Traditional and ML approaches to generate and understand implied volatility surfaces

by

Redouan Ochalhi

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Tuesday June 8, 2021 at 3:00 PM.

Student number:	4582764	
Project duration:	Sep 1, 2020 – May 31, 2021	
Thesis committee:	Prof. dr. ir. C.W. Oosterlee,	TU Delft, Responsible Full professor
	Dr. T. Nane,	TU Delft
	Dr. Ir. M.B. van Gijzen,	TU Delft
	N. Mucke MSc.	CWI Amsterdam, Daily supervisor
	T. Gragert MSc.	True Partner Capital
	A. Theiakos MSc.	True Partner Capital

Preface

First of all, I would like to take the opportunity to thank True Partner Capital which, despite the COVID-19 crisis, made it possible for me to carry out my research remotely. In particular, I would like to thank Thorsten Gragert and Alexios Theiakos for their critical eye and dedication throughout this project. I also would like to thank Nikolaj Mucke for his help regarding the implementation of the models and his knowledge of neural networks. In addition, I would like to thank Kees Oosterlee for his academic and critical perspective and his skills which steered the project in the right direction. Finally, I would like to thank Tina Nane and Martin van Gijzen for being part of the examination committee.

Redouan Ochalhi
Delft, June 2021

Abstract

In this research, different models are used to construct volatility surfaces and these models are compared with each other in terms of accuracy. The models range from the SSVI to neural networks. Specifically, we look at the SSVI, the feedforward neural network and the gated neural network. Attention is also paid to the incorporation of financial conditions in the considered models. In addition, we propose a framework which uses two neural networks in combination with the weighted mean squared error as a loss function to construct a volatility surface belonging to one trading day. We found out that this approach appears to be very accurate and outperforms all the other approaches. The thesis is structured in such a way that it starts with the construction of a volatility surface for 1 trading day, later in the research for the sake of examining the robustness of the models we used data belonging to several trading days.

Contents

Chapter 1

Introduction

1.1 Thesis statement

For traders and market makers in financial markets all over the world, the implied volatility surface is a very useful tool. Traders for example, use implied volatilities in option pricing to determine if they believe a financial product is under or overpriced. From the well-known Black-Scholes Model [1], which takes as input the volatility, strike and maturity of an option contract to compute the option value, it is possible to calculate the implied volatility by using a numerical method if we know the option value, but not the volatility. If this numerical method is applied for different combinations of strikes and expirations, it becomes clear that one of the conditions of the Black-Scholes Model is not being met, namely that the volatility σ should have been constant. In this way, it is possible to define a volatility surface based on the Black-Scholes Model, thus creating a 3-dimensional plot. The way investors use this surface is by pricing options based on the implied volatility corresponding to the expiration and strike of an option contract. Investors could form an opinion based on the construction of the volatility surface. However, the question then arises as to how such a volatility surface moves over time and when a recalibration of the chosen model is required.

In recent years, a wide range of techniques has been proposed to calibrate implied volatility surfaces. Nowadays, traditional methods such as *Surface Stochastic Volatility Inspired* (SSVI) [2] and simple spline-based algorithms are the ones that are widely used in practice. However, the demand for more accurate and robust techniques is increasing. As a result, scholars tend to investigate the applicability of different types of neural networks. The aim of this research is to compare different models with respect to their ability to fit the market while adhering to financial conditions such as no-arbitrage conditions. At the same time, the models will also be linked to the criteria that are important for a trader to use or not use a particular model, like the accuracy of the model at certain areas of a volatility curve.

Specifically, emphasis will be put on a wide range of different models that attempt to describe a volatility surface. It is interesting that it is possible to bring in financial conditions that have to be met in practice into a model to make sure that such a model takes them into account. One can think of conditions like arbitrage or limit conditions that such a surface must meet. In addition, the robustness of a model is of paramount importance in determining to what extent a model can be used in practice and thus can have added value for an investor. To validate this, a large amount of data is needed to test the models for different market conditions. During this research, the following models will be studied:

- SSVI model: a parametric formula which is calibrated by using real market data [3].
- Feedforward neural network: Network which is proposed in [4].
- Gated neural networks: two different models are described in [5].
  - Single gated neural network model
  - Multi gated neural network model

  This thesis considers two different gated neural network models. One of them is known as the 'single model' and the other one is known as the 'multi-model'.

The focus will lie on how to implement these models, understand them, adjust them according to investor's desires and find good metrics to compare the output of these models.

1.2 Literature

In practice, we can distinguish between two types of models for an implied volatility surface: the *indirect models* and the *direct models* [6]. The indirect models are those that depend on other dynamic models, including stochastic volatility models and Lévy models [7]. However, we see in practice that this type of model is not always usable [7, 8, 9, 10]. The methods we will be concerned with are the direct models, where implied volatility will be explicitly defined. Here again we can distinguish between static direct methods and dynamic direct methods. Dynamic direct methods make certain assumptions about the course of the surface over time [11], while static direct methods use a parametric approach.

A model that is used extensively in the industry today is one that takes the form of the so-called *Stochastic Volatility Inspired* (SVI) model [2]. For a predetermined expiration, a slice is constructed using a parametric formula, the underlying parameters of which must in turn be optimized using an optimization method. After the invention of this method, a simplification has been sought regarding the incorporation of no-arbitrage conditions. This has led to the revelation of the Surface Stochastic Volatility Inspired, also abbreviated as SSVI [3], which is popular among practitioners. Beyond this model, non-parametric methods are also available that use polynomials of sigmoid functions [12]. Moreover, several Machine Learning Models have also been applied to the field of volatility surfaces. The application of different types of neural networks (NN) [5, 13, 14, 4] is advocated in recent years. Most of these research papers focus on the performance, robustness and accuracy in comparison with traditional models, such as the SSVI method.

In [15] kernel machines are utilized to sort out the volatility surface for option pricing. In [16] an Artificial Intelligence (AI) based model is proposed to demonstrate that the risk-aversion character of an investor explains the implied volatility smile. In [17] an approach is presented to predict implied volatility surfaces using regression trees and the loss function is minimized using more trees. A point that emerges in each of the Machine Learning based studies is that Machine Learning models depend heavily on good and accurate data. Due to the growing amount of data and greatly improved computing power in recent years, it is certainly an area where attention should be paid.

1.3 Thesis overview

First of all, an important choice will be about the different technologies to use to conduct the experiments. To construct a neural network using Python, TensorFlow 2.0, Keras [18] and SciKit-Learn [19] are libraries that are widely used for dealing with concepts such as complicated configurations, custom-loss functions and efficient neural network training. In order to perform the experiments in this thesis, it is imperative that we have access to the data needed to enable the models to create volatility surfaces. To obtain this data, it should be sorted out which source to use and how to aggregate this data to obtain the desired features. This thesis also focuses on the complications of using neural networks, such as overfitting and underfitting. In addition, attention will also be paid to making sure that a model will perform excellently in certain regions, as this may be a requirement from traders.

In Chapter 2 we will present the mathematical backbone of the different models we will discuss. Next, in Chapter 3, we will apply the models to a limited data set. This will give us an idea of the computational nature of the models and in addition an impression of how to set the models with respect to the parameters. In Chapter 4 we will then aim to find a robust hyperparameter set by means of a hyperparameter optimization. In chapter 4, we will also spend time on a dual neural network approach which uses the weighted mean squared error to find out if we can arrive at fits with good accuracy. Chapter 5 concludes the research conducted in this thesis.

Chapter 2

Theoretical Background

In this chapter, we will describe the financial and mathematical background of all the models and concepts that will be used throughout the research. Specifically, this chapter will also focus on the SSVI Model, neural networks and the numerical solvers used to arrive at reasonable results. In addition, proofs of some mathematical results to strengthen the understanding of all models are shown.

2.1 Options Theory

As mentioned earlier, the focus of this research will be on financial options. There are two different types of options: call options and put options. A European call option gives the holder the right to buy, but not the obligation to buy an asset, at a specified price K, also known as the *strike*, at some specific time of *maturity* T. This means that the payoff of such an option at maturity T with stock-price S_t at time t can be written as $\max(S_T - K, 0)$.

A European put option gives holders of such an option the right, but not the obligation, to sell a specified amount of an underlying asset at a specified price K, also known as the strike, at some specific time of maturity T. This means that the payoff of such an option at maturity T can be written as $\max(K - S_T, 0)$. *In The Money* (ITM) options refer to options where their intrinsic value, $\max(S_t - K, 0)$ in case of a call option and in case of a put option $\max(K - S_t, 0)$, is positive at time t, whereas *Out The Money* (OTM) options refer to options for which the intrinsic value is equal to zero.

A mathematically more challenging derivative is the American option, which gives the holder the right to exercise at any time before the maturity. More specifically, this means that the holder has more freedom in choosing when to exercise the option.

In order to price European options, some model should be fitted that aims to describe the price-path S_t of the asset with W_t^{P} a stochastic variable known as the Brownian motion. One way to do this is by assuming a Geometric Brownian Motion [20]. The dynamics of such a price-path can be described as

$$dS_t = \mu S_t dt + \sigma S_t dW_t^{\mathrm{P}}, \tag{2.1}$$

where μ and σ are constants and are known as the *drift* and the *volatility* parameters, respectively. The following Partial Differential Equation (PDE) for the European option value $V(S_t)$ can be derived assuming that the return on the riskless asset equals r and the volatility of asset returns σ is constant and that the stock does not pay dividends:

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S_t^2 \frac{\partial^2 V}{\partial S_t^2} + r S_t \frac{\partial V}{\partial S_t} - rV = 0,$$

where

$$V^C(S_T) = \max(S_T - K, 0),$$

in case of a call option with maturity T and

$$V^P(S_T) = \max(K - S_T, 0),$$

in case we are dealing with a put option with maturity T. For the sake of completeness, a derivation of this PDE is included in this thesis and will be supported by lemma 1. In [21] a derivation of this equation is shown as follows:

Lemma 1. *Suppose S_t follows an Ito process:*

$$dS_t = \mu S_t dt + \sigma S_t dW_t,$$

with μS_t the drift term and σS_t the diffusion term. Let now $g(t, S_t)$ be a function with continuous derivatives (up to second order). Then $V(S_t) := V_t = g(t, S_t)$ follows an Ito process with the same Brownian Motion process W_t,

$$dg_t = \left(\frac{\partial g}{\partial t} + \mu S \frac{\partial g}{\partial S} + \frac{1}{2}(\sigma S)^2 \frac{\partial^2 g}{\partial S^2}\right) dt + \sigma S \frac{\partial g}{\partial S} dW_t.$$

To evaluate how the option value $V(S_t)$ evolves as a function of the asset price S and time t, lemma 1 can be applied. This results in the following expression:

$$dV_t = \left(\mu S \frac{\partial V}{\partial S} + \frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2}\right) dt + \sigma S \frac{\partial V}{\partial S} dW_t. \tag{2.2}$$

Consider a portfolio that consists of one short option and $\frac{\partial V}{\partial S}$ long shares of the asset at time t. This portfolio is also known as the *delta-hedge* portfolio. The value Π of this portfolio can be described by:

$$\Pi_t = -V_t + \frac{\partial V}{\partial S} S_t.$$

Now consider the total change in profit during some time period $[t, t + \Delta t]$, denoted by $\Delta \Pi_t$:

$$\Delta \Pi_t = -\Delta V_t + \frac{\partial V}{\partial S} \Delta S_t.$$

Now take the limit $\Delta t \to 0$, then we write dt, dS, dV and $d\Pi$ instead of Δt, ΔS, ΔV and $\Delta \Pi$, respectively. This leads to the following expression:

$$d\Pi = -dV + \frac{\partial V}{\partial S} dS. \tag{2.3}$$

The expressions for dS (2.1) and dV (2.2) should now be substituted in the expression for $d\Pi$ (2.3). This results in the following expression:

$$d\Pi = \left(-\frac{\partial V}{\partial t} - \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} \right) dt.$$

Note that the Brownian Motion process increment dW is deleted, so the portfolio is riskless. Assume that our portfolio with value Π has a return that is equal to the return r of an asset without any risk involved. So we have that:

$$r\Pi dt = d\Pi.$$

As we possess two expressions for $d\Pi$, we can put them both in an equation:

$$\left(-\frac{\partial V}{\partial t} - \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} \right) dt = r \left(-V + S\frac{\partial V}{\partial S} \right) dt.$$

Simplifying the equation leads to the desired result:

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0. \tag{2.4}$$

Subsequently, a closed-form expression can also be found for call and put options [22] by solving (2.4). The well-known Black-Scholes formulas for a call price V^C and a put price V^P are defined as follows:

$$V^C = S_0 N(d_1) - Ke^{-rT} N(d_2), \tag{2.5}$$

$$V^P = Ke^{-rT} N(-d_2) - S_0 N(-d_1), \tag{2.6}$$

where

$$d_1 = \frac{\ln \frac{S_0}{K} + \left(r + \frac{\sigma^2}{2} \right) T}{\sigma\sqrt{T}}, \tag{2.7}$$

$$d_2 = d_1 - \sigma\sqrt{T}, \tag{2.8}$$

$T - t$ represents the time left for the option to expire, S_0 stands for the price of the asset when purchasing an option, r denotes the interest rate, K is the strike of the option and N represents the cumulative distribution function (cdf) of a standard normal distribution.

