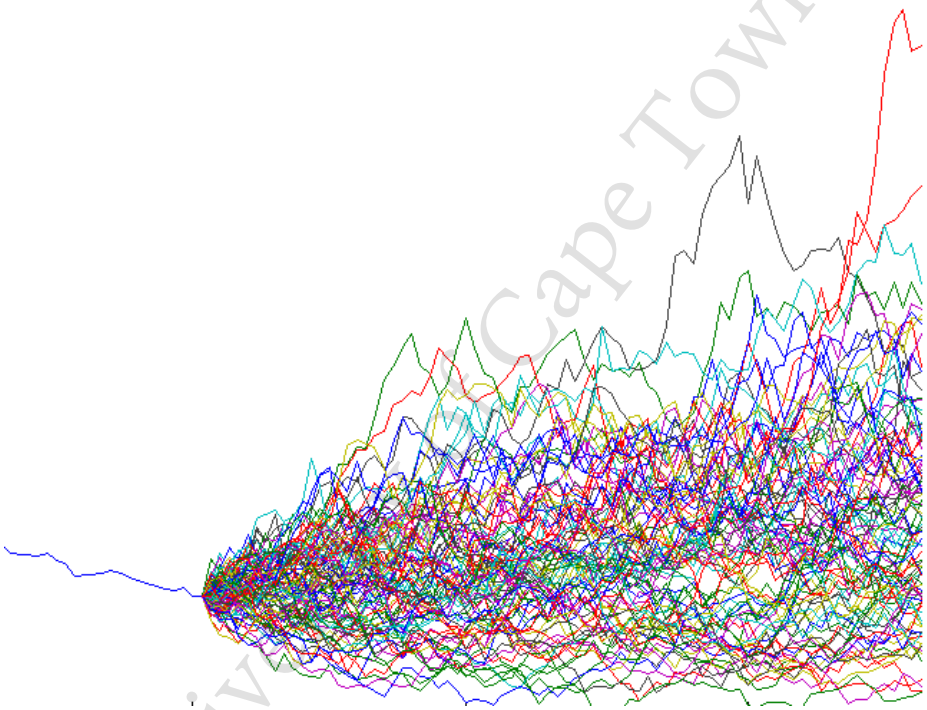
Neural Network Libor Market Model for Pricing and Hedging Interest Rate Derivatives



Author: Yuri Robbertze
Department of Statistical Sciences

December 13, 2021

Acknowledgments

Firstly, I would like to thank my supervisor Melusi Mavuso for the guidance provided throughout the course of this dissertation. The advice and support provided was paramount to me completing this dissertation.

Secondly, I would like to thank Jake Stangroom for inspiration throughout. I had the pleasure of co-supervising his honours thesis and the effort he put in kept me focused toward my goal. His interesting solutions to hyper-parameter optimisation, such as using a tensorboard, inspired my own.

Thirdly, I would like to make a personal remark. In the course of my dissertation the dreaded COVID-19 pandemic struck, I had two surgeries, lectured and tutored. I am eternally grateful to have been able to complete this challenging dissertation despite the circumstances that arose.

Finally, the financial assistance of the National Research Foundation (NRF) towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the NRF.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

1.1 Abstract

One of the most important interest rate models is the *LIBOR Market Model* (LMM). This model was first developed in [Brace et al., 1997] and is now used extensively in the financial industry to price and hedge interest rate derivatives. It is also used to price and hedge exotic options which are embedded in insurance contracts, making the development of this model's accuracy paramount to society as a whole.

The LMM is a *market model*, as it directly models the forward rates. The model assumes that the forward rates, $L_1, \ldots, L_n$, each satisfy the following SDE

$$dL_i(t) = L_i(t) \sum_{k=q+1}^{i} \frac{\delta_k L_k(t)}{1 + \delta_k L_k(t)} \sigma_i(t)\sigma_k(t)\rho_{ik}dt + L_i(t)\sigma_i(t)dW_i^B(t), \quad q < i,$$

under the appropriate pricing measure.

The only unknown is the volatility function, σ, and various specifications are used in practice. The goal of this dissertation is to fit and study the performance of σ that is specificed by a neural network, i.e.

$$\sigma(t) \approx \sigma^{NN}(t)$$

where $\sigma^{NN}(t)$ represents the neural network approximation of the real underlying volatility.

In order to test the viability of this model we will also fit another volatility model, which is common in the financial industry, and compare the hedging performance of the two models. This will require us to use a few hedging techniques and compare each of them in turn.

The hedging test in theory is usually conducted on multiple simulated sample paths or on a daily profit and loss scale with one observed real world path. We would like to circumvent this issue by exploring data generation options which seek to generate sample paths from the same distribution as the observed path.

In this dissertation, we will introduce a new formulation of variational auto-encoders in order to generate the data we require. Our variational auto-encoder is based on data generation principles from elementary probability i.e. finding the inverse cumulative distribution function and using uniform inputs to generate samples from the distribution. Like all autoencoders, the goal is to reduce the dimensionality in the kernel and use this to describe the data features in the generation. Our formulation will use a kernel which transforms the outputs of the encoder into

multi-dimensional uniformly distributed variables, which in turn will learn the cumulative distribution function (in the case of a one dimensional latent space) or the relationship of variables to copula input uniforms (in the case of a multi-dimensional latent space). The decoder will then train to learn the inverse of the encoder and this will then be used to generate data.

1.2 Literature Review

In this section, due to our interest in replacing the deterministic volatility in the LIBOR market model with a suitable neural network model, we will discuss what interest rate models exist and where other applications of neural networks have already been used within the financial industry.

1.2.1 Interest Rate Models

An interest rate model is a mathematical model that tries to explain the future evolution of the interest rate. Most models usually focus on modeling the instantaneous force of interest (r_t), however, the model of most significance for this dissertation will be the LIBOR Market Model which describes the evolution of simple interest rates.

We analyze some of these existing models. A neat summary of most of these models can be found on page 57 of [Brigo and Mercurio, 2006].

Vasicek Model

In [Vasicek, 1977] a general form of the term stucture of interest rates is derived under two main assumptions:

1. r_t follows a diffusion process.
2. r_t is mean-reverting.

The author shows that the expected rate of return in excess of r_t is proportional to its standard deviation. The example used to illustrate the model was:

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

where α, θ and σ are constants with $\alpha > 0$.
Here we have that the instantaneous drift, $\alpha(\theta - r_t)$, represents mean-reversion to it's long-term mean, θ, which is proportional to the deviation of the process from its mean.

Rendleman–Bartter Model

In [Rendleman and Bartter, 1980] a binomial "distribution free" model was introduced that approximated the lognormal dynamics:

$$dr_t = \theta r_t dt + \sigma r_t dW_t$$

where θ and σ are constants. It was noted that actually [Brennan and Schwartz, 1977] used this model. The model was used with $\theta = 0$ to illustrate their theory of pricing bonds. The main advantage of this model was the positivity of the interest rates.

Cox–Ingersoll–Ross Model

[Cox et al., 1985] expressed their interest rate model as a "square-root process":

$$dr_t = \alpha(\theta - r_t)dt + \sigma\sqrt{r_t}dW_t$$

where the model has the same general features of the Vasicek model, however, the diffusion term was proportional to $\sqrt{r_t}$. This model exhibits mean-reversion with a bonus of non-negative interest rates. The authors note that for $\sigma^2 > 2\alpha\theta$ the rates may reach 0 but in the case where $\sigma^2 \leq 2\alpha\theta$ the upward drift is sufficiently large to make the origin inaccessible.

Exponential Vasicek Model

The Exponential Vasicek Model is a slight adjustment to the original Vasicek model where instead it is assumed that:

$$d\ln(r_t) = \alpha(\theta - \ln(r_t))dt + \sigma dW_t$$

Applying Ito's Lemma to the above we can rewrite this as:

$$dr_t = r_t(\alpha(\theta - \ln(r_t)) + \frac{\sigma^2}{2})dt + \sigma r_t dW_t$$

and finally, changing notation, as:

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

where θ, α and σ are constant.
This model is lognormally distributed so that the rates are always positive, additionally, it has mean-reverting properties. This model is thus an adjustment of both the Vasicek model and the Rendleman-Bartter model. A disadvantage of this model, however, is that a simple discount bond price does not even have a closed form solution.

Hull-White Model

In [Hull and White, 1990], both the Vasicek and Cox-Ingersoll-Ross models were extended to include deterministic functions for α, θ and σ. These models were then shown to still be as tractable as the models they were derived from. The actual Hull-White model is deemed to be a more simplified version of these, written as:

$$dr_t = \alpha(\theta_t - r_t)dt + \sigma dW_t.$$

This model comes with the advantage of mean-reversion to a deterministic adjusting mean but comes with the disadvantage of potential negative rates.

Black–Karasinski model

[Black and Karasinski, 1991] take the same approach as Hull and White but adjust the Exponential Vasicek Model with deterministic functions for α, θ and σ. In their paper, they note that the model actually comes from a model known the Black-Derman-Toy one-factor model, however, we shall not go into this detail. Like the Hull-White model, a simplified model in the form:

$$dr_t = r_t(\theta_t - \alpha\ln(r_t))dt + \sigma r_t dW_t$$

is deemed the Black-Karasinski model. This model extends the Exponential Vasicek Model to have a mean-reverting process to a deterministic mean but unfortunately comes with the intractability of the original model.

LIBOR Market Model

The LIBOR market model was first introduced in [Brace et al., 1997]. In contrast to models that evolve the instantaneous short rate (discussed above), which are not directly observable in the market, the objects modeled using the LIBOR market model are observable quantities (LIBOR forward rates). This makes the LIBOR market model quite popular in practice.

The LIBOR market model can be used to price any instrument whose pay-off can be decomposed into a set of forward rates. Each forward rate has a time dependent volatility and time dependent correlations with the other forward rates. After specifying these volatilities and correlations, the forward rates can be evolved using Monte Carlo simulation.

Let $T_{-1}, T_0, \ldots, T_n$ represent a fixed set of increasing maturities where $T_{-1} := 0$ is the valuation date. We define $L_i(t) := F(t, T_{i-1}, T_i)$ as the simple compounded forward rate resetting at time T_{i-1} with maturity at T_i for each $i = 1, \ldots, n$ i.e. the forward rate which applies for the period $[T_{i-1}, T_i]$. Furthermore, let $P_i(t) := P(t, T_i)$ denote a zero coupon bond, valued at some time t, maturing at T_i and let $\delta_i := T_i - T_{i-1}$.

Under a common measure $\mathbb{P}^*$, the LIBOR Market Model is expected to satisfy the following stochastic differential equation

$$dL_i(t) = L_i(t)\mu_i(t, \mathbf{L}(t))dt + L_i(t)\sigma_i(t)dW^*(t), \quad i = 1, \ldots, n$$

where μ_i is some deterministic function of time and all the LIBOR rates, $\mathbf{L}(t) = [L_1(t), \ldots, L_n(t)]$.

This model will be the basis of the rest of this dissertation. To this end, we examine the LIBOR market model in further detail in its own chapter.

1.2.2 Neural Networks in Finance

The application of neural networks has become a popular endeavor due to the predictive capacity that they provide: given a continuous function, there exists a neural network that approximates it to arbitrary precision. A proof of this remarkable result can be found in [Hornik, 1989].

In finance, neural networks are used to extract the potential true patterns and give insights into the observable data to inform the model holder of decisions to be made. A brief account to some specific cases of neural networks in finance is provided below. The basic structure was derived from [Qi, 1996] and [Swamy et al., 1997], however, more texts were consulted.

What is a neural network?

Definition 1.2.1. A *feedforward neural network* having N neurons arranged in a single hidden layer is a function $f^{NN} : \mathbb{R}^d \to \mathbb{R}$ of the form

$$f^{NN}(x) = \sum_{k=1}^{N} \alpha_k \gamma(w_k^T x + b_k)$$

where $w_k, x \in \mathbb{R}^d$, $\alpha_k, b_k \in \mathbb{R}$ and γ is a nonlinear function called the *activation function.* Furthermore, let $\mathcal{N}$ denote the set of all feedforward neural networks.

These special systems of functions are universal approximators, which means that they can approximate any continuous function to an arbitrary precision. The Universal Approximation Theorem for Neural Networks makes this statement precise.

Theorem 1.2.1 (The Universal Approximation Theorem for Neural Networks)**.**
For each $n \in \mathbb{N}$ let $C([0,1]^n)$ denote the set of all continuous functions defined on the cube $[0,1]^n$. Then, the set of all feed-forward neural networks, $\mathcal{N}$, is dense in $(C([0,1]^n), d_\infty)$.

Let's take a look at some applications of neural networks in our finance.

Stock Price Predictions

Neural networks are often designed and trained to extract patterns in historical data to predict future stock price trends which supplements the primitive time series analysis. A neural network can extract the underlying trajectory, furthermore, it can be used to recognise stocks that consistently outperform the market. In [Kozdraj, 2009], it is noted that a non-linear model is beneficial due to the complexity of stock data and that a large set of input series is required to explain the dynamics of a stock, which makes a neural network a suitable model candidate. An older paper, [Kryzanowski et al., 1993], showed that a neural network was able to correctly classify 72% of the positive/negative returns. The neural networks accuracy, however, was below this level when trying to predict a 3-state outcome.

Option Pricing

According to [Qi, 1996], a well-known application of neural networks to option pricing was done by [Hutchinson et al., 1994]. In this study, the authors noted that given the flexibility of neural networks and their power to approximate complex nonlinear relations, derivative securities is a natural application even when they have closed form solutions. Given a two year training set of Black-Scholes stock paths, the network was able to learn the pricing formulae, furthermore, when applied to the S&P500 from 1987 to 1991 future options the network outperformed the Black-Scholes model in both pricing and delta hedging. A newer study by [Robbertze, 2019] uses recurrent neural networks to price and hedge a standard call option produced by the Black-Scholes model and then the Black-Scholes-Merton model and it is shown that when the stock process includes more realistic assumptions, like a jump, the network can outperform the standard Black-Scholes model.

Hedging

A recent study by [Buehler et al., 2018] developed the ideas of hedging via neural networks, specifically using reinforcement learning. In this paper, the authors noted that the modern quantitative finance has developed a wide variety of tools used to handle derivative pricing and hedging in a complete market setting. It, however, has not been successful at creating a scalable approach under more realistic assumptions. Many instances of risk management in reality require manual input from traders who understand the issues around existing derivative tools.

The key advantages of the reinforcement learning approach over existing techniques were summarised as follows:

1. Scalability. No existing method is yet available to assess the impact of market frictions with reasonable computational effort.

2. Model independence. The choice of market dynamics is removed from the architecture of the hedging engine.

The conclusions of this paper were that using reinforcement learning in a market with frictions resulted in better performance and that learning the optimal hedge for a portfolio was faster than for a single instrument.

In [Robbertze, 2019] a similar method was considered but using an LSTM-like recurrent neural network. This was attempted to try and reduce the computational efforts significantly, however, it was never tested on real stock price data. The results from the simulated data did seem to imply that the technique was worth pursuing further.

Bond Rating Predictions

In [Moody and Utans, 1994], a study was conducted in predicting corporate bond ratings via neural networks. The main task here was to display selection procedures of neural network architectures, however, a case study was provided at the end toward bond rating predictions. In this study, they compared a two layer neural network to a standard linear regression model, and found that the neural network was far superior. In [Tan, 2000], the author uses self-organising maps to cluster and predict the credit ratings of companies via their financial statements. The final results showed that it was possible to predict credit ratings, but only to a certain extent. Even though the model was not suitable to classify the companies very accurately, it gave a good insight into the credit rating procedure. Thus, using cluster analysis could help predict the change of ratings and also give insight to whether this will have a positive/negative effect on the rating.

Bankruptcy Prediction

The standard tool for bankruptcy prediction is that of general linear models with a logit link function. This technique suggests that it is natural to apply a neural network with bounded activation functions, the added potential being the networks ability to work with more complicated data structures. [O'Leary, 1998] explored the use of neural networks in bankruptcy predictions and found that although neural networks perform as well as classical techniques, these other techniques seemed to outperform them overtime. In [du Jardin, 2010], the author visits multiple techniques for predicting bankrupty, and concludes that a logistic model produced up to a 90% accuracy score which was nothing in comparison to a neural networks whopping 94.03%. The author notes that all the experiments that were been done with large samples showed that probability of bankruptcy behaved in a non-linear manner. This was most likely why a neural network approach turned out to be the best.

Exchange Rate Forecasting

[Qi, 1996] notes that exchange rates are notorious for their unpredictability due to linear time series models being the usual tools. In [Parot et al., 2018], the authors note that econometric models have been another widely used method for this task but are usually heavily contested as the majority are linear and work under assumptions that restrict them. Exchange rates, like many other financial data, have a high nonlinear complexity. [Galeshchuk, 2015] explores the use of neural networks in exchange rate predictions with a data set that contains the following exchange rates: USD/EUR, JPN/USD and USD/GBP. The author concludes that the neural network can be applied in practical systems to predict the exchange rates one step ahead whether steps are daily, monthly or quarterly.

Fraud Detection

[Fanning et al., 1995] speaks about the ever increasing need for better models in fraud detection as it is an important issue facing auditing companies. Herewith, they analyze a neural network approach to such a problem. In the paper, they show that the neural network can distinguish between fraudulent and non-fraudulent companies much better than a logit model from another paper. They also note that fraud is a multi-dimensional issue and that auditors cannot just focus on one aspect or predictor when developing their audit plans. [Hamdan et al., 2015] apply cluster

analysis to the problem of identifying illegal and fraud transactions. The proposed model which was a mixture of K-MEANS, DBSCAN and AGGLOMERATIVE clustering algorithms predicted 68.75% of fraudulent transactions with online dynamic modeling and 81.25% in offline mode. These predictions may not seem highly accurate but they are competitive with other algorithms in this area. The models most importantly do provide insight into the data and the important partitioning features can be used to investigate further.

1.3 Dissertation Outline

The remainder of this dissertation is structured in the following way.

In Chapter 2, we begin with a brief introduction to trading in continuous time. This is used to set up the theory of pricing and hedging in continuous time and prepares us to introduce the LIBOR Market Model.

In Chapter 3, we will study the LIBOR Market Model in further detail. We begin with the model description and then address financial instruments pertaining to discrete interest rate tenors such as Forward Rate Agreements and Swaptions. We then outline the idea of volatility structures prior to the calibration of these models to Swaptions.

In Chapter 4, we provide a potential solution to the problem surrounding most model testing procedures, the lack of sample paths. We do this by introducing a new type of variational auto-encoder for data generation which learns both the copula and inverse copula.

In Chapter 5, we explore the main problem. We fit multiple volatility models and test each via the pricing and hedging of interest rate derivatives.

Lastly, in the final chapter, we conclude the dissertation with a summary of our findings throughout, some pitfalls of our model and potential extensions.

Chapter 2

Preliminaries

In this chapter, we will explore the mathematics needed to develop the LIBOR Market Model as well as the theory behind the pricing and hedging of options.

2.1 Stochastic Calculus

The format and content for this section was mainly derived from [Mavuso, 2018], [Mavuso, 2019c], and [Ouwehand, 2020].

2.1.1 Preliminary Definitions

Consider a time horizon $\mathbb{I} \subseteq [0, \infty]$. Let $(\Omega, \mathcal{F}, \mathbb{F}, \mathbb{P})$ be a filtered probability space which consists of a sample space Ω of possible outcomes, a σ-algebra $\mathcal{F}$ of measurable outcomes from subsets of Ω, a filtration $\mathbb{F} := \{\mathcal{F}_t \subseteq \mathcal{F} : t \in \mathbb{I}\}$ of increasing sub-σ-algebras of $\mathcal{F}$ which could be thought of as the knowledge obtained at and prior to some time t which has been gained over time and lastly a probability measure $\mathbb{P} : \mathcal{F} \to [0, 1]$ which assigns a probability to each set within the σ-algebra. We will assume for the rest of this dissertation that $\mathcal{F}_0$ is trivial ($\mathcal{F}_0 = \{\emptyset, \Omega\}$), i.e. at time 0 we have no information.

This subsection will consist of definitions that the major theory in the sections to follow will be need.

Definition 2.1.1. Let $X : \Omega \to \mathbb{R}$ be a function and let $\mathcal{B}(\mathbb{R})$ represent the borel σ-algebra generated by $\mathbb{R}$. We say that X is a random variable if $X^{-1}(B) \in \mathcal{F}$ for all $B \in \mathcal{B}(\mathbb{R})$. We denote this measurability criteria via $X \in m\mathcal{F}$.

A stochastic process is an indexed collection of random variables $\{X_t : t \in \mathbb{I}\}$ on the same probability space $(\Omega, \mathcal{F}, \mathbb{P})$. We call a stochastic process adapted if X_t is $\mathcal{F}_t$-measurable ($X_t \in m\mathcal{F}_t$) for all t.

Definition 2.1.2. Let τ be a random variable. We say that τ is a *stopping time* if $\{\tau \leq t\} \in \mathcal{F}_t$ for all $t \in \mathbb{I}$.

Definition 2.1.3. We say that a stochastic process is left continuous if for fixed ω the left limit exists with respect to t, i.e.

$$X_t(\omega) = X_{t^-}(\omega) = \lim_{s \to t^-} X_s(\omega)$$

Furthermore, we define the predictable σ-algebra, $\mathcal{P}$, to be the smallest σ-algebra that makes all left continuous and adapted stochastic processes measurable and call a stochastic process predictable if it is $\mathcal{P}$-measurable.

