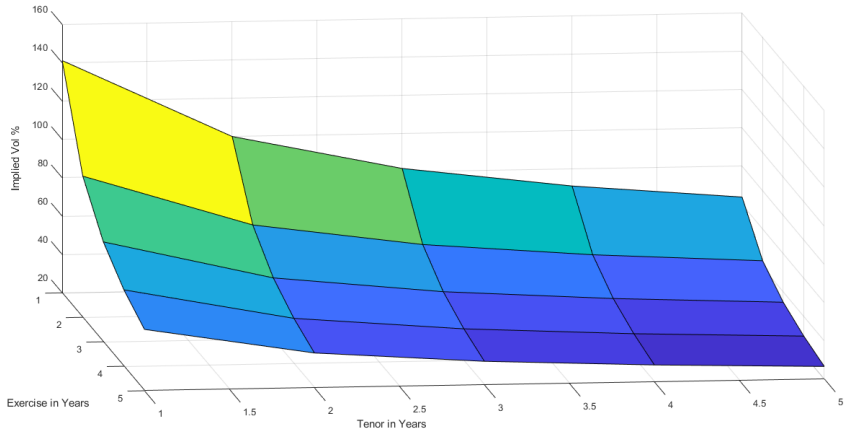


Fig. 4.3: Simulated ATM swaption volatility surface for the two factor model.

The sample data for the swaptions was generated in `matlab` due to the long simulation times in `python`. A total time of 28 hours was used to generate 100 000 sample points of swaption volatilities under the two-factor model.

4.2 Calibration procedure

Calibration frameworks are required to be robust, fast and accurate. We attempt to adopt the procedure to deliver these properties using neural networks. Two different neural network calibration techniques are considered, the direct calibration, and indirect calibration, which are discussed in section 4.2.1 and 4.2.2 respectively. Our main focus is to find suitable neural network architecture which holds a balance between computational time, complexity and accuracy.

4.2.1 Direct calibration

This method involves a direct inversion of the standard pricing neural network. The input of the neural network consists of market observables, which are the yield curves and implied volatilities. The size of the inputs is dependent on the options being considered. In the case of calibrating to caplets, we would have a vector with 20 caplet implied volatilities, and 21 distinct yield curve vertices, giving an input vector of length 41. Consequently, incorporating the addition of swaptions, the input vector would consist of an additional 25 swaption prices, and a total of 40 distinct yield curve vertices, giving an input vector of length 85. The output vectors would then consist of the parameters being calibrated, with the length of the vector dependent on the option pricing model.

This set up would define an input and output observation pair for one sample. This is then combined to have N samples, giving an input matrix of $[N, x]$ and output matrix of $[N, y]$, where x and y are the input and output vector lengths respectively. Each of the input column vectors are scaled to have a mean of zero and variance of one, this scaling improves the neural network model stability and performance. The output values are scaled to be in a comparable range (e.g ρ is scaled to $[0, 0.02]$ by adding 1 and dividing by 25). The resulting network is trained with the selected hyperparameters and an appropriate loss function. In this case, we use the parameter mean squared error (PMSE) cost function which is defined as

$$\text{PMSE}(\phi, \hat{\phi}) := \frac{1}{K} \sum_{i}^{K} (\phi_i - \hat{\phi}_i)^2 \tag{4.1}$$

where ϕ and $\hat{\phi}$ are the true and calibrated parameter values respectively, and K is the number of parameters being calibrated. In the case of the one-factor and two-factor models, we have (α, σ) and $(a, b, \sigma, \eta, \rho)$ respectively.

4.2.2 Indirect calibration

The indirect framework is split into two consecutive components, the pricing process and the calibration process. The pricing process entails estimation of the option pricing function using a trained neural network; this is defined as the forward pass. The latter phase uses the resulting neural network applied within a calibration procedure to fit model parameters from observed market prices, defined as the backward pass. Using this two-step approach, we attempt to leverage a pre-trained forward pass neural network, whose efficacy well validated when compared to the traditional option pricing methods (Ruf and Wang, 2019).

The aim of the forward pass is to learn the mapping between the model parameters and implied volatilities. In other words, it can be considered as a bypass to using the standard option pricing methods and Brent's root finding algorithm to obtain implied volatilities. The neural network created for this process would consist of yield curves and model parameters in the input layer, and market observable volatilities in the output layer. Analogous to the direct case, the size of the input and output layers would be governed by the type of contract and model being calibrated. Before initializing the training process, the input matrix is normalized to have a mean of zero and variance of one. The resulting neural network is then trained with the selected hyperparameters and a customized loss function. The customized loss function minimizes the relative mean-squared-error between the model and market implied volatilities. We define the volatility relative mean

squared error (VRMSE) by

$$\text{VRMSE}(V) = \frac{1}{M}\sum_{i=1}^{M}\left(\frac{\left(IV_i^{NN}(\theta) - IV_i^{mkt}\right)^2}{IV_i^{mkt}}\right) \tag{4.2}$$

where θ are the model parameters, and M is the number of implied volatilities (equivalent to length of output vector).

The choice of the modified cost function is to ensure the loss is not dominated by more expensive options. In addition to that, when considering the calibration of the combination of caplets and swaptions, it is more practical to define them in relative terms rather than individual contract prices.

While training the forward pass using an input-output observation pair, the hidden layers are optimized to obtain appropriate weights and biases. During this phase, only the hidden layers are the trainable units in the network, as can be seen in Figure 4.4 below.

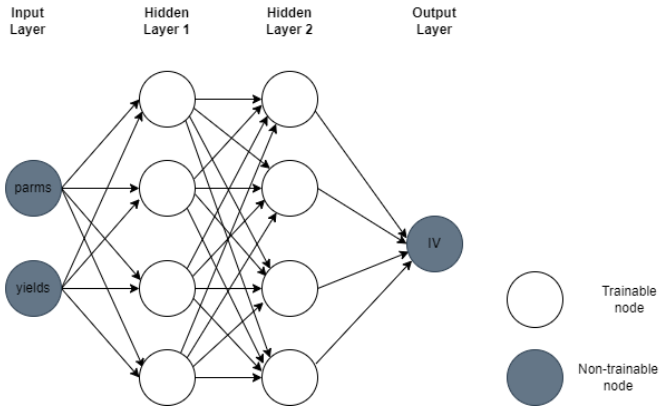


Fig. 4.4: Neural network architecture during the pricing phase.

The training process during the calibration phase relies on the pre-trained model in a backward manner. During the calibration phase, the weights in the hidden layers from the forward pass are frozen, and the input layer is transformed into a learnable layer. This is achieved by the addition of an input layer to the existing neural network structure from the forward pass, such that the initial input layer is transformed into the first hidden layer. Therefore, the first hidden layer in the calibrator would now consist of trainable model parameters and a non-trainable yield curve. In that case, the input layer to the backward pass network would consist of a vector of [1] and, the yield curve. Figure 4.5 below illustrates the calibrator neural network obtained after modifying the forward pass. Since this is a customized neural network, the `Keras Functional API` was used in its construction which allows for the addition of user defined architectures.

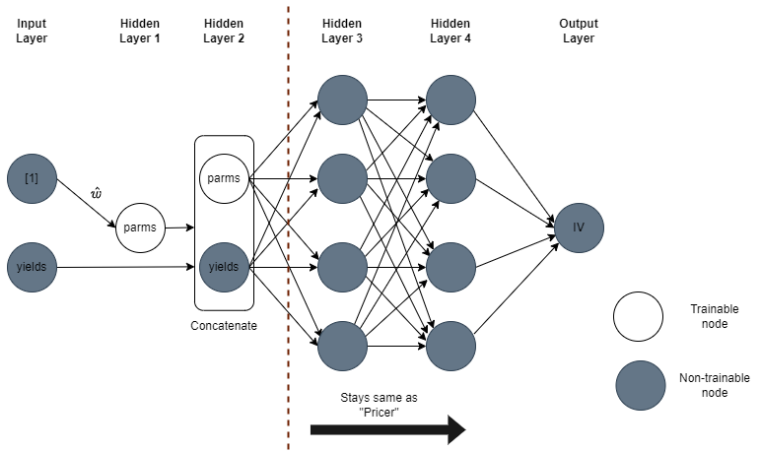


Fig. 4.5: Neural network architecture during the calibration phase.

To summarize this calibration framework, the forward pass network with x-many parameters is trained in an offline space to learn the pricing function. The calibration network is then obtained by adding a new layer with constant input of 1 to the forward pass, and making only the weights emanating from this new input layer trainable. As a result, the network has as many trainable weights as the number of parameters, yielding to an easy optimization. The training of this layer finds parameters which match the implied volatilities, i.e. training = online calibration.

To facilitate the training process, it is essential to create customized activation functions within the estimated range for the model parameters. Since these parameters are normalized, the bounds of the scaled parameters were $[-5, 5]$. Through trial and error, it was found that modifying the sigmoid activation function was best suited for this application. The neural network is then trained by defining the input as suggested above, and the output as a vector of the implied volatilities. It is important to note, the optimization of the weights $\hat{w}$ would only take place in the trainable nodes, whose biases are set to zero . The model parameters are then recovered by applying the activation function on each of the weights. For instance, the one-factor model parameters are recovered by

$$\alpha = \psi_1(w_1) \qquad \sigma = \psi_2(w_2)$$

where ψ_n represents the customized activation function for the n^{th} node.