2.1.1 Implied Volatility Surface

The Black-Scholes equation derivation relies heavily on a constant volatility σ, while in practice it is shown that σ is not constant. So the assumption that σ is constant does not hold in practice. The Black-Scholes equation is often used for quoting option prices in terms of the underlying volatility, which serves as a risk indicator. More specifically, this means that every combination of strike K and maturity T has its own volatility value $\sigma_{BS}(T, K)$ that can be used in risk calculations. The implied volatility is written as a function of the strike K and the time to maturity T:

$$\sigma_{BS}(T, K) : (T, K) \mapsto \sigma_{BS}(T, K)$$

The implied volatility surface is also described using the *log-moneyness* k, which is defined as $k = \ln(\frac{K}{S_t})$, and the expiration T:

$$\sigma_{BS}(T, k) : (T, k) \mapsto \sigma_{BS}(T, k).$$

Implied volatilities, when shown as a function of strike K or log-moneyness k, show a local maximum for extreme strike prices which results in a smile. As a result, when discussing implied volatilities for a fixed maturity T, one frequently uses the term 'volatility smile'. In figure 2.1, a simple example of a volatility smile for some expiration T can be observed.

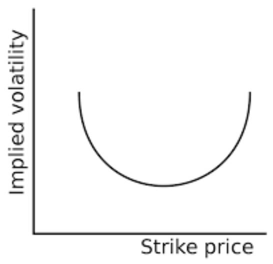


Figure 2.1: A simplistic example of a volatility smile for a given expiration T. This example is borrowed from [23].

Figure 2.2 shows us an example of how an implied volatility surface looks like. Depending on the market and type of product considered, the volatility surface's skewness and smile features differ.

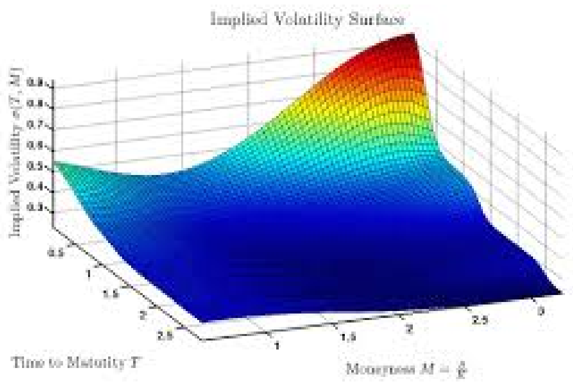


Figure 2.2: An implied volatility surface of the SI&P500 in March 1999. This example is borrowed from [11].

Now that we have been able to find an analytical expression for pricing European options using (2.5) and (2.6), it is possible to infer the implied volatility σ if S_0, r, T, K and V are available. Since there is no analytical expression for the volatility σ, the use of a numerical method is necessary in order to get an approximation for the volatility σ belonging to an option contract with log-moneyness k and expiration T. In this thesis, we will restrict ourselves to a number of simple numerical methods which will enable us to calculate the implied volatilities for a large number of option contracts. Below we describe some numerical methods which aim to solve the equation $f(\sigma) = 0$ numerically. In our case, f can be defined by bringing the right side of (2.5) and (2.6) to the left side and filling in the available data.

Bisection method

In [24], the bisection method is explained. The bisection is a simple, but a time-consuming method which can be used to obtain the root of a function f on an interval $[a, b]$, the procedure can be expressed as follows: The required input of the procedure are f, a, b, $f(a)$ and $f(b)$ and the procedure looks as follows:

1. Start with computing the midpoint by $\sigma = \frac{a+b}{2}$.
2. Compute $f(\sigma)$.
3. If convergence is satisfactory; return x and stop procedure.
4. Investigate the sign of $f(\sigma)$ and replace either $(a, f(a))$ or $(b, f(b))$ with $(\sigma, f(\sigma))$ so that 0 lies in between f(a) and f(b).

Newton's method

In [24] the Newton's method is described as well. The Newton's method to approximate roots is defined as follows:

$$\sigma_{n+1} = \sigma_n - \frac{f(\sigma_n)}{f'(\sigma_n)}. \tag{2.9}$$

The derivation of this method relies on the Taylor approximation and is structured as follows: Let σ be a root of a function$f \in C^2[a, b]$ and σ_0 be an initial approximation to σ. Using Taylor Series we have

$$f(\sigma) = f(\sigma_0) + (\sigma - \sigma_0) f'(\sigma_0) + O\left((\sigma - \sigma_0)^2\right).$$

The approximation is performed by only considering the linear terms. This leads to the following approximation:

$$f(\sigma) = 0 \approx f(\sigma_0) + (\sigma - \sigma_0) f'(\sigma_0),$$

$$\sigma - \sigma_0 \approx -\frac{f(\sigma_0)}{f'(\sigma_0)} \Rightarrow \sigma \approx \sigma_1 = \sigma_0 - \frac{f(\sigma_0)}{f'(\sigma_0)}$$

This leads to the iteration formula (2.9).

Secant Method

The Secant Method is also defined in [24] and is defined by the following recurrence relation:

$$\sigma_n = \sigma_{n-1} - f(\sigma_{n-1}) \frac{\sigma_{n-1} - \sigma_{n-2}}{f(\sigma_{n-1}) - f(\sigma_{n-2})} = \frac{\sigma_{n-2} f(\sigma_{n-1}) - \sigma_{n-1} f(\sigma_{n-2})}{f(\sigma_{n-1}) - f(\sigma_{n-2})}.$$

This method is derived by the following bunch of operations. Let's start with initial points σ_0 and σ_1 and build a line through $(\sigma_0, f(\sigma_0))$ and $(\sigma_1, f(\sigma_1))$. The equation corresponding to this line can be represented by

$$y = \frac{f(\sigma_1) - f(\sigma_0)}{\sigma_1 - \sigma_0} (\sigma - \sigma_1) + f(\sigma_1). \tag{2.10}$$

The root of (2.10) is equal to

$$\sigma = \sigma_1 - f(\sigma_1) \frac{\sigma_1 - \sigma_0}{f(\sigma_1) - f(\sigma_0)}.$$

The following step would be to use this value as our σ_2 and repeat the same process using σ_1 and σ_2 instead of σ_0 and σ_1. The process stops when the convergence criteria are met; when the difference between two consecutive function values is small enough.

Now that we can obtain the corresponding implied volatility for each contract, we are able to construct a volatility surface for combinations of the log-moneyness k and the expiration T. A bunch of trading strategies depend on certain assumptions being made for a volatility surface. For this reason one uses models to approximate surfaces and smiles. For example for the SSVI model, where it is assumed that a volatility smile can be described by a number of parameters and is widely used in practice because of the ability of these models to satisfy financial conditions analytically. Neural networks are also a possible way of modeling implied volatility surfaces, because of the possibility of setting up the so-called cost function in such a way that it ensures that certain conditions that a trader wants to meet can easily be forced. Therefore, in this thesis, two types of neural networks will be considered: gated neural networks and feedforward neural networks.

2.1.2 Put-Call parity

There is a relationship between European call options and European put options with the same strike price and expiration, which is widely known as the *Put-Call parity.* Mathematically, the relationship can be described by

$$V^C - V^P = D(F - K) \tag{2.11}$$

where V^C denotes the call price, V^P stands for the put price, K the strike, D is a discount factor, which can be computed by using the interest rate r and maturity T as $D = e^{-rT}$ and F is the forward price of the asset. The present value of the asset is given by

$$S = D \cdot F.$$

Intuitively, (2.11) means that a portfolio consisting of a long call option and a short put option has the same value as a forward contract at the same strike price K and expiration T.

2.2 SSVI-Model

One of the models to construct volatility surfaces that is widely used is one that is parametric in nature. This model is also known as the SSVI model. Modelling of the SSVI is performed by parametrizing the total implied variance. In this section, we will summarize the theorems, definitions and lemmas described in [3].

2.2.1 Notation

- In this thesis, $\sigma_{\text{BS}}(k,T)$ will be referred to as the volatility surface, where T represents the time left to expiration and k is the *log-moneyness* which is defined as $\ln(\frac{K}{S_t})$, where K is the strike and S_t the price of the underlying asset at time t.
- The 2-dimensional function:

$$IV(k,T) = \sigma^2_{\text{BS}}(k,T) \cdot T,$$

  will be referred to as the *total implied variance surface.*
- For a specific time to maturity T, $\Lambda(k)$ represents a *total implied variance slice* and is defined as $\Lambda(k) = IV(k,T)$.

2.2.2 Static arbitrage

If a portfolio has value 0, with zero probability of becoming a negative value, and a non-negative probability of increasing the portfolio value, we speak of an *arbitrage* opportunity [25]. If at some discrete point in time, there is an arbitrage opportunity, we speak of *static arbitrage.* Regarding the SSVI model, it is possible to find certain conditions for the parameters to satisfy no-arbitrage conditions. In [3], theorem 1, lemma 2 and theorem 2 are introduced and proved.

Theorem 1. *A total implied variance surface is free of static arbitrage iff the following holds*

1. *The surface is free of calendar spread arbitrage,*
2. *Every slice is free of butterfly arbitrage.*

Lemma 2. *The total implied variance surface IV is free of calendar spread arbitrage, if and only if* $\frac{\partial IV(k,T)}{\partial T} \geq 0$, *for all* $k \in \mathbb{R}$ *and* $T > 0$.

In [3], it is stated that theorem 2 relies on the function $g : \mathbb{R} \to \mathbb{R}$, which is defined by:

$$g(k) := \left(1 - \frac{k\frac{\partial \Lambda}{\partial k}(k)}{2\Lambda(k)}\right)^2 - \frac{(\frac{\partial \Lambda}{\partial k}(k))^2}{4}\left(\frac{1}{\Lambda(k)} + \frac{1}{4}\right) + \frac{\frac{\partial^2 \Lambda}{\partial k^2}(k)}{2}, \tag{2.12}$$

and reads as follows:

Theorem 2. *A total implied variance slice* Λ *is butterfly arbitrage-free, if and only if* $g(k) \geq 0$ *for all* $k \in \mathbb{R}$ *with* $g(k)$ *as in (2.12) and* $\lim_{k\to+\infty} -\frac{k}{\sqrt{\Lambda(k)}} + \frac{\sqrt{\Lambda(k)}}{2} = -\infty$

Proof. The structure of the proof is similar to [3]. From [26], it follows that the probability density p from the Black-Scholes formula V^C as in (2.5), can be derived by:

$$p(k) = \left.\frac{\partial^2 V^C(k, \Lambda(k))}{\partial K^2}\right|_{K=S_t e^k}, \quad \text{for any } k \in \mathbb{R}.$$

Applying explicit differentiation results in:

$$p(k) = \frac{g(k)}{\sqrt{2\pi\Lambda(k)}} \exp\left(-\frac{-k/\sqrt{\Lambda(k)} \pm \sqrt{\Lambda(k)}/2^2}{2}\right)$$

This density does not necessarily integrates to 1. The following asymptotic conditions should be imposed to enforce this: $\lim_{k\to+\infty} -\frac{k}{\sqrt{\Lambda(k)}} + \frac{\sqrt{\Lambda(k)}}{2} = -\infty$. [27] can be referenced to observe why this should be enforced. ■

2.2.3 Formulations

In an attempt to parametrize the total implied variance slice Λ, a number of models has been created. Gatheral [2] published the stochastic volatility implied or *SVI* parametrization of the implied volatility smile, which has become very popular. We will start with the original raw SVI formulation proposed in [2], then move on to some alternate (but equivalent) formulations which are introduced in [3]. The reason for utilizing different formulations is because it is extremely difficult to establish conditions for the raw parameterization to prevent arbitrage, whereas these conditions for alternate formulations, such as the natural parameterization and the SVI-Jump Wings parameterization, are much easier to find.