Definition 2.1.4. We define the expectation of a random variable X, $\mathbb{E}[X]$, as the following integral

$$\mathbb{E}[X] = \int_\Omega X d\mathbb{P}$$

and the conditional expectation of X given a sub-σ-algebra $\mathcal{G} \subseteq \mathcal{F}$, $\mathbb{E}[X|\mathcal{G}]$, as the almost surely unique $\mathcal{G}$-measurable random variable which satisfies the partial averaging property

$$\int_G X d\mathbb{P} = \int_G \mathbb{E}[X|\mathcal{G}] d\mathbb{P}$$

for all $G \in \mathcal{G}$.

We now introduce the most important continuous-time stochastic process: *Brownian motion*. A Brownian motion plays a similar role played by a random walk in discrete-time stochastic process, in fact, the existence of a Brownian motion can be proven via limits of a sequence of simple random walks as the step sizes get smaller.

Definition 2.1.5. Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space. A stochastic process $W = \{W_t : t \geq 0\}$ is called a *Brownian motion* or *standard Brownian motion* if it satisfies the following:

1. $W_0 = 0$ $\mathbb{P}$-a.s.
2. W has continuous sample paths.
3. W has independent increments; i.e., for each

   $$0 < t_0 < t_1 < t_2 < \ldots < t_{n-1} < t_n,$$

   the random variables

   $$W_{t_0} - W_0, W_{t_1} - W_{t_0}, W_{t_2} - W_{t_1}, \ldots, W_{t_n} - W_{t_{n-1}}$$

   are independent.
4. W has normally distributed, stationary increments: $W_t - W_s \sim N(0, t-s)$ for $s < t$.

If $(\Omega, \mathcal{F}, \mathbb{F}, \mathbb{P})$ is a filtered probability space and W is adapted, then we call
$W = \{W_t : t \geq 0\}$ a Brownian motion with respect to $\mathbb{F}$ if it satisfies 1 - 4 and the stronger condition that

5. For $0 \leq s < t$, $W_t - W_s$ is independent of $\mathcal{F}_s$.

Definition 2.1.6. We say that a stochastic process $X = \{X_t : t \geq 0\}$ is a *martingale* if X is $\mathbb{F}$-adapted, integrable and for $s \leq t$,

$$\mathbb{E}(X_t|\mathcal{F}_s) = X_s.$$

For a submartingale, the $\geq$ inequality replaces the above equality.

It is well-known and easy to prove that Brownian Motion is a martingale. In fact, Brownian Motion is a square integrable martingale which brings us to our next definitions.

Let X be a square integrable martingale; i.e., $\mathbb{E}(X_t^2) < \infty$ for all $t \geq 0$. From Jensen's inequality, the process X^2 is a non-negative submartingale, hence it is of a special class that has a unique decomposition. The Doob-Meyer decomposition states that there exists a unique martingale M and a unique predictable increasing process A, both null at zero, such that

$$X^2 = X_0^2 + M + A.$$

The process A is called the *predictable quadratic variation* of the square integrable martingale X and is denoted by $\langle X \rangle$. That is, $\langle X \rangle$ is the unique predictable increasing process null at zero such that $X^2 - \langle X \rangle$ is a martingale.

Futhermore, let X and Y be square integrable martingales. We define the *predictable covariation process* of X and Y via polarization as:

$$\langle X, Y \rangle := \frac{1}{4} \left(\langle X + Y \rangle - \langle X - Y \rangle \right).$$

The covariation process behaves like a real-valued inner product:

1. $\langle X, X \rangle \geq 0$
2. $\langle X, Y \rangle = \langle Y, X \rangle$
3. $\langle aX, Y \rangle = a \langle X, Y \rangle$
4. $\langle X, Y + Z \rangle = \langle X, Y \rangle + \langle X, Z \rangle$

Note that $\langle X, X \rangle = \langle X \rangle$.

It is easy to see that $\langle X, Y \rangle$ is the unique predictable process null at zero such that $XY - \langle X, Y \rangle$ is a martingale. Also, XY is a martingale if and only if $\langle X, Y \rangle = 0$, and in that case, we say that X and Y are *strongly orthogonal*.

2.1.2 The Ito Integral

In this subsection, we introduce the *stochastic integral* of a stochastic process φ with respect to another process X. We will denote this by

$$(\varphi \bullet X) \text{ or } \int_0^{\cdot} \varphi_s \, dX_s.$$

We call X the *integrator* and φ the *integrand*.

Let us first define this integral for *simple processes*.

Definition 2.1.7. We say that φ is a *simple* process if we can write it as

$$\varphi_t = \sum_{k=1}^{n} H_k I_{(\tau_{k-1}, \tau_k]}(t),$$

where $0 = \tau_0 < \tau_1 < \ldots < \tau_n$ are stopping times and each H_k is bounded and $\mathcal{F}_{\tau_{k-1}}$-measurable for each k. We still denote the class of all simple processes by $\mathbb{S}$.

Definition 2.1.8. For a simple process φ, we define the *stochastic integral* of φ with respect to M as the stochastic process $(\varphi \bullet M) = \{(\varphi \bullet M)_t : 0 \leq t \leq T\}$ defined by

$$(\varphi \bullet M)_t := \sum_{k=1}^{n} H_k \left(M_{t \wedge \tau_k} - M_{t \wedge \tau_{k-1}} \right).$$

We can extend this definition to a larger class of processes. Define

$$L^2(M) := \{\varphi : \varphi \text{ is predictable and } ||\varphi||_M < \infty\},$$

where

$$||\varphi||_M := \left(\mathbb{E}\left(\int_0^T \varphi_s^2 \, d\langle M\rangle_s\right)\right)^{\frac{1}{2}}.$$

It can be shown that each $\mathbb{S} \subseteq L^2(M)$ and for each $\varphi \in L^2(M)$, there exists a sequence (φ^n) in $\mathbb{S}$ such that $||\varphi^n - \varphi||_M \to 0$ as $n \to \infty$. We then define $(\varphi \bullet M)$ as an appropriate limit of the sequence $(\varphi^n \bullet M)$, and this definition is independent of the chosen sequence. For more details, see [Kunita and Watanabe, 1967].

Remark 2.1.1. We will sometimes denote the stochastic integral by

$$\int_0^t \varphi_s \, dM_s.$$

The stochastic integral satisfies the following (for M a square integrable martingale and $\varphi \in L^2(M)$):

1. $\varphi \mapsto (\varphi \bullet M)$ is linear
2. $(\varphi \bullet M)$ is a square integrable martingale with

$$\langle(\varphi \bullet M)\rangle_t = \left(\varphi^2 \bullet \langle M\rangle\right)_t,$$

   which can also be written as

$$\left\langle \int_0^{\cdot} \varphi_s \, dM_s \right\rangle_t = \int_0^t \varphi_s^2 \, d\langle M\rangle_s.$$

3. Ito isometry

$$\mathbb{E}\left(\left(\int_0^t \varphi_s \, dM_s\right)^2\right) = \mathbb{E}\left(\int_0^t \varphi_s^2 \, d\langle M\rangle_s\right).$$

2.1.3 Ito Diffusion Processes

We now shift our attention to a specific class of stochastic processes: *Ito Diffusion Processes.*

Definition 2.1.9. An m-dimensional Brownian motion $W = (W^1, \ldots, W^m)$ is a process such that each W^i is a Brownian motion process and $[W^i, W^j]_t = \rho_{ij} t$ for every t.

Likewise we will define a d-dimensional *Ito process* $X = \left(X^1, \ldots, X^d\right)$ as a process such that for each $i = 1, \ldots, d$,

$$dX_t^i = \mu_t^i dt + \sum_{j=1}^m \sigma_t^{ij} \, dW_t^j$$

for some processes μ^i and σ^{ij}. We will sometimes write this in matrix/vector notation as

$$dX_t = \mu_t dt + \sigma_t dW_t,$$

where $\mu = \left(\mu^1, \ldots, \mu^d\right)$ and $\sigma = (\sigma^{ij})$ is a $d \times m$ matrix. The process μ is called the *drift* term and σ is called the *diffusion* term of X. Note that if $\sigma \in L^2(W)$, then X is a martingale if and only if $\mu \equiv 0$; i.e, if and only if X is driftless.

Definition 2.1.10. If X is an Ito process with differential

$$dX_t = \mu_t dt + \sigma_t dW_t,$$

we will define the integral of a stochastic process φ with respect to X as

$$\int_0^t \varphi_s \, dX_s := \int_0^t \varphi_s \mu_s \, ds + \int_0^t \varphi_s \sigma_s \, dW_s,$$

provided the integrals on the right hand side exist.

One of the most important results in stochastic calculus is *Ito's Lemma*. Before we state it, we need to introduce the quadratic variation of a stochastic process.

Definition 2.1.11. Let X and Y be stochastic processes. We define the *quadratic covariation* of X and Y as the stochastic process $[X, Y]$ defined by

$$[X, Y]_t := \lim_{||P|| \to 0} \sum_{i=1}^{n} \left(X_{t_i} - X_{t_{i-1}} \right) \left(Y_{t_i} - Y_{t_{i-1}} \right),$$

where $P = \{t_0, t_1, \ldots, t_n\}$ is a partition of $[0, t]$, provided the limit exists (in probability). We call $[X] := [X, X]$ the *quadratic variation* of X.

Example 2.1.1. If W is a Brownian motion, then $[W]_t = \langle W \rangle_t = t$.

Remark 2.1.2. In general, for any continuous martingale M, $[M] = \langle M \rangle$, that is why $[\cdot]$ and $\langle \cdot \rangle$ are often used interchangeably (in the case of a continuous martingale).

The quadratic variation, $[\cdot]$, behaves in a similar way to the predictable quadratic variation, $\langle \cdot \rangle$. In general, the quadratic variation of a process of finite variation is zero.

We can now state Ito's lemma:

Theorem 2.1.1 (Ito's Lemma)**.** Let X be a d-dimensional Ito process and $f : \mathbb{R}^{d+1} \to \mathbb{R}$ be a function that is twice continuously differentiable. Then the process Y defined by $Y_t := f(t, X_t)$ is also an Ito process and

$$dY_t = \frac{\partial f}{\partial t} dt + \sum_{i=1}^{d} \frac{\partial f}{\partial x_i} dX_t^i + \frac{1}{2} \sum_{i=1}^{d} \sum_{j=1}^{d} \frac{\partial^2 f}{\partial x_i \partial x_j} d \left[X^i, X^j \right]_t$$

Ito's Lemma is used to 'solve' stochastic differential equations via substitution.

We end this subsection by mentioning that the same construction of the Ito integral in the previous subsection can be carried out in the same manner using the quadratic variation $[M]$ in place of $\langle M \rangle$.

2.1.4 Martingale Representation

In this subsection, we turn to a result that is of great importance in hedging derivative securities.

Let W be a Brownian motion on $(\Omega, \mathcal{F}, \mathbb{F}, \mathbb{P})$. We know that if $\varphi \in L^2(W)$, then the process X such that $dX_t = \varphi_t dW_t$ is a martingale. The Martingale Representation Theorem (MRT) says that, under certain conditions, the converse is also true; that is, all martingales are just stochastic integrals with respect to W.

Theorem 2.1.2 (MRT)**.** Let M be a martingale that is adapted to the natural filtration of $W = (W^1, \ldots, W^m)$. Then there exists predictable processes φ^j, $j = 1, \ldots, m$, such that

$$M_t = M_0 + \sum_{j=1}^{m} \int_0^t \varphi_s^j \, dW_s^j = M_0 + \int_0^t \varphi_s \, dW_s, \quad \text{for every } t.$$

From the above, we receive an interesting result which will be of utmost importance in the section on trading.

Theorem 2.1.3. Let H be an $\mathcal{F}_T^W$-measurable random variable with $\mathbb{E}(|H|) < \infty$. Then there exists a predictable process φ such that

$$H = \mathbb{E}(H) + \int_0^T \varphi_s \, dW_s.$$

If $\mathbb{E}(H^2) < \infty$ then $\varphi \in L^2(W)$.

The proof just applies the MRT to $M_t = \mathbb{E}(H|\mathcal{F}_t)$ (remember that we assume that $\mathcal{F}_0$ is trivial).

2.2 Trading in Continuous Time

This section has plenty of parallels to trading in discrete-time which was examined in detail in [Robbertze, 2019].[1] The format and content in this section mimics that found in [Mavuso, 2019a].

2.2.1 Introduction to Continuous Trading

We begin the section with a brief account of some important concepts and notation of continuous time trading.

Let $T > 0$ represent the time-horizon and assume that the trading dates are in the interval $[0, T]$.

Let $S = \left(S^0, S^1, \ldots, S^d\right)$ be a stochastic process that represents the prices of the $d+1$ assets, i.e. for each $i = 0, 1, \ldots, d$, $S^i = \{S_t^i : t \in [0, T]\}$ is a continuous-time stochastic process.

Let $(\Omega, \mathcal{F}, \mathbb{F}, \mathbb{P})$ be the real-world filtered probability space and assume S is adapted to $\mathbb{F}$, and that S is a $(d+1)$-dimensional stochastic process such that the Ito integral with respect to this process is defined. We then call the tuple, $((\Omega, \mathcal{F}, \mathbb{F}, \mathbb{P}), S)$, a *market*.

In general we assume that the components of S are positive and that $\mathcal{F}_0$ contains only trivial events at initialization since the price process is only known at this time. Furthermore, without loss of generality, we assume $S^0 > 0$ and express the prices of the other d assets in units of S^0. In this case, S^0 is called the *numeraire*.

Definition 2.2.1. The *discounted prices*, $X = \left(X^0, X^1, \ldots, X^d\right)$, are:

$$X^i := \frac{S^i}{S^0}, \ i = 0, 1, \ldots, d$$

From this point onward, we only work with the discounted prices and note that

$$X_t^0 = 1 \ \ \forall t \in [0, T].$$

[1] Request a copy: yuri.robbertze@gmail.com

Definition 2.2.2. A *trading strategy* is a predictable X-integrable process $\varphi = \left(\varphi^0, \ldots, \varphi^d\right)$, where φ_t^i denotes the units of X^i held at time t, and we define $\Theta := \{\varphi : \varphi \text{ is a trading strategy}\}$.

Definition 2.2.3. The *value* of a trading strategy φ is the stochastic process $V(\varphi) = \{V_t(\varphi) : t \in [0,T]\}$ defined by:

$$V_t(\varphi) := \varphi_t X_t = \varphi_t \cdot X_t = \sum_{i=0}^{d} \varphi_t^i X_t^i, \quad t \geq 0.$$

i.e. the dot product of the strategy at the time with the asset values.

Definition 2.2.4. The *gains process* is the stochastic process $G(\varphi) = \{G_t(\varphi) : t \in [0,T]\}$ defined by:

$$G_0(\varphi) := 0, \quad G_t(\varphi) := (\varphi \bullet X)_t = \int_0^t \varphi_s dX_s, \quad t > 0.$$

A *self-financing* portfolio is that the value of a strategy only changes due to the gains process i.e. there are no injections into or leaks from the funds at any point in time.

Definition 2.2.5. A self-financing trading strategy is a strategy, φ, whereby:

$$V_t(\varphi) = V_0(\varphi) + G_t(\varphi), \quad \forall t > 0.$$

When a strategy φ is self-financing, we often write it as a pair (v_0, φ) where v_0 refers to the value of the trading strategy at time 0, i.e. $v_0 := V_0(\varphi)$.

A trading strategy should mimick real life situations, thus, we should consider defining some useful bounds.

Remark 2.2.1. Let $((\Omega, \mathcal{F}, \mathbb{F}, \mathbb{P}), X)$ be a market. Suppose there exists a pair of assets A, namely $A = (A^1, A^2)$, such that at some t their prices satisfy

$$A_t^1 = \alpha A_t^2, \quad \alpha > 0$$

in their current state.
We can sell αN of the first asset and purchase N of the second asset with no additional cost to the portfolio. This can occur $\forall N \in \mathbb{N}$, thus φ would be unbounded. As this is unreasonable, we ignore situations like this and claim some bound exists, once again enforcing a bounded strategy.

The bound that is usually enforced can be defined as follows:

$$G_t(\varphi) \geq -C$$

for some $C \in \mathbb{R}^+$. This bound is usually referred to as the *finite credit line* as it states there is some bound on the amount of money one can borrow from the bank. We call such strategies *admissible*.

From now onwards, we will assume that all trading strategies are self-financing and admissible.

2.2.2 Arbitrage

In this subsection we introduce equivalent martingale measures and the notion of arbitrage in continuous time.

We continue to work on the filtered probability space $(\Omega, \mathcal{F}, \mathbb{F}, \mathbb{P})$ and the assets $X = \left(X^1, \ldots, X^d\right)$ where X is a d-dimensional process that satisfies the conditions in the previous subsection.

Definition 2.2.6. A probability measure $\mathbb{P}^*$ on $(\Omega, \mathcal{F})$ is called an *equivalent martingale measure* for X (EMM) if X is a $\mathbb{P}^*$ martingale. Likewise, we say that $\mathbb{P}^*$ is an *equivalent local martingale measure* for X (ELMM) if X is a $\mathbb{P}^*$ local martingale. Furthermore, define

$$P_{\text{loc}} := \{\mathbb{P}^* : \mathbb{P}^* \text{ is an ELMM for } X\}$$

We will use these ELMMs to characterize markets without arbitrage and to price derivative contracts.

Definition 2.2.7. A trading strategy φ is an *arbitrage strategy/opportunity* if:

$$V_0(\varphi) = 0, \quad \mathbb{P}(V_T(\varphi) \geq 0) = 1, \text{ and } \mathbb{P}(V_T(\varphi) > 0) > 0.$$

Which is to say that an arbitrage strategy is a strategy that starts with a no cost ($V_0(\varphi) = 0$) and ends with a non-negative value ($\mathbb{P}(V_T(\varphi) \geq 0) = 1$), with the possibility of making a gain ($\mathbb{P}(V_T(\varphi) > 0) > 0$).

Definition 2.2.8. We say that a market model is *arbitrage-free* or that the model satisfies the *no arbitrage* (NA) condition if under the model no arbitrage strategies/opportunities exist.

In discrete time, the *Fundamental Theorem of Asset Pricing I* (FTAP I) shows that the converse of the above remark is also true, however, in continuous time we need to strengthen the notion of no arbitrage in order to obtain the reverse implication.

Definition 2.2.9. We say that a sequence of (admissible and self-financing) strategies (v_0^k, φ^k) admits a *free lunch with vanishing risk* if

1. $v_0^k = 0$
2. $V_T((v_0^k, \varphi^k)) \geq -\frac{1}{k}$ for every k
3. There exists $\epsilon > 0$ and $\delta > 0$ such that $\mathbb{P}\left(V_T((v_0^k, \varphi^k)) > \delta\right) > \epsilon$ for each k.

Definition 2.2.10. We say that a market satisfies *no free lunch with vanishing risk* (NFLVR) if there are no sequences of strategies as above.

Let us take a look at one of the most important theorems of mathematical finance.

Theorem 2.2.1 (FTAP I)**.** For a financial market with non-negative assets, the following are equivalent:

1. The market satisfies the NFLVR condition
2. $|P_{\text{loc}}| \neq 0$.

i.e. by proving that a local martingale measure exists, we know that the market satisfies the NFLVR condition.

2.2.3 Pricing and Completeness

We now consider the pricing of *contingent claims* H, which are simply $\mathcal{F}$-measurable random variables.

Definition 2.2.11. Let H be a contingent claim. A strategy (v_0, φ) is said to *replicate* H if $V_T((v_0, \varphi)) = H$ $\mathbb{P}$-a.s.. If such a replicating strategy exists, we say that H is *replicable* or *attainable*.

Definition 2.2.12. The market $((\Omega, \mathcal{F}, \mathbb{F}, \mathbb{P}), X)$ is said to be *complete* if every bounded random variable H is attainable.

Definition 2.2.13. A (d-dimensional) stochastic process X is said to have the *predictable representation property* (PRP) with respect to a measure $\mathbb{P}^*$ if every $\mathbb{P}^*$ local martingale M can be written as

$$M_t = M_0 + \int_0^t \varphi_s \, dX_s$$

for some predictable X-integrable process φ.

At this point, we can now make use of the above definitions to explore the second fundamental theorem, which links completeness to the uniqueness of an ELMM and the PRP property of X. This theorem is called *The Fundamental Theorem of Asset Pricing* II (FTAP II).

Theorem 2.2.2 (FTAP II)**.** Let $((\Omega, \mathcal{F}, \mathbb{F}, \mathbb{P}), X)$ be a market whose components are non-negative and assume that NFLVR holds. Then the following are equivalent:

1. The market is complete
2. $|\mathcal{P}_{\text{loc}}| = 1$
3. X satisfies PRP with respect to at least one ELMM for X

This theorem is of utmost importance since if the market satisfies the NFLVR condition and $\mathbb{P}^*$ is an ELMM for X. Letting H be a bounded and attainable contingent claim with replicating portfolio (v_0, φ). Then

$$\pi(H) = \mathbb{E}^*(H) = v_0,$$

since $G(\varphi)$ is a local martingale. Hence, the starting capital v_0 is the same for every replicating portfolio. We will call $\pi(H)$ the *price* of H.

Girsanov's Theorem does something similar with Brownian motions. We will let $||x||$ denote the Euclidean norm of a vector x in $\mathbb{R}^d$.

Theorem 2.2.3 (Girsanov's Theorem)**.** Let $(\Omega, \mathcal{F}, \mathbb{F}, \mathbb{P})$ be a filtered space and $W = \left(W^1, \ldots, W^d\right)$ be a d-dimensional Brownian motion on this space. Let $\gamma = (\gamma_1, \ldots, \gamma_d)$ be a vector of progressive processes satisfying

$$\mathbb{P}\left(\int_0^T \left(\gamma_s^i\right)^2 \, ds < \infty\right) = 1, \quad i = 1, \ldots, d.$$

Define the continuous local martingale Z by

$$Z_t := \exp\left(\sum_{i=1}^d \int_0^t \gamma_s^i \, dW_s^i - \frac{1}{2} \int_0^t ||\gamma_s||^2 \, ds\right).$$

Assume that Z is a martingale and define a new probability measure $\mathbb{P}^*$ by

$$\frac{d\mathbb{P}^*}{d\mathbb{P}} = Z_T.$$

Then the process $\tilde{W} = \left(\tilde{W}^1, \ldots, \tilde{W}^d\right)$ defined by

$$\tilde{W}_t^i := W_t^i - \int_0^t \gamma_s^i \, ds$$

is a d-dimensional Brownian motion on $(\Omega, \mathcal{F}, \mathbb{F}, \mathbb{P}^*)$.