4.2.3 Model testing

During the training process, each of the neural networks are monitored by suitable metrics, such as the PMSE, VRMSE defined by Equation 4.1 and 4.2 respectively. In

addition to these, we define the mean absolute error (MAE) to monitor the neural network training given by

$$\text{MAE}(y, \bar{y}) = \frac{1}{W} \sum_{i}^{W} |y_i - \bar{y}_i| \tag{4.3}$$

where $y, \bar{y}$ are the actual and model predicted quantities (parameters or implied volatilities) and W is the number of quantities. We abbreviate the mean absolute error between parameters (volatilities) by PMAE (VMAE).

Each of the trained neural networks is tested after the training process to evaluate the model performance and ensure the neural network does not over fit the training data. This done by the respective metrics introduced above.

In addition to that, we conduct a further out of sample testing using actual market yield curves from March 2011 to October 2018. For this time horizon, we make use of the end of month yield curves which amounts to ninety-nine data points. Option implied volatilities are then synthetically generated using each of these yield curves as described in section 4.1.3. These are used as inputs to the trained neural networks to obtain the calibrated model parameters. Consequently, the calibrated parameters are used in the option pricing framework to obtain the predicted implied volatilities. Each of these calibration frameworks is compared by time series plots, regression plots, and the mean relative implied volatility error (MRIVE) defined by

$$\text{MRIVE}(V) = \frac{1}{N} \sum_{i}^{N} \frac{IV_i^{mkt} - IV_i^{N}}{IV_i^{mkt}} \tag{4.4}$$

where IV_i^{mkt} is the market implied volatility for the i^{th} option, and IV_i^{N} is the implied volatility derived by repricing the i^{th} option using calibrated parameters.

4.3 Computational resources

The training of the neural network is performed using the `TensorFlow Keras` (Abadi *et al.*, 2016) library in `Python`. `TensorFlow` is an open source library developed by google that focuses on deep neural networks. The training and calibration were run using Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz with an installed RAM of 8GB. It is recommended to run the training process under a GPU, however this was not available in the computer used. Due to the limited computational power, this dissertation was geared towards simulations with low computational power requirements.

4.4 Model architecture optimization

The success of the neural network training process lies in the careful design of the architecture. In machine learning, the size of the input and output layer is predetermined according to the training data set. The number of hidden layers, nodes, activation functions, and many other features are dependent on the choice of the user. For this work, we create a standard architecture to be used in the calibration framework. The models will be optimized by a specific selection of hyperparameters according to the training data set. The standard architecture is given by:

- Splitting the data sets: As mentioned, it is essential to split the data into three distinct sets. The training and testing data is initially split using the `sklearn train_test_split` function (0.9 training and 0.1 testing). The training data is further split into training (0.9) and validation (0.1), when fitting the model.
- Learning rate scheduler: This is a pre-defined framework that reduces the learning rate as the training epochs progress. Presence of the scheduler improves the convergence of the model. This work makes use of the `ReduceLROnPlateau` scheduler available in the `keras` library.
- Optimizer: As discussed before, an optimizer is an algorithm used to minimize the loss function. We use the `Adam` optimizer, which incorporates the adaptive momentum estimation approach. Furthermore, the Adam optimizer is proven to be computationally efficient with low memory requirements.
- Early Stopping: It is a form of regularization technique to prevent over-fitting. This halts the training process when the selected metric has not improved for a certain number of iterations. We monitor the validation loss, and set the minimum improvement to 1e-4 for 50 epochs. This means, if the validation loss has not improved by 1e-04 in 50 epochs, the training process will be stopped.
- Loss functions: The loss functions are determined by the type of calibration framework used. In the case of the direct calibration, the loss is given by Equation 4.1 and for the indirect framework, the loss is given by Equation 4.2.
- Activation function: ReLU activation is used for the direct calibration. This function is the most computationally efficient, as it does not activate all the neurons at the same time. Neurons are deactivated when the output of the linear transformation is negative. A modified sigmoid activation function is

used for the first hidden layer in the indirect calibration with the output range modified to [-5,5] to accommodate the normalization of input parameters.

- Hidden layers and neurons: The size of the hidden layers and neurons are very sensitive in estimating the calibration function. The architecture of these parameters will be dependent on the retrieved values after optimizing the hyperparameters.

- Batch size: This refers to the number of training samples utilized in one iteration. It controls the accuracy of the gradient error when training the neural network. This is also a sensitive parameter, which will also be found after optimizing the hyperparameters.

- Batch normalization: It is a technique that standardizes the inputs to a layer for each minibatch. In some cases, the addition of a batch normalization can accelerate the training process. However, this can also bring a negative effect as it breaks the independence between the training samples in the minibatch. Therefore, to find the optimum model, we will consider models with and without batch normalization.

- Dropout: Another regularization technique which refers to ignoring randomly selected neurons during the training process. The presence of a dropout layer aids to prevent overfitting the model and accelerates the training process, making it computationally cheap. However, this presence could also impact the convergence rate of the model. Therefore, this parameter will also be considered in the hyperparameter optimization.

4.4.1 Hyperparameter optimization

A relatively dense neural network yields to more accurate models with a cost of more expensive computations. To maximize the performance of the network with reasonable computational powers, an optimum combination of hyperparameters are required to be selected. We covered the different types of hyperparameter tuning in section 2.4. To obtain the best performing neural network, it would be recommended to use a combination of a manual search followed by a sparse grid search to obtain the best initial parameters applied in the Bayesian optimization. However, the time taken to run a grid search exponentially increases as the number of hyperparameters increase. Therefore, due to limited computational resources, this work only considers hyperparameter tuning using a manual search and Bayesian optimization.

The manual search is performed by changing each hyperparameter and evaluating the training performance for a relatively low number of epochs. The best model obtained from the manual search is used as the initial point of the Bayesian optimization. In this work, the Bayesian optimization is performed using the `Keras Tuner` library. The search space is generated according to the hyperparameters chosen to be optimized. This is then fed into function which is called 40 times.

Direct framework optimization

Various combinations of neural network architectures can be used in this framework, from a standard feed forward neural network to more complex models. We adopt two different approaches for modelling the architecture. The first approach entails the training of each parameter separately. This means, the input layer of the neural network is split into n-many branches, where n is the number of parameters being calibrated. Each of the branches is trained separately, after which the n branches are combined back into a single layer before reaching the output layer. To illustrate this, Figure 4.6 shows a schematic of this architecture. The second approach consisted of the standard feed forward neural with multiple hidden layers. The architecture included a potential presence of a batch normalization and dropout layer after each dense layers in the network. For completeness, we show the schematic of this architecture in Figure 4.7. The two approaches were compared using a manual search. It was found that the configuration with the standard neural network had improved performances compared to training the parameters separately. Therefore, the Bayesian optimization was performed in the standard feed forward neural network.

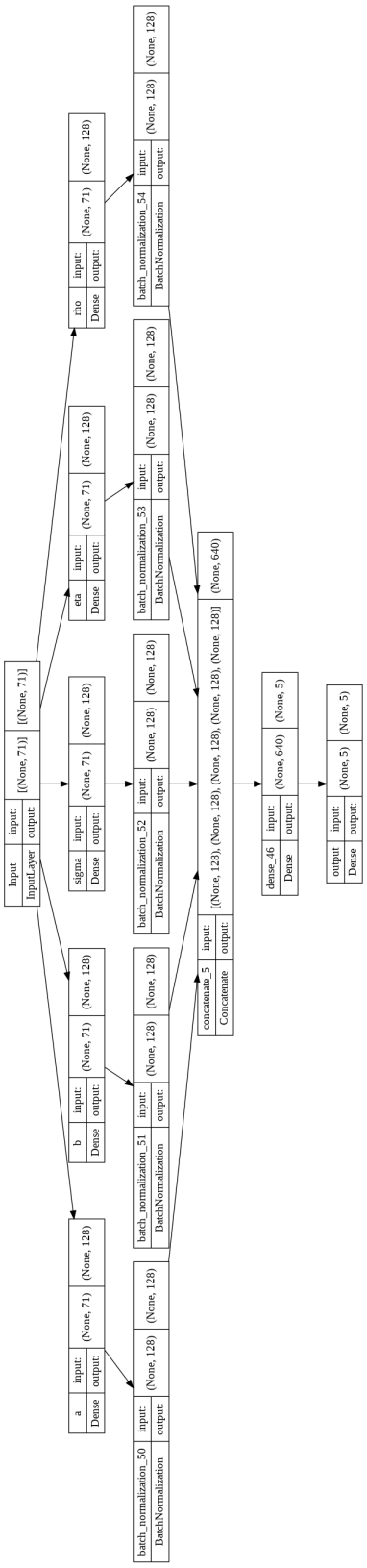


Fig. 4.6: Direct calibration architecture for training the G2++ parameters separately.