The raw SVI parametrization

The total implied variance slice Λ can be modelled by the *raw SVI* parametrization; the aim of this formulation is to find appropriate values for the parameters set: $\chi_R = \{a, b, \rho, m, \sigma\}$, and the formulation from [2] reads as follows:

$$\Lambda(k; \chi_R) = a + b\left\{\rho(k-m) + \sqrt{(k-m)^2 + \sigma^2}\right\},$$

where $a \in \mathbb{R}$, $b \geq 0$, $|\rho| < 1$, $m \in \mathbb{R}$, $\sigma > 0$ and the condition $a + b\sigma\sqrt{1-\rho^2} \geq 0$, which ensures that the total implied variance slice is not negative. In [3], it is stated that a major advantage of using a parameterization is that changes in the parameters will have an expected effect:

- Increasing a results in a vertical movement of the smile.
- Increasing b tightens the smile by increasing the slope of the left and right wing.
- Increasing ρ decreases the slope of the left wing.
- Increasing m moves the smile to the right.
- Increasing σ reduces the curvature in the smile.

In figure 2.3, an example of the effect of changes in the parameters, can be found.

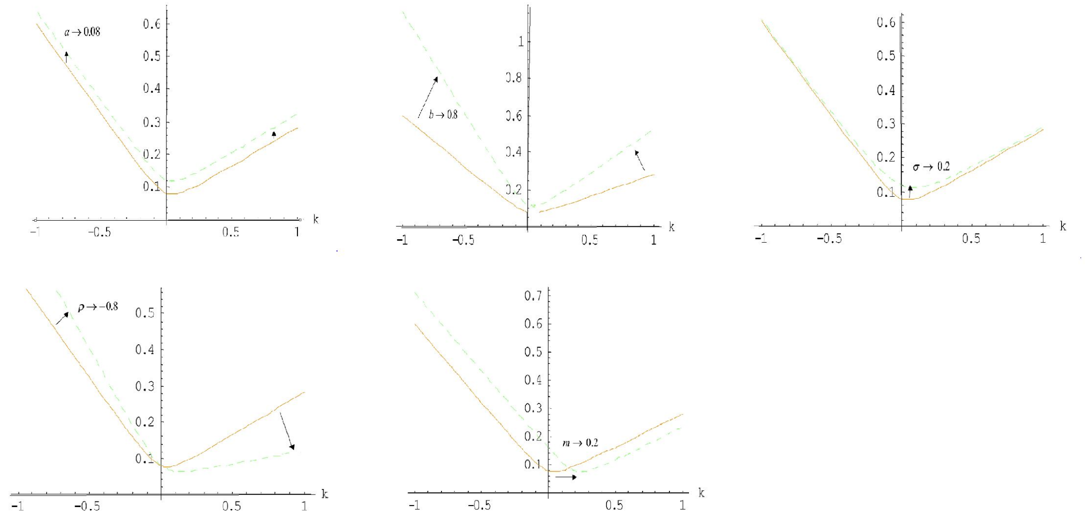


Figure 2.3: Effect of the changes of the Raw parametrization parameters on the implied volatility surface. This example is borrowed from [2].

The natural SVI parametrization

The natural SVI parametrization is an alternative formulation introduced in [3] and can be used to parametrize the total implied variance slice Λ as well. The aim of this formulation is find appropriate values for the parameters set: $\chi_N = \{\Xi, \mu, \rho, \omega, \zeta\}$, and the formulation reads as follows:

$$\Lambda\left(k; \chi_N\right) = \Xi + \frac{\omega}{2}\left\{1 + \zeta\rho(k-\mu) + \sqrt{(\zeta(k-\mu)+\rho)^2 + (1-\rho^2)}\right\},$$

where $\omega \geq 0$, $\Xi \in \mathbb{R}$, $\mu \in \mathbb{R}$, $|\rho| < 1$ and $\zeta > 0$.

There is a strong relation between the parameters of the Heston model, a model which describes the movement of the volatility of an underlying asset, and the natural SVI parameterization [28]. In [3], an equivalence relationship between the natural and raw parametrization, which is described by lemma 3, is introduced.

Lemma 3. *The following mapping should be applied to find a mapping of parameters between the raw and the natural SVI:*

$$(a, b, \rho, m, \sigma) = \left(\Xi + \frac{\omega}{2}\left(1-\rho^2\right), \frac{\omega\zeta}{2}, \rho, \mu - \frac{\rho}{\zeta}, \frac{\sqrt{1-\rho^2}}{\zeta}\right)$$

and its inverse transformation, between the natural and the raw SVI:

$$(\Xi, \mu, \rho, \omega, \zeta) = \left(a - \frac{\omega}{2}\left(1-\rho^2\right), m + \frac{\rho\sigma}{\sqrt{1-\rho^2}}, \rho, \frac{2b\sigma}{\sqrt{1-\rho^2}}, \frac{\sqrt{1-\rho^2}}{\sigma}\right)$$

The natural parametrization is not as useful implementation wise as the the SVI-Jump Wings parameterization.

The SVI Jump-Wings parametrization

The final parameterization is one that may look somewhat complex at first glance, but can certainly be very useful in practice. The *SVI Jump-Wings* parametrization is defined from the raw parametrization. The SVI-JW parameters, $\chi_J = \{v_T, \psi_T, p_T, c_T, \widetilde{v}_T\}$, are defined for some time to expiry $T > 0$ from the raw SVI parametrization as follows:

$$
\begin{aligned}
v_T &= \frac{a + b\left\{-\rho m + \sqrt{m^2 + \sigma^2}\right\}}{T}, \\
\psi_T &= \frac{1}{\sqrt{v_T \cdot T}} \cdot \frac{b}{2}\left(-\frac{m}{\sqrt{m^2+\sigma^2}} + \rho\right), \\
p_T &= \frac{1}{\sqrt{v_T \cdot T}} \cdot b(1-\rho), \\
c_T &= \frac{1}{\sqrt{v_T \cdot T}} \cdot b(1+\rho), \\
\widetilde{v}_T &= \frac{1}{T}\left(a + b \cdot \sigma \cdot \sqrt{1-\rho^2}\right).
\end{aligned}
$$

Interesting fact about this parameterization is that it explicitly depends on the expiration T and can be seen as a generalization of the raw SVI parametrization. The parameters can be interpreted as follows:

- v_t gives the total implied variance at the log-moneyness value 0.
- ψ_t gives a measure for the skew at the log-moneyness value 0.
- p_t gives a measure of the slope of the left wing.
- c_t gives a measure of the slope of the right wing.
- $\widetilde{v}_t$ is the minimum total implied variance that can be attained.

There is also an inverse mapping which is introduced and proved in [3]. Lemma 4 provides the inverse mapping for the SVI Jump Wings parametrization.

Lemma 4. *Assume that $m \neq 0$. For any $T > 0$, define the following quantities:*

$$\beta := \rho - \frac{2\psi_T\sqrt{v_T \cdot T}}{b} \quad \text{and} \quad \alpha := \operatorname{sign}(\beta)\sqrt{\frac{1}{\beta^2} - 1},$$

under the assumption that $\beta \in [-1, 1]$. In this case, the raw SVI and SVI-Jump Wings parameters are related in the following way:

$$
\begin{aligned}
b &= \frac{\sqrt{v_T \cdot T}}{2}\left(c_t + p_T\right), \\
\rho &= 1 - \frac{p_T\sqrt{v_T \cdot T}}{b}, \\
a &= \tilde{v}_T T - b\sigma\sqrt{1-\rho^2},
\end{aligned}
$$

$$m = \frac{(v_T - \tilde{v}_T)\, T}{b\left\{-\rho + \operatorname{sign}(\alpha)\sqrt{1+\alpha^2} - \alpha\sqrt{1-\rho^2}\right\}},$$

$$\sigma = \alpha m.$$

If $m = 0$ then the formula above for b, ρ and a still holds, but $\sigma = \frac{(v_T T - a)}{b}$.

Now that we have defined the parametrizations, we can look at the static arbitrage conditions. A condition for the absence of calendar spread arbitrage is established by lemma 5 and again obtained from [3].

Lemma 5. *The raw SVI-parametrization is free of calendar-spread arbitrage if polynomial (2.13 in the proof) has no real root.*

Proof. The structure of the proof is similar to the one that is given in [3]. Assume that for every two expirations T_1 and T_2 for which holds that $T_1 < T_2$, the corresponding slices Λ^{T_1} and Λ^{T_2}, respectively, do not have an overlapping point. These two slices can be characterized by two parameters sets, namely χ_1 and χ_2:

$$\chi_1 := \{a_1, b_1, \sigma_1, \rho_1, m_1\}$$

and

$$\chi_2 := \{a_2, b_2, \sigma_2, \rho_2, m_2\}.$$

So it is obvious that we need to determine the roots of the following equation:

$$\Lambda^{T_1}(k) = \Lambda^{T_2}(k).$$

Using the raw SVI parametrization, this can be rewritten to

$$a_1 + b_1\left\{\rho_1(k-m_1) + \sqrt{(k-m_1)^2 + \sigma_1^2}\right\} = a_2 + b_2\left\{\rho_2(k-m_2) + \sqrt{(k-m_2)^2 + \sigma_2^2}\right\}.$$

Rearranging leads to

$$2b_2(\alpha + \beta k)\sqrt{(k-m_2)^2 + \sigma_2^2} = b_1^2\left\{(k-m_1)^2 + \sigma_1^2\right\} - b_2^2\left\{(k-m_2)^2 + \sigma_2^2\right\} - (\alpha + \beta k)^2,$$

where $\alpha := a_2 - a_1 + b_1\rho_1 m_1 - b_2\rho_2 m_2$ and $\beta := b_2\rho_2 - b_1\rho_1$.

Note that squaring this equation leads to the following polynomial:

$$\pi_4 k^4 + \pi_3 k^3 + \pi_2 k^2 + \pi_1 k + \pi_0 = 0. \tag{2.13}$$

The coefficients π_1, π_2, π_3 and π_4 can be expressed in terms of the parameter set. Note that in case the derived equation has no real root, then the considered slices do not overlap and we have no calendar spread arbitrage. ■

SSVI model

In practice, it seems to be quite hard to impose conditions on the parameters of each of the parametrizations which ensure that the parametrization leads to static-free arbitrage. In order to accomplish this, in [3] a new class of volatility surfaces is introduced, which is known as the Surface Stochastic Volatility Inspired (SSVI) model. This extension of the SVI parametrization gives us the space to choose parameter sets that guarantee the absence of the aforementioned arbitrages.

For each maturity $T \geq 0$, the *At the Money* (ATM) total implied variance is defined as:

$$\theta_T := IV(0, T)$$

and serves as an important component of the SSVI surface, which is obtained from [3] and reads as follows:

Definition 1. *Let φ be a continuous and differentiable function from $\mathbb{R}_+^*$ to $\mathbb{R}_+^*$ such that the limit $\lim_{T\to 0} \theta_T \varphi(\theta_T)$ exists in $\mathbb{R}$. The SSVI surface for expiration T, which is a parametrization for the total implied variance slice $\Lambda_T(k)$ for expiration T, is then defined by:*

$$\Lambda_T(k) = \frac{\theta_T}{2}\left\{1 + \rho\varphi(\theta_T)k + \sqrt{(\varphi(\theta_T)k + \rho)^2 + (1-\rho^2)}\right\}. \tag{2.14}$$

It can be shown that the SSVI model can be related to the natural parametrization by the following parametrization: $\chi_N = \{0, 0, \rho, \theta_T, \varphi(\theta_T)\}$ [29].