We will use Girsanov's Theorem to change the numeraire of the LIBOR Market Model which in turn will change the dynamics of the model appropriately, yielding a unique price for interest rate derivatives as well as the existence of a replicating portfolio.

Chapter 3

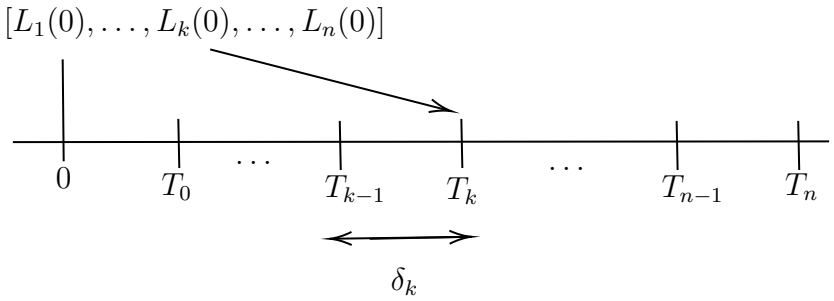
LIBOR Market Model

Up until this point we have only looked at preliminary content as well as some prior works in finance relating to our research. We would now like to go into depth on the LIBOR market model, which is of utmost importance to this dissertation. The following chapter follows largely from content found in [Moodliyar, 2016], [Brace et al., 1997] and [Brigo and Mercurio, 2006].

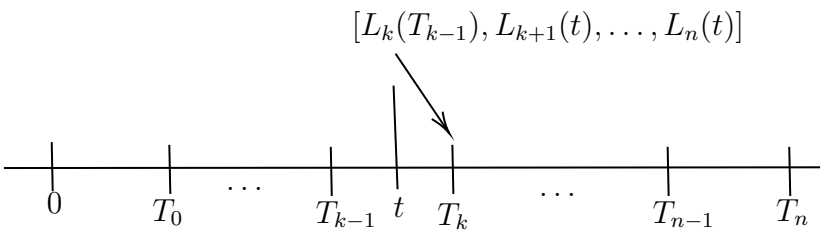
3.1 Market Model Specification

Following the same set-up and notation as subsection 1.2.1. under the "Libor Market Model" heading, we can then illustrate the majority of the notation below via diagrams that were replicated from [Glasserman, 2003] and [Moodliyar, 2016].

At $t = 0$, all the rates are "alive". This is depicted by:



At an arbitrary t between two maturities T_{k-1} and T_k, only rates after and including L_k are "alive" and those before are "dead". This is depicted by:



Remark 3.1.1. Let $L_i(t)$ denote the LIBOR forward rate at time t which applies for the period $[T_{i-1}, T_i]$, then

$$L_i(t) = \frac{1}{\delta_i} \frac{P_{i-1}(t) - P_i(t)}{P_i(t)}, \quad i = 1, \ldots, n$$

Now, consider an asset $H_i(t) := \frac{1}{\delta_i}[P_{i-1}(t) - P_i(t)]$ in a market model which satisfies the NFLVR condition. Using $P_i(t)$ as the numeraire, we get that

$$L_i(t) = \frac{H_i(t)}{P_i(t)} = \mathbb{E}^i\left[\frac{H_i(T)}{P_i(T)}\middle|\mathcal{F}_t\right] = \mathbb{E}^i\left[L_i(T_{i-1})|\mathcal{F}_t\right]$$

where the ELMM associated with the numeraire $P_i(t)$ is $\mathbb{P}^i$ with Brownian motion W_i.

It is clear now that under $\mathbb{P}^i$, we have that the LIBOR forward rate process L_i is a martingale. Therefore, in this dissertation we will assume that the dynamics of L_i take on the form:

$$dL_i(t) := L_i(t)\sigma_i(t)dW_i^i(t), \quad i = 1, \ldots, n$$

where σ_i represents the instantaneous volatility of the i^{th} forward rate independent of the other $n-1$ forward rates and $d\langle W_i, W_j\rangle_t := \rho_{ij}$ is the instantaneous correlation between the brownian motions.

We have now observed that the LIBOR process L_i is a martingale under the forward pricing measure $\mathbb{P}^i$. We also specified a specific model form i.e. Geometric Brownian Motion.

Remark 3.1.2. When performing Monte Carlo simulations, it is crucial that all LIBOR rates $L_1, \ldots, L_n$ are specified under one common pricing measure. We will deal with this in the next section.

3.2 Market Model under a common measure

In a previous section, we observed that the LIBOR process L_i is a martingale under the forward pricing measure $\mathbb{P}^i$. We also specified a specific model form i.e. Geometric Brownian Motion. However, when performing Monte Carlo simulations we need to define the dynamics of all the LIBOR rates under a common pricing measure with a single numeraire. The most commonly used measures are the Terminal and Spot measures. These change of measures will result in the alteration of the drift component of the stochastic differential equation leaving the volatility of the asset unmodified.

Under a common measure $\mathbb{P}^*$, the LIBOR Market Model is expected to satisfy the following stochastic differential equation

$$dL_i(t) = L_i(t)\mu_i(t, \mathbf{L}(t))dt + L_i(t)\sigma_i(t)dW^*(t), \quad i = 1, \ldots, n$$

where μ_i is some deterministic function of time and all the LIBOR rates, $\mathbf{L}(t) = [L_1(t), \ldots, L_n(t)]$.

To relate the measures $\mathbb{P}^i$ and $\mathbb{P}^{i-1}$, we have that

$$\frac{d\mathbb{P}^{i-1}}{d\mathbb{P}^i} = Z_i^{i-1}(T)$$

where

$$Z_i^{i-1}(t) = \frac{P_i(0)}{P_i(t)}\frac{P_{i-1}(t)}{P_{i-1}(0)} = \frac{P_i(0)}{P_{i-1}(0)}[1 + \delta_i L_i(t)]$$

as we change the numeraire between $P_i(t)$ and $P_{i-1}(t)$.

From this, we get that the dynamics of the density process under $\mathbb{P}^i$ is

$$dZ_i^{i-1}(t) = \frac{P_i(0)}{P_{i-1}(0)}\delta_i dL_i(t) = \frac{P_i(0)}{P_{i-1}(0)}\delta_i L_i(t)\sigma_i(t)dW^i(t)$$

or

$$\frac{dZ_i^{i-1}(t)}{Z_i^{i-1}(t)} = \frac{\delta_i L_i(t)\sigma_i(t)}{1+\delta_i L_i(t)} dW^i(t)$$

Thus, from Girsanov's Theorem we get

$$dW^i(t) = \frac{\delta_i L_i(t)\sigma_i(t)}{1+\delta_i L_i(t)} dt + dW^{i-1}(t)$$

Now declaring that we will model the LIBOR rates with respect to the terminal measure $\mathbb{P}^n$ and applying the above iteratively, we obtain

$$dW^i(t) = -\sum_{k=i+1}^{n} \frac{\delta_k L_k(t)\sigma_k(t)}{1+\delta_k L_k(t)} dt + dW^n(t)$$

The dynamics of the LIBOR rates under the common measure $\mathbb{P}^n$, now take on the form

$$dL_i(t) = -L_i(t) \sum_{k=i+1}^{n} \frac{\delta_k L_k(t)}{1+\delta_k L_k(t)} \sigma_i(t)\sigma_k(t)\rho_{ik} dt + L_i(t)\sigma_i(t) dW_i^n(t)$$

for $i = 1, \ldots, n$ and $d\langle W_i, W_j \rangle_t := \rho_{ij}$.

Denoting the spot measure by $\mathbb{P}^B$, we can construct a self-financing portfolio that represents a bank account numeraire which is only rebalanced at each of the discrete tenor maturities. This is achieved by rolling over the shortest remaining maturity bond. The strategy is as follows: At T_0 we invest one unit of currency into a T_1 bond and at any further maturity date T_{k-1}, $k = 2, \ldots, n$, we sell the T_{k-1} bond and we invest all our earnings into a T_k bond. This strategy is repeated until T_n. The value of this discretely rebalanced bank account is given by:

$$B(t) = P_q(t)B(T_q) \quad \text{for} \quad T_{q-1} \leq t < T_q$$

where

$$B(T_q) = \prod_{i=1}^{q} (1 + \delta_i L_i(T_{i-1}))$$

The only stochastic part of this numeraire is the discount bond $P_q(t)$. Hence, the dynamics under $\mathbb{P}^q$ is the focus of the solution. Once again, we can take an iterative approach to the problem as follows:

$$dW^i(t) = \sum_{k=q+1}^{i} \frac{\delta_k L_k(t)\sigma_k(t)}{1+\delta_k L_k(t)} dt + dW^q(t)$$

Therefore under the Spot measure $\mathbb{P}^B$, the dynamics of L_i now take on the form

$$dL_i(t) = L_i(t) \sum_{k=q+1}^{i} \frac{\delta_k L_k(t)}{1+\delta_k L_k(t)} \sigma_i(t)\sigma_k(t)\rho_{ik} dt + L_i(t)\sigma_i(t) dW_i^B(t)$$

This formulation will be the one used in our simulations for both pricing and hedging. Now, let us have a look at some common market instruments held on LIBOR rates.

3.3 Market Instruments

The pricing of caps, floors and swaptions is generally done with the Black-76 formula in financial markets. The Black-76 formula assumes, among other things, that the underlying asset is log-normally distributed at its maturity under a specific risk-neutral measure. These simplifying assumptions allow one to use a standardised formula for pricing options. We now take a look at a few options and their pricing formulas under Black-76. In what follows, Φ represents the CDF of the standard normal distribution.

Definition 3.3.1 (Cap)**.**
A cap is a contract which at each of the tenors $T_1, \ldots, T_n$ gives the holder the amount $X_i = \delta_i \max\{L_i(T_{i-1}) - K, 0\}$ at each T_i for $i = 1, \ldots, n$ where K is the cap rate at resettlement dates. We see that a cap is just a portfolio of caplets $X_1, \ldots, X_n$ where each caplet X_i is a call option on the underlying spot rate.

Using the LMM, the price for a caplet $X_i = \delta_i \max\{L_i(T_{i-1}) - K, 0\}$ is given by:

$$\text{Caplet}_i^{LMM}(t) = \delta_i P_i(t) \left[L_i(t)\Phi(d_1) - K\Phi(d_2) \right], \quad i = 1, \ldots, n$$

where

$$\begin{aligned} d_1 &= \frac{1}{v_i(t, T_{i-1})} \left[\ln\left(\frac{L_i(t)}{K} \right) + \frac{1}{2} v_i^2(t, T_{i-1}) \right] \\ d_2 &= d_1 - v_i(t, T_{i-1}) \end{aligned}$$

for

$$v_i^2(t, T) = \int_t^T \|\sigma_i(s)\|^2 ds$$

Hence, the price of a cap is the sum of the prices of relevant caplets involved.

An intuitive explanation of a cap is that the holder of the option wants the right but not obligation to swap a fixed rate (K) for a floating rate (L_i) at time T_i. The holder of the option will swap K for L_i if only if the floating rate is greater than the fixed rate.

Similarly to the call options, a portfolio of put options on the underlying spot rate can be established with the interest rate floor. In this situation, the holder will swap the floating rate for the fixed rate if and only if the fixed rate is greater than the floating rate.

Definition 3.3.2 (Floor)**.**
A floor is a contract which at each of the tenors $T_1, \ldots, T_n$ gives the holder the amount $X_i = \delta_i \max\{K - L_i(T_{i-1}), 0\}$ at each T_i for $i = 1, \ldots, n$ where K is the floor rate at resettlement dates.

Using the LMM, the price for a floorlet $X_i = \delta_i \max\{K - L_i(T_{i-1}), 0\}$ is given by:

$$\text{Floorlet}_i^{LMM}(t) = \delta_i P_i(t) \left[K\Phi(-d_2) - L_i(t)\Phi(-d_1) \right], \quad i = 1, \ldots, n$$

where

$$\begin{aligned} d_1 &= \frac{1}{v_i(t, T_{i-1})} \left[\ln\left(\frac{L_i(t)}{K} \right) + \frac{1}{2} v_i^2(t, T_{i-1}) \right] \\ d_2 &= d_1 - v_i(t, T_{i-1}) \end{aligned}$$

The final instrument we consider here is the swaption; however, it is prudent to cover swap instruments to begin since this is the underlying asset. A swap is a contract whereby two parties

agree to exchange different types of cash flows in the future. An interest rate swap, in particular, exchanges a fixed rate of interest for a floating rate of interest at a pre-specified date for a pre-determined amount of time. The holder of a receiver swap with tenor $(T_\beta - T_\alpha)$ will receive the fixed leg and pay the floating leg at dates $T_{\alpha+1}, \ldots, T_\beta$. The payments in a payer swap move in the counter direction.

Definition 3.3.3 (Payer Swap)**.**
The payments in a $T_\alpha \times (T_\beta - T_\alpha)$ *payer swap* are as follows:

- Payments will be made at times $T_{\alpha+1}, \ldots, T_\beta$.
- For each period $[T_i, T_{i+1}]$, $i = \alpha, \ldots, T_{\beta-1}$, the LIBOR rate $L_{i+1}(T_i)$ is fixed at time T_i and the floating leg $\delta_{i+1}L_{i+1}(T_i)$ is received at T_{i+1}.
- For the same period the fixed leg $\delta_{i+1}K$ is paid at T_{i+1}.

The t-net value, $t < T_\alpha$, for the $T_\alpha \times (T_\beta - T_\alpha)$ payer swap is given by

$$\text{Swap}(t) = \sum_{i=\alpha+1}^{\beta} \delta_i P_i(t) L_i(t) - K \sum_{i=\alpha+1}^{\beta} \delta_i P_i(t)$$

which, using the definition of the LIBOR rate, can be simplified as

$$\text{Swap}(t) = P_\alpha(t) - P_\beta(t) - K \sum_{i=\alpha+1}^{\beta} \delta_i P_i(t)$$

Definition 3.3.4 (Swap Rate)**.**
The par or forward swap rate $S_\alpha^\beta(t)$ of the $T_\alpha \times (T_\beta - T_\alpha)$ swap is the value of K for which $\text{Swap}(t) = 0$, i.e.

$$S_\alpha^\beta(t) = \frac{P_\alpha(t) - P_\beta(t)}{\sum_{i=\alpha+1}^{\beta} \delta_i P_i(t)}$$

Now that we have covered swaps, we can consider swaption instruments. *Swaptions* are options that give the holder the right but not obligation to enter into a particular interest rate swap at a pre-specified time in the future. Market practice is to compute swaption prices once again through the use of the Black-76 formula. The prices are then quoted in the market as implied Black volatilities which are typically for par swap rates.

Definition 3.3.5 (Payer Swaption)**.**
A $T_\alpha \times (T_\beta - T_\alpha)$ European payer swaption with strike K is a contract which at the exercise date T_α gives the holder the right but not the obligation to enter into a $T_\alpha \times (T_\beta - T_\alpha)$ swap with a fixed swap rate K.

The payer swaption is a T_α contingent claim defined as

$$\text{Swaption}(T_\alpha) = \max\{\text{Swap}(T_\alpha), 0\} = \sum_{i=\alpha+1}^{\beta} \delta_i P_i(T_\alpha) \max\{S_\alpha^\beta(T_\alpha) - K, 0\}$$

that when expressed under the numeraire $\sum_{i=\alpha+1}^{\beta} \delta_i P_i(t)$ can be considered a call option $S_\alpha^\beta(t)$ with strike K.

The Black-76 payer swaption price for a $T_\alpha \times (T_\beta - T_\alpha)$ swaption is

$$\text{Swaption}_{\alpha,\beta}(t) = \sum_{i=\alpha+1}^{\beta} \delta_i P_i(t) \left[S_\alpha^\beta(t)\Phi(d_1) - K\Phi(d_2) \right]$$

where

$$d_1 = \frac{1}{\hat{\sigma}_{\alpha,\beta}^{swap}\sqrt{T_\alpha - t}} \left[\ln\left(\frac{S_\alpha^\beta(t)}{K}\right) + \frac{1}{2}(\hat{\sigma}_{\alpha,\beta}^{swap})^2(T_\alpha - t) \right]$$
$$d_2 = d_1 - \hat{\sigma}_{\alpha,\beta}^{swap}\sqrt{T_\alpha - t}$$

and the constant $\hat{\sigma}_{\alpha,\beta}^{swap}$ is the implied Black volatility.

This price formula requires the assumption that $S_\alpha^\beta(t)$, under the measure $\mathbb{P}_\alpha^\beta$ corresponding to the numeraire $\sum_{i=\alpha+1}^{\beta} \delta_i P_i(t)$, assumes the following lognormal dynamics

$$dS_\alpha^\beta(t) = \hat{\sigma}_{\alpha,\beta}^{swap} S_\alpha^\beta(t) dW_\alpha^\beta(t)$$

This formulation is unfortunately not consistent with our formulation of the lognormal forward rates, however, prices in the market are generally quoted in terms of implied volatilities and thus this will prove useful in the approximation to be studied later.

3.4 Instantaneous Volatility Structures

In practice, we often have to assume the forward rate volatility follows a specific structure to reduce the number of unknown variables. In this subsection, we shall explore some of these well-known volatility structures. Ultimately, the LMM calibration should display a generally realistic, smooth and qualitatively stable evolution of the term structure. The most prevalent specifications along with our new proposed model for $\sigma_i(t)$ are provided in Table 3.1 below.

Table 3.1: Instantaneous Volatility Specifications

Specification	Description
A	Assume volatility is constant in time $\sigma_i(t) = \sigma_i$ for $0 \le t \le T_{i-1}$
B	Assume volatility is piecewise constant $\sigma_i(t) = \sigma_{ij}$ for $T_{j-1} \le t \le T_j$, $j = 0, \ldots, i-1$
C	Assume volatility depends only on time to maturity $\sigma_i(t) = \eta_{i-j}$ for $T_{j-1} \le t \le T_j$, $j = 0, \ldots, i-1$
D	Assume volatility depends on maturity and time to maturity $\sigma_i(t) = \phi_i \psi_{i-j}$ for $T_{j-1} \le t \le T_j$, $j = 0, \ldots, i-1$
E	Assume volatility follows a functional parameterised form $\sigma_i(t) = \phi_i \left([a(T_{i-1} - t) + d]\, e^{-b(T_{i-1}-t)} + c \right)$
F	(Proposed) Assume volatility can be approximated via a neural network $\sigma_i(t) = \sigma_i^{NN}(t)$ for $0 \le t \le T_{i-1}$

Specification B in Table 3.1 is often referred to as the general piecewise constant parameterisation, under this assumption the instantaneous volatilities can be structured as per Table 3.2.

Table 3.2: General Piecewise Constant Parameterisation

Forward Rate	$(0, T_0]$	$(T_0, T_1]$	$(T_1, T_2]$	$\dots$	$(T_{n-1}, T_n]$
$L_1(t)$	σ_{11}	expired	expired	$\dots$	expired
$L_2(t)$	σ_{21}	σ_{22}	expired	$\dots$	expired
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$L_n(t)$	σ_{n1}	σ_{n2}	σ_{n3}	$\dots$	σ_{nn}

Specification D can be split into a term structure that depends on maturity time (ϕ) and time until maturity (ψ). This deconstruction into the two dependents ensures in parameter reduction compared to specification B. Specification E consists of the same components as D, however, ψ is locally altered parametric form for each maturity with parameter ϕ. Through optimisation techniques, we can solve for the parameters of specifications D, E and F. Solving for the parameters in any formulation usually requires the use of the *Rebonatto approximation* which we will cover in detail within the next section. After covering the relevant approximation, we will derive an algorithm that solves analytically for the parameters of specification B and then use this to explain how we will optimise for the parametric forms E and F.

3.5 Calibration to Swaptions

3.5.1 Rebonatto Approximation

The next subsection will focus on solving for the forward rate volatilities given swaption implied volatilities. In this subsection, we shall explore the Rebonatto approximation which will prove to be a powerful tool in the calibration process.