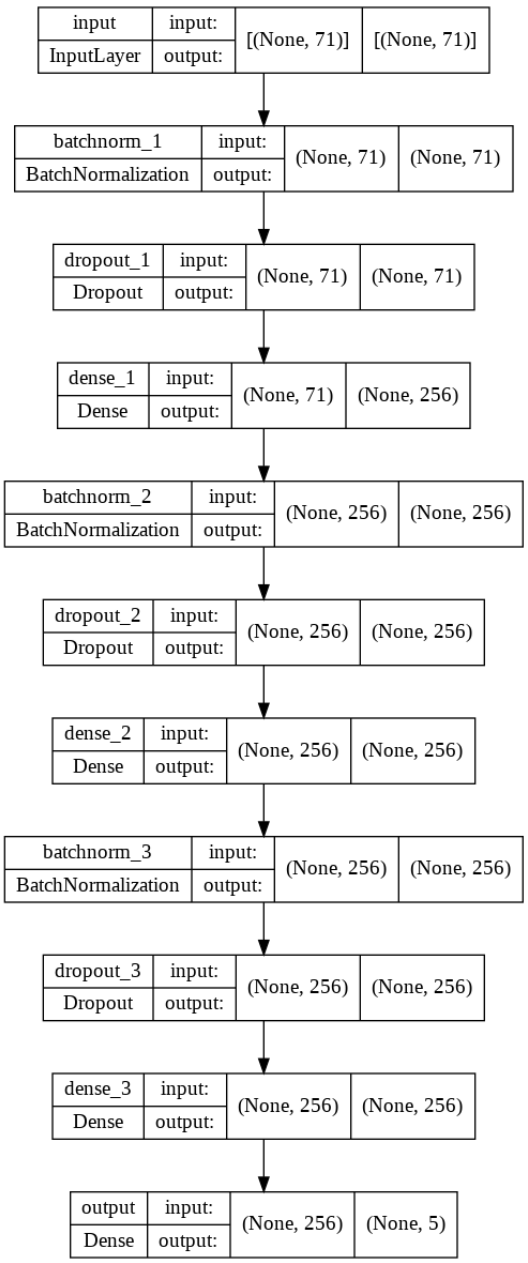


Fig. 4.7: Direct calibration feed forward architecture for the G2++ model.

Indirect framework optimization

The architecture of the indirect calibration is not as flexible as the direct framework, due to the reliance between the two phases. The hyperparameter tuning for the pricer/calibrator requires two different optimizations, firstly on the forward pass, followed by an optimization on the backward pass. Figure 4.8 shows the basic architecture of the indirect framework.

The forward pass used in this network resembles the standard feed forward neural network. The optimum architecture was obtained by the Bayesian search. Using these optimized parameters, the backward pass was optimized by a manual search. Through the manual search, it was determined the addition of a training loop in the backward pass substantially improves the performance of the neural. However, considering that this training takes place in an online space, it is more practical to have a minimum number of training loops to provide a reasonably efficient model.

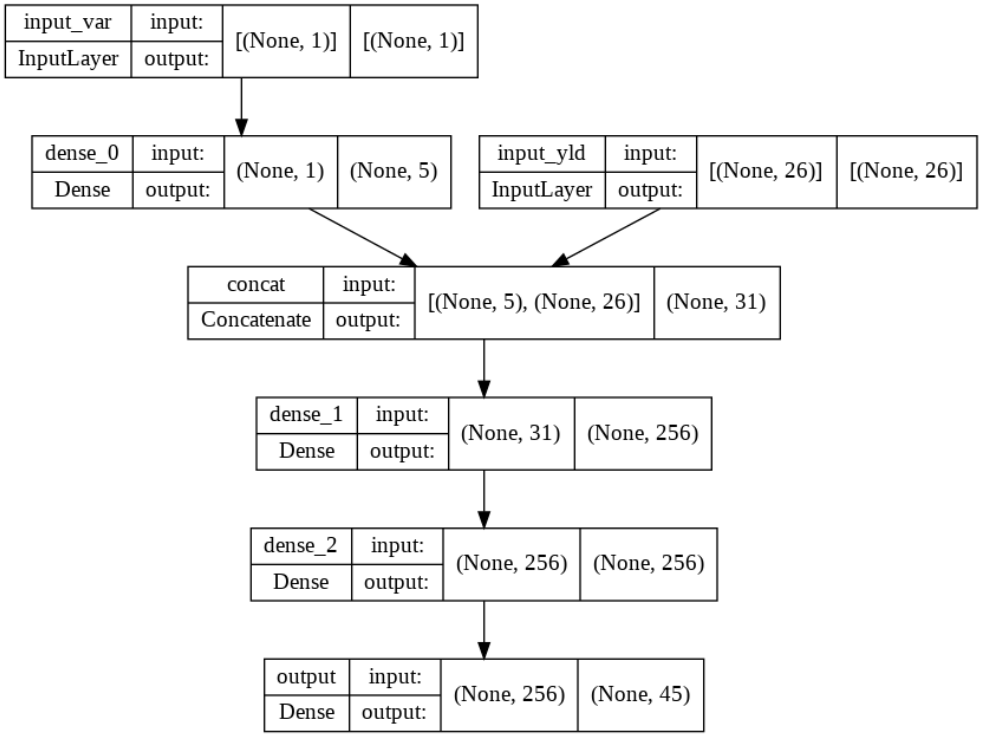


Fig. 4.8: Indirect calibration framework architecture for the G2++ model, which has five parameters.

Chapter 5

Calibrating The Short-Rate Models Using Neural Networks

In this chapter, the neural network calibration is put into practice. The one-and-two factor models are calibrated using the direct and indirect approach. A data set of 100 000 observation pairs is used in training each of the networks. This is followed by an out of sample testing with actual yield curves data. Finally, the efficacy of each of the models is compared to the standard numerical calibration techniques.

We outline the general process followed for the calibration process for each of these models. The process begins by utilizing the Bayesian optimization procedure to obtain the hyperparameters for the neural network. For each of the models, batch normalization and dropouts were found not to be optimal using the Bayesian search. The neural network is then trained using the optimum set of hyperparameters for 300 epochs. In instances where the early stopping criteria were satisfied, the neural network training was stopped before reaching 300 epochs. The training and validation losses of the neural network are monitored by observing the respective metrics. The respective loss plots for each of these models are shown in Figure A.1 to A.4. From the obtained plots, a general observation noted was the identical trend between the training and validation loss, which shows that the model does not over-fit the data. In addition to that, a stability point is quickly reached, after which the losses gradually decreases as the number of epochs progress. It is also noted that the mean-squared error (relative mean squared error) stabilizes much faster compared to the mean absolute error.

Ninety-nine actual market yield curves (monthly consecutive from March 2011 to October 2018) are used to test the efficiency of the network. These yield curves together with randomly chosen parameters using bounds from Table 4.2 for the respective model were used to obtain the test implied volatilities. These drawn parameters are labelled as the actual parameters. The trained neural network is then used to predict the model parameters. A trader would like this process to be

as efficient as possible in terms of the accuracy and calibration time taken.

The respective losses obtained during this calibration are recorded together with the time taken per calibration. To visualize the performance of the network, a plot of the actual and calibrated parameters over the trading years is displayed. The trading years considered are from March 2011 to October 2018. The predicted parameters from the neural network are then applied to the respective option pricing model to infer the implied volatilities. The actual volatilities are then compared to the neural network volatilities by means of regression plots and time series plot of the mean relative implied volatility error. Finally, the respective option pricing model is calibrated using the traditional numerical methods. The respective losses and time taken to calibrate are then compared to the neural network calibration frameworks.

5.1 Calibrating the one-factor Hull-White model

Calibration of the one-factor model was achieved by caplet volatilities. For each observation pair, a row vector of 20 caplet implied volatilities was generated for option expiry's ranging from 0.25 years to 5 years with a tenor length of 0.25 years.

5.1.1 Direct: one-factor

The direct method uses the market data as the input to the network, and calibrated parameters as the outputs. The cost function optimized in this framework is defined by Equation 4.1. The hyperparameter optimization was performed using the base architecture shown in Figure 4.7. The optimum set of hyperparameters obtained through the Bayesian optimization are illustrated in Table 5.1 below.

Tab. 5.1: Optimum parameters for direct calibration using one-factor model.

Hyperparameter	Value
Learning Rate	1.22e-02
Layers	5
Neurons per Layer	416
Batch Size	32

The training of this model was monitored by observing the PMSE and PMAE between the actual parameters and predicted parameters. The losses and the time taken for the training and calibration phases are summarized in Table 5.2 below. It is important to note that the time shown for the calibration phase is the average

time per calibration. In addition, Figure 5.1 and 5.2 shows the comparison plots obtained using this calibration framework.

Tab. 5.2: Performance measures of the NN training and calibration using the direct calibration for the one-factor model.