In [3] it is stated that intuitively the SSVI surface is free of calendar arbitrage if the skew in total variance terms is monotonically increasing in trading time and the skew in implied variance is monotonically decreasing in trading time. Theorem 3, which is again introduced and proved in [3], formalizes this intuition.

Theorem 3. *The SSVI surface (2.14) is free of calendar spread arbitrage if and only if*

1. $\frac{\partial\theta_T}{\partial T} \geq 0$, *for all* $T \geq 0$.
2. $0 \leq \frac{\partial(\theta\varphi(\theta))}{\partial\theta} \leq \frac{1}{\rho^2}\left(1 + \sqrt{1-\rho^2}\right)\varphi(\theta)$, *for all* $\theta > 0$.

In [3], the relation between SVI-Jump Wings parametrization and SSVI is also defined. The relation between the SVI-JW parametrization and SSVI can be defined through lemma 6 and is a consequence of what we have seen in lemma 3 and 4.

Lemma 6. *The SVI-JW parameters related to the SSVI surface are defined as*

$$\begin{aligned}
v_T &= \theta_T/T, \\
\psi_T &= \frac{1}{2}\rho\sqrt{\theta_T}\varphi(\theta_T), \\
p_T &= \frac{1}{2}\sqrt{\theta_T}\varphi(\theta_T)(1-\rho), \\
c_T &= \frac{1}{2}\sqrt{\theta_T}\varphi(\theta_T)(1+\rho), \\
\widetilde{v}_T &= \frac{\theta_T}{T}\left(1-\rho^2\right).
\end{aligned}$$

We will now move to the sufficient conditions that we could impose on the SSVI surface to get rid of butterfly arbitrage which are introduced in [3] and summarized in theorem 4, lemma 7 and lemma 8.

Theorem 4. *The SSVI volatility surface is free of butterfly arbitrage if the following points are met for all $\theta > 0$:*

1. $\theta\varphi(\theta)(1+|\rho|) < 4$
2. $\theta\varphi(\theta)^2(1+|\rho|) \leq 4$

Proof. To gain some insights in how the proof is structured, [3] can be consulted. ■

Lemma 7. *SSVI is free of butterfly arbitrage only if*

$$\theta\varphi(\theta)(1+|\rho|) \leq 4, \quad \theta > 0$$

If it also holds that

$$\theta\varphi(\theta)(1+|\rho|) = 4$$

then the surface is free of butterfly arbitrage if

$$\theta\varphi(\theta)^2(1+|\rho|) \leq 4$$

Proof. By using the function g, which is introduced in theorem 2, we have

$$g(k) = \begin{cases} \frac{16-\theta^2\varphi(\theta)^2(1+\rho)^2}{64} + \frac{4-\theta\varphi(\theta)^2(1+\rho)}{8\varphi(\theta)k} + \mathcal{O}\left(\frac{1}{k^2}\right), & k \to +\infty \\ \frac{16-\theta^2\varphi(\theta)^2(1-\rho)^2}{64} - \frac{4-\theta\varphi(\theta)^2(1-\rho)}{8\varphi(\theta)k} + \mathcal{O}\left(\frac{1}{k^2}\right), & k \to -\infty \end{cases}$$

The remaining part of the proof follows by inspection of these two equations. ■

The following lemma serves as a result of all the lemmas and theorems that we have seen so far.

Lemma 8. *SSVI is free of static arbitrage if the following points are satisfied:*

1. $\frac{\partial\theta_T}{\partial T} \geq 0$, *for all* $T > 0$
2. $0 \leq \frac{\partial(\theta\varphi(\theta))}{\partial\theta} \leq \frac{1}{\rho^2}\left(1+\sqrt{1-\rho^2}\right)\varphi(\theta)$, *for all* $\theta > 0$
3. $\theta\varphi(\theta)(1+|\rho|) < 4$, *for all* $\theta > 0$
4. $\theta\varphi(\theta)^2(1+|\rho|) \leq 4$, *for all* $\theta > 0$

The remaining obvious question would be how to choose φ appropriately. It can be shown that certain choices $\varphi(\theta)$ can guarantee a static free surface if their parameters are chosen wisely. For example, the choice

$$\varphi(\theta) = \frac{\eta}{\theta^\gamma(1+\theta)^{1-\gamma}}, \tag{2.15}$$

gives a surface that is completely free of static arbitrage provided that $\eta(1+|\rho|) \leq 2$.

2.3 Feedforward neural networks

In this section, we aim to cover the basics of machine learning, with a focus on the feedforward neural network model. The properties and setup of this model will be explained, followed by an explanation of how this model can be used to construct volatility surfaces.

2.3.1 Introduction

Machine learning is a type of data analysis that automates the creation of analytical models. It's a field of artificial intelligence based on the premise that computers can learn from data, recognize patterns, and make judgments with little or no human input. The neural network is one model that has attracted a lot of attention in recent decades. This is due to the ability to make significant advancements in areas like speech recognition, text mining, and image processing [30, 31]. Because of the fast-paced nature of financial markets and the widespread usage of different models, practitioners are always on the lookout for models that can accurately describe specific moves at any particular time. In recent years, for example, research has been performed into the use of neural networks in the construction of volatility surfaces. This entails searching for suitable neural networks configurations, which maximizes the accuracy. This accuracy is determined by an objective function, also known as the *cost function* that can be specified by a practitioner's knowledge and competence.

In machine learning, there are two types of learning: supervised learning and unsupervised learning. Supervised learning is about learning an input-output mapping based on input-output examples. Specifically, this means that each data sample is an object of input values and desired output value. Based on these, an approximation method is sought that should approximate the output values based on input values and which should be accurate for unseen data; so it requires to generalize from the training data to unseen data, which makes it extremely hard in case of lack of training data. On the contrary, unsupervised learning is not based on input-output examples, but is about recognizing patterns in unlabeled data. In this thesis, supervised learning will be considered to construct volatility surfaces. The inputs are represented by strike prices and expirations and the outputs are represented by implied volatilities, which are a result of applying a numerical method in combination with the Black-Scholes formula to obtain the implied volatilities.

An early version of a neural network is the feedforward/regular neural network. One advantage compared to traditional fitting methods is that neural networks are flexible in discerning hidden patterns in raw data. A network can be subdivided into different components, better known as *layers*. The first layer is the input layer and the last layer is the output layer, which define the input and the output, respectively. The other layers are called *hidden layers*. Each of the layers consist of neurons. In a neural network setting, each neuron receives inputs, uses some function to process this input and passes the output forward. This processing function is known as the *activation function*. The activation function is chosen based on domain context. The neurons are connected by edges which carry some weight. The activation function computes the input of a neuron using a weighted sum from the outputs of its predecessor neurons and the weights for the connections between the neuron and its predecessors. As an example, in figure 2.4 the input x_4 of a neuron is calculated as follows:

$$x_4 = A(w_1x_1 + w_2x_2 + w_3x_3),$$

where A represents the activation function and w_i represents the weight corresponding to input x_i.

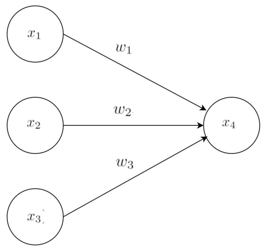


Figure 2.4: Example to illustrate how to use the activation function

Figure 2.5 shows a neural network with 1 hidden layer, 3 hidden neurons, 5 input neurons and 1 output neuron.

The weights of a neural network can be trained by using a so-called *backpropagation algorithm* to construct implied volatility surfaces. After applying a numerical method to calculate implied volatilities for the available option data, one might choose to train a neural network. However, choices have to be made in terms of choosing the right hyperparameters, such as the number of hidden layers, the number of neurons, activation function and other hyperparameters. In addition, a trader has to make the choice whether each expiration will be modeled separately, i.e. a 1-dimensional model that tries to construct a volatility smile for a certain expiration $T > 0$, or whether the trader will try to fit a 2-dimensional model, i.e. a model that takes the log-moneyness k and the expiration T as input. The choice of the cost function is important as it basically indicates how a trader evaluates a fit and what properties a fit should satisfy. Another network that can be used to generate volatility surfaces is the gated neural network, which is an example of a complex architecture and appears to be very accurate at constructing volatility surfaces.

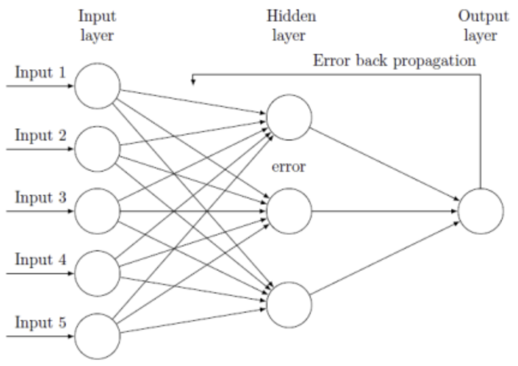


Figure 2.5: A visual representation of a neural network. This figure is obtained from [32].

2.3.2 Activation functions

As mentioned earlier, the activation function is an important component of a neural network, because it is closely related to the output that a neural network can take. In addition, the choice of certain activation function ensures that the learning process can run smoother and better. The following activation functions $A(x)$ are the commonly used ones:

- **Sigmoid activation function**

  The *sigmoid* function it is particularly useful in models where a probability must be predicted as an output. The function looks as follows:

$$A(x) = \frac{1}{1+e^{-x}}.$$

- **Tanh activation function**

  This function is known as the hyperbolic tangent *tanh* activation function. This activation returns values between -1 and 1. Depending on the phenomenon that is studied this could be a useful activation function. The advantage of using a tanh function is that negative inputs will result in a negative output, and zero inputs will result in an output close to zero. This activation function is defined as follows:

$$A(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}.$$

- **ReLU activation function**

  The most used activation function is the *ReLU* activation function. The key benefit of employing the ReLU function over other activation functions is that it does not simultaneously activate all of the neurons. This means the neurons will only be silenced if the linear transformation's output is less than 0. Another benefit is the avoidance of vanishing gradients [33]. The ReLU function can not output values that are negative and is described as follows:

$$A(x) = \max(0, x).$$

- **SoftPlus activation function**

  The sigmoid function is the *SoftPlus* function's derivative. Except around 0, where the SoftPlus is smooth and differentiable, ReLU and SoftPlus are basically comparable. ReLU and its derivative are more simple and faster to compute than the SoftPlus activation function. The SoftPlus activation function looks as follows:

$$A(x) = \ln\left(1 + e^{x}\right).$$

2.3.3 Learning through backpropagation

Before starting to learn appropriate weights for some neural network structure, some hyperparameters should be fixed. For example, the number of hidden layers, the number of neurons in each of the hidden layers and the batch size are choices that should be made in advance. Training a neural network is about computing appropriate weights to ensure that the difference between reality and the neural network is as minimal as possible. More specifically, the learning process will rely on a function that takes into account the difference between the desired output and the output of the neural network. The function that computes this error is called the *cost function* or *loss function.* The backpropagation algorithm aims to find the weights that minimize this error of a neural network. Different optimization algorithms require gradients of the model, which are computationally expensive to compute. The backpropagation algorithm computes gradients efficiently and fast and is mathematically defined by repeated use of the chain rule [34]. Different backpropagation procedures are all structured in such a way that they aim to find w that minimizes some objective function:

$$Q(w) = \frac{1}{n}\sum_{i=1}^{n} Q_i(w),$$

where i stands for the i-th observation in the data set and each function Q_i is associated with the i-th observation in the data set.

(Stochastic) Gradient Descent

The *(Stochastic) Gradient Descent* (SGD) [35] updates the weights by means of the following iteration formula:

$$w := w - \eta \nabla Q(w) = w - \frac{\eta}{n}\sum_{i=1}^{n} \nabla Q_i(w),$$

where η represents the learning rate.

The SGD is an extension of the Gradient Descent [36] where the whole training data set is used to adjust the weights, whereas in the case of using the SGD a batch with size known as the *Batch size* is passed to the iteration formula. As the size of this batch is smaller than the whole data set, the SGD is much faster compared to the Gradient Descent algorithm.