In the previous section we saw that the Black-76 swaption prices that are derived under the numeraire $\sum_{i=\alpha+1}^{\beta} \delta_i P_i(t)$ require the swap rate $S_\alpha^\beta(t)$ to follow the dynamics

$$dS_\alpha^\beta(t) = \sigma_{\alpha,\beta}^{swap} S_\alpha^\beta(t) dW_\alpha^\beta(t)$$

We also showed that the definition of a swap rate implies that

$$S_\alpha^\beta(t) = \frac{P_\alpha(t) - P_\beta(t)}{\sum_{i=\alpha+1}^{\beta} \delta_i P_i(t)}$$

which can also be written as

$$S_\alpha^\beta(t) = \frac{\sum_{i=\alpha+1}^{\beta} \delta_i P_i(t) L_i(t)}{\sum_{i=\alpha+1}^{\beta} \delta_i P_i(t)} = \sum_{i=\alpha+1}^{\beta} w_i(t) L_i(t)$$

where

$$w_i(t) = \frac{\delta_i P_i(t)}{\sum_{i=\alpha+1}^{\beta} \delta_i P_i(t)}.$$

In order to relate the swaption volatilities with that of the forward rates correlations and volatilities, we apply Itô's formula to obtain

$$\left(\sigma_{\alpha,\beta}^{swap} S_\alpha^\beta\right)^2 = \sum_{j=\alpha+1}^{\beta} \sum_{k=\alpha+1}^{\beta} \frac{\partial S_\alpha^\beta}{\partial L_j} \frac{\partial S_\alpha^\beta}{\partial L_k} \sigma_j L_j \sigma_k L_k \rho_{jk}$$

Now, even though the partial derivatives are rather complicated, a very good approximation for short to medium option lengths can be made via

$$w_j^* = \frac{\partial S_\alpha^\beta}{\partial L_j} \approx w_j$$

We can now write the instantaneous volatility for the swap rate as

$$\sigma_{\alpha,\beta}^{swap}(t)^2 = \sum_{j=\alpha+1}^{\beta} \sum_{k=\alpha+1}^{\beta} \zeta_{jk}(t) \rho_{jk}(t) \sigma_j(t) \sigma_k(t)$$

where

$$\zeta_{jk}(t) = \frac{w_j^*(t) L_j(t) w_k^*(t) L_k(t)}{\left(\sum_{i=\alpha+1}^{\beta} w_i(t) L_i(t)\right)^2}.$$

The implied volatility can be linked to the instantaneous volatility through the following equation

$$(\hat{\sigma}_{\alpha,\beta}^{swap})^2 T_\alpha = \int_0^{T_\alpha} \sigma_{\alpha,\beta}^{swap}(t)^2 dt$$

where T_α is the time the swaption should be exercised.

The problem involving the fact that the lognormal forward rates are incompatible with the lognormal swap rates are of little concern. If L, w and w^* are deterministic and are calculated using the current yield curve we have the following constant weight approximation for the swap rate volatility

$$\sigma_{\alpha,\beta}^{swap}(t)^2 = \sum_{j=\alpha+1}^{\beta} \sum_{k=\alpha+1}^{\beta} \zeta_{jk}^{cwa}(0) \rho_{jk}(t) \sigma_j(t) \sigma_k(t)$$

where

$$\zeta_{jk}^{cwa}(0) = \frac{w_j(0) L_j(0) w_k(0) L_k(0)}{\left(\sum_{i=\alpha+1}^{\beta} w_i(0) L_i(0)\right)^2}.$$

Although more elaborate approximations exist, this approximation will be sufficient for the problems to follow.

3.5.2 Cascade Calibration

The algorithm that we shall use provides an analytical procedure that does not require the use of simulation or optimisation. To derive the algorithm we assume the forward rate volatilities follow the general piecewise constant parameterisation. Furthermore, we need to use the constant weight approximation as well as assume the correlations are constant in time. If one can invert the approximation formula, the precise input swaption prices can be retrieved without any calibration error. The most important feature of the Cascade algorithm is the injective correspondence

between the instantaneous forward rate volatilities and the market swaption volatilities. This correspondence is indicated within Table 3.3 below.

Table 3.3: Swaption and Forward Rate Volatilities in Cascade Calibration

	Length		
Maturity	1 Year	2 Years	3 Years
1 Year	$\hat{\sigma}_{0,1}^{swap}$	$\hat{\sigma}_{0,2}^{swap}$	$\hat{\sigma}_{0,3}^{swap}$
	$\boxed{\sigma_{1,1}}$	$\sigma_{1,1}, \boxed{\sigma_{2,1}}$	$\sigma_{1,1}, \sigma_{2,1}, \boxed{\sigma_{3,1}}$
2 Years	$\hat{\sigma}_{1,1}^{swap}$	$\hat{\sigma}_{1,2}^{swap}$	$\hat{\sigma}_{1,3}^{swap}$
	$\sigma_{2,1}, \boxed{\sigma_{2,2}}$	$\sigma_{2,1}, \sigma_{2,2}, \sigma_{3,1}$	$\sigma_{1,1}, \sigma_{2,1}, \sigma_{3,1}$
		$\boxed{\sigma_{3,2}}$	$\sigma_{3,2}, \boxed{\sigma_{4,1}, \sigma_{4,2}}$
3 Years	$\hat{\sigma}_{2,1}^{swap}$	$\hat{\sigma}_{2,2}^{swap}$	$\hat{\sigma}_{2,3}^{swap}$
	$\sigma_{3,1}, \sigma_{3,2}, \boxed{\sigma_{3,3}}$	$\sigma_{3,1}, \sigma_{3,2}, \sigma_{3,3}$	$\sigma_{3,1}, \sigma_{3,2}, \sigma_{3,3}$
		$\sigma_{4,1}, \sigma_{4,2}, \boxed{\sigma_{4,3}}$	$\sigma_{4,1}, \sigma_{4,2}, \sigma_{4,3}$
			$\boxed{\sigma_{5,1}, \sigma_{5,2}, \sigma_{5,3}}$

Let's begin by rewriting the approximation formula as

$$\begin{aligned}\left(\hat{\sigma}_{\alpha,\beta}^{swap}\right)^2 T_\alpha &= \int_0^{T_\alpha} \sigma_{\alpha,\beta}^{swap}(t)^2 dt \\ &\approx \sum_{j=\alpha+1}^{\beta} \sum_{k=\alpha+1}^{\beta} \zeta_{jk}^{cwa}(0)\rho_{jk} \int_0^{T_\alpha} \sigma_j(t)\sigma_k(t)dt \\ &\approx \sum_{j=\alpha+1}^{\beta} \sum_{k=\alpha+1}^{\beta} \zeta_{jk}^{cwa}(0)\rho_{jk} \sum_{i=0}^{\alpha} \delta_i \sigma_{j,i+1}\sigma_{k,i+1}\end{aligned}$$

Starting with the first market swaption volatility $\hat{\sigma}_{0,1}^{swap}$, we obtain via simplification

$$\left(\hat{\sigma}_{0,1}^{swap}\right)^2 \approx \boxed{\sigma_{1,1}^2}.$$

Moving to the following volatility $\left(\hat{\sigma}_{0,1}^{swap}\right)^2$ we achieve the following quadratic equation

$$\begin{aligned}S_0^2(0)^2 \left(\hat{\sigma}_{0,2}^{swap}\right)^2 \approx w_1(0)^2 L_1(0)^2 \sigma_{1,1}^2 + w_2(0)^2 L_2(0)^2 \boxed{\sigma_{2,1}^2} \\ 2\rho_{1,2} w_1(0) L_1(0) w_2(0) L_2(0) \sigma_{1,1} \boxed{\sigma_{2,1}}.\end{aligned}$$

In general, we can solve for the upper triangular volatilities via a quadratic equation of the form

$$A_{\alpha,\beta}\sigma_{\beta,\alpha+1}^2 + B_{\alpha,\beta}\sigma_{\beta,\alpha+1} + C_{\alpha,\beta} = 0.$$

The constants involved in the quadratic equation can be solved for by expanding the approximation as follows:

$$\begin{aligned}
S_\alpha^\beta(0)^2 \left(\hat{\sigma}_{\alpha,\beta}^{swap}\right)^2 T_\alpha = & \sum_{j=\alpha+1}^{\beta-1} \sum_{k=\alpha+1}^{\beta-1} w_j(0) L_j(0) w_k(0) L_k(0) \rho_{j,k} \sum_{i=0}^{\alpha} \delta_i \sigma_{j,i+1} \sigma_{k,i+1} \\
& + 2 \sum_{j=\alpha+1}^{\beta-1} w_j(0) L_j(0) w_\beta(0) L_\beta(0) \rho_{j,\beta} \sum_{i=0}^{\alpha-1} \delta_i \sigma_{j,i+1} \sigma_{\beta,i+1} \\
& + w_\beta(0)^2 L_\beta(0)^2 \sum_{i=0}^{\alpha-1} \delta_i \sigma_{\beta,i+1}^2 \\
& + 2 \sum_{j=\alpha+1}^{\beta-1} w_j(0) L_j(0) w_\beta(0) L_\beta(0) \rho_{j,\beta} \delta_\alpha \sigma_{j,\alpha+1} \boxed{\sigma_{\beta,\alpha+1}} \\
& + w_\beta(0)^2 L_\beta(0)^2 \delta_\alpha \boxed{\sigma_{\beta,\alpha+1}^2}
\end{aligned}$$

Thus, the constants in

$$A_{\alpha,\beta} \sigma_{\beta,\alpha+1}^2 + B_{\alpha,\beta} \sigma_{\beta,\alpha+1} + C_{\alpha,\beta} = 0$$

are

$$A_{\alpha,\beta} = w_\beta(0)^2 L_\beta(0)^2 \delta_\alpha$$

$$B_{\alpha,\beta} = 2 \sum_{j=\alpha+1}^{\beta-1} w_j(0) L_j(0) w_\beta(0) L_\beta(0) \rho_{j,\beta} \delta_\alpha \sigma_{j,\alpha+1}$$

$$\begin{aligned}
C_{\alpha,\beta} = & \sum_{j=\alpha+1}^{\beta-1} \sum_{k=\alpha+1}^{\beta-1} w_j(0) L_j(0) w_k(0) L_k(0) \rho_{j,k} \sum_{i=0}^{\alpha} \delta_i \sigma_{j,i+1} \sigma_{k,i+1} \\
& + 2 \sum_{j=\alpha+1}^{\beta-1} w_j(0) L_j(0) w_\beta(0) L_\beta(0) \rho_{j,\beta} \sum_{i=0}^{\alpha-1} \delta_i \sigma_{j,i+1} \sigma_{\beta,i+1} \\
& + w_\beta(0)^2 L_\beta(0)^2 \sum_{i=0}^{\alpha-1} \delta_i \sigma_{\beta,i+1}^2 \\
& - S_\alpha^\beta(0)^2 \left(\hat{\sigma}_{\alpha,\beta}^{swap}\right)^2 T_\alpha
\end{aligned}$$

3.5.3 Optimisation of Parametric Forms

In order to discuss the optimisation procedure for specification D, E and F, assume that $\sigma(t)$ takes on one of these forms. For ease of notation, we classify all of these specifications into a class of deterministic functions such that

$$\sigma(t) = (\sigma_1(t), \sigma_2(t), \ldots, \sigma_n(t))$$

where each σ_i satisfies the conditions of the chosen parameterisation.

We then define the *rebonatto volatility* of this specification as

$$\sigma^{reb}_{\alpha,\beta}(T_\alpha) = \sqrt{\frac{1}{T_\alpha}\sum_{j=\alpha+1}^{\beta}\sum_{k=\alpha+1}^{\beta}\zeta^{cwa}_{jk}(0)\rho_{jk}\int_0^{T_\alpha}\sigma_j(s)\sigma_k(s)ds}.$$

which can be simplified depending on the specification in question.

In order to optimise this functional form to fit the swaption implied volatilities we will need an objective function. Let $\mathcal{AB}$ denote the set of all (α, β) such that $\hat{\sigma}^{swap}_{\alpha,\beta}$ exists in our data set.

The objective function is then defined as

$$\mathcal{C}(\theta) = \sum_{(\alpha,\beta)\in\mathcal{AB}}\left(\hat{\sigma}^{swap}_{\alpha,\beta} - \sigma^{reb}_{\alpha,\beta}(T_\alpha)\right)^2$$

where θ is the vector of parameters of $\sigma(t)$.

We are then tasked with finding θ such that

$$\theta = \underset{\theta}{\operatorname{argmin}}\,\mathcal{C}(\theta)$$

which can be done via standard well-known optimisation techniques.

After finding these optimal parameters, they can then be used in the simulation for pricing and hedging of options accordingly.

Remark 3.5.1. If the focus is on pricing and hedging is on long dated maturity products, such as insurance products, the large number of forward rates results in slow optimisation to solve for these parameters.

After discussing all the optimisation procedures available, let's have a look at how we will approach our neural network model.

3.5.4 Neural Network Volatility Structure

In this dissertation, we would like to test whether a neural network volatility model is a viable option, i.e.

$$\sigma(t) \approx \sigma^{NN}(t)$$

where $\sigma^{NN}(t)$ represents the neural network approximation of the real underlying volatility. The neural network volatility model parameters can ultimately be solved for using the procedure discussed in the previous subsection. However, before optimising a cost function solely there are first other questions that need to be answered. These questions are as follows:

1. How many layers should the network have?
2. How many nodes should exist within each layer?
3. Which activation functions should be used on each layer?
4. Should the network be regularised and if so what should the regularisation parameter be?
5. What optimisation algorithm should be used?

In order to answer these questions, we must perform a preliminary grid search through each combination of these hyper-parameters and decipher which would be optimal.

In machine learning it is standard procedure to split the dataset into a training and test set in which one trains each model on the training set and then checks which performs the best on the out of sample test set. Due to the lack of data constraints that we are faced with in the financial industry, we will be performing this procedure slightly differently.

Our data set consists of bond prices with different maturities which we then use to obtain Forward Rate Agreement prices. The Swaption prices will be simulated and will act as our training set in which each model is trained to fit them as accurately as possible (using the cost function in the previous section) then we shall test the models ability to predict unseen data by evaluating the price discrepancy of the real and fitted Forward Rate Agreements.

A tensorboard will be used to display a time series of each epoch versus both the training loss and the validation loss for each hyper-parameter choice. The choice of the best hyper-parameters will depend on the ability of the model to accurately predict the test sets prices whilst showing signs that the model has begun to converge (minor fluctuations in the time series only) near the end of the training process.

We will use the tensorboard to systematically exclude combinations of parameters and optimisation algorithms which indicate that they are unstable or that they may take too long to converge. In the end, we will be left with the optimal hyper-parameter choice and will revert back to solving for the inner parameters of this chosen model via one final optimisation routine.

Chapter 4

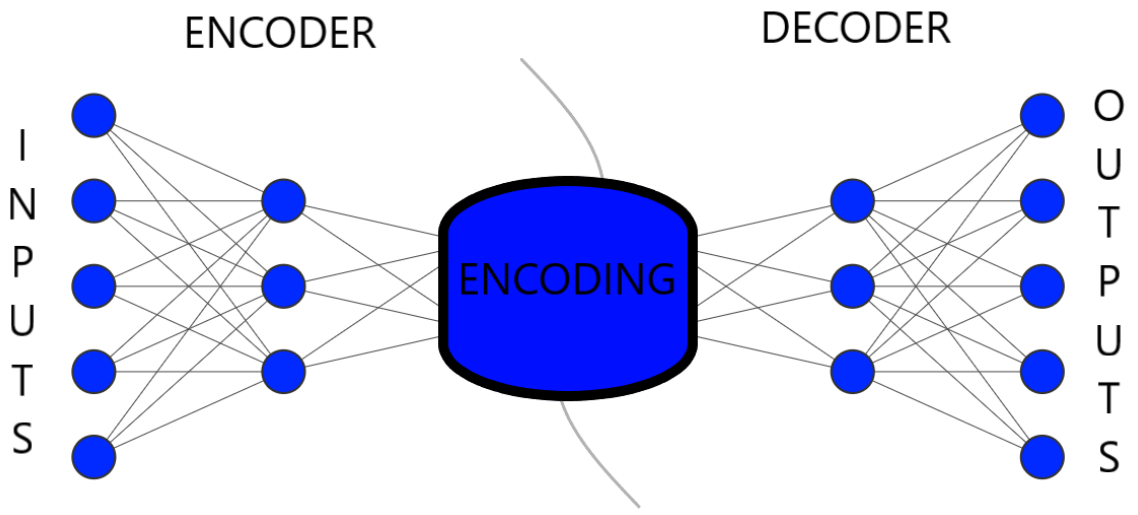
Variational Auto-Encoders

In order to attack the issue surrounding the lack of data in terms of sample paths of price processes we need to look into methods of data generation. The first two sections following are mainly compiled from [Rifai et al., 2011], [Xu et al., 2017] and [Hou et al., 2017]. The third section is a new proposed variational autoencoder which we deem to be a better formulated version of the existing variational autoencoders.

4.1 Autoencoders

An autoencoder comprises of two connected networks, an encoder and a decoder. An encoder network takes in an input and attempts to extract its primary features i.e., reduce it's dimensionality. The decoder network then tries to rebuild the original input from this reduced feature space. If the decoder can reconstruct the inputs completely then we have successfully reduced the dimension of the data. The network structure is represented in Figure 4.1 below.

Figure 4.1: Standard Autoencoder Structure



Both components of the network are usually trained together, using a single loss function which penalises the whole network for poorly reconstructing the original inputs. The loss function is usually defined as the mean-square error or the cross-entropy between the inputs and outputs.

4.1.1 Limitations of autoencoders for data generation

Autoencoders are generally capable of generating lower dimensional representations of the input data, and consequently reconstruct their inputs fairly well. However, apart from some select applications, like denoising data and signal processing, their use tends to be limited.

When using autoencoders for purpose of data generation, the encoded vectors, which are in the latent space, are neither continuous nor easily interpretable in most cases.

When training an autoencoder, clusters within the latent space can be expected to form. These clusters are logically formed since distinct groups improve the decoder's ability to decode effectively. If you are just trying to replicate the data for dimensionality reduction, this is fine but when you are building a generative model you are not only trying to replicate the data but also sample new data points. In order to generate new data points, you want to be able to randomly sample from a continuous latent space and pass it to the decoder in order to generate new variations of the existing data.

Following the process described above, where the latent space contains discontinuities, any points sampled from within the discontinuity would give rise to unrealistic synthetic data. This is because the discontinuity creates 'jumps' between the continuous portions of the latent space which results in the network behaving unpredictably in those regions.

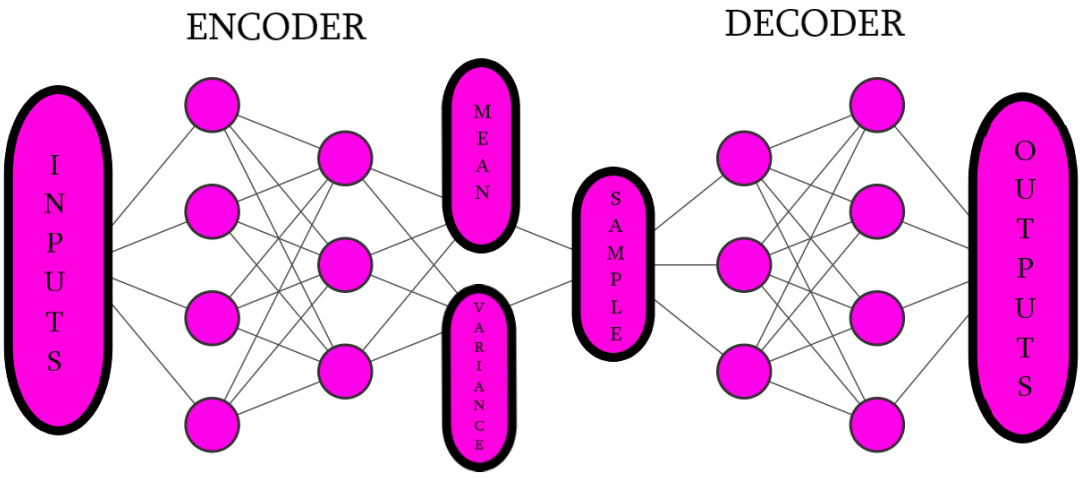
Variational autoencoders, which are covered in the next section, deal with the problem of discontinuous latent spaces by forcing some sort of distributional continuity.

4.2 Variational Autoencoders

Variational Autoencoders (VAEs) are by design better generative models than standard autoencoders. The property which makes them so useful is the continuity of the latent space which allows for easy sampling and interpolation.

The VAE achieves this goal by having the encoder output two vectors, one mean vector (μ) and one standard deviation vector (σ), instead of an n-dimensional encoding vector. The network structure is represented in Figure 4.2 below.

Figure 4.2: Variational Autoencoder Structure



The mean and standard deviation vectors form parameters for an n-dimensional random variable, where the i^{th} elements of μ and σ are the mean and standard deviation of the i^{th} random variable, X_i, from which we sample to obtain the sampled encoding for the decoder.

This process is stochastic which means that, even for the same inputs, the sampled encodings passed to the decoder may differ.

Intuitively, the mean vector determines where the centre of the encoding lies while the standard deviation vector determines the distance we allow the encoding to vary from its centre. As encodings are normally distributed, the decoder learns to associate the near neighbours of each sample point with the same class.

The model is now exposed to some variations of the data by varying the encoding from one sample to another. This process results in continuous latent spaces on a local scale for similar samples. Ideally, we want encodings which are as close as possible, while remaining distinct, which would allow for easy interpolation for the network. However, having no limitations on the values for μ and σ, the encoder may generate very different values for μ for different clusters and may minimise σ which would cluster distinct groups far apart.

In order to force the network to find the optimal parameter configuration where the groups are as close as possible but yet distinct we need to introduce the Kullback–Leibler divergence into the loss function.

Given two probability distributions $\mathbb{P}$ and $\mathbb{Q}$ where $\mathbb{P}$ usually refers to the data or precisely measured probability distribution and $\mathbb{Q}$ refers to the theoretical distribution used to approximate $\mathbb{P}$. The Kullback–Leibler divergence is then interpreted as the average difference of the number of bits required for encoding samples of $\mathbb{P}$ using a code optimized for $\mathbb{Q}$ rather than one optimized for $\mathbb{P}$.

Definition 4.2.1 (Kullback–Leibler Divergence)**.** If $\mathbb{P}$ and $\mathbb{Q}$ are probability measure over the measurable space $(\Omega, \mathcal{F})$, and $\mathbb{P}$ is absolutely continuous with respect to $\mathbb{Q}$, then the relative entropy from $\mathbb{Q}$ to $\mathbb{P}$ is defined as

$$D_{KL}(\mathbb{P}||\mathbb{Q}) = \int_{\Omega} \ln\left(\frac{d\mathbb{P}}{d\mathbb{Q}}\right) d\mathbb{P}$$

where $\frac{d\mathbb{P}}{d\mathbb{Q}}$ is the radon-nikodym derivative of $\mathbb{P}$ with respect to $\mathbb{Q}$.