One-factor-ANN	PMSE	PMAE	MRIVE	Time
Training	1.27e-08	8.08e-05	-	9 mins
Calibration	1.13e-08	7.76e-05	9.33e-04	0.073 s

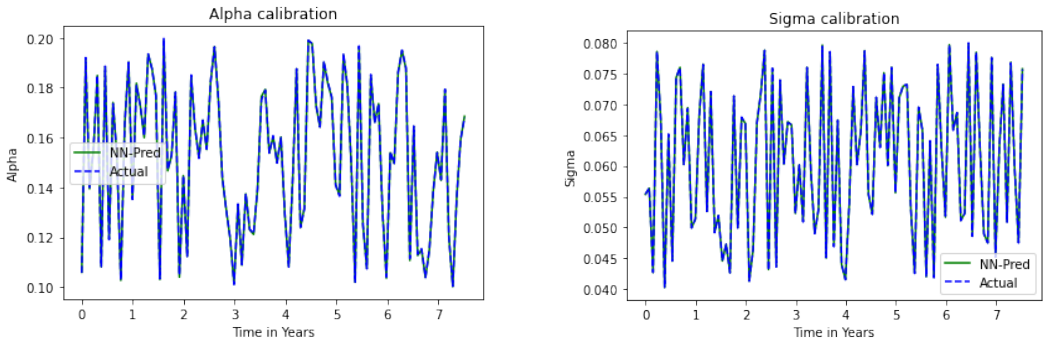
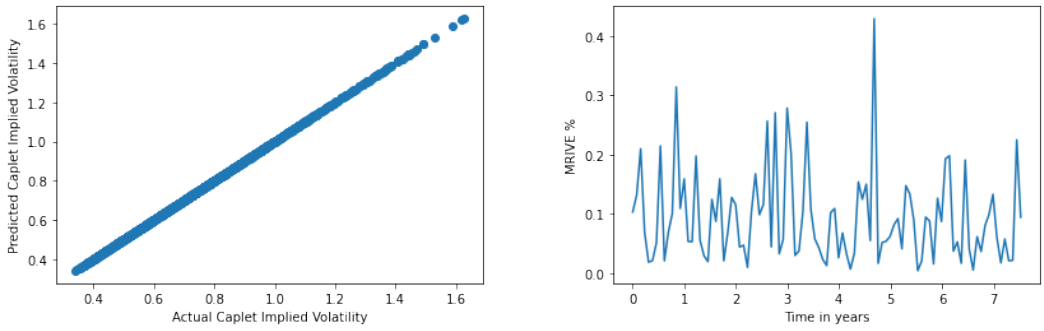


Fig. 5.1: One-factor direct, actual versus calibrated parameters over the trading years.



(a) Regression plot for 99 trading days

(b) MRIVE over trading years

Fig. 5.2: One-factor direct, estimated implied volatility plots using NN.

5.1.2 Indirect: one-factor

The indirect framework has two distinct phases. The pricing phase constitutes the offline model training process, and the backward pass would be performed in an

online space. The forward pass uses an input vector of the model parameters (α, σ) and yield curves, with the output set to the caplet implied volatilities. The optimum set of hyperparameters obtained for this model through the Bayesian optimization are shown in Table 5.3 below.

Tab. 5.3: Optimum parameters for indirect calibration using one-factor model.

Phase	Hyperparameter	Value
Forward Pass	Learning Rate	2.66e-03
	Layers	4
	Nodes	480
	Batch Size	80
Two-Factor	Learning Rate	1e-02
	No of Loops	10

The forward pass is first trained using the optimized architecture. Since the forward pass constitutes of the major training process, the training and validation losses are monitored by observing the VRMSE and VMAE between the market volatilities and predicted volatilities. As before, these losses and the time taken to train the model are displayed in Table 5.4.

Tab. 5.4: Performance measures of the NN training and calibration using the indirect framework for the one-factor model.

One-factor-ANN	VRMSE	VMAE	MRIVE	Time
Training	7.73e-07	4.81e-04	-	13 mins
Calibration	1.34e-09	2.20e-05	1.55e-05	7.27 s

The pre-trained model is then used in the online space for the calibration process. The weights from the previous model are copied over to the "new" model whilst being set to non-trainable. Consequently, the resulting model is trained using a training loop for one observation pair, with 100 epochs per loop. The losses and time taken during the calibration phase are also shown in Table 5.4. For consistency in testing, this model is tested using the same test data set used for the direct one-factor model, the relevant plots obtained are shown in Figure 5.3 and 5.4. A significant improvement of the losses is observed after the application of the backward pass. It should be noted that the loss function of this model did not change from the pricing phase to the calibration phase.

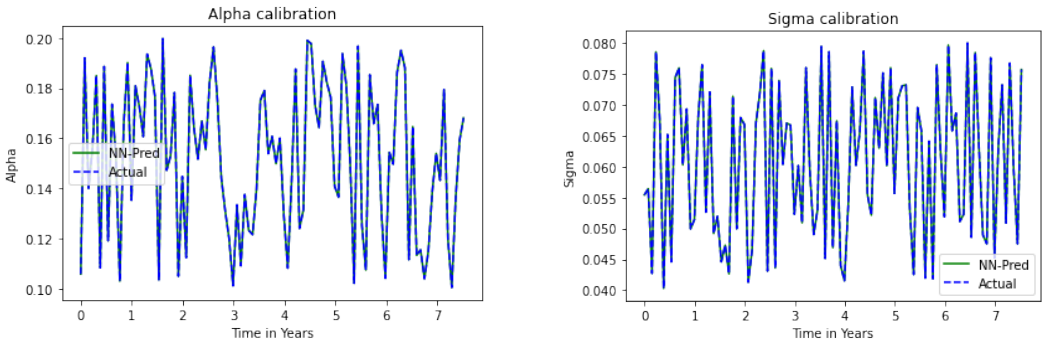
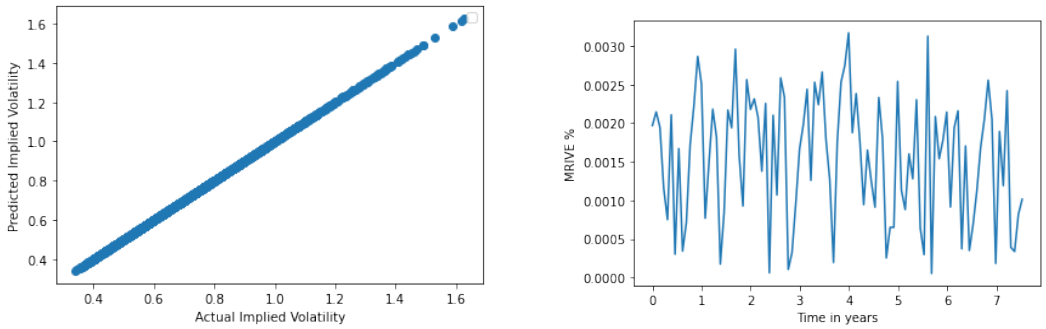


Fig. 5.3: One-factor indirect, actual vs calibrated parameters over the trading years.



(a) Regression plot for 99 trading days

(b) MRIVE over trading years

Fig. 5.4: One-factor indirect, estimated implied volatility plots using NN.

5.1.3 Numerical calibration: one-factor model

The numerical calibration was achieved by minimizing the actual and predicted caplet prices. A calibration to prices was preferred to volatilities, to save the optimization time taken. The root finding algorithm for stripping the implied volatilities would have to be called within the optimization function, which would take additional time in the calibration phase. Instead, we utilize the Black 76 pricing model to obtain caplet prices using observed volatilities in the market. In turn, we define a calibration function that minimizes the relative mean squared prices between the actual and model prices. For consistent comparisons with the neural network, the same loss function as for the indirect method was used, given by Equation 4.2, but using prices instead of volatilities. The optimization of the loss function was performed by the Nelder-Mead algorithm using the `scipy optimize` library in `Python`.

Generally, for numerical calibration, practitioners have to run an optimizer from several initial points to obtain a global minimum. Since this research focuses on

NN calibration, this process was omitted from the methodology. Therefore, in this work, we set up the model and run the minimization algorithm with a manually selected optimal initial point.

The calibrated parameters obtained through the minimization were used to determine the implied volatilities, and thus the MRIVE. A MRIVE of 2.48e-05 was obtained with an average calibration time of 0.13 seconds. Figure 5.5 and 5.6 below shows parameter recovery plots and the MRIVE for the numerical calibration over each trading day respectively.

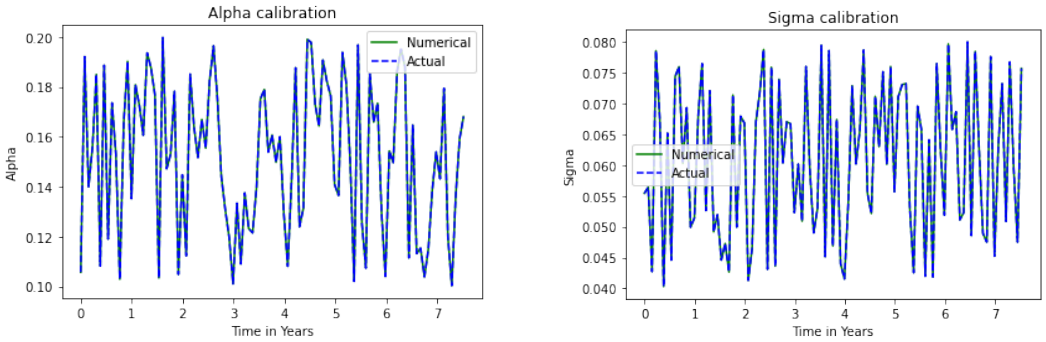
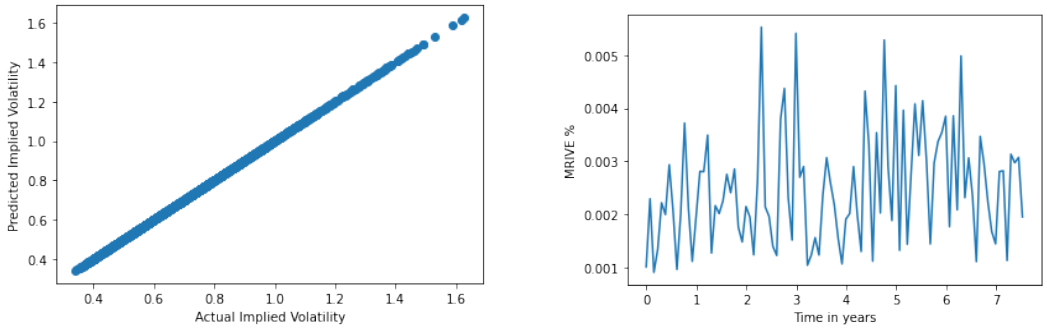


Fig. 5.5: One-factor numerical calibration, actual vs calibrated parameters over the trading years.