ADAgrad algorithm

Another algorithm that is widely used is the *ADAgrad* [37], which was first published in 2011. The learning rate η is still part of the algorithm, but the difference lies in the fact that this learning rate is multiplied with a vector G which can be obtained by

$$G = \nabla Q_i(w) \cdot \nabla Q_i(w)^T.$$

The weights w are then updated by

$$w := w - \eta \, \mathrm{diag}(G)^{-\frac{1}{2}} \circ \nabla Q_i(w).$$

Note that $\circ$ denotes an element-wise product.

Adagrad is mainly used for applications, where there is sparse data. This has to do with the fact that the learning rate adapts to the parameters. In other words, weights which belong to important features will have a low learning rate, and weights which belong to unimportant features will have a high learning rate.

2.3.4 Cost function

The cost function is an essential component of the learning process. A neural network is trained using a batch sample; n samples from the original data set that will be propagated through the network, where n is known as the batch size. The cost function is essential, because of the fact that it is, in a sense, responsible for determining whether a neural network is performing well during the training process. The following cost functions are commonly used:

1. The *Mean Absolute Error* (MAE) measures the average absolute value of the differences between the desired values and model values. This metric looks as follows:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i| \,,$$

   where y_i denotes the desired value for the i-th obervation and $\hat{y}_i$ the model value belonging to the i-th observation.

2. The *Mean Squared Error* (MSE) measures the average square of the differences between the desired values and model values. This metric looks as follows:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \,,$$

   where y_i denotes the desired value for the i-th obervation and $\hat{y}_i$ the model value belonging to the i-th observation.

3. The *Root Mean Squared Error* (RMSE) measures the square root of the errors of our batch size. This metric looks as follows:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} (y_i - \hat{y}_i)^2},$$

   where y_i denotes the desired value for the i-th obervation and $\hat{y}_i$ the model value belonging to the i-th observation.

There is also a possibility to create a customized cost function, which incorporates the opinion of a practitioner about a good fit.

2.3.5 Overfitting and Underfitting

Overfitting is a widely known problem within machine learning. Overfitting occurs when an algorithm learns the structure and noise of a data set and is not able to generalize well with regard to unseen data. In more detail, this entails that the noise that is part of the data set will be embedded in the model. One possible way to prevent overfitting is by dividing the data in at least two parts;

a training part and a validation part; to see how the trained model behaves on the validation data set. The model should be trained until it behaves properly for both data sets. Another way to prevent overfitting is by augmenting the considered data set. We can use several algorithms [38, 39] that study the data set and come up with a sampling model to augment the data set. Removing some layers or adjusting the number of neurons in each layer could also help to prevent overfitting. Another way to prevent overfitting, is using regularization. This is a technique which is used to create a less complex model. We will discuss two regularization techniques: L1-Regularization and L2-Regularization. L1-Regularization works by adding a fraction of the sum of the absolute values of the weights of a neural network to the loss function. L2-regularization works by adding a fraction of the sum of the squared weight values to the loss function. This fraction is known as the regularization term λ. In this thesis, we will only consider L2-regularization. On the contrary, underfitting refers to a model that does not represent the phenomena that is studied. Underfitting occurs when we observe that the neural network's training error is significantly larger than the expected error. Figure 2.6 shows an example of overfitting, underfitting and an ideal example.

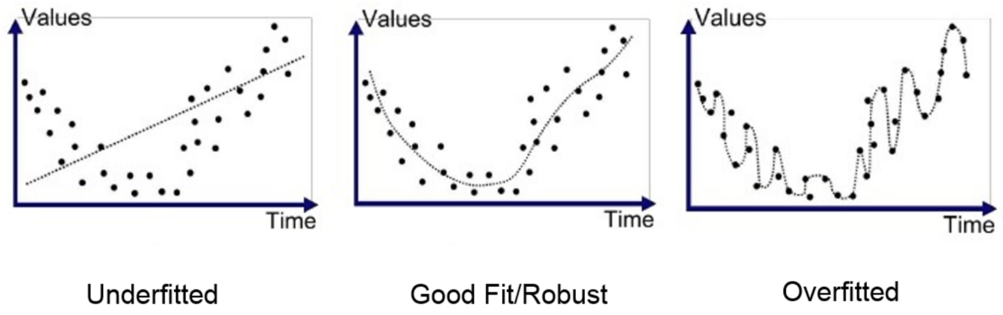


Figure 2.6: The first representation clearly shows that the model is not correct, the second plot shows an ideal fit, while the third representation shows us an example of overfitting. This example is borrowed from [40].

2.4 Gated neural network

The gated neural network presented in [5] is one that has caught our attention given the accuracy of the gated neural network compared to benchmark models, such as the SSVI. For this reason, the choice is made to take a closer look at the gated neural network. The gated neural network presented in this paper is known for its complex architecture which is specifically intended to model implied volatility surfaces $\hat{\sigma}_{BS}(k, T)$. The focus will be on two types of gated neural networks, namely multi gated neural networks and single gated neural networks. The multi gated neural network is constructed in such a way that it consists of several single gated neural networks. A schematic overview of a multi gated neural network is shown in figure 2.7. In this figure, we see 3 different single gated neural networks and a 2-dimensional feedforward neural network which outputs 3 weights. Subsequently, these 3 weights are combined with the outputs of the single gated neural neural networks as a weighted sum. The single gated neural network in figure 2.7

is described as follows: The expiration and the log-moneyness are both linked to 3 neurons and then a different activation function is applied for both the neurons linked to the expiration and the neurons linked to the log-moneyness. Each of the 3 neurons belonging to the expiration are then multiplied with one of the 3 neurons belonging to the log-moneyness and an additional weight term is incorporated in this multiplication. These 3 terms are then added up.

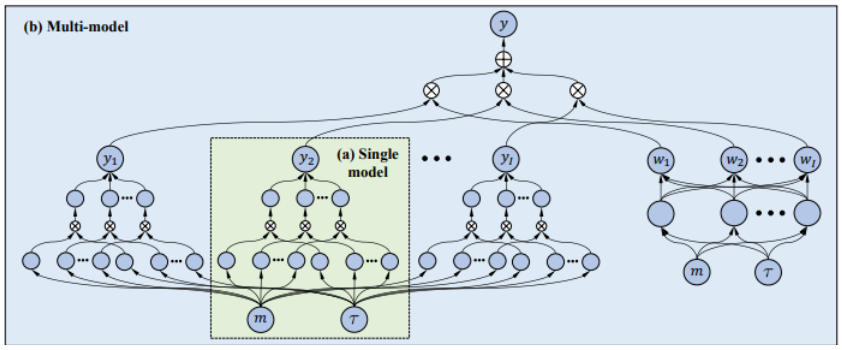


Figure 2.7: Multi gated neural network, borrowed from [5]. Note that in this figure m represents the log-moneyness and τ stands for the expiration.

Mathematically, the single gated neural network is defined in [5] and is used to model the implied volatility surface $\hat{\sigma}_{BS}(k,T)$, as follows:

$$\hat{\sigma}_{BS}(k,T) = \sum_{j=1}^{J} \phi\left(k\bar{W}_{1,j} + \bar{b}_j\right) \psi\left(T\tilde{W}_{1,j} + \tilde{b}_j\right) \mathrm{e}^{\hat{W}_{j,1} + \mathrm{e}^{\hat{b}}},$$

where $\psi(\cdot)$ is the sigmoid function, J is the number of neurons in the hidden layer, $\bar{b}, \bar{W}, \tilde{b}\ \tilde{W}, \hat{W}, \hat{b}$ are network parameters which correspond to bias and weight factors and $\phi(\cdot)$ is known as the *smile function* and is defined as follows:

$$\phi(z) = \sqrt{z \tanh\left(z + \frac{1}{2}\right) + \tanh\left(-\frac{1}{2}z + \epsilon\right)}, \quad z \in \mathbb{R}.$$

In figure 2.8, we can observe a visual representation of the smile function. This function represents a skew pattern which is similar to the structure of a volatility smile. Figure 2.9 shows a visual representation of a single gated neural network.

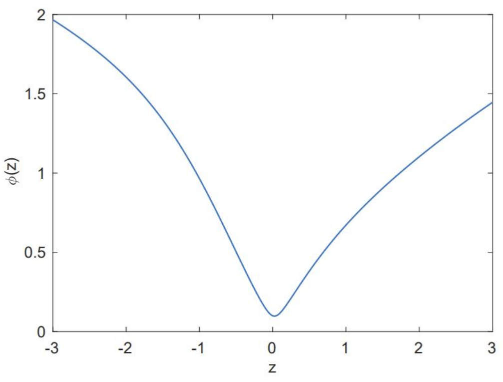


Figure 2.8: Smile function $\phi(z)$ where $\epsilon = 0.01$. This example is borrowed from [5].

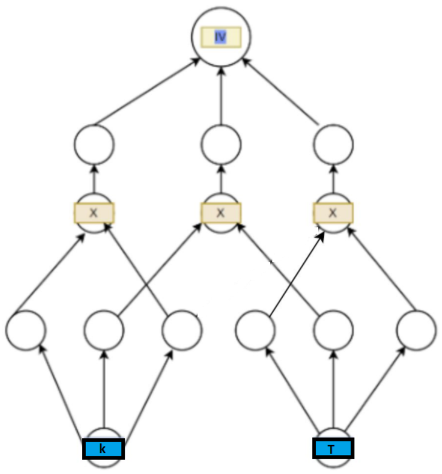


Figure 2.9: Single gated neural network. Note that in this specific case $J = 3$

One possible reason for the success of the gated neural network is that this network incorporates a custom activation function $\phi(z)$, which provides a proper structure for constructing volatility surfaces. As mentioned earlier, the single gated neural network serves as a starting point for a more complex architecture, which will be referred to as the multi gated neural network. This model is introduced in [5] and can be expressed as follows:

$$\hat{\sigma}_{BS}(k,T) = \sum_{i=1}^{I} y_i(k,T) w_i(k,T), \tag{2.16}$$

$$y_i(k,T) = \sum_{j=1}^{J} \phi\left(k\bar{W}_{1,j}^{(i)} + \bar{b}_j^{(i)}\right) \psi\left(T\tilde{W}_{1,j}^{(i)} + \tilde{b}_j^{(i)}\right) \mathrm{e}^{\hat{W}_{j,1}^{(i)}} + \mathrm{e}^{\hat{b}^{(i)}}, \tag{2.17}$$

$$w_i(k,T) = \frac{\mathrm{e}^{\sum_{a=1}^{K} \psi\left(k\dot{W}_{1,a}+T\dot{W}_{2,a}+\dot{b}_a\right)\ddot{W}_{a,i}+\ddot{b}_i}}{\sum_{i=1}^{I} \mathrm{e}^{\sum_{a=1}^{K} \psi\left(k\dot{W}_{1,a}+T\dot{W}_{2,a}+\dot{b}_a\right)\ddot{W}_{k,i}+\ddot{b}_i}}, \tag{2.18}$$

where $\dot{W}, \dot{b}, \ddot{W}, \ddot{b}$ are the newly added parameter terms of the network for weighting single models. The dimensions of $\dot{W}$, $b, \ddot{W}, \ddot{b}$ are $2 \times K, K \times 1, K \times I$, and $I \times 1$, respectively.

In [5], financial conditions are made part of the model development part and the training phase, in the sense that the loss-function that is used to train a neural network is composed of some mathematical properties that a volatility surface should comply with. The following mathematical properties related to the implied volatility surface $\sigma_{BS}(k,T)$ are studied: no static arbitrage, asymptotic slope, limiting boundaries. Lemma 9 is introduced in [5], where the mathematical properties are decomposed and summarized into 8 conditions which will later be embedded in a loss function. Note that these conditions are not limited to the gated neural network, but can be applied to other neural networks as well.