For VAEs the KL loss is equivalent to the sum of all KL divergences between component $X_i \sim N(\mu_i, \sigma_i^2)$ in X and a standard normal i.e.

$$\text{loss} = \sum_{i=1}^{n} (\sigma_i^2 + \mu_i^2 - \ln(\sigma_i) - 1)$$

which is minimized when $\mu_i = 0$ and $\sigma_i = 1$.

Due to symmetry, the KL loss encourages the encoder to distribute all encodings evenly around the center of the latent space. If it clusters them far apart into non-overlapping regions, away from the origin, it will be penalized. This network captures the stochastic nature of a generator pretty well and is quite effective for image generation, however, we would like to learn the real distribution more effectively without prior assumption on the underlying.

In the next section we formulate a new proposed VAE to help us in this regard.

4.3 Proposed Copula Variational Autoencoder

In order to formulate this Variational Autoencoder, we will need to revisit some theory on simulating random variables. We will use [Mavuso, 2019b] and [Gebbie, 2020] to define and describe some of the theoretical results needed to formulate these generator machines.

4.3.1 Simulating Univariate Random Variables

To begin our search for the ultimate random variable simulating machine, we need to start at the beginning with univariate random variables. In elementary probability courses we are introduced to the concept of a cumulative distribution function (CDF) which is used to completely classify a random variable.

Definition 4.3.1 (Cumulative Distribution Function)**.** If X is a random variable, the Cumulative Distribution Function (CDF) of X is a function $F_X : \mathbb{R} \to [0, 1]$ defined by

$$F_X(x) := \mathbb{P}(\{X \leq x\}),\ x \in \mathbb{R}$$

The CDF contains all information about the probability distribution of X i.e. knowledge about F_X allows us to calculate all probabilities relating to X, no matter what X is.

There are many properties of the CDF that will not be displayed here since we are solely interested in the ability it provides us to simulate samples of random variables. The following result summarises this property.

Theorem 4.3.1 (Probability integral transform)**.**

1. If X is a continuous random variable and $U := F_X(X)$, then U is uniformly distributed between 0 and 1.
2. Conversely, suppose U has a uniform distribution between 0 and 1 and let $F : \mathbb{R} \to [0, 1]$ be a continuous CDF. Then $X := F^{-1}(U)$ is a random variable with CDF equal to F (i.e., $F_X = F$)

This result tells us that given a uniform random number U, we can simulate a random number X from a distribution with CDF F by letting $X = F^{-1}(U)$.

If we could somehow use a neural network to approximate the CDF, then we will automatically gain access to the simulating properties it possesses by simultaneously approximating its inverse.

4.3.2 Simulating Multivariate Random Variables

In order to finalise our search for the ultimate random variable simulating machine, we need to visit some theory on multivariate random variables. As before we start by defining the CDF for multivariate random variables (random vectors).

Definition 4.3.2 (Cumulative Distribution Function)**.** If $X = (X_1, \ldots, X_n)$ is a random vector, the Cumulative Distribution Function (CDF) of X is a function $F_X : \mathbb{R}^n \to [0, 1]$ defined by

$$F_X(x) := \mathbb{P}(\{X \leq x\}),\ x \in \mathbb{R}^n$$

which can also be written as

$$F_X(x_1, \ldots, x_n) := \mathbb{P}\left(\bigcap_{i=1}^{n} \{X_i \leq x_i\}\right),\ x_1, \ldots, x_n \in \mathbb{R}$$

The CDF, as before, contains all information about the probability distribution of X, however, we will need the concept of a copula before fully grasping the generative power we are looking for.

The copula definition we will look at is usually introduced as a theorem after a more theoretical approach. Given the nature of our problem, this is enough.

Definition 4.3.3 (Copula)**.**
Let F be an n-dimensional multivariate distribution and F_i be the i^{th} marginal for $i = 1, \ldots, n$ then F has a unique copula $C : [0, 1]^n \to \mathbb{R}$ defined by

$$F(x_1, \ldots, x_n) := C(F_1(x_1), \ldots, F_n(x_n))$$

From this definition, we can reformulate in terms of uniform random variables. Let $u_1, \ldots, u_n$ be uniformly distributed between 0 and 1 then

$$C(u_1, \ldots, u_n) = F(F_1^{-1}(u_1), \ldots, F_n^{-1}(u_n))$$

i.e. The copula is a function that links independent uniform random variables to correlated random variables.

If we could somehow use a neural network to approximate this link, then we will once again automatically gain access to the simulating properties. In the case of univariate random variables, the link is just the CDF and inverse CDF as described previously.

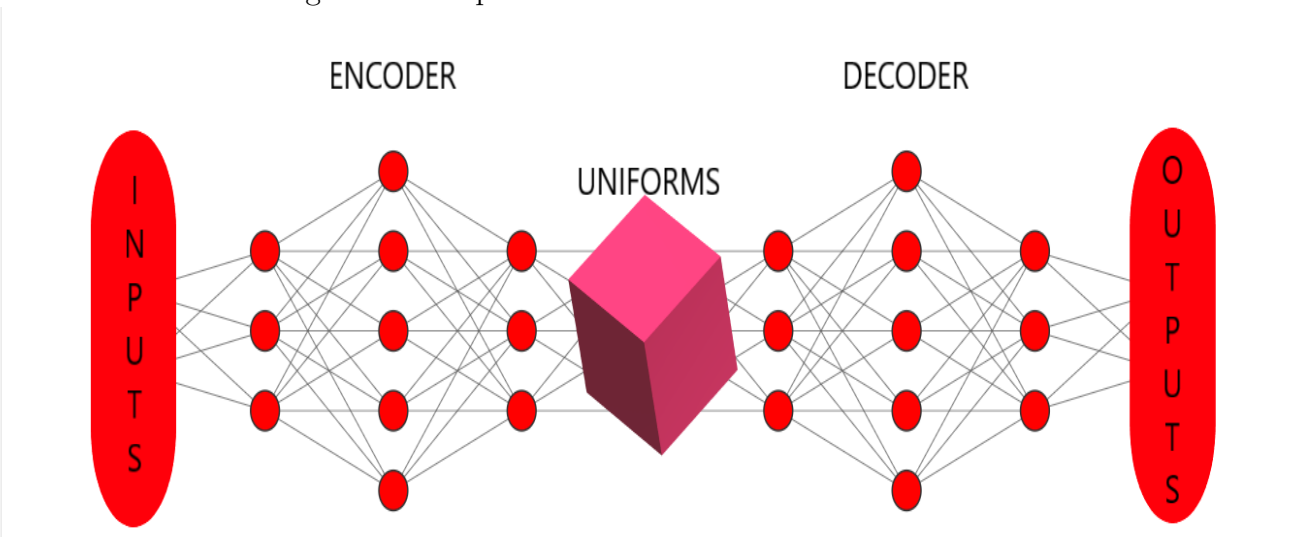
In the next subsection, we will formulate our copula variational autoencoder and then show some examples of how it can be used to simulate correlated random variables.

4.3.3 The Copula Variational Autoencoder

Much like the standard autoencoder and the original variational autoencoder, the structure of our copula VAE consists of an encoder and a decoder. The one big difference is that our latent space will be uniformly distributed by design.

The network structure is represented in Figure 4.3 below.

Figure 4.3: Copula Variational Autoencoder Structure



To this end, suppose $X_1, \ldots, X_n$ is a random sample from the distribution of the d-dimensional random vector $X : \Omega \to \mathbb{R}^d$ with an unknown joint CDF $F_X : \mathbb{R}^d \to [0,1]$. In what follows, we describe a way of simulating additional observations from this unknown distribution using artificial neural networks.

We want to find a dimension $d' \leq d$ and two functions, F and G, such that

1. $F : \mathbb{R}^d \to [0,1]^{d'}$ and $G : [0,1]^{d'} \to \mathbb{R}^d$
2. $G = F^{-1}$
3. $F(X)$ is uniformly distributed on the cube $[0,1]^{d'}$
4. For $U \sim U([0,1]^{d'})$, $G(U)$ has the same distribution as X.

Our contribution is to use Neural Networks to approximate F and G. To find the parameters of the neural network approximation to F, we use condition 3 above to minimize the distance between an empirical density of the observations $F(X_1), \ldots, F(X_n)$ and the uniform density on the cube $[0,1]^{d'}$ (which is just the indicator function of the cube).

For the empirical density, we use a kernel density estimate as follows. Let $K : \mathbb{R}^{d'} \to \mathbb{R}$ be a probability density function e.g., a Gaussian kernel, which is defined by

$$K(x) = (2\pi)^{-\frac{d}{2}} e^{-\frac{1}{2}\|x\|^2}$$

for $x \in \mathbb{R}^{d'}$. Given a sample $y_1, \ldots, y_n \in \mathbb{R}^{d'}$, we define the *Kernel density estimate* of the sample as

$$\hat{f}_h(y) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{y - y_i}{h}\right), \quad y \in \mathbb{R}^{d'},$$

where $h > 0$ is the smoothing parameter called the *bandwidth*.

Then if F_θ^{NN} is a neural network approximation of F with parameters θ, and $x_1, \ldots, x_n$ are the observed values of the given sample, we find θ by minimising the mean-square error loss given by

$$MSE(\theta) = \frac{1}{n} \sum_{k=1}^{n} \left(\frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{F_\theta^{NN}(x_k) - F_\theta^{NN}(x_i)}{h}\right) - 1 \right)^2$$

which also can be made stronger by adding the terms

$$\frac{1}{n} \sum_{k=1}^{n} \left(\frac{1}{nh^2} \sum_{i=1}^{n} K'\left(\frac{F_\theta^{NN}(x_k) - F_\theta^{NN}(x_i)}{h}\right) \right)^2$$

and

$$\frac{1}{n} \sum_{k=1}^{n} \left(\frac{1}{nh^3} \sum_{i=1}^{n} K''\left(\frac{F_\theta^{NN}(x_k) - F_\theta^{NN}(x_i)}{h}\right) \right)^2$$

i.e. we find the parameters, θ, which sets the density of the distribution to the indicator function of the cube and make this stronger by ensuring that the first and second derivatives of the density are zero.

To find G, we use point 2 that $G = F^{-1}$. Let G_θ^{NN} be a neural network approximation of G and, with some abuse of notation, denote the best neural network approximation of F (found above)

by F. Finally, let $y_1, \ldots, y_N$ be a large sample of $U \sim U([0,1]^{d'})$ variates. Then we find θ by minimising

$$MSE = \frac{1}{N} \sum_{k=1}^{N} \left(F(G_\theta^{NN}(y_k)) - y_k \right)^2$$

The training process of the encoder involves taking the inputs and learning a function that will map them into a continuous uniformly distributed latent space.

The training process of the decoder then learns the inverse of the encoder in order to take the uniform random variables and generate variables from the underlying distribution. In order to perform this training procedure, we map the uniforms obtained from the encoder back to the input as well as newly generated uniforms through the decoder and then encode them back to themselves.

Later in this dissertation, we will use this formulation of variational autoencoders in order to generate "real-world" sample paths. This will allow us to thwart the issue existing in financial model testing: only one sample path exists in reality.

In the next two subsections, we will take a look at how the copula VAE performs in the univariate and bivariate case.

4.3.4 Example of Generating Exponential Random Variables

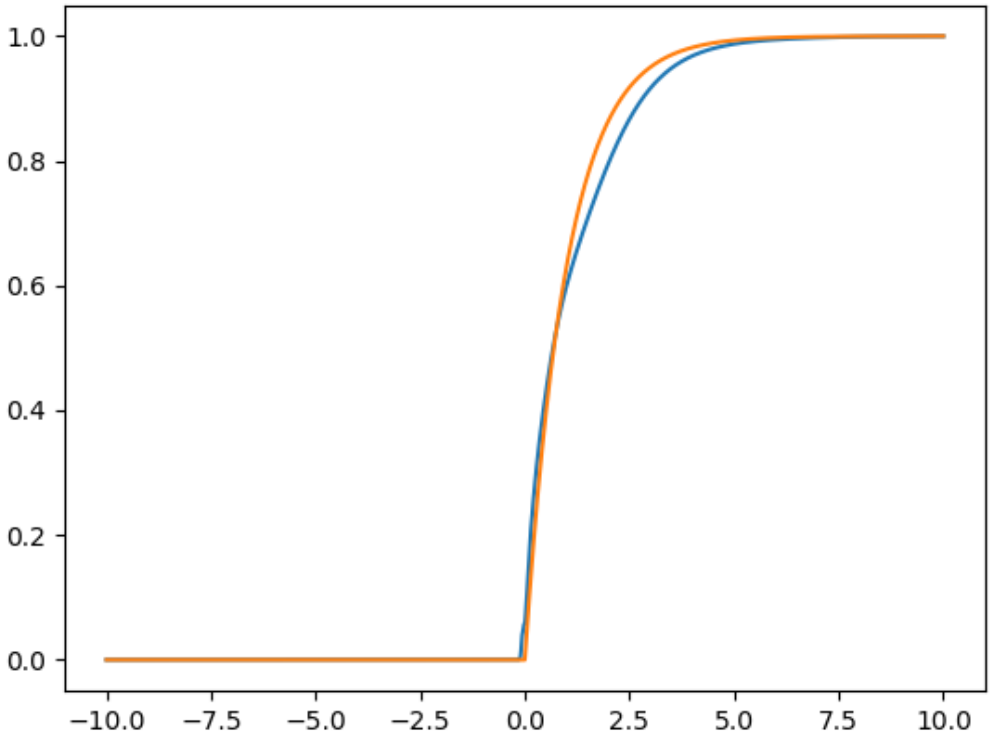
After formulating the network, we are ready to view examples of the network learning the CDF of a given distribution based on observations of the random variable. Furthermore, we are interested in its generative abilities, thus we also view how well it simulates realisations from this distribution.

In order to observe the results, we simulate 1000 exponential realisations with rate $\lambda = 1$. We performed no hyper-parameter optimisation as we were mainly interested in whether the network could work as we would like and not what the best choice of parameters would be.

We arbitrarily chose a $(60, 60, 60)$-network for both the encoder and decoder. The encoder's activation functions were arctan, swish, relu and sigmoid. The decoder's activation functions were tan, tanh, relu and linear. We train the encoder for 100 epochs and then we train the decoder against the encoder to find the inverse for 1000 epochs.

In Figure 4.4, for input values $x \in [-10, 10]$, we display the encoders learned CDF as well as the real CDF of an exponential random variable with rate $\lambda = 1$.

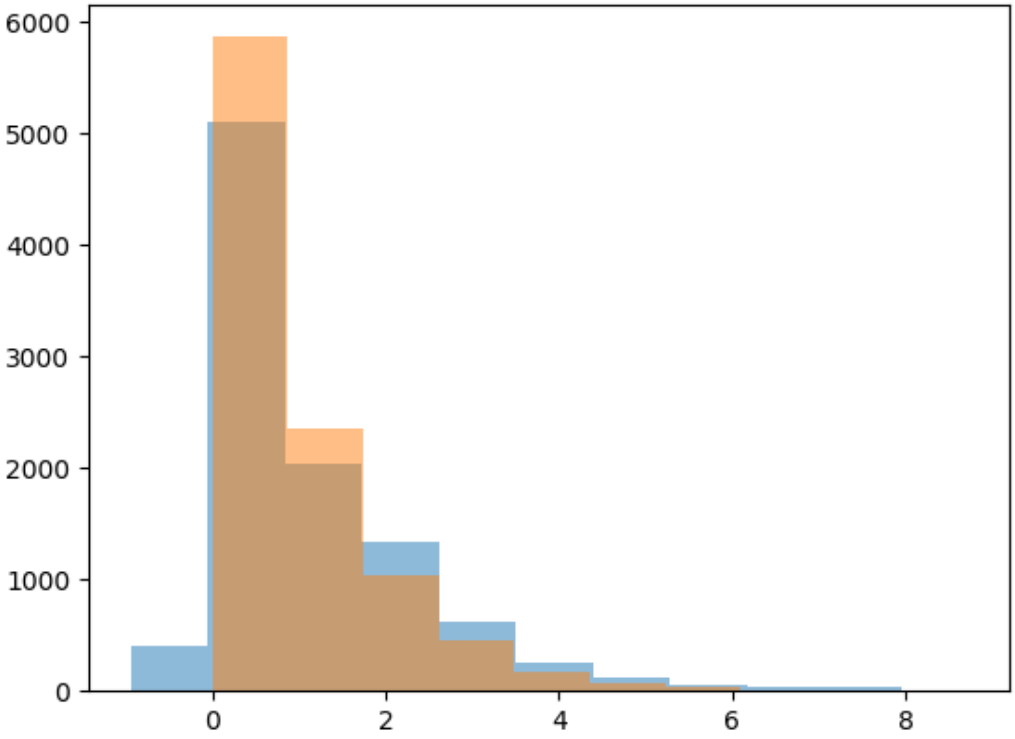
Figure 4.4: Exponential CDF - Exact (Orange) and Learned (Blue)



As previously discussed, we are mainly interested in the generative abilities of our network.

We generate 10000 uniform random variables between 0 and 1 and feed them into the decoder. In Figure 4.5 the histogram displaying the relevant frequencies of the simulated variables as well as simulations from the exponential function in python's numpy class is shown.

Figure 4.5: Exponential Simulation: Exact (Orange) and Learned (Blue)



From the figures one can see that the network performs exceptionally well with one important issue being raised: the negative observations generated. However, this is not actually an issue since knowledge about the realisations being positive and optimising the hyper-parameters accordingly would rid us of this affair.

4.3.5 Example of Generating Correlated Random Variables

Now that we have seen that the network can deal with learning the CDF and Quantile functions of random variables, the next step is to see if it can handle random vectors with correlated components.

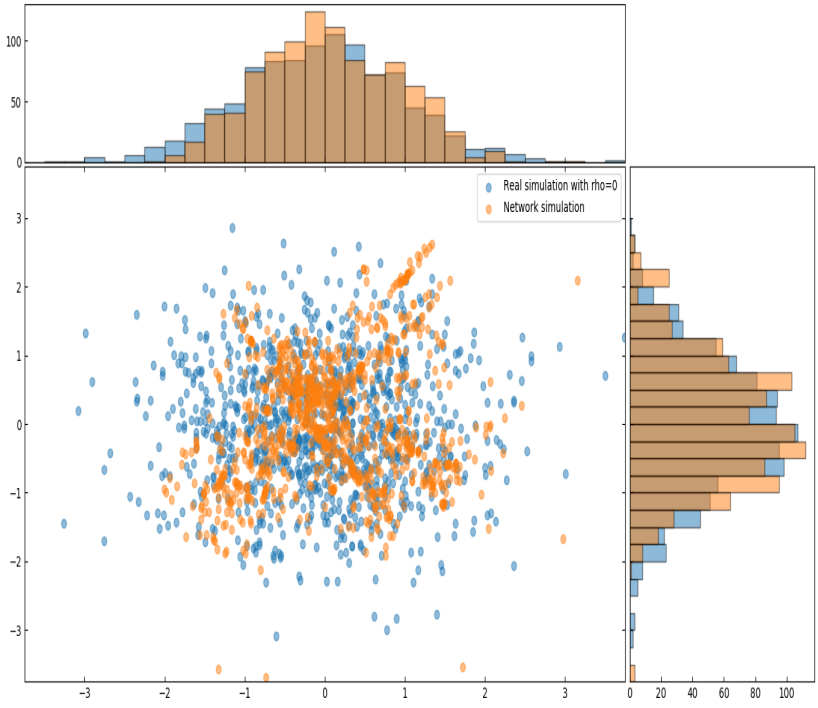
In order to generate correlated random variables with correlation coefficient ρ, we generate two uncorrelated standard normal random variables Z_1 and Z_2 and perform the following linear transformation

$$\begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} \rho & \sqrt{1-\rho^2} \\ 1 & 0 \end{bmatrix} \begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix}$$

then X_1 and X_2 are both standard normal random variables with correlation ρ.

We begin with uncorrelated standard normal random variables in order to see whether the network can handle random vectors. The generation is displayed in Figure 4.6.

Figure 4.6: Simulation of uncorrelated normal random variables



From the figure we can see that our network performs exceptionally well once again and that the scatter plot is random as we would like.

On the following page, in Figure 4.7 we choose $\rho = 0.99$ and in Figure 4.8 we choose $\rho = -0.85$. From these diagrams we can see the network start to learn how to correlate it's observations as expected.

If we were to optimise the hyper-parameters of the network, this would prove to be an outstanding tool for generative purposes. In the last chapter of this dissertation we will finally use this network to obtain more sample paths to test our other results on.