(a) Regression plot for 99 trading days

(b) MRIVE over trading years

Fig. 5.6: One-factor numerical calibration, estimated implied volatility plots.

5.1.4 Comparison and remarks for the one-factor model calibration

The close correlation between the actual and predicted parameters for the calibration framework suggests the loss function was minimized appropriately to find the global minima in the calibration process. Additionally, the linear relationship between the regression plots indicates a successful calibration in minimizing the

distance between true and predicted volatilities.

The performance of the calibration using two different architectures have two distinct observations. The direct method produced a significantly faster calibration time compared to the indirect method. In contrast to that, the indirect method had a substantially improved accuracy. Table 5.5 below shows a summary of the different calibration frameworks considered in this dissertation for the one-factor model.

Tab. 5.5: Comparison of different calibration frameworks for the one-factor model.

Model	Framework	MRIVE	Time
One Factor	Direct	9.33e-04	0.073 s
	Indirect	1.55e-05	7.27 s
	Numerical	2.48e-05	0.13 s

Comparing the direct approach to the numerical calibration, it can be seen the time taken for calibrating improved with a degradation in the accuracy. This is not apprehended in calibration of option pricing models where a trader requires the hedging to be as accurate as possible. To leverage on the significant enhancement of calibration time, the accuracy of the direct method needs to improved. A more accurate neural network architecture can be devised by considering a more dense model with neurons varying across the hidden layers together with a larger search space for the Bayesian optimization. In addition to that, the neural network could be trained with a much larger data set to improve the accuracy. The implementation of this was not explored in this paper due to the limited computational resources. Generally, a more dense network would take longer to train, but the online calibration time would not be impacted significantly. Therefore, if the alterations of the architecture can match the accuracy of numerical methods, the direct method would prove to be the most efficient calibration framework.

When considering the indirect method, the accuracy marginally improved from the numerical calibration, but taking considerably longer to calibrate. The increase in time is due to the model being trained in an online phase. However, this can be seen as an improvement to the numerical calibration as one does not need to run an optimizer from different starting points for the neural network. Furthermore, despite the accuracy being better than the numerical calibration, this can further be improved as suggested above. Therefore, it is possible that the indirect neural network calibration method leads to an improvement of the numerical calibration in terms of both accuracy and time taken for the one-factor model.

5.2 Calibrating the two-factor model

The two-factor model is calibrated to caplet and swaption implied volatilities. As described before, this was done explicitly to pin certain parameters in the calibration framework which cannot be achieved by caplet volatilities only. For each observation pair, a row vector of 25 swaption volatilities and 20 caplet volatilities are used as the market observables, alongside the yield curves. Same procedures are followed as for the direct and indirect one-factor model to obtain the respective results under the two-factor model. However, since the calibration was considered for a combination of caplets and swaptions, the MRIVE for each of them is individually defined by CMRIVE and SMRIVE respectively. The results obtained under this model are shown in section 5.2.1 and 5.2.2 respectively.

5.2.1 Direct: two-factor

Tab. 5.6: Optimum parameters for direct calibration using two-factor model.

Hyperparameter	Value
Learning Rate	0.0955
Layers	4
Neurons per Layer	416
Batch Size	176

Tab. 5.7: Performance measures of the NN training and calibration using the direct framework for the two-factor model.

One-factor-ANN	PMSE	PMAE	CMRIVE	SMRIVE	Time
Training	6.088e-06	1.96e-03	-	-	5 mins
Calibration	6.38e-06	1.99e-03	2.88e-02	3.39e-03	0.078 secs

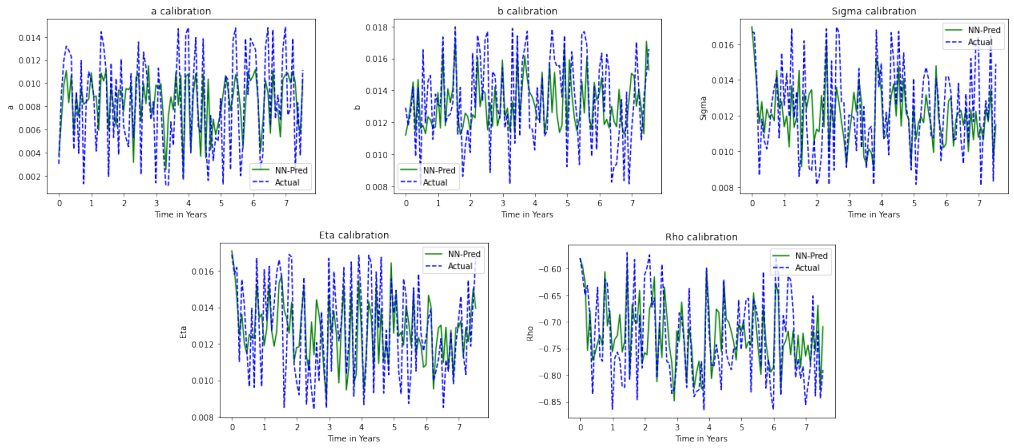
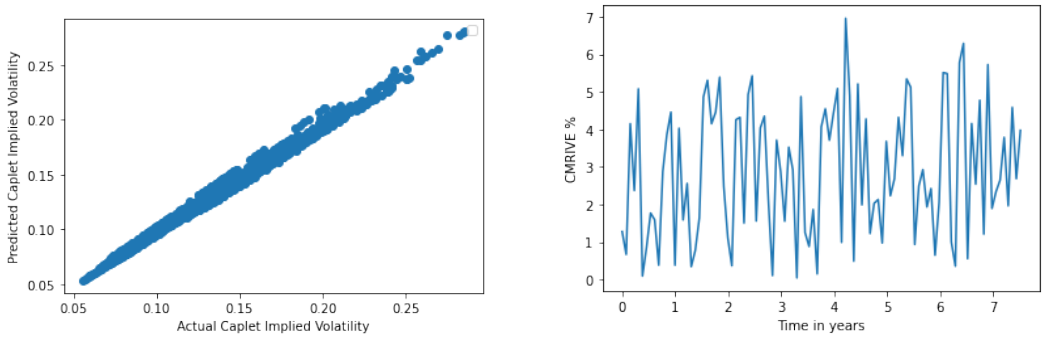
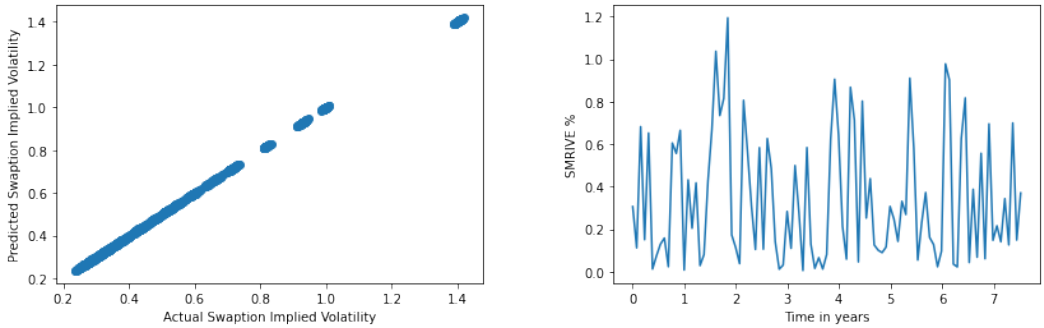


Fig. 5.7: Two-factor direct, actual vs calibrated parameters over the trading years.



(a) Caplet regression plot for 99 trading days **(b)** CMRIVE over trading years

Fig. 5.8: Two-factor direct, estimated caplets implied volatility plots using NN.



(a) Swaption regression plot for 99 trading days **(b)** SMRIVE over trading years

Fig. 5.9: Two-factor direct, estimated swaption implied volatility plots using NN.

5.2.2 Indirect: two-factor

Tab. 5.8: Optimum parameters for indirect calibration using two-factor model.

Phase	Hyperparameter	Value
Forward Pass	Learning Rate	2.15e-05
	Layers	4
	Nodes	320
	Batch Size	192
Backward Pass	Learning Rate	1e-02
	No of Loops	12

Tab. 5.9: Performance measures of the NN training and calibration using the indirect for the-two factor model.