Lemma 9. *Let $d_{\pm}(k,T) = -\frac{k}{\sqrt{T}\hat{\sigma}_{BS}(k,T)} \pm \frac{1}{2}\sqrt{T}\hat{\sigma}_{BS}(k,T)$, and $n(.)$ be the standard normal distribution. Note that T represents the time to expiration and k stands for the log-moneyness. The following eight conditions for the implied volatility surface should hold:*

1. ***Positivity** - For $(k,T) \in \mathbb{R} \times \mathbb{R}^{+}, \hat{\sigma}_{BS}(k,T) > 0$.*

2. ***Twice Differentiability** - For $T > 0$, the function $k \to \hat{\sigma}_{BS}(k,T)$ is twice differentiable on $\mathbb{R}$.*

3. ***Monotonicity** - For $k \in \mathbb{R}, T \to \sqrt{T}\hat{\sigma}_{BS}(k,T)$ is increasing on $\mathbb{R}^{+}$, then $\hat{\sigma}_{BS}(k,T) + 2T\frac{\partial \hat{\sigma}_{BS}(k,T)}{\partial T} \geq 0$*

4. ***Butterfly Arbitrage-free** - For $(k,T) \in \mathbb{R} \times \mathbb{R}^{+}$,*
$$\left[1 - \frac{k\frac{\partial \hat{\sigma}_{BS}(k,T)}{\partial k}}{\hat{\sigma}_{BS}(k,T)}\right]^2 - \frac{1}{4}\left[\hat{\sigma}_{BS}(k,T)T\frac{\partial \hat{\sigma}_{BS}(k,T)}{\partial k}\right]^2 + \tau\hat{\sigma}_{BS}(k,T)\frac{\partial^2 \hat{\sigma}_{BS}(k,T)}{\partial k^2} \geq 0$$

5. ***Limit condition** - If $T > 0$, then*
$$\lim_{k \to +\infty} d_{+}(k,T) = -\infty$$

6. ***Right Boundary** - If $k \geq 0$, then*
$$N\left(d_{-}(k,T)\right) - \sqrt{T}\frac{\partial \hat{\sigma}_{BS}(k,T)}{\partial k} n\left(d_{-}(k,T)\right) \geq 0$$

7. ***Left Boundary** - If $k < 0$, then*
$$N\left(-d_{-}(k,T)\right) + \sqrt{T}\frac{\partial \hat{\sigma}_{BS}(k,T)}{\partial k} n\left(d_{-}(k,T)\right) \geq 0$$

8. ***Asymptotic Slope** - For $T > 0$, $2|k| - \hat{\sigma}_{BS}^2(k,T) \cdot T > 0$.*

In order to ensure that these conditions are satisfied by a gated neural network, the training procedure should take the conditions from lemma 9 into account. One way to do this is by defining a cost function that represents these conditions. The following cost function is presented in [5]:

$$\ell = \ell_0 + \gamma\ell_1 + \delta\ell_2 + \eta\ell_3 + \rho\ell_4 + \omega\ell_5, \tag{2.19}$$

where the hyperparameters γ, δ, η, ρ should be chosen carefully.

The term ℓ_0 is a joint loss function which combines the Mean Squared Log Error (MSLE) and the Mean Squared Percentage Error (MSPE. Joint loss functions are mostly used in applications with sensitive data. The function ℓ_0 is described as follows:

$$\ell_0 = \alpha\underbrace{[\frac{1}{N}\sum_{n=1}^{N}\left(\log\left(\sigma_{BS}(k_n,T_n)\right) - \log\left(\hat{\sigma}_{BS}(k_n,T_n)\right)\right)^2]}_{\text{MSLE}} + \beta\underbrace{[\frac{1}{N}\sum_{n=1}^{N}\left(\frac{\sigma_{BS}(k_n,T_n) - \hat{\sigma}_{BS}(k_n,T_n)}{\sigma_{BS}(k_n,T_n)}\right)^2]}_{\text{MSPE}},$$

where α and β are hyperparameters.

The loss-function ℓ_1 has to do with the monoticity condition and $a(k,T) := \sigma_{BS}(k,T) + 2T\frac{\partial\sigma_{BS}(k,T)}{\partial T}$, where the objective is to push $a(k,T)$ to become non-negative by sampling P values from the domain of the log-forward moneyness k and Q values from the domain of the maturity T. The term ℓ_1 is defined as:

$$\ell_1 = \sum_{p=1}^{P}\sum_{q=1}^{Q}\max\left\{0, -a\left(k_p, T_q\right)\right\},$$

where k_p represents a sampled strike and T_q represents a sampled expiration.

The second term ℓ_2 specifies the absence of butterfly arbitrage and is defined as:

$$\ell_2 = \sum_{p=1}^{P}\sum_{q=1}^{Q}\max\left\{0, -b\left(k_p, T_q\right)\right\},$$

where $b(k,T) := \left(1 - \frac{k\frac{\partial\sigma_{BS}(k,T)}{\partial k}}{\sigma_{BS}(k,T)}\right)^2 - \frac{\left(\sigma_{BS}(k,T)T\frac{\partial\sigma_{BS}(k,T)}{\partial k}\right)^2}{4} + T\sigma_{BS}(k,T)\frac{\partial^2\sigma_{BS}(k,T)}{\partial k^2}$.

The left and right boundary conditions are specified by the ℓ_3 term

$$\ell_3 = \sum_{p_1=1}^{P_1}\sum_{q=1}^{Q}\max\left\{0, -c_1\left(k_{p_1}, T_q\right)\right\} + \sum_{p_2=1}^{P_2}\sum_{q=1}^{Q}\max\left\{0, -c_2\left(k_{p_2}, T_q\right)\right\},$$

where P_1 and P_2 unique values are sampled from the domain of k and Q unique values are sampled from the domain of T and

$$c_1(k,T) = N\left(d_-(k,T)\right) - \sqrt{T}\frac{\partial\sigma_{BS}(k,T)}{\partial k}n\left(d_-(k,T)\right),$$

$$c_2(k,T) = N\left(-d_-(k,T)\right) + \sqrt{T}\frac{\partial\sigma_{BS}(k,T)}{\partial k}n\left(d_-(k,T)\right).$$

The asymptotic conditions are captured by the following loss-term ℓ_4:

$$\ell_4 = \sum_{p=1}^{P}\sum_{q=1}^{Q} \max\left\{0, -\left(g\left(k_p, T_q\right) - \epsilon\right)\right\},$$

where $g(k,T) := 2|k| - \sigma_{BS}^2(k,T)T$ and $\epsilon = 10^{-5}$ and ℓ_5 is a regularization term to avoid overfitting and is defined as:

$$\ell_5 = \left\{ \begin{array}{ll} \|\bar{W}\|_F^2 + \|\tilde{W}\|_F^2 + \|\hat{W}\|_F^2, & \text{for the single model,} \\ \sum_{i=1}^{I} \left\|\bar{W}^{(i)}\right\|_F^2 & \text{for the multi-model.} \\ +\sum_{i=1}^{I} \left\|\tilde{W}^{(i)}\right\|_F^2 & \\ +\sum_{i=1}^{I} \left\|\hat{W}^{(i)}\right\|_F^2 & \\ +\|\dot{W}\|_F^2 + \|\ddot{W}\|_F^2, & \end{array} \right\}$$

where $\|\cdot\|_F^2$ is the square of Frobenius norm, e.g., $\|W\|_F^2 = \frac{1}{2}\sum_{i=1}^{I}\sum_{j=1}^{J} W_{i,j}^2$ for an $I \times J$ weight matrix.

In this thesis, the domains for the time to expiration T and the log-moneyness k from which is sampled are $[0, 1.2]$ and $[-1.2, 1]$, respectively.

2.5 Numerical optimizers

Clearly, a variety of techniques are required to bridge the gap between theory and numerical results. Consider determining the optimal parameters for the SSVI method or finding the optimal configuration for a neural network. In this section, we will look at two approaches that can help us searching for a proper neural network setup as well as finding excellent parameters for the SSVI method.

2.5.1 Differential Evolution Algorithm

In this thesis, we will have to find the SSVI parameters that minimize or maximize a given objective function. We could achieve this using a global optimizer called the differential evolution algorithm. The differential evolution algorithm is known for the fact that it makes minimal, if any, assumptions about the problem to be solved and can search a vast number of potential solutions. Formally, let $f : \mathbb{R}^n \to \mathbb{R}$ be the fitness function which must be minimized. A basic variant starts with an initial set of possible solutions. Next, a mathematical operation is performed on each element of this set, aiming that the new combination gives a better result with respect to the measure of quality. If this is the case, the old element is replaced by the new element. Important factors that play a role in the success of this procedure are, of course, the set of candidate solutions, the measure of quality and the mathematical operation.

A procedure that is often used in practice and explained in [41] is described below:

1. Choose a random initialization for the candidate solutions set and choose a number p_{cr} between 0 and 1 and F between 0 and 2.

2. Start with picking an element A from the candidate solutions set
    - Pick three different candidate solutions at random that are distinct from each other. Call them $B = [b_1, b_2, ..., b_n]$, $C = [c_1, c_2, ..., c_n]$ and $D = [d_1, d_2, ..., d_n]$.
    - Subsequently, choose a random integer R between 1 an n, the dimension of the problem that is optimized.
    - The new position $Y = [y_1, y_2, ..., y_n]$ of A is calculated by
        - For all integers i in between 1 and n, choose a standard uniformly distributed random number and call it $r_i \sim U(0, 1)$.
        - If $r_i < p_{cr}$ or $i = R$ then set $y_i = b_i + F(c_i - di)$, otherwise $y_i = x_i$.
3. In case $f(Y) \leq f(A)$ then replace A by Y.

2.5.2 Bayesian optimization for hyperparameter optimization

One can imagine that while doing an optimization, complexities may arise when it comes to finding hyperparameter set $\mathbf{x}$ that causes the neural network to reach a minimum for the loss function. f will denote the neural network which takes as an input the hyperparameters and outputs the value of the loss function after training the neural network. One would prefer to keep the number of times we need to sample from the domains of the hyperparameter sets as small as possible because of the costs that come with training a neural network. As an example, for a neural network, we are looking for an appropriate combination of the number of hidden layers, the number of neurons, the activation function, the learning rate and the number of epochs which yields a good performance. These so-called hyperparameters must be found and a Bayesian optimization technique might assist here to provide a hyperparameter set which yields good performance. Broadly speaking, prior belief is made part of the optimization process and this prior is iteratively adjusted by means of samples drawn from the hyperparameters domain in order to eventually obtain a posterior that better estimates the neural network f and its output.

The model used to estimate f is known as the *surrogate model.* In this thesis, a Gaussian processes [42] will be used; this model is known for its low-cost evaluation nature and is thus used to propose points in the search domain that will lead to an improvement. In addition, Bayesian optimization also makes use of acquisition functions that are responsible for proposing favorable hyperparameter sets. These functions trade off between exploration and exploitation; this means that in areas where the surrogate model estimates a high value an exploitation procedure is used, while in cases that in a certain region there is quite uncertainty regarding the predictive power of the surrogate model the exploration step is activated. In [43], the following explanation of the Bayesian optimization procedure is given:

Mathematically, f is sampled at hyperparameter set at iteration t, and the chosen hyperparameter set is $\mathbf{x}_t = \arg\max_{\mathbf{x}} u\left(\mathbf{x} \mid \mathcal{D}_{1:t-1}\right)$, where u represents the acquisition function and

$$\mathcal{D}_{1:t-1} = \{\left(\mathbf{x}_1, f\left(\mathbf{x}_1\right) + \epsilon_1\right), \ldots, \left(\mathbf{x}_{t-1}, f\left(\mathbf{x}_{t-1}\right) + \epsilon_{t-1}\right)\},$$

represents the $t-1$ samples drawn from f, where ϵ_i represents noise that is added to the function value. Many different acquisition functions u could be chosen, but during this research the *Expected*

Improvement function EI will be utilized. The procedure of choosing a hyperparameter set at iteration t is described as follows:

$$\mathbf{x}_t = \arg\max_{\mathbf{x}} \mathrm{EI}(\mathbf{x}) = \arg\max_{\mathbf{x}} \mathbb{E}\left[\max\left(f(\mathbf{x}) - f\left(\mathbf{x}^+\right), 0\right) | \mathcal{D}_{1:t-1}\right],$$

where $f\left(\mathbf{x}^+\right)$ is the function value of the best hyperparameter set that is observed so far. Under the assumption that the Gaussian process is adopted as the surrogate model, it can be derived that the following closed-form expression holds [44]:

$$\mathrm{EI}(\mathbf{x}) = \begin{cases} \left(\mu(\mathbf{x}) - f\left(\mathbf{x}^+\right) - \xi\right)\Phi(Z) + \sigma(\mathbf{x})\phi(Z) & \text{if } \sigma(\mathbf{x}) > 0 \\ 0 & \text{if } \sigma(\mathbf{x}) = 0 \end{cases},$$

where

$$Z = \begin{cases} \frac{\mu(\mathbf{x}) - f\left(\mathbf{x}^+\right) - \xi}{\sigma(\mathbf{x})} & \text{if } \sigma(\mathbf{x}) > 0 \\ 0 & \text{if } \sigma(\mathbf{x}) = 0 \end{cases},$$

where $\mu(\mathbf{x})$ and $\sigma(\mathbf{x})$ represent the mean and standard deviation for a Gaussian process at hyperparameter set $\mathbf{x}$, Φ and ϕ the cumulative distribution function (cdf) of a standard normal distribution and the probability distribution function (pdf) of a standard normal distribution, respectively.