Figure 4.7: Simulation of correlated normal random variables with $\rho = 0.99$

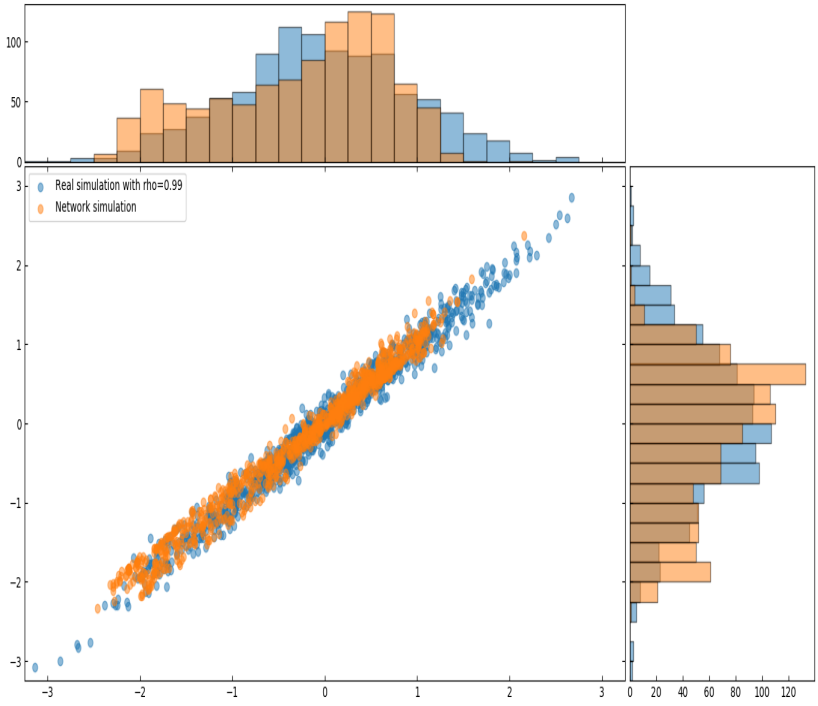
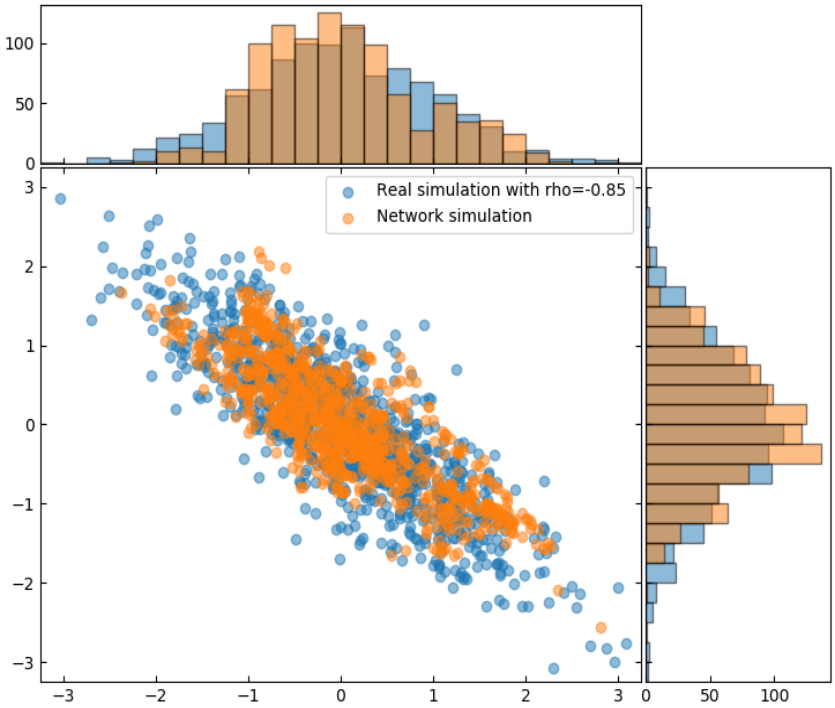


Figure 4.8: Simulation of correlated normal random variables with $\rho = -0.85$



Chapter 5

Results and Discussion

In this chapter we will fit two volatility models to the LIBOR market model. We will then test each of these in turn via hedging and evaluate the performance of these models based on a metric.

A disclaimer before getting into any details is that this chapter may seem unusually short compared to the prior chapters. This is due to the fact that all the heavy lifting was covered previously and most of the weight of this chapter was in the code implementation.

5.1 Proposed Volatility Structures

In this dissertation, we shall fit two different volatility structures. In Chapter 3, we discussed many structures but our chosen models will be constant and the neural network approximation. See A and F in Table 3.1 to be reminded of the forms we will fit. In Section 3.5, we discussed which procedures we will use to fit the parameters in these models.

5.2 Proposed Hedging Techniques

As discussed in Chapter 1, we will need a way to compare the performances of the proposed volatility stuctures; this will be done via different hedging techniques.

In the preliminaries it was shown that in a complete market where all bounded claims are attainable you can find the hedging strategy via the predictable representation property which states that for a local martingale V we can write

$$V_t = V_0 + \int_0^t \varphi_s \, dX_s$$

for some predictable X-integrable process φ where in our case V represents the value of the hedging portfolio and X represents the tradeable assets.

To find such a φ, we perform different approximations and consider them to be our trading strategies.

Sensitivity Approximation

Assuming that V is a function of both X_t and t, the first approach we take is to note that we can write

$$V(X_t, t) = V(X_0, 0) + \int_0^t \frac{\partial V(X_s, s)}{\partial X_s} \, dX_s$$

i.e. that

$$\varphi_s = \frac{\partial V(X_s, s)}{\partial X_s}$$

which can be approximated via finite differencing as follows:

$$\varphi_s = \frac{\partial V(X_s, s)}{\partial X_s} \approx . \frac{V(X_s + h, s) - V(X_s - h, s)}{2h}$$

for small h.

Quadratic Variation Approximation

The Kunita-Watanabe Decomposition states that under the pricing measure if X is a continuous local martingale, then any continuous local martingale V can be written as

$$V_t = V_0 + \int_0^t \varphi_s \, dX_s + L_t = (\varphi \bullet X)_t + L_t$$

where L_t is strongly orthogonal to X.

We note now that

$$\langle V, X \rangle_t = \langle (\varphi \bullet X), X \rangle_t + \langle L_t, X \rangle_t = (\varphi \bullet \langle X \rangle)_t .$$

Another way of writing this is to say that

$$d\langle V, X \rangle_t = \varphi_t d\langle X \rangle_t$$

which we loosely write as

$$\varphi_t = \frac{d\langle V, X \rangle_t}{d\langle X \rangle_t}.$$

The approximation we use to hedge via this theory is to say that

$$\varphi_t \approx \frac{\mathbb{E}\left[(V_{t+dt} - V_t)(X_{t+dt} - X_t) \,|\mathcal{F}_t\right]}{\mathbb{E}\left[(X_{t+dt} - X_t)^2 \,|\mathcal{F}_t\right]}$$

which assumes that V_t and X_t are continuous.

This will require us to simulate multiple one step ahead values for X_{t+dt} and via this find the price of the options one step ahead i.e. V_{t+dt}.

5.3 Results

Data - Yield Curves

The yield curves needed for our sample paths were constructed from seven bonds which were obtained from Bloomberg terminal. The bonds of interest were generic South African bonds for multiple maturities, namely 2, 3, 7, 10, 15, 20, and 30 year maturity bonds. The tickers for these bonds were GSAB2YR, GSAB3YR, GSAB7YR, GSAB10YR, GSAB15YR, GSAB20YR and GSAB30YR. The data consisted of 713 trading days ranging from 14 April 2016 to 6 March 2020.

In order to generate new yield curves, we trained our VAE with the following parameters:

1. Encoder

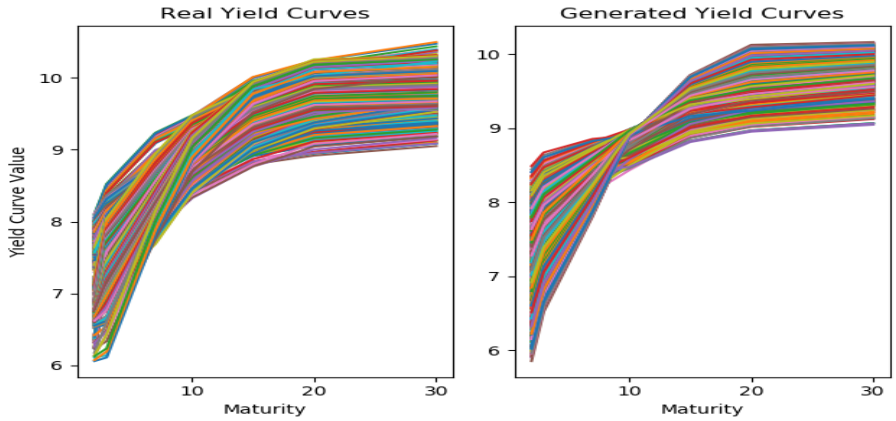
(a) 3 Hidden layers
(b) 10 Nodes per layer
(c) Arctan, Swish, Relu and Sigmoid activations
(d) 10000 uniforms for the kernel
(e) 200 Epochs
(f) Adam optimizer with a learning rate of 0.006

2. Decoder

(a) 3 Hidden layers
(b) 10 Nodes per layer
(c) Tan, Tanh, Relu and Linear activations
(d) 10000 uniforms for extra training units
(e) 2000 Epochs
(f) Adam optimizer with a learning rate of 0.006

Contrary to the belief that 3 factors describe the yield curve, a latent space of one was used in order to learn the distribution and generate the curves. In Figure 5.1, we display the real yield curves obtained along with one sample path of yield curves (713 days worth of yield curves) generated from the VAE.

Figure 5.1: One sample of yield curves



5000 such sample paths were obtained via this technique i.e. 5000 more sample paths than the market was able to provide.

Data - Swaption Volatility

The last set of data we would need, is the swaption volatility matrices relating to the bonds in question.

Due to unforeseen circumstances in the data collection process, the swaption volatility matrices were not obtained. To thwart this issue, we assumed that the swaption prices would have come from a stochastic volatility model, namely the Heston model.

The Heston model assumed for the swap rates is as follows:

$$dS_\alpha^\beta(t) = \sqrt{\nu_t} S_\alpha^\beta(t) dW_t^S$$

where

$$d\nu_t = \kappa(\theta - \nu_t)dt + \xi\sqrt{\nu_t}dW_t^{\nu}$$

such that $dW_t^S dW_t^{\nu} = \rho dt$.

We chose the following parameters:

1. $S_\alpha^\beta(0)$ obtained from yield curves
2. $\nu_0 = 0.01$
3. $\theta = 0.01$
4. $\kappa = 2$
5. $\xi = 0.1$
6. $\rho = 0$

Remark 5.3.1. These parameters may or may not be realistic, however, the dependence on S_α^β holds true as in the real markets. This dependence may not be the strongest but without assurances on market efficiency, we cannot truly say that the market volatilities are accurate either.

Via simulation, we obtain the prices of the relevant at-the-money swaptions. We then use them to solve for the volatility which satisfies the Black-76 formula for swaptions introduced in Section 3.3 by equating the price obtained to the pricing formula i.e. through root finding algorithms we find $\hat{\sigma}_{\alpha,\beta}^{swap}$ for each relevant α and β which satisfies

$$\pi_{\alpha,\beta}^*(0) = \sum_{i=\alpha+1}^{\beta} \delta_i P_i(0) \left[S_\alpha^\beta(0)\Phi(d_1) - K\Phi(d_2)\right]$$

where

$$d_1 = \frac{1}{\hat{\sigma}_{\alpha,\beta}^{swap}\sqrt{T_\alpha}}\left[\ln\left(\frac{S_\alpha^\beta(0)}{K}\right) + \frac{1}{2}(\hat{\sigma}_{\alpha,\beta}^{swap})^2(T_\alpha)\right]$$
$$d_2 = d_1 - \hat{\sigma}_{\alpha,\beta}^{swap}\sqrt{T_\alpha}$$

and $\pi_{\alpha,\beta}^*(0)$ is the simulated Swaption price.

Hyper-parameter tuning

Now that the data has been discussed, we need to obtain the optimal neural network hyper-parameters for our volatility model.

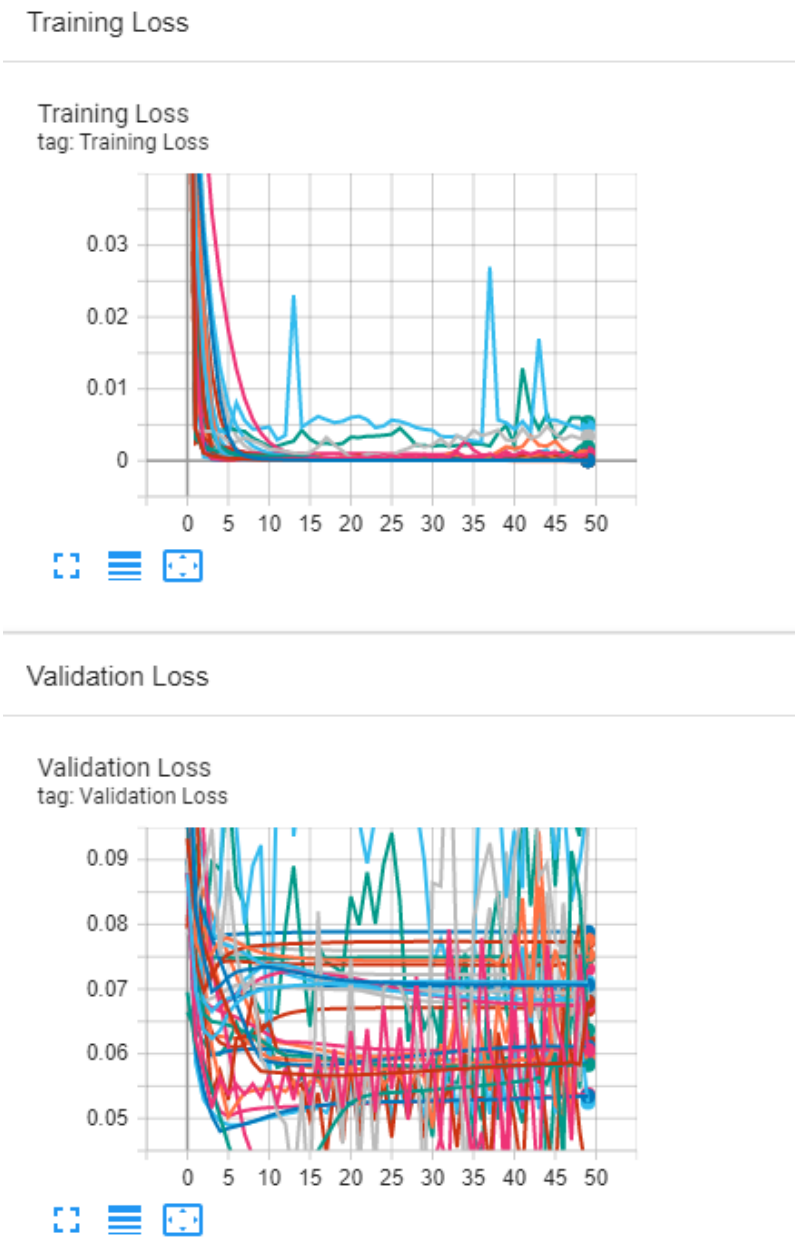
The issues with some of the recent machine learning enthusiasts is their tendency to over fit their models. Although neural networks are prone to over fit on seen data, we can nudge the process by optimising these parameters on unseen data. A validation process was performed via a tensorboard that plotted the training loss (the ability to fit the swaption volatilities) and validation loss (how well these new volatilities price caplets) per epoch. We decided to use a three hidden layer neural network with a fixed output activation being sigmoid and optimise for the rest of the hyper-parameters.

The grid was set up with the following as a sample space for the choices of hyper-parameters:

1. Neurons1 = Neurons2 = {60, 120}
2. Activation1 = Activation2 = {'relu','sigmoid','tanh'}
3. Learning rate = {0.01}
4. Optimiser = {RMSprop}

A bigger search with different learning rates and more optimisers was done prior. We noted that RMSprop with a learning rate of 0.01 out performed the rest quite signifcantly and to save time only ran these 36 combinations.

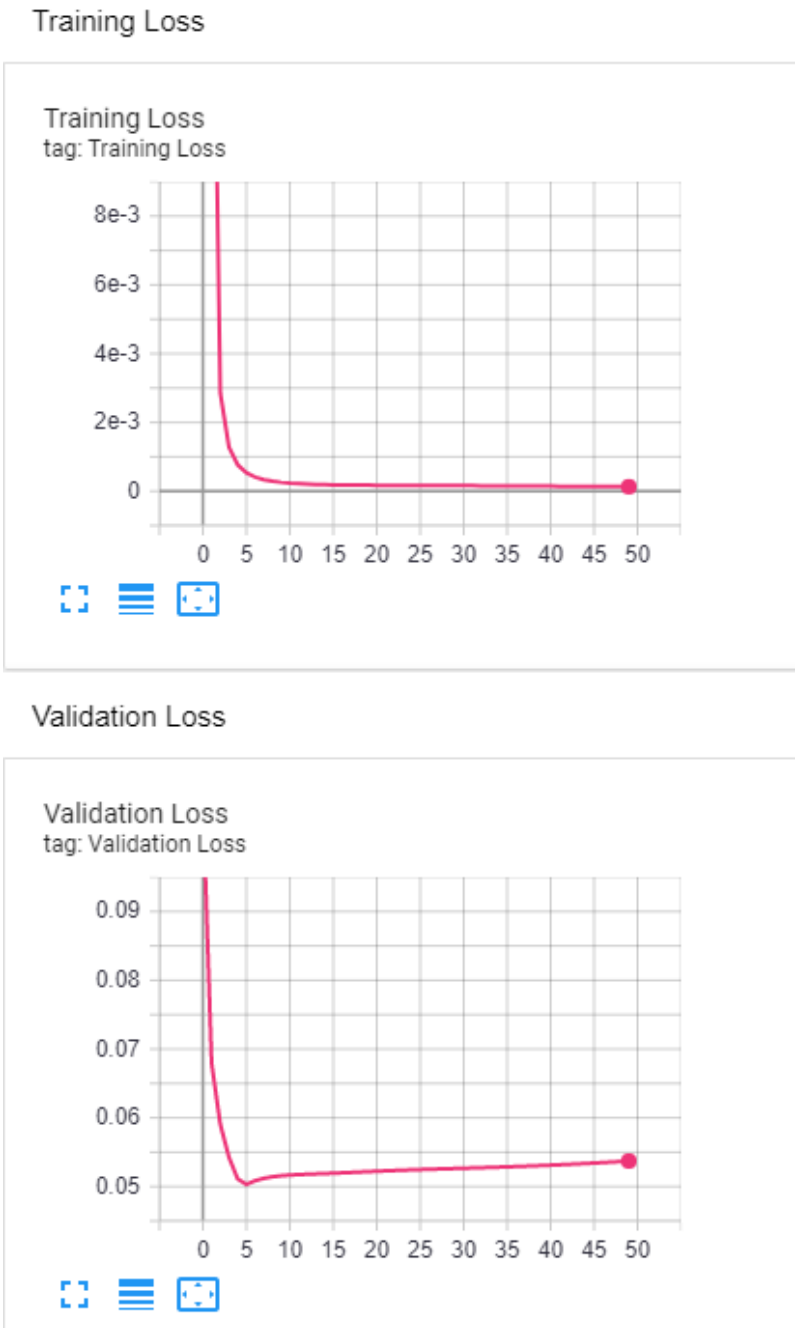
Figure 5.2: The interactive tensorboard can be viewed via the link:
`https://tensorboard.dev/experiment/KfLf6aH4Rn6uenWi0okYxw/`



The full tensorboard displayed in Figure 5.2 may not be very appealing as a standalone diagram, however, we will display the losses of each model, we deem important to discuss, per epoch in turn as we analyse them throughout.

The optimal configuration of hyper parameters chosen was model 5 i.e the configuration
{60, 60, 'sigmoid', 'sigmoid', 0.01, 'RMSprop'}
which has a training loss of 1.2962×10^{-4} and a validation loss of 0.05372. The parameters showed the tendency to converge relatively quickly without much fluctuations from epoch to epoch. The out of bag error also seemed to move smoothly toward the correct prices unlike some other models tested. The losses per epoch are displayed in Figure 5.3.

Figure 5.3: Model 5: {60, 60, 'sigmoid', 'sigmoid', 0.01, 'RMSprop' }



The ability to learn swaption volatilities while extending comfortably to other options is an extremely desirable trait that can only be achieved by training and testing multiple models. Neural networks give us this seemlessly by providing the illusion that they are one model, yet as a matter of fact they are an infinite class of models.

An additional training procedure will precede the testing of each of the volatility models.

Comparing volatility structures

In order to test our volatility models, we hedge the 5000 paths of a simple caplet, trading the available Forward Rate Agreements to do so. One issue that was picked up was the inability to trade each trading day, thus we only traded on terminal dates.[1]

In order to select the optimal model between the two being tested, we first need to define what the metric used to decide this would be. The metric commonly used to in practice is the mean square replication error (MSRE).

Definition 5.3.1. The expected mean square replication error is defined as

$$\mathbb{E}[(V(\varphi) - H)^2].$$

The mean square replication error is the point estimate of this quantity provided by the data at hand.

The best model would be the model that produces the lowest MSRE. The profit and loss histograms for both the trading strategies are shown in Figure 5.4 and Figure 5.5.

Figure 5.4: Profit and Loss with Quadratic Variation Hedging with Different Volatilities

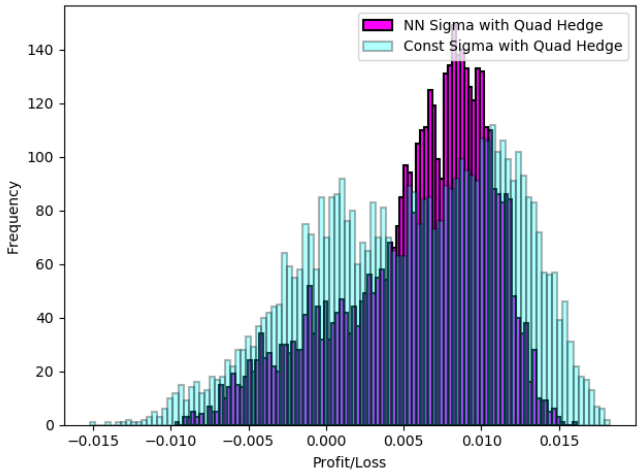
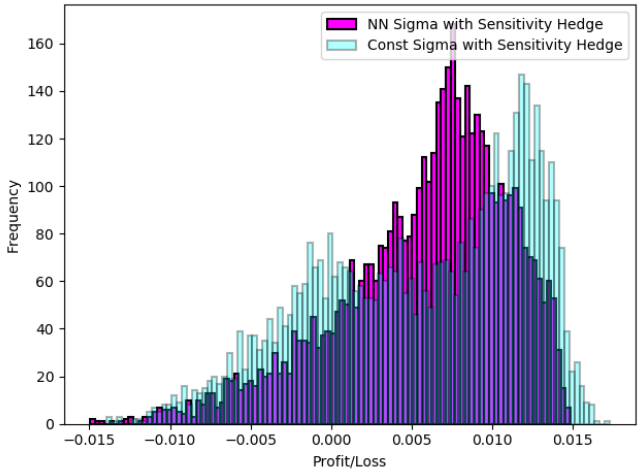


Figure 5.5: Profit and Loss with Sensitivity Hedging with Different Volatilities



[1]Using the discretely reblanced bank account as a numeraire comes with such a disadvantage and it is common in the industry to test the models across the terminal dates.