Two-factor-NN	VRMSE	VMAE	CMRIVE	SMRIVE	Time
Training	1.59-06	2.01e-04	-	-	19 mins
Calibration	5.39e-07	1.89e-04	9.72e-04	1.74e-04	7.85 s

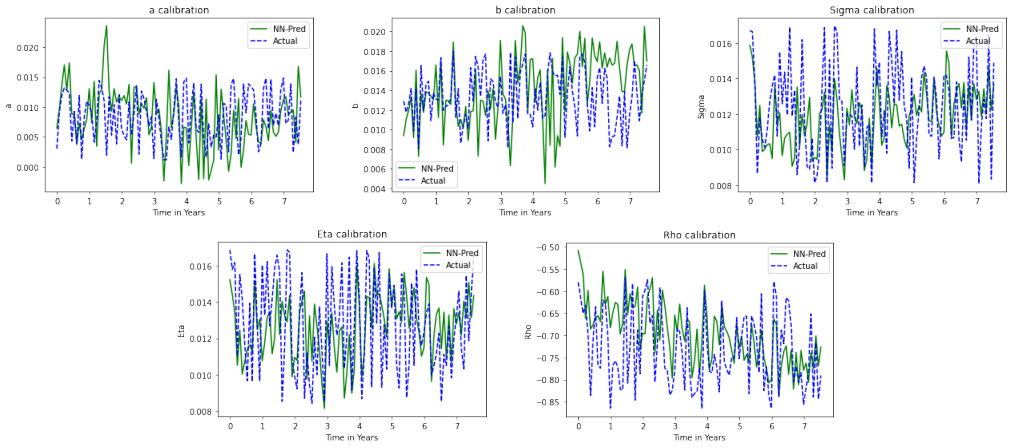
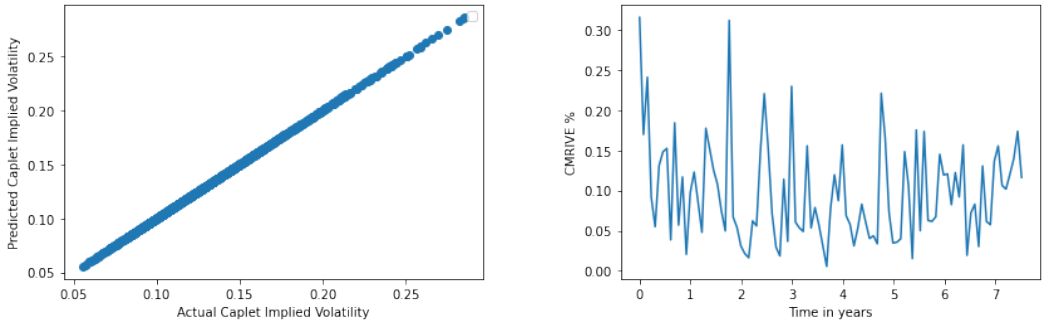
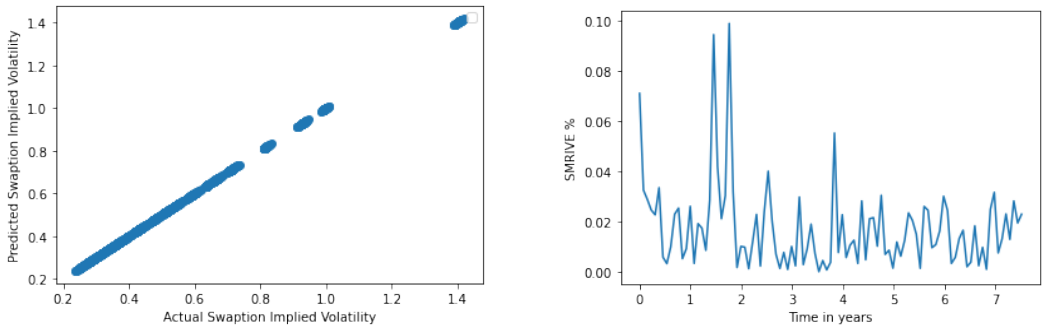


Fig. 5.10: Two-factor indirect, actual vs calibrated parameters over the trading years.



(a) Caplet regression plot for 99 trading days **(b)** CMRIVE over trading years

Fig. 5.11: Two-factor indirect, estimated caplets implied volatility plots using NN.



(a) Swaption regression plot for 99 trading days **(b)** SMRIVE over trading years

Fig. 5.12: Two-factor indirect, estimated swaption implied volatility plots using NN.

5.2.3 Numerical calibration: two-factor

The numerical calibration for the two-factor model was performed by minimizing the actual and predicted caplets and swaption prices simultaneously. Since pricing of swaption was more efficient in `matlab`, therefore the optimization was also performed using the `fminsearch` function in `matlab`.

The calibrated parameters obtained through the minimization were used to obtain the implied volatilities, and thus the CMRIVE and SMRIVE. A CMRIVE and SMRIVE of 6.66e-04 and 2.21e-04 were obtained with an average calibration time of 204 seconds. The parameter recovery plots, together with the respective regression plots for the implied volatilities and the associated MRIVE plots are shown below.

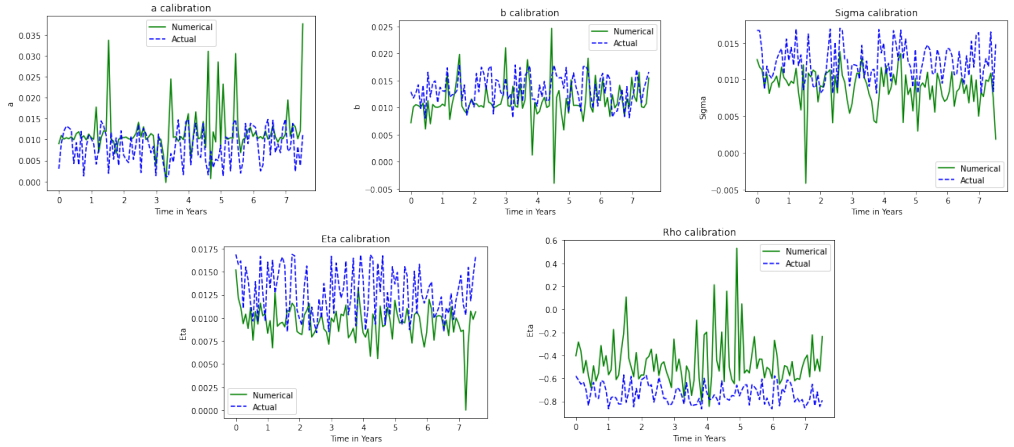
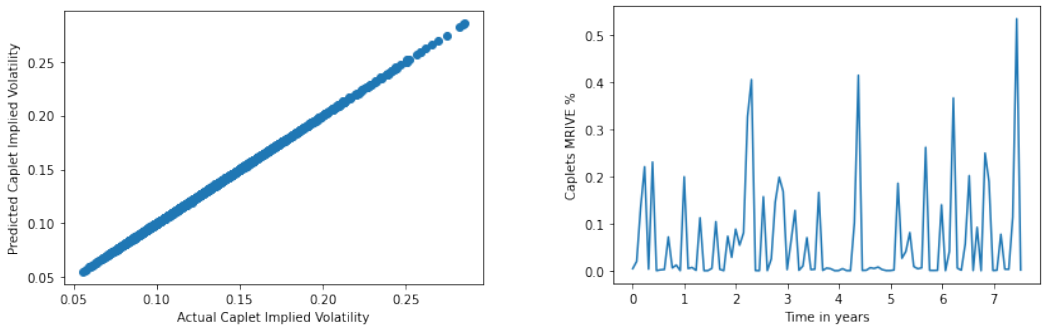


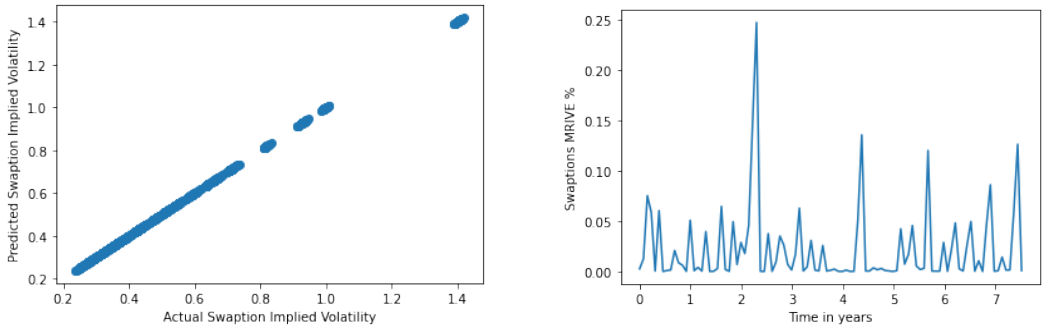
Fig. 5.13: Two-factor numerical calibration, actual vs calibrated parameters over the trading days.



(a) Caplet regression plot for 99 trading days

(b) CMRIVE over trading years

Fig. 5.14: Two-factor numerical calibration, estimated caplets implied volatility plots



(a) Swaption regression plot for 99 trading days

(b) SMRIVE over trading years

Fig. 5.15: Two-factor numerical calibration, estimated swaption implied volatility plots.