Note that ξ represents the exploration part; higher ξ leads to more exploration since improvements which are predicted by the Gaussian Process mean $u(\mathbf{x})$ are less important relative to improvements in regions where there is some high uncertainty, which is represented by a high $\sigma(\mathbf{x})$. So in order to evaluate the next hyperparameter set, we should optimize the closed-form expression. This can be done by using a method called *DIRECT*, which is introduced in [45]. After a certain amount of iterations T, one could make a choice for a hyperparameter set based on the obtained samples $\mathcal{D}_{1:T}$.

Chapter 3

Implied volatility surface construction for one trading day

In this chapter, the aim is to use the SSVI, the feedforward neural network and the gated neural network to generate implied volatility surfaces based on real market data and analyze the properties of these models. The different sections of this chapter will be dedicated to the data setup, SSVI, feedforward neural network and the gated neural network. In this chapter, we limit ourselves to one data set to see how the different models perform relative to each other.

3.1 Preprocessing

In this section, we will take look at the steps required to obtain implied volatilities from market data using the Black-Scholes formula. We will do this by means of two steps: The future price and interest rate will be calibrated from data and then we will use them for our numerical method to obtain the implied volatilities.

3.1.1 Data

Options data from the S&P 500 index of October 2nd 2020 will be used to investigate the ability of the different models to construct volatility smiles and volatility surfaces. The S&P 500 is a stock market index that tracks 500 large-cap firms in the United States. It reflects the stock market's success by monitoring the biggest corporations' risks and returns. Initially, the intention is to apply the SSVI, the feedforward neural network and the gated neural network on one specific trading day in order to understand the models before expanding to a data set that covers multiple days. Our original data set contains the following features: the bid price, the ask price, the strike price, the expiration and whether an option is a call or put option.

Before using this data, some operations should be applied on it. First of all, the option contracts whose bid price is less than 0.375 have been removed. The reason for this is that the price 0.375 comes close to the minimum price change between the bid and ask price of an option [5]. Subsequently, a number of features, such as the log-moneyness, expiration value and the mid-price, the price that lies in the middle of the bid and the ask price, are calculated and added to the data set. The considered data set consists of 9 different expirations, which can be found in table 3.1.

Expiration nr.	**Expiration value**	**Expiration date**
Expiration 1	0.085	11/02/20
Expiration 2	0.090	11/04/20
Expiration 3	0.096	11/06/20
Expiration 4	0.115	11/13/20
Expiration 5	0.134	11/20/20
Expiration 6	0.153	11/27/20
Expiration 7	0.164	11/30/20
Expiration 8	0.211	12/18/20
Expiration 9	0.247	12/31/20

Table 3.1: The features of the original data set

Figure 3.1 shows that the majority of the option contracts in our data set have strikes with values ranging from 2000 to 4000. It is important to note that a lack of data will have possible consequences on the performance of a neural network and therefore more attention should be paid when it comes to deep OTM (log-moneyness value way less than 0) and deep ITM data (log-moneyness value much greater than 0), as there are relatively fewer data points for these extremes.

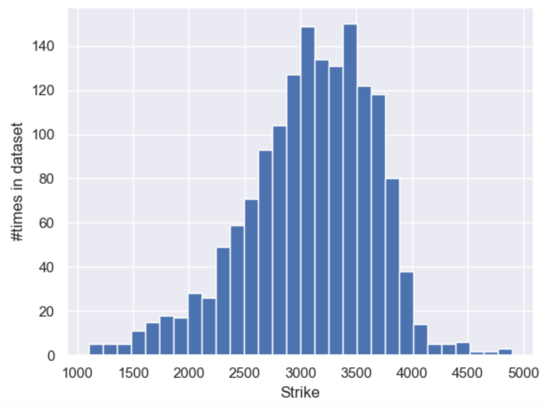


Figure 3.1: Frequency plot of the considered dataframe corresponding to the first trading day of October 2020.

3.1.2 Calibration of future prices and interest rates

We lack two main input values needed to use the Black-Scholes formula to obtain implied volatilities. These are the interest rate and the index's price. There is, however, a possibility of providing an approximation for these values by using the put-call parity (2.11). This relationship can be used to calibrate the future price of the index, by considering call and put options with the same expiration and strike price. The put-call parity can be rearranged to:

$$F = \frac{V^C - V^P}{D} + K, \tag{3.1}$$

where $D = e^{-rT}$, K denotes the strike, V^C represents the call price, V^P stands for the put price, r is the interest rate and T the time left to maturity. Note that the put-call parity shows a linear relationship between the put-call difference, $V^C - V^P$, and the strike K. This follows from the following rearrangement:

$$V^C - V^P = DF - DK. \tag{3.2}$$

Since for every combination of a put option and a call option with the same expiration and strike, we not only possess the mid-price for both the put and call, but also the bid and ask prices, we could obtain different put-call differences, $V^C - V^P$, from our data set by combining bid prices and ask prices. We will obtain this put-call difference in the following 3 ways:

- The bid price for a call option with expiration T and strike K and the ask-price for a put option with expiration T and strike K.
- The ask price for a call option with expiration T and strike K and the bid price for a put option with expiration T and strike K.
- The mid price for a call option with expiration T and strike K and the mid-price for a put option with expiration T and strike K.

We will then use these put-call differences to obtain a future price and an interest rate for each of the 9 expirations separately. A linear regression model will be utilized to achieve this. Recall that applying linear regression on a data set,

$$\left\{\left(K_i, (V_i^C - V_i^P)\right), i = 1, \ldots, n\right\},$$

consisting of only 1 expiration, is described by the underlying relationship between $(V_i^C - V_i^P)$ and K_i involving an error term ε_i and is expressed by:

$$V_i^C - V_i^P = \alpha + \beta K_i + \varepsilon_i. \tag{3.3}$$

A linear regression algorithm could now be easily applied to the data set to find the best α and β. It then follows immediately from (3.2) that $\alpha = -D$ and $\beta = -\alpha F$. From D, it is then possible to obtain r, as D is defined by $D = e^{-rT}$. This process will be applied for each of the 9 expirations.

In table 3.2, the obtained values for F and r can be found and the corresponding linear regression fits for each of the 9 expirations can be found in figure 3.2.

Expiration	F	r
Expiration 1	3346.97	0.0008
Expiration 2	3347.00	$-5.82 \cdot 10^{-6}$
Expiration 3	3346.04	0.003
Expiration 4	3334.29	0.004
Expiration 5	3342.95	0.004
Expiration 6	3342.32	0.007
Expiration 7	3341.61	0.006
Expiration 8	3339.18	0.005
Expiration 9	3339.33	0.005

Table 3.2: Calibrated interest rate and future price for all the 9 expirations in increasing order

As can be seen in figure 3.2, the linear regression fits approximate the put-call differences accurately for two expirations. The plots corresponding to the other expirations have a similar structure.

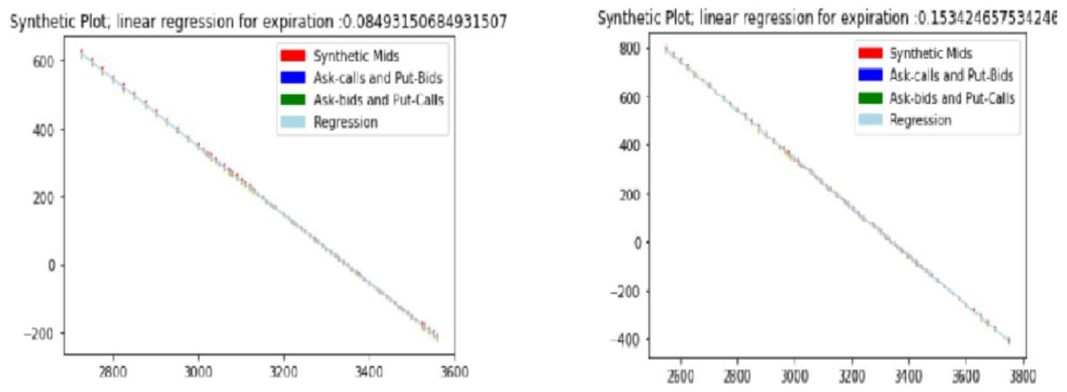


Figure 3.2: For two chosen expirations the linear regression fit for the put-call differences is created and shown within this figure. The different colors correspond to the different ways of how the put-call differences are calculated. The x-axis reflects the strike price and the y-axis the put-call difference.

3.1.3 Black-Scholes Implied Volatility from market data

Note that the Black-Scholes formula requires the underlying index price, while we have just made an attempt to estimate futures prices. What we now need is a way to obtain the index price from the future price and vice versa. In other words, we wish to model the index price in terms of the future price. The modelling choice we adopt for the future price is described as follows:

$$F = S \cdot e^{rT},$$

where T stands for the time to expiration, F denotes the future price and S represents the index price.

Initially the Newton's method is used as our numerical solver and applied on the bid, ask and mid-prices to obtain the implied volatilities. If Newton's method resulted in a computational error, we switched to the bisection method. In order to gain an impression for the obtained implied volatility smiles for 2 expirations, figure 3.3 can be consulted.

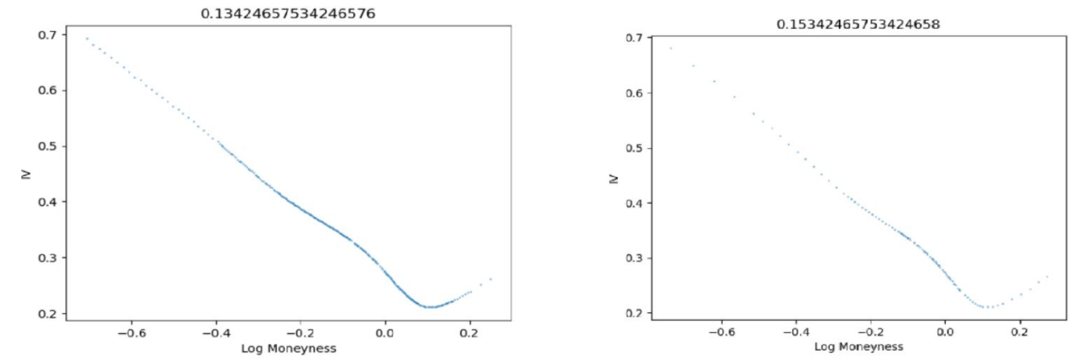


Figure 3.3: Implied volatility smiles corresponding to two maturities corresponding to 2nd of October 2020.

3.2 SSVI model calibration

In this section, the SSVI model is studied from two different perspectives. The first approach uses restrictions on parameters, whereas the second approach utilizes *Differential Evolution* (DE) to construct volatility smiles.

3.2.1 SSVI model using restrictions on parameters

The parametrization introduced in Definition 1 is utilized to calibrate the volatility smile. In order to ensure that the fitted curves comply with the static arbitrage conditions, η and ρ are chosen in such a way that $\eta(1 + |\rho|) \leq 2$, described as (2.15). Using this condition and an optimization algorithm known as *fmincon* [46], which is a trust-region optimization algorithm, the SSVI volatility smile fits for the 9 considered maturities are obtained. A visual representation of each of these fits can be found in appendix A. The SVI-Jump Wings parameters for each expiration are summarized in table 3.3. These parameters are found by applying lemma 6.