The images alone, however, do not paint the complete picture. We use Table 5.1 to summarise these results as follows:

Table 5.1: Hedging results summary

Model	Hedge	Mean	Standard Deviation	MSRE
Neural Network	Quadratic	0.005864	0.004838	5.7799×10^{-5}
Neural Network	Sensitivity	0.005799	0.005276	6.1478×10^{-5}
Constant	Quadratic	0.005491	0.006382	7.0886×10^{-5}
Constant	Sensitivity	0.005657	0.006529	7.4635×10^{-5}

Using the metric described in the beginning of this subsection, we see that the Neural Network volatility model seems to outperform the standard constant volatility model by a comfortable margin independent of the hedging strategy used.

Other figures to assist with our analysis are that of Figure 5.6 and 5.7.

Figure 5.6: Profit and Loss with Quadratic Variation Hedging with Different Volatilities

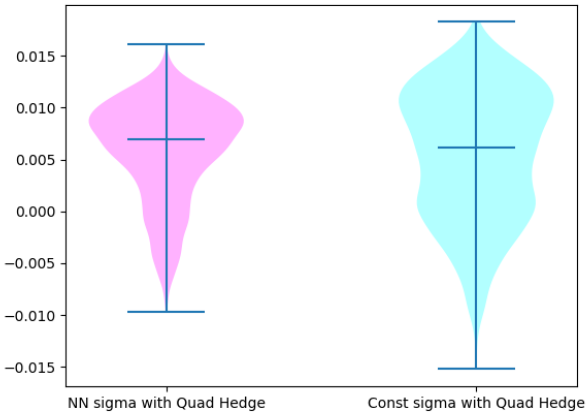
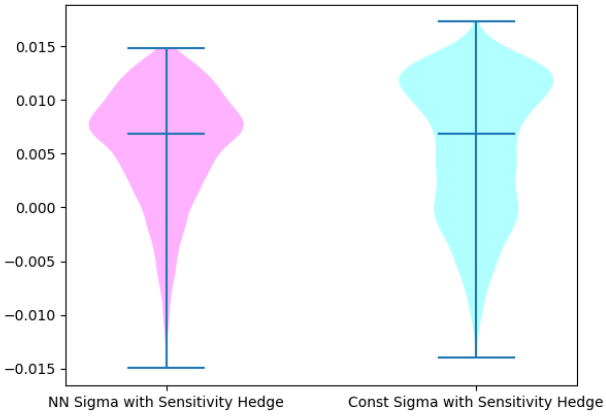


Figure 5.7: Profit and Loss with Sensitivity Hedging with Different Volatilities



To interpret the above plots, we must first describe what a violin plot is.

A violin plot is a simple box plot with kernel densities added on either side i.e. while a box plot only shows summary statistics such as median and interquartile ranges, the violin plot also shows the full distribution of the data.

Given this additional data summary technique, we see that while both medians within the profit/loss results have similar values, it is clear that Neural Network has tighter interquartile ranges.

Another note from these plots is the clear mono-modal distribution surrounding the Neural Network model and a nearing bi-modal distribution surrounding the constant volatility model. Multi-modal distrbutions with similar means to mono-modal distributions tend to have higher variance than the mono-modal counterpart.

Another remark to be made is that no matter which volatility model and/or hedging strategy was chosen, the profit/loss means tend to be positive and with similar magnitude. This observation suggests that the option hedged may be overpriced in the market as the price is the starting value of the portfolio and we expect a mean of zero in theory to be regular.

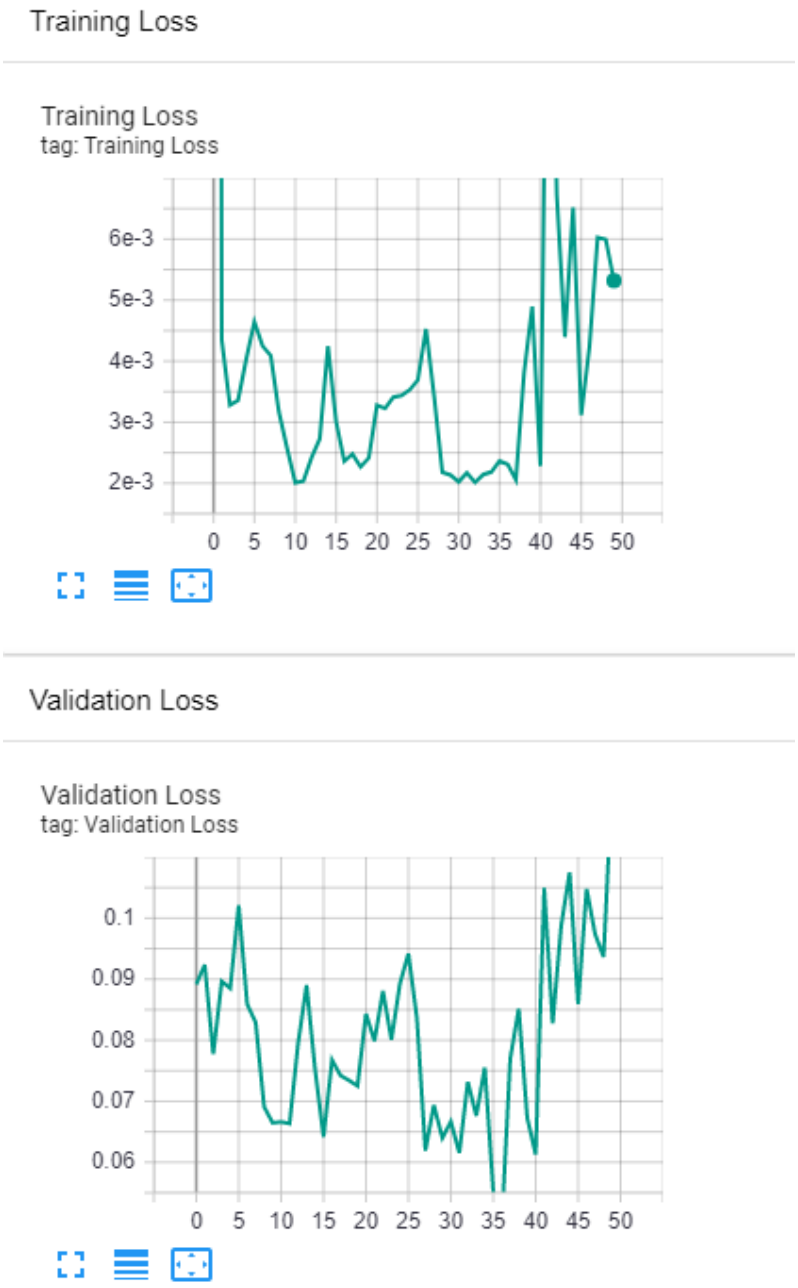
The evidence is now clear that a neural network volatility structure is a viable model when pricing and hedging options. It was able to comfortably outperform that of the constant volatility model while maintaining many desirable properties within the hedging outcomes.

Other neural network models considered

The models in this section were chosen for displaying various aspects, including those which are undesirable, during the training procedure. The layout for this section is as follows: we shall note and display the various aspects displayed for each model, summarise the hedging via sensitivity results obtained from each model and then discuss these results.

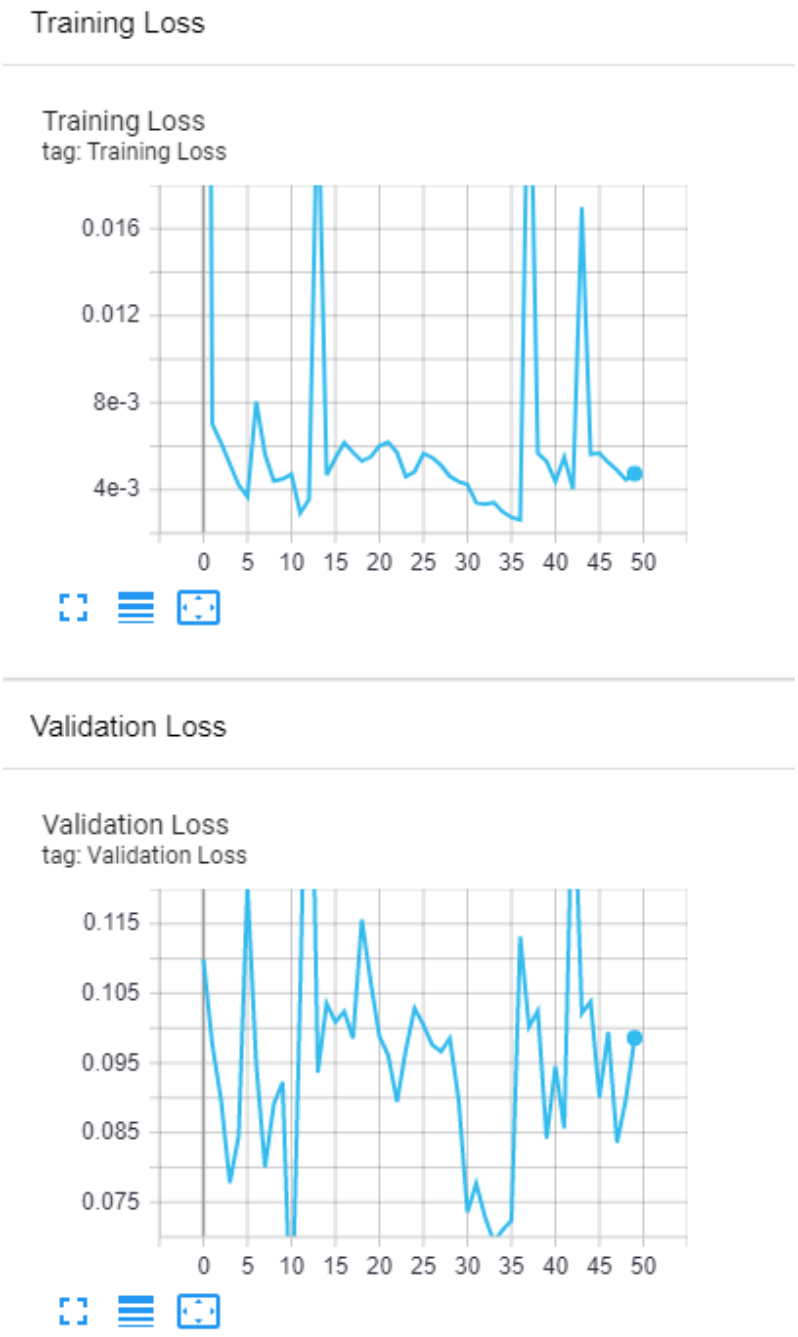
The first model chosen was model 33. This model displayed sporadic training and validation loss as seen in Figure 5.8.

Figure 5.8: Model 33: {120, 120, 'sigmoid', 'tanh', 0.01, 'RMSprop' }



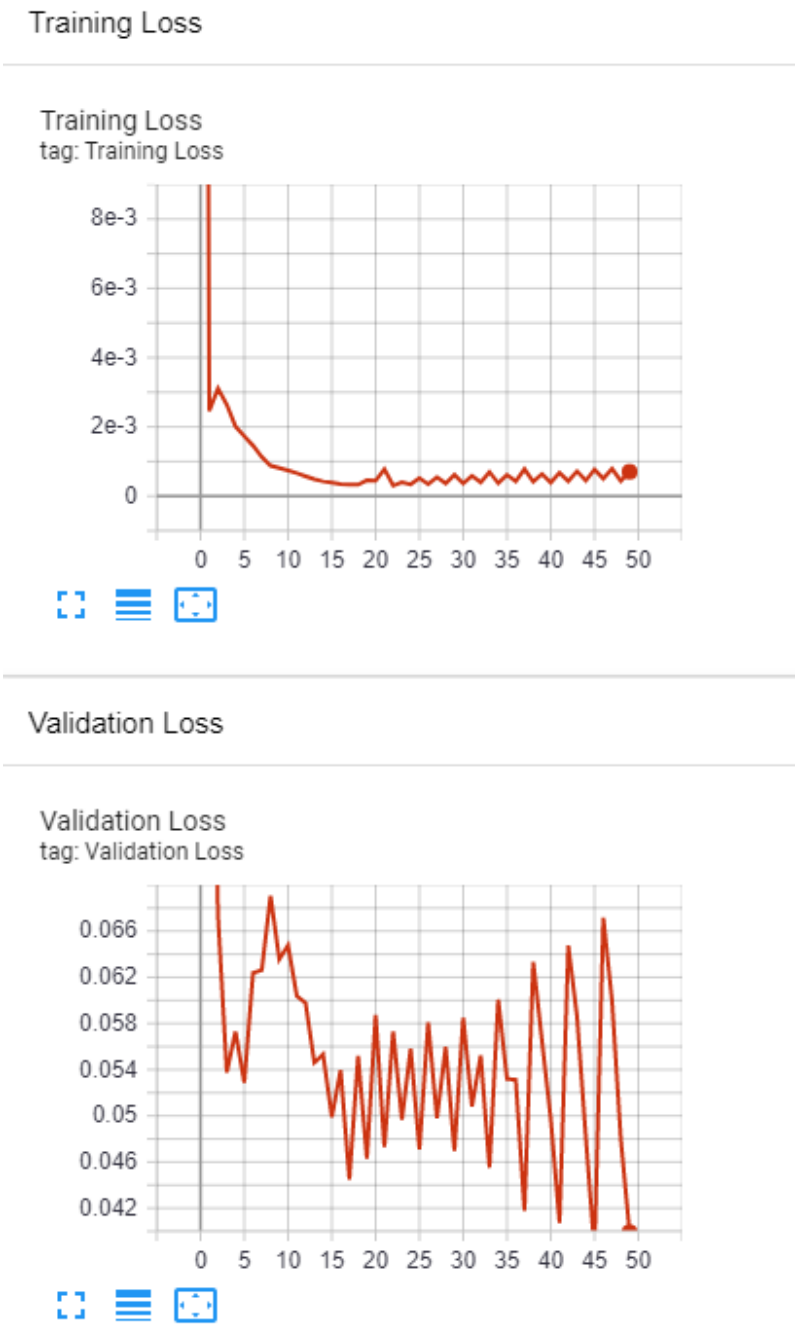
The second model chosen was model 31. This model displayed a less sporadic training loss than model 33 but the validation loss was just as terrible. This can be seen in Figure 5.9.

Figure 5.9: Model 31: {120, 120, 'sigmoid', 'relu', 0.01, 'RMSprop' }



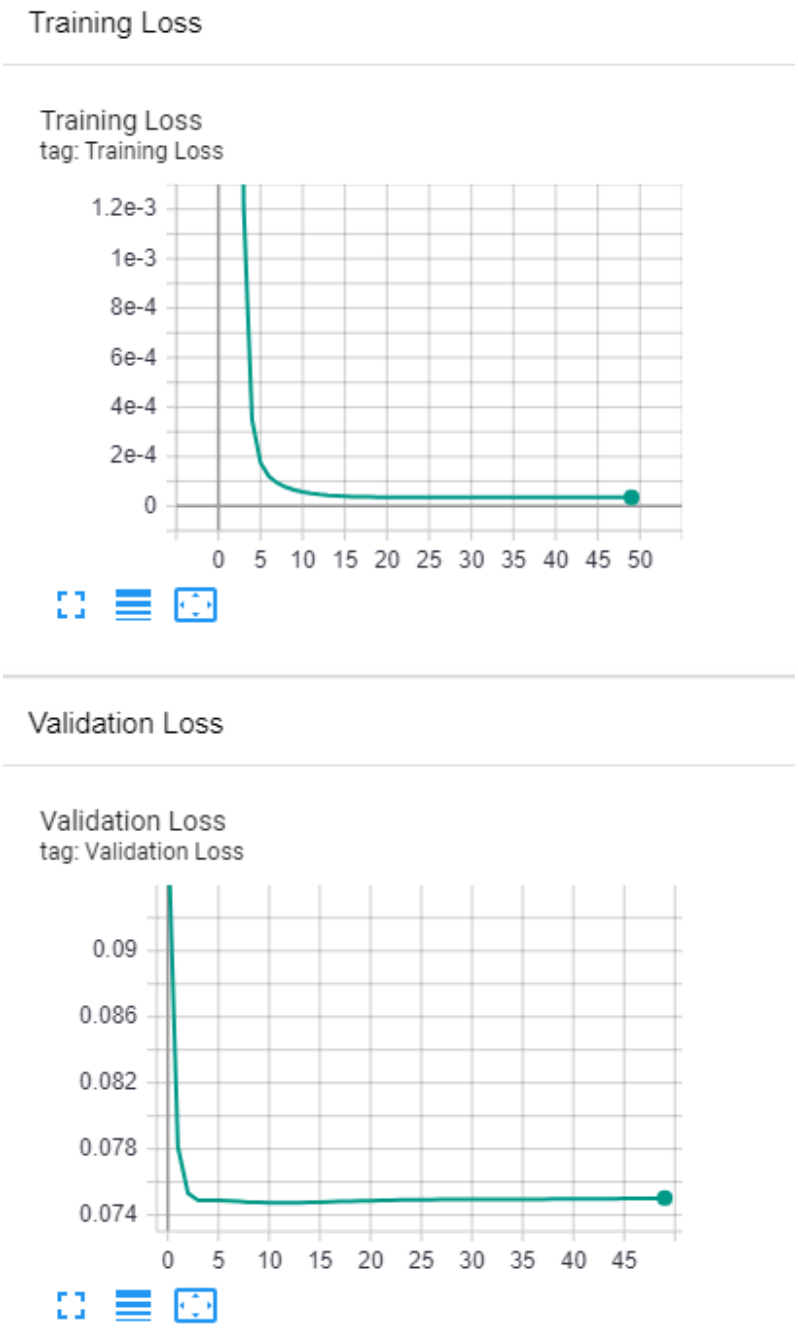
Thirdly, we have model 24. This model displayed a relatively well converging training loss with few fluctuations. The final validation loss was the lowest out of all models at merely 0.03992. This can be seen in Figure 5.10.

Figure 5.10: Model 24: {120, 60, 'sigmoid', 'tanh', 0.01, 'RMSprop' }



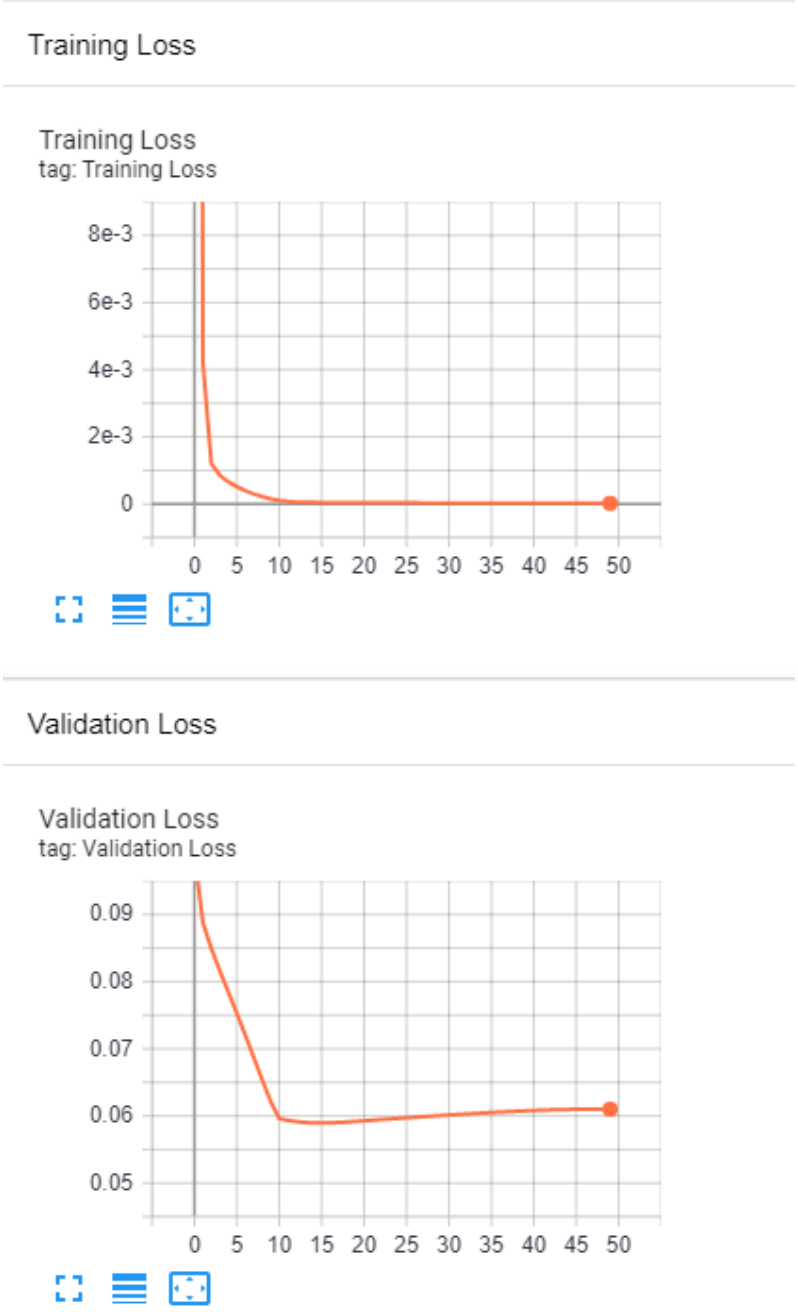
Model 27 was chosen due to its stable validation and training loss. These losses were, however, not as good as our final model discussed previously. In Figure 5.11 we display this phenomenon.