5.2.4 Comparison and remarks for the two-factor model calibration

Despite the addition of swaption prices to the calibration framework, the variance between the actual and calibrated parameters for the two-factor model is significant. Typically in calibration, it is expected to have a close correlation between the two, as we observed for the one-factor model. In order to verify if this is expected for the two-factor calibration, we compare it to the parameter recovery of the two-factor model using the numerical calibration methods, shown in Figure 5.13. It is observed that the numerical method also fails to recover the model parameters, infact the performance is even poor compared to the neural network calibration. In addition to that, Alaya *et al.* (2021) attempted to recover the G2++ model parameters using a covariance matrix structure between forward rates of different maturities. However, they also failed to recover the actual model parameters, with their PMSE higher than the ones obtained in this work. Failure to recover model parameters accurately does not mean a calibration framework is inconsistent. In reality, traders never know the "actual" parameters during calibration. All that matters is that the parameters recover market prices reasonably well. For this model, it can be observed that different parameter sets to the "actual parameters" recover prices with a sufficient accuracy. This suggests the two-factor model is under-determined, meaning that more information from the market is needed to pin down the model parameters.

Further investigations showed that the model parameters can be recovered with sufficient accuracy by the addition of a covariance/correlation matrix structure. To perform the calibration using this addition on a given day, historical yield curves would be required on as many days as necessary to obtain an accurate assessment of the actual covariances. In this case, the outputs of the model would depend on

current and past market data. For calibration of option pricing models, we typically want to price according to the current market observable, whereas, the model should be able to accommodate sudden changes in market volatilities. The yield curves and the covariance structure would not change drastically. Thus, the parameters estimated would be very similar for each day regardless of the sudden changes in the volatilities. Using a neural network trained by incorporating the covariance structure would lead to incorrect pricing and hedging of the option pricing models. In that case, despite a more accurate parameter recovery, this method was not investigated further.

We now compare the performance of the different calibration frameworks under the two-factor model, using a summary as presented in Table 5.10.

Tab. 5.10: Comparison of different calibration frameworks for two-factor model.

Model	Framework	CMRIVE	SMRIVE	Time
Two Factor	Direct	2.28e-02	3.39e-03	0.078 s
	Indirect	9.72e-04	1.74e-04	7.85 s
	Numerical	6.66e-04	3.59e-04	204 s

The comparison of the direct and indirect model is analogous to the one observed for the one-factor case. Since we would not want to compromise a degradation in accuracy, the indirect method is preferred.

The accuracy of the indirect method is almost identical to the numerical calibration. However, it is noted that the NN approximates swaptions more accurately than caplets. A significant improvement in the calibration time is observed using the indirect neural network, which is twenty-six times faster than the numerical calibration technique. Therefore, in terms of model efficiency, the neural network calibration using the indirect method is observed to have a better performance than the conventional calibration techniques used.

Chapter 6

Conclusion

In this paper, we explored multiple facets of neural networks in the context of calibrating short rate models. This began with an introduction to deep learning and the underlying process to obtain an optimum neural network architecture. A theoretical background to the one-and-two factor Hull-White models was provided and the analytical solutions were stated for the respective calibration instruments. A direct and indirect calibration frameworks were devised to invert the option pricing models using neural networks. The models were trained with synthetically generated data (by combining information from historical market data) and tested by an out-of-sample data consisting of actual market yield curves for ninety-nine trading days for a horizon of 7.5 years.

In terms of efficacy of the models, the direct calibration framework performed significantly faster, with an online calibration time of less than 0.1 seconds. This observation was consistent with Hernandez (2016), where they suggested the online calibration phase consists of multiplying matrices which can quickly be performed by any computer. However, a drawback using the direct method was the degradation of the calibration accuracy (mean relative error of 2.88%) which is despised by traders. It was deduced that, if one can improve the accuracy of this method by manipulating the architecture of the network or training with a larger data set, then the direct method could be the most efficient calibration framework to be used in an online space. On the other hand, the indirect method proved to be an improvement to the numerical calibration. The accuracy of the neural network was comparable to the traditional technique, where the mean relative implied volatility error remained below a tenth of one percent throughout the out-of-sample testing. This provides an indication that neural networks can be used as a calibration tool to effectively map the non-linear relationship between market prices and model parameters for a given financial model. In terms of the time taken to calibrate the indirect method, an immediate improvement of the one-factor model may not be seen. However, the online calibration of the two-factor model was twenty-six times faster than the

numerical method. The significant improvement in the calibration time without sacrifice in accuracy indicates the potential benefits of using neural networks for model calibration where numerical techniques take too long for practical use.

Several aspects of neural network calibration were explored in this paper, in particular, the application of this framework to a complex financial short rate model (two-factor model). It is evident much more research can be done to build on this foundation. Firstly, the hyperparameter optimization carried out in this research together with the ones by Hernandez (2016) and Mavuso *et al.* (2017) had a limited search space, therefore it is not clearly indicative what constitutes as a good neural architecture for calibration. Additionally, it might be worthwhile to explore different neural network algorithms other than the feed forward network. As such, Hernandez (2016) explored the use of convolutional networks in the calibration of the one-factor Hull-White model, and suggested they do not significantly improve the calibration accuracy despite taking longer to train. However, application of networks such as the recurrent neural networks could prove to be more efficient, which remains a potential area to investigate. Finally, to further validate the application of neural networks in the context of calibration frameworks, it is essential to investigate the performance of a fully trained network for pricing and hedging at a trading desk level, and compare it to the numerical calibration techniques.

Bibliography

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M. *et al. (2016). Tensorflow: A system for large-scale machine learning,* 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), *pp. 265–283.*

Alaya, M. B., Kebaier, A. and Sarr, D. (2021). Deep calibration of interest rates model, arXiv preprint arXiv:2110.15133 .

Amat, S., Busquier, S. and Plaza, S. (2004). Review of some iterative root-finding methods from a dynamical point of view, Scientia **10***(3): 35.*

Bayer, C. and Stemper, B. (2018). Deep calibration of rough stochastic volatility models. 2018, arXiv preprint arXiv:1810.03399 .

Betrò, B. and Schoen, F. (1991). A stochastic technique for global optimization, Computers & Mathematics with Applications **21***(6-7): 127–133.*

Björk, T. (1998). Arbitrage Theory in Continuous Time, *Oxford University Press Oxford.*

Black, F. (1976). The pricing of commodity contracts, Journal of Financial Economics **3***(1-2): 167–179.*

Black, F. and Karasinski, P. (1991). Bond and option pricing when short rates are lognormal, Financial Analysts Journal **47***(4): 52–59.*

Black, F. and Scholes, M. (1973). The pricing of options and corporate liabilities, Journal of Political Economy **81***(3): 637–654.*

Brigo, D. and Mercurio, F. (2006). Interest rate models-theory and practice: with smile, inflation and credit, *Springer.*

Büchel, P., Kratochwil, M., Nagl, M. and Rösch, D. (2021). Deep calibration of financial models: turning theory into practice, Review of Derivatives Research *pp. 1–28.*

Burgess, N. (2014). An Overview of the Vasiček Short Rate Model, Available at SSRN 2479671 .

Canto, R. (2008). Modelling the term structure of interest rates: A literature review, Available at SSRN 1640424 .

Chauhan, N. S. (2021). Optimization Algorithms in Neural Networks , https://dphi.tech/blog/optimization-algorithms-in-neural-networks/.

Chen, L. (1996). A three-factor model of the term structure of interest rates, Interest Rate Dynamics, Derivatives Pricing, and Risk Management, *Springer, pp. 1–36.*

Cohen, S. N., Snow, D. and Szpruch, L. (2021). Black-box model risk in finance, Available at SSRN 3782412 .

Cont, R. (2010). Model calibration, Encyclopedia of quantitative finance .

Cox, J. C., Ingersoll, J. E. and Ross, S. A. (1985). A theory of the term structure of interest rates, Econometrica **53***(2): 385–407.*

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function, Mathematics of control, signals and systems **2***(4): 303–314.*

Dothan, L. U. (1978). On the term structure of interest rates, Journal of Financial Economics **6***(1): 59–69.*

Driessen, J., Klaassen, P. and Melenberg, B. (2003). The performance of multi-factor term structure models for pricing and hedging caps and swaptions, Journal of Financial and Quantitative Analysis **38***(3): 635–672.*

Falcó, A., Navarro, L. and Nave, J. (2009). The Hull-White model and Multiobjective Calibration with Consistent Curves: Empirical Evidence, Springer **103***(2): 235–249.*

Fama, E. F. and Bliss, R. R. (1987). The information in long-maturity forward rates, The American Economic Review **77***(4): 680–692.*

Gibson, R., Lhabitant, F.-S. and Talay, D. (2010). Modeling the term structure of interest rates: a review of the literature, Now Publishers Inc, Available at SSRN 275076 .

Glova, J. (2010). Matrix theory application in the bootstrapping method for the term structure of interest rates, Economic Analysis **43***(1-2): 44–49.*

Haykin, S. and Lippmann, R. (1994). Neural networks, a comprehensive foundation, International Journal of Neural Systems **5***(4): 363–364.*

Hernandez, A. (2016). Model calibration with neural networks, IBM Risk Analytics, Available at SSRN 2812140 .

Ho, T. S. and Lee, S.-B. (1986). Term structure movements and pricing interest rate contingent claims, Journal of Finance **41***(5): 1011–1029.*

Horvath, B., Muguruza, A. and Tomas, M. (2021). Deep learning volatility: A deep neural network perspective on pricing and calibration in (rough) volatility models, Quantitative Finance, Available at SSRN 3322085 **21***(1): 11–27.*

Hull, J. C. and White, A. D. (1994). Numerical procedures for implementing term structure models ii: Two-factor models, Journal of Derivatives **2***(2): 37–48.*

Hull, J. and White, A. (1990). Pricing interest-rate-derivative securities, Review of Financial Studies **3***(4): 573–592.*

Itkin, A. (2019). Deep learning calibration of option pricing models: some pitfalls and solutions, arXiv preprint arXiv:1906.03507 .

Itô, K. (1944). Stochastic integral, Proceedings of the Imperial Academy **20***(8): 519–524.*

Liu, S., Borovykh, A., Grzelak, L. A. and Oosterlee, C. W. (2019). A neural network-based framework for financial model calibration, Journal of Mathematics in Industry **9***(1): 1–28.*

Longstaff, F. A. and Schwartz, E. S. (1992). Interest rate volatility and the term structure: A two-factor general equilibrium model, Journal of Finance **47***(4): 1259–1282.*

Mavuso, M., Marr, A., Mitoulis, N. and Singh, A. (2017). Model Calibration with Neural Networks, *ACQuFRR,* http://www.aifmrm.uct.ac.za/wp-content/uploads/2017_Financial_Mathematics_Team_Challenge_Research_Reports.pdf.

Meng, Q., Kaplin, A. and Levy, A. (2013). Estimating parameters in the singlefactor hull-white model using historical data, Technical report, Moody's Analytics – Quantitative Research Group .

Merton, R. C. (1973). Theory of rational option pricing, The Bell Journal of Economics and Management Science *pp. 141–183.*

Miller, S. J. (2006). The method of least squares, Mathematics Department Brown University **8***: 1–7.*

Mooney, C. Z. (1997). Monte carlo simulation, *Sage.*

Moré, J. J. (1978). The levenberg-marquardt algorithm: implementation and theory, Numerical analysis, *Springer, pp. 105–116.*

O'Malley, T. and Bursztein, E. (2019). Kerastuner, https://github.com/keras-team/keras-tuner.

Park, F. (2004). Implementing interest rate models: A practical guide, Capital Markets and Portfolio Research Inc (CMPR) .

Patro, S. and Sahu, K. K. (2015). Normalization: A preprocessing stage, arXiv preprint arXiv:1503.06462 .

Perez Cruz, L. and Treisman, D. (2018). Ai turning points and the road ahead, International Joint Conference on Computational Intelligence, *Springer, pp. 89–107.*

Ravikumar, M., Cheng, H. and McCourt, M. (2020). Bayesian optimization, https://sigopt.com/blog/bayesian-optimization-101/.

Rebonato, R. (1999). On the pricing implications of the joint lognormal assumption for the swaption and cap markets, Journal of Computational Finance **2***(3): 57–76.*

Rendleman, R. J. and Bartter, B. J. (1980). The pricing of options on debt securities, Journal of Financial and Quantitative Analysis **15***(1): 11–24.*

Ruf, J. and Wang, W. (2019). Neural networks for option pricing and hedging: a literature review, arXiv preprint arXiv:1911.05620 .

Scheffer, M. and Zacharias, M. (2018). A comparative study of the 1-Factor Hull-White and the G2++ interest rate model, Milliman .

Schumann, G. W. (2016). Trolle-schwartz hjm interest rate model, AIFMRM, MPhil Dissertation, University of Cape Town. .

Senior, A., Heigold, G., Ranzato, M. and Yang, K. (2013). An empirical study of learning rates in deep neural networks for speech recognition, 2013 IEEE international conference on acoustics, speech and signal processing, *IEEE, pp. 6724–6728.*

Snoek, J., Larochelle, H. and Adams, R. P. (2012). Practical Bayesian Optimization of machine learning algorithms, Advances in neural information processing systems, arXiv preprint arXiv:1206.2944 .

Szandała, T. (2021). Review and comparison of commonly used activation functions for deep neural networks, Bio-inspired Neurocomputing, *Springer, pp. 203–224.*

Vasiček, O. (1977). An equilibrium characterization of the term structure, Journal of Financial Economics **5***(2): 177–188.*

Wu, J., Chen, X.-Y., Zhang, H., Xiong, L.-D., Lei, H. and Deng, S.-H. (2019). Hyperparameter optimization for machine learning models based on Bayesian optimization, Journal of Electronic Science and Technology **17***(1): 26–40.*

Appendix A

Training Loss Plots

After the neural network calibration, the loss plots are observed to ensure the model does not over fit the data in the training set. The loss plots for each of the neural network calibrations are displayed below.

One-factor model

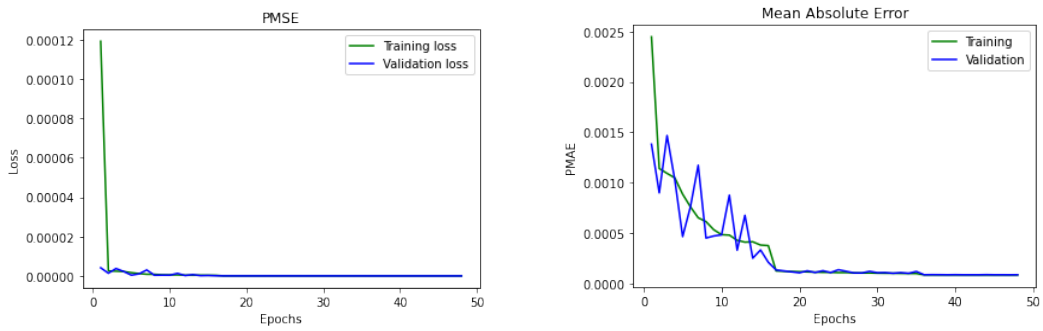


Fig. A.1: One-factor direct NN calibration losses

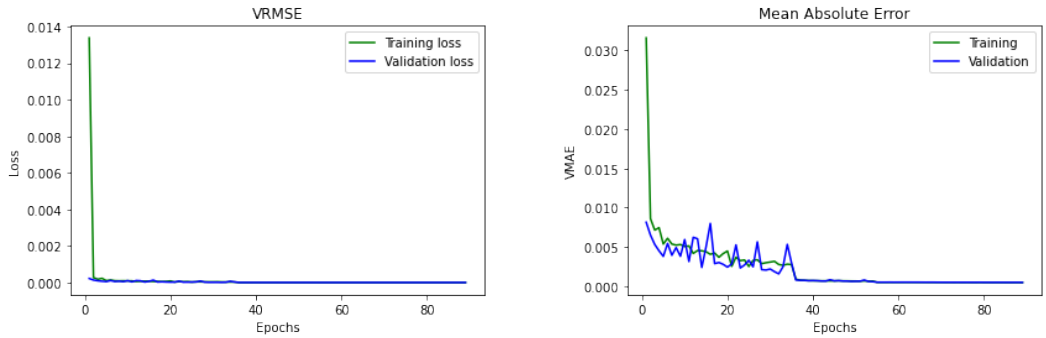


Fig. A.2: One-factor indirect NN calibration losses

Two-factor model

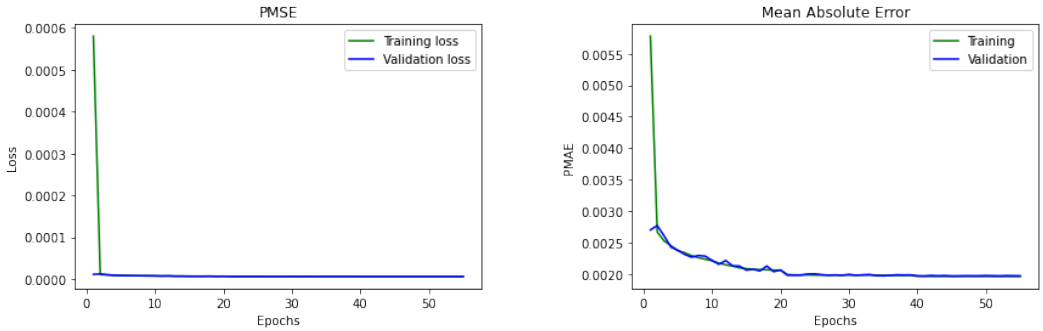


Fig. A.3: Two-factor direct NN calibration losses

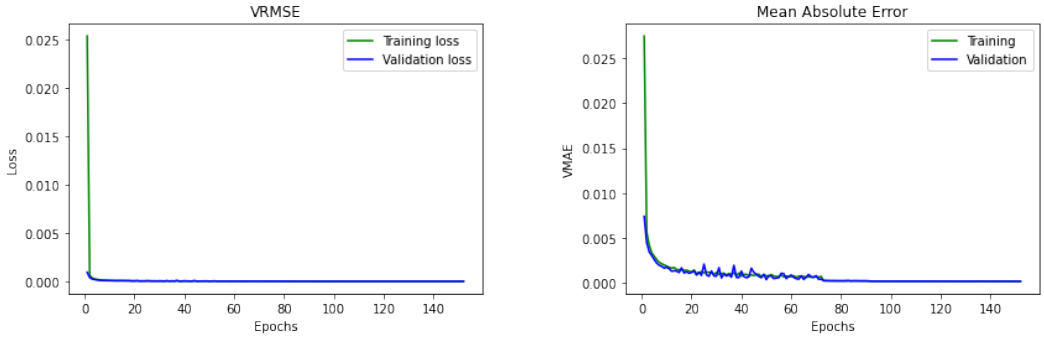


Fig. A.4: Two-factor indirect NN calibration losses