Figure 5.11: Model 27: {120, 60, 'tanh', 'tanh', 0.01, 'RMSprop' }



The last model chosen based on its interesting training results was model 35. This model displayed the lowest final training loss out of all models at 2.1457×10^{-5} and had quite a decent validation loss. These results are displayed in Figure 5.12.

Figure 5.12: Model 35: {120, 120, 'tanh', 'sigmoid', 0.01, 'RMSprop' }



There are also two other models that were chosen that were not in the validation process: a wide model and a deep model.

The wide and deep models were chosen to have the same hyper-parameters as the final chosen model except for their distinct differences. The wide model was chosen to have 600 nodes per hidden layer instead of 60 and the deep model was chosen to have 12 hidden layers instead of 3.

The profit/loss results of these results are displayed in Table 5.2, Figure 5.13 and Figure 5.14.

Figure 5.13: Profit and Loss with Sensitivity Hedging with Different Volatilities

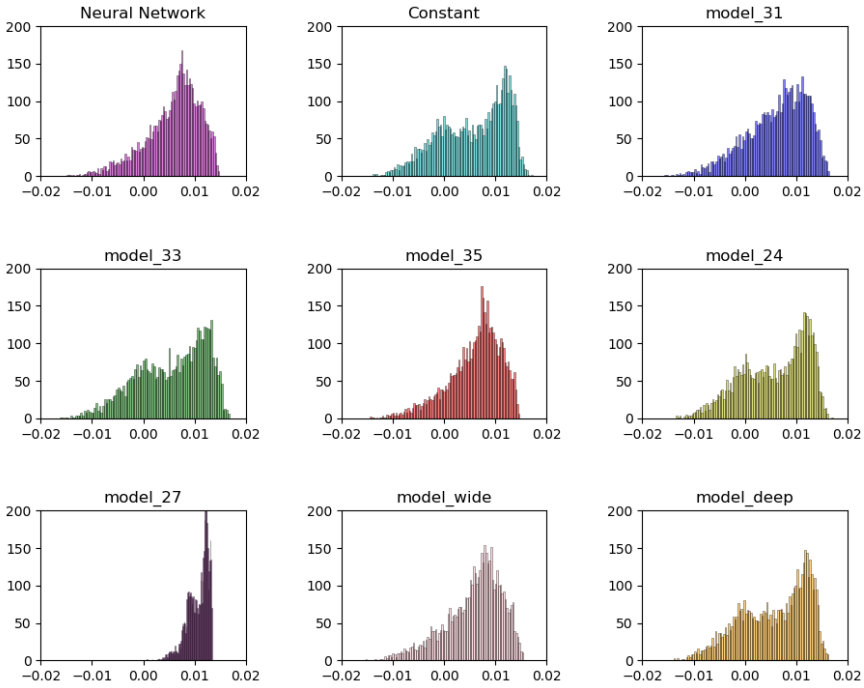


Figure 5.14: Profit and Loss with Sensitivity Hedging with Different Volatilities

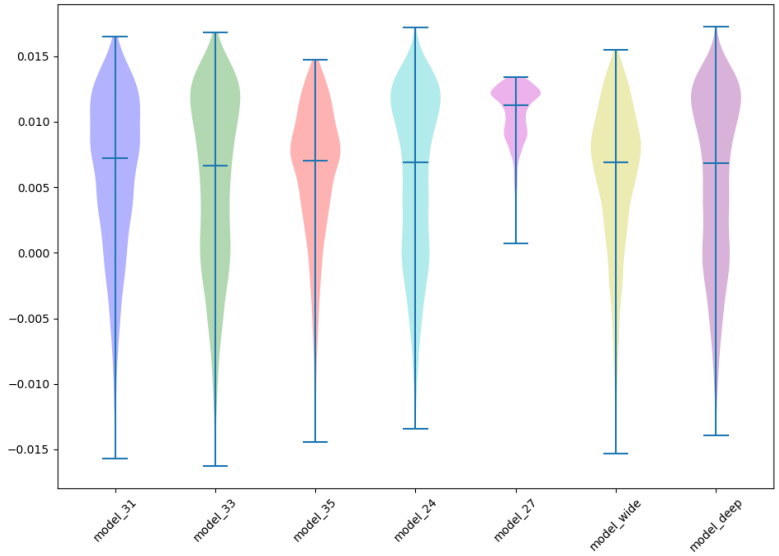


Table 5.2: Summary of hedging results for multiple models

Model	Hedge	Mean	Standard Deviation	MSRE
Neural Network	Sensitivity	0.005799	0.005276	6.1478×10^{-5}
Constant	Sensitivity	0.005657	0.006529	7.4635×10^{-5}
model 31	Sensitivity	0.006150	0.005924	7.2933×10^{-5}
model 33	Sensitivity	0.005475	0.006737	7.5369×10^{-5}
model 24	Sensitivity	0.005741	0.006438	7.4417×10^{-5}
model 27	Sensitivity	0.010657	0.002071	0.000117
model 35	Sensitivity	0.005961	0.005162	6.2193×10^{-5}
model wide	Sensitivity	0.005786	0.005538	6.4157×10^{-5}
model deep	Sensitivity	0.005657	0.006529	7.4635×10^{-5}

From these results we can conclude a few things.

The final chosen neural network model was indeed the best neural network model with an MSRE of 6.1478×10^{-5} when hedged via sensitivity hedging.

Model 27 which had the most stable training and validation loss proved to have very low variance, however, due to the convergence being relatively weak it ended up with a poor MSRE score and mean further from 0 than all other models.

The models 31, 33 and 24 were chosen with quite sporadic training and/or validation losses per epoch. These models displayed that if you were to choose in this way, even if the training loss were good, the MSRE would be weak.

Model 35 had all good training aspects but were not at the same level as the final chosen model. This model fended well for itself, however, although proving better than the constant volatility model it did not outperform the final chosen neural network model.

The wide model incapsulates the chosen model if we were to set many of the weights to 0. Thus, training this model well proved to be quite competitive in the hedging. However, due to the complexity of the model it may have been too large to have finished training and we would most likely see convergence to the final chosen model if trained for longer.

The deep model had the most weights of any model. In fact, looking at the results displayed, the model performed exactly the same as the constant volatility model. The mean, variance and MSRE of the deep model and that of the constant volatility model were exactly the same. The models deepness blinded the training process and caused the first few layers' weights to be set to 0 which resulted in a constant function which matched that of the constant volatility.

Chapter 6

Conclusions

6.1 Dissertation results conclusion

This dissertation has proven that the Libor market model constructed in Chapter 3 is a powerful tool when dealing with interest rate derivatives. The focus of this dissertation was to investigate the viability of a neural network volatility structure in the Libor market model. This was achieved through theory developed in section 3.5 which described how to calibrate these volatility structures to Swaptions. Additionally, a new variational autoencoder was introduced and used to generate more "real world" sample paths to test our model. Astonishingly, this variational autoencoder learned to describe the yield curve with a latent space of one, contrary to the belief that it takes at least three factors to do so.

Upon hedging a simple caplet via buying and selling forward rate agreements in the market, it was concluded that the neural network volatility model outperformed the constant volatility model. This conclusion has led to the belief that a neural network volatility structure is a viable choice when pricing and hedging options.

In the dissertation thus far, we have mainly given the positive feedback on our models. Before ending with the possible extensions, let us revisit the model and discuss the pitfalls.

6.2 Pitfalls

1. The art of machine learning

   The task of choosing a specific neural network configuration for any circumstance has become an artistic front. Whether it be a classifcation problem or a regression problem, neural networks come in many forms. The forms that one can choose from range between simple feed-forward neural networks, recurrent neural networks, convolutional neural networks and many more. Each neural network form comes with advantages and disadvantages and these should be weighed against which is more preferential for the problem at hand.

   After deciding what form of neural network one would like to use, there are many other questions one would need to answer before continuing. The artistic nature of the problem stems from these questions.

   The questions that need to be answered are:

   (a) How many layers should the network have?

   (b) How many nodes should exist within each layer?

(c) Which activation functions should be used on each layer?

(d) Should the network be regularised and if so what should the regularisation parameter be?

(e) What optimisation algorithm should be used?

Many of these questions can be answered via a simple grid search optimisation, however, considering all possible combinations is not feasible since there an infinite number of combinations eg. The set of all possible activation functions is uncountably infinite.

A good artist should always prepare their pallet prior to painting anything, this is the same for machine learning. Answering these questions efficiently requires us to select a few sets of optimal answers for these questions and test each combination in turn to decide which is better.

This procedure of educated guesses could be quite tricky.

2. Hyper-parameter tuning

Hyper-parameter tuning is a task set out to find the correct combination of hyper-parameters.

Choosing the correct hyper-parameters to tune falls into the pitfall previously discussed, however, physically performing this task needs a mention of its own.

As mentioned previously, a simple grid search will do the trick once the parameters have been chosen.

Setting up a grid search is farely straight forward: apply each of the combination of parameters, train the model using them and calculate a validation loss. Upon choosing a metric, one can choose which of the parameters are best.

Choosing the number of epochs to use for each combination and knowing how much time each would take may be a critical step in this procedure. Hyper-parameter tuning is very costly in terms of time and if not enough epochs are chosen, you may not get a clear picture on which is the optimal set.

Another pitfall in hyper-parameter tuning stems from the training and testing procedure performed. Depending on the outcome you would like to achieve, it isn't always as clear cut as K-fold validation etc. As an example, in our procedure, the lack of data required us to use different functional forms of the data found separately for the training and validation sets.

3. Long training duration

Training a parametric volatility model requires long durational training sessions for the optimisation.

If long dated maturity options are the goal, then many forward rates will be required. If many rates are needed, the time for training can increase dramatically.

Depending on the goal, simplicity or accuracy, these training sessions could be quite a burden.

4. Overfitting of neural networks

   Overfitting occurs when a model is trained on data that is either very scarce or too noisy. A model that has too many parameters or is overly complex for the problem at hand can result in overfitting.

   Overfitting results in a highly inaccurate model as the predictions do not match that of reality.

   One can tell if a model is overfitted when it produces accurate results on the training data and performs poorly on unseen data.

   The goal of any machine learning algorithm is to generalise well to unseen data.

   There are plenty of ways to avoid/minimise overfitting. Choosing which technique one should use sometimes depends on the problem itself and can be quite tricky to decide which is the best or most accurate in the situation.

   Let's briefly discuss a few techniques one could use to mitigate overfitting. These techniques each have their own advantages and disadvantages which we shall not discuss here. These techniques are as follows:

   - Simplifying The Model

     The first step when dealing with overfitting is to try and accommodate the rule of parsimonious models.

     Parsimonious models are simple models with great explanatory predictive power. They explain data with a minimum number of parameters, or predictor variables. The goal is to find the simplest model that results in the predictive power necessary to answer the questions at hand.

     Thus, the first step to control overfitting is to decrease the complexity of the model. There are many things one can do to achieve a more parsimonious model. The most obvious complexity control is to remove layers or reduce the number of neurons to make the network smaller.

     There is no general rule on how much to remove or how large your network should be, however, if the network is overfitting it is likely to be overly complex.

   - Early Stopping

     Early stopping is a technique used as an alternative to regularization while training a model in an iterative manner. Since all neural networks learn exclusively by using gradient descent, early stopping can be used for any problem.

     While training a model, it is commonly seen that the model tends to fit the test data sufficiently well up to a certain point whilst the training loss usually continues to decrease overtime. The generalization loss decreases as the model begins to learn the pattern from the data but there exists a point at which the generalization loss starts to increase. It is best to stop the training process when the model begins to decrease

in generalization ability.

Early stopping rules provide guidance as to how many iterations can be run before the model begins to overfit.

- Use Regularization

  Regularization is a technique to reduce the complexity of the model. It does so by adding a penalty term to the loss function. The most common techniques are known as L1 and L2 regularization:

  L1 regularization:

$$L(\theta) = C(\theta) + \lambda \sum_{i=1}^{n} |\theta_i|$$

  L2 regularization:

$$L(\theta) = C(\theta) + \lambda \sum_{i=1}^{n} \theta_i^2$$

  Where C represents the original cost function and L represents the penalized loss function.

  In table 6.1, we compare both the regularization techniques.

Table 6.1: Differences in types of regularizaion techniques

L1 Regularization	L2 Regularization
L1 penalizes the sum of absolute values of the parameters	L2 penalizes the sum of squared values of the parameters
L1 generates a simple and interpretable model	L2 is able to learn more complex patterns in data
L1 is robust to outliers	L2 is not robust to outliers

- Use Dropouts

  Dropout os a form of regularization which deals with modifying the neural network itself and not just the cost function.

  Regularization methods like L1 and L2 reduce overfitting by modifying the cost function. Dropout on the other hand, modifies the network itself.

  Dropout randomly selects neurons from the model during each training iteration and sets there weights temporarily to zero i.e. we drop the impact of the neuron on the predictive ability.

  Dropping different sets of neurons each at each epoch is equivalent to training different neural networks. Each neural network will overfit in a different way, thus the net effect of dropout will end in a reduced overfit for the model itself.

To conclude we discuss possible extensions relating to the dissertation as a whole.

6.3 Extensions

1. Apply proposed variational autoencoder in other generative situations

   There are many situations where data generation could be of use.

   VAEs are typically used within feature extraction and image generation.

   In this dissertation, we focused on neither of these tasks individually. We were tasked to extract import latent features solely for its generative ability.

   The VAE proposed in this dissertation could be of much use in each of these fields and it would be interesting to conduct research further on each of them individually.

   The surface was only scratched with regard to the capabilities this new proposed VAE has.

2. Test wider and/or deeper network structures

   In most situations we are compelled to find a parsimonious model, however, in other situations a complex model is necessary.

   It would be intriguing to test neural networks of alternative depths and widths to see what would happen within each situation.

   Testing these differences could lead to discovering certain properties one could be interested in when encountering different problems.

3. Test more volatility structures against our model

   In this dissertation, we only compared our model to the standard constant variance model.

   The constant variance model is not known to be the best model to exist. It would be beneficial to not only know the viability of our model but also the strengths and weaknesses against the other models that exist.

4. Hedge some exotic derivatives with longer terms such as those embedded in insurance products eg. Guaranteed Annuity Options

   In this dissertation, a relatively short term option was chosen.

   Choosing a long dated security will enable us to test not only the capability of our model in the long-run but also to test the rebonatto approximation when it comes to longer termed interest rate options.

   Insurance products are typically the longest options available to hedge. Along with testing our models and the approximation, this would open up avenues into its usefulness to everyday problems in the industry.

5. Compare more hedging strategies

   Many more trading strategies exist within the world of mathematical finance. It would be naive to assume that the two hedging strategies we used were the best available.

One such hedging strategy that could give an interesting outcome would be that using a neural network to predict holdings at trading date. Many different neural networks exist which gives us many different avenues to look into.

Bibliography

[Black and Karasinski, 1991] Black, F. and Karasinski, P. (1991). *Bond and Option Pricing When Short Rates Are Lognormal*. Financial Analysts Journal.

[Brace et al., 1997] Brace, A., Gatarek, D., and Musiela, M. (1997). *The market model interest rate dynamics*. Mathematical Finance.

[Brennan and Schwartz, 1977] Brennan, M. J. and Schwartz, E. S. (1977). *SAVINGS BONDS, RETRACTABLE BONDS AND CALLABLE BONDS*. Journal of Financial Economics.

[Brigo and Mercurio, 2006] Brigo, D. and Mercurio, F. (2006). *Interest Rate Models – Theory and Practice*. Springer.

[Buehler et al., 2018] Buehler, H., Gonony, L., Teichmanny, J., Wood, B., and Mohan, B. (2018). *Deep Hedging: Hedging Derivatives Under Generic Market Frictions Using Reinforcement Learning*. Swiss Finance Institute.

[Cox et al., 1985] Cox, J. C., Ingersoll, J. E., and Ross, S. A. (1985). *A THEORY OF THE TERM STRUCTURE OF INTEREST RATES*. Econometrica.

[du Jardin, 2010] du Jardin, P. (2010). *Predicting bankruptcy using neural networks and other classification methods: the influence of variable selection techniques on model accuracy*. Munich Personal RePEc Archive.

[Fanning et al., 1995] Fanning, K., Cogger, K. O., and Srivastava, R. (1995). *Detection of Management Fraud: A Neural Network Approach*. International Journal of Intelligent Systems in Accounting, Finance & Management.

[Galeshchuk, 2015] Galeshchuk, S. (2015). *Neural networks performance in exchange rate prediction*. Elsevier B.V.

[Gebbie, 2020] Gebbie, T. (2020). *Modelling Random Events*. University of Cape Town.

[Glasserman, 2003] Glasserman, P. (2003). *Monte Carlo Methods in Financial Engineering*, pages 166–84. Springer.

[Hamdan et al., 2015] Hamdan, A. R., Vadoodparast, M., and Hafiz (2015). *Fraudulent Electronic Transaction Detection Using Dynamic KDA Model*. International Journal of Computer Science and Information Security.

[Hornik, 1989] Hornik, K. (1989). *Multilayer Feedforward Networks are Universal Approximators*. Pergamon Press plc.

[Hou et al., 2017] Hou, X., Shen, L., Sun, K., and Qiu, G. (2017). *Deep Feature Consistent Variational Autoencoder*. IEEE Computer Society.

[Hull and White, 1990] Hull, J. and White, A. (1990). *Pricing Interest-Rate-Derivative Securities*. The Review of Financial Studies.

[Hutchinson et al., 1994] Hutchinson, J. M., Lo, A. W., and Poggio, T. (1994). *A Nonparametric Approach to Pricing and Hedging Derivative Securities Via Learning Networks*. The Journal of Finance.

[Kozdraj, 2009] Kozdraj, T. (2009). *Using Artificial Neural Networks to Predict Stock Prices*. ACTA UNIVERSITATIS LODZIENSIS.

[Kryzanowski et al., 1993] Kryzanowski, L., Galler, M., and Wright, D. W. (1993). *Using Artificial Neural Networks to Predict Stock Prices*. Financial Analysts Journal.

[Kunita and Watanabe, 1967] Kunita, H. and Watanabe, S. (1967). *On Square Integrable Martingales*. Nagoya Mathematical Journal.

[Mavuso, 2018] Mavuso, M. (2018). *Order out of Chaos 2*. University of Cape Town.

[Mavuso, 2019a] Mavuso, M. (2019a). *Mathematical Finance*. University of Cape Town.

[Mavuso, 2019b] Mavuso, M. (2019b). *Order out of Chaos 1*. University of Cape Town.

[Mavuso, 2019c] Mavuso, M. (2019c). *Stochastic Calculus*. University of Cape Town.

[Moodliyar, 2016] Moodliyar, L. (2016). *Calibrating The Libor Market Model To Swaptions With An Extension For Illiquidity In South Africa*. African Institute of Financial Markets and Risk Management - University of Cape Town.

[Moody and Utans, 1994] Moody, J. and Utans, J. (1994). *Architecture Selection Strategies for Neural Networks: Application to Corporate Bond Rating Prediction*. Oregon Graduate Institute.

[O'Leary, 1998] O'Leary, D. E. (1998). *Using Neural Networks to Predict Corporate Failure*. International Journal of Intelligent Systems in Accounting, Finance & Management.

[Ouwehand, 2020] Ouwehand, P. (2020). *Probability Theory*. African Institute of Financial Markets and Risk Management - University of Cape Town.

[Parot et al., 2018] Parot, A., Michell, K., and Kristjanpoller, W. D. (2018). *Using Artificial Neural Networks to forecast Exchange Rate, including VAR-VECM residual analysis and prediction linear combination*. International Journal of Intelligent Systems in Accounting, Finance & Management.

[Qi, 1996] Qi, M. (1996). *Handbook of Statistics 14: Statistical Methods in Finance*, chapter 18, pages 544–547. Elsevier Science BV.

[Rendleman and Bartter, 1980] Rendleman, R. J. and Bartter, B. J. (1980). *The Pricing of Options on Debt Securities*. The Journal of Financial and Quantitative Analysis.

[Rifai et al., 2011] Rifai, S., Vincent, P., Muller, X., Glorot, X., and Bengio, Y. (2011). *Contractive Auto-Encoders: Explicit Invariance During Feature Extraction*. International Conference on Machine Learning.

[Robbertze, 2019] Robbertze, Y. (2019). *Mean-Variance Hedging with Recurrent Neural Networks in Incomplete Markets*. University of Cape Town.

[Swamy et al., 1997] Swamy, K., Pashley, M., and Gilbert, E. (1997). *Neural Network Applications in Finance*. Conference paper.

[Tan, 2000] Tan, R. (2000). *Credit Rating Prediction Using Self-Organizing Maps*. Erasmus University Rotterdam: Faculty of Economics.

[Vasicek, 1977] Vasicek, O. (1977). *An equilibrium characteristic of the term structure.* Journal of Financial Economics.

[Xu et al., 2017] Xu, W., Sun, H., Deng, C., and Tan, Y. (2017). *Variational Autoencoder for Semi-Supervised Text Classification.* Association for the Advancement of Artificial Intelligence.