Expiration	v	ψ	p	c	$\widetilde{v}$
Expiration 1	0.051	-0.233	0.712	0.247	0.039
Expiration 2	0.063	-0.241	0.703	0.222	0.046
Expiration 3	0.069	-0.234	0.711	0.244	0.052
Expiration 4	0.071	-0.243	0.711	0.224	0.052
Expiration 5	0.066	-0.257	0.758	0.243	0.049
Expiration 6	0.070	-0.256	0.744	0.233	0.051
Expiration 7	0.071	-0.263	0.748	0.222	0.050
Expiration 8	0.064	-0.308	0.924	0.309	0.048
Expiration 9	0.071	-0.294	0.849	0.261	0.051

Table 3.3: SVI-JW parameters corresponding to the 9 different maturities for trading day October 2

To successfully use the fmincon algorithm, it is of importance to establish an objective function that reflects a specific measure of the goodness of fit. The adopted objective function is defined as follows:

$$OF = \frac{1}{N}\sum_{i=0}^{N}(\sigma_{BS}(k_i, T_i) - \hat{\sigma}_{BS}(k_i, T_i))^2, \tag{3.4}$$

where $\hat{\sigma}_{BS}(k_i, T_i)$ stands for the SSVI value and $\sigma_{BS}(k_i, T_i)$ represents the desired implied volatility.

We will elaborate on 3 of the 9 expirations to see how the SSVI performs in detail. In figure 3.4, a fit for maturity 0.084932 is generated. Besides the SSVI fit, there are also graphs for the Black-Scholes implied volatility generated by the bid and ask price. Ideally, the fit should lie exactly between these 'bid and ask spreads'. One can imagine that when the bid and ask spreads are very tight, it will become harder to ensure that the fit will lie in the spreads. The assumptions we made on the search area of the parameters may have a relatively large impact on the fits as well. We observe that for log-moneyness values around 0 the fit looks accurate, whereas when looking at the extremes (deep OTM/ITM), some discrepancies can be observed.

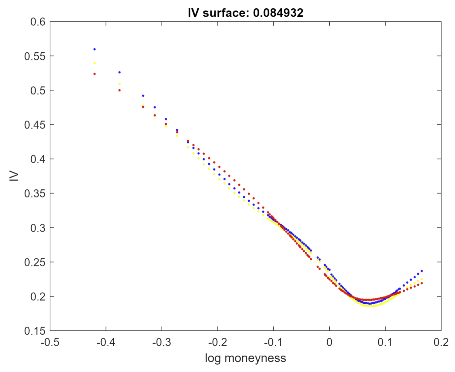


Figure 3.4: SSVI fit based on maturity 0.084932. The red graph corresponds to the fit, while the blue line and yellow line stand for the ask-IV and bid-IV, respectively.

In figure 3.5, a fit for maturity 0.13425 is generated. The fit looks similar to the one in figure 3.4, however, it appears that, for small log-moneyness values, the discrepancies are larger, whereas for higher log-moneyness values, the SSVI fit appears to be more accurate.

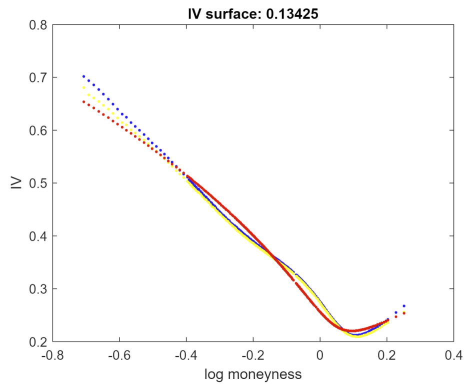


Figure 3.5: SSVI fit based on maturity 0.13425. The red graph corresponds to the fit; the blue line and yellow line stand for the ask-IV and bid-IV, respectively.

Figure 3.6 shows that the structure of the fits are similar to the fits of figure 3.5 and 3.6.

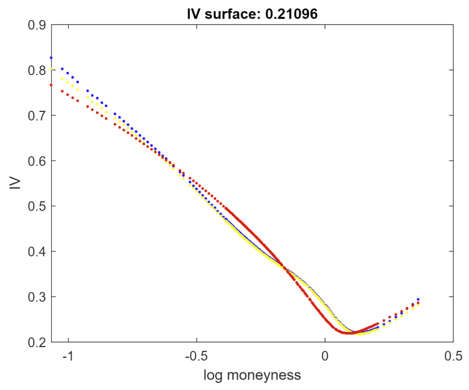


Figure 3.6: SSVI fit based on maturity 0.21096. The red graph corresponds to the fit; the blue line and yellow curves represent the ask-IV and bid-IV respectively.

It is obvious that the tighter the bid and ask spreads are, the more difficult it becomes for an algorithm to find parameters that ensure that the SSVI fits will lie exactly in the bid-ask spreads. One possible way to achieve this, is to match the objective function to satisfy this. Nevertheless, the question is to what extent the restriction $\eta(1 + |\rho|) \leq 2$, influences the shape of the fits and thus possibly precludes that we will end up with SSVI fits which lie exactly in the bid-ask spreads. Moreover, it could be the case that the parameters found by fmincon yield a local optimum. Further research should assess what the impact on the fits is, when using the restriction $\eta(1+|\rho|) \leq 2$ in combination with the fmincon algorithm.

From figure 3.7, we can learn more about the absolute errors of the fits in relation to the mid-volatilies for two chosen expirations. It immediately stands out that for the two chosen expirations, small log-moneyness values tend to have larger discrepancies. The structure of the errors are similar for all of the considered expirations.

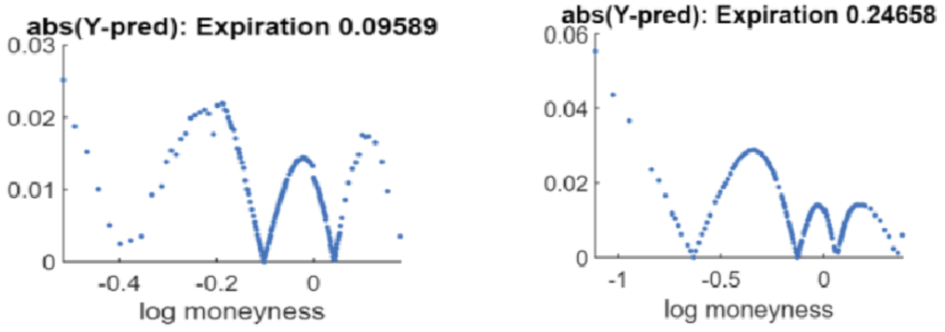


Figure 3.7: Absolute error plots for two randomly chosen expirations corresponding to October 2.

3.2.2 SSVI model using the differential evolution algorithm

In section 3.2.1, we have used the fmincon algorithm to find parameters which optimize the objective function (3.4). In this section, we leverage on the differential evolution [47, 48] algorithm, which is described in section 2.5.1, to find parameters for the SSVI parameterization. However, the restriction $\eta(1+|\rho|) \leq 2$ is not made part of the differential evolution algorithm. The objective function (3.4) is re-used. The following setting is used for the configuration of the differential evolution algorithm: F is taken equal to 1.5, N is taken as 5000 and p_{cr} is 0.7.

Figure 3.8 shows for all the considered expirations the performance of the SSVI method which is optimized by the differential evolution algorithm. It can be observed that for most of the expirations the obtained smiles are quite accurate, but not able to fully follow the shape of the curve implied by the market data.

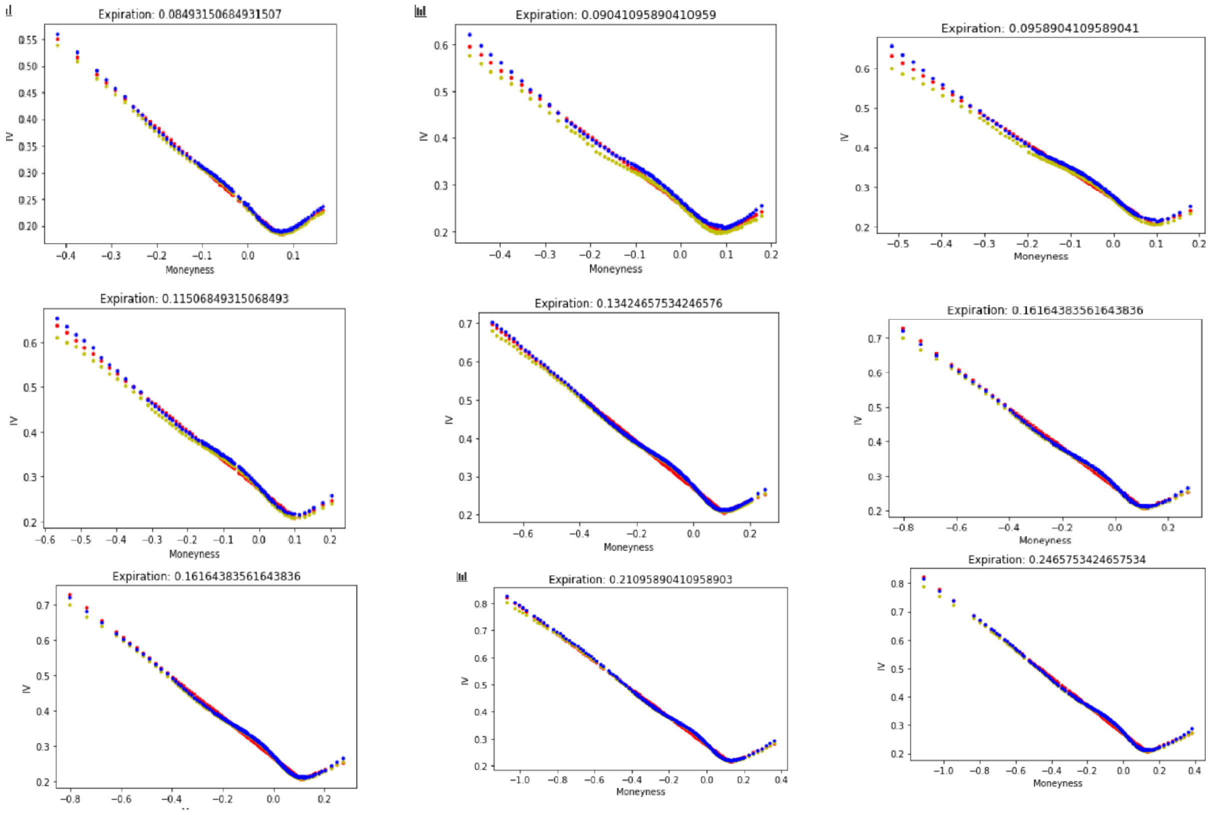


Figure 3.8: SSVI fits based on differential evolution. The red dots represent the ask-volatilities, the green dots represent the bid-volatilities and the blue dots represent the SSVI-values using Differential Evolution

The SVI-Jump Wings parameters parameters discovered by this method are shown in table 3.4.

Expiration	v	ψ	p	c	$\widetilde{v}$
Expiration 1	0.179	0.645	-0.170	0.071	0.011
Expiration 2	0.120	0.618	-0.167	0.087	0.011
Expiration 3	0.206	0.543	-0.288	0.091	$2.08 \cdot 10^{-8}$
Expiration 4	0.202	0.531	-0.215	0.104	$7.25 \cdot 10^{-10}$
Expiration 5	0.202	0.470	-0.286	0.106	$1.15 \cdot 10^{-6}$
Expiration 6	0.205	0.471	-0.215	0.115	$1.15 \cdot 10^{-8}$
Expiration 7	0.205	0.382	-0.297	0.131	0.007
Expiration 8	0.212	0.403	-0.271	0.126	0.011
Expiration 9	0.206	0.382	-0.297	0.131	0.007

Table 3.4: SVI-JW parameters corresponding to the different maturities using differential evolution algorithm

In order to gain a deeper understanding of how the SSVI method, which is optimized by the differential evolution algorithm, performs, figure 3.9 can be consulted where one can observe the absolute error plots for two expirations. The absolute error plots for all the expirations have a similar structure; the accuracy is fluctuating. It becomes clear that the errors for using the differential evolution algorithm are much lower compared to using the fmincon algorithm in combination with the restriction $\eta(1+|\rho|) \leq 2$. In any case, this method does not seem to be entirely useful in practice since it took too long in terms of computing time to obtain reasonable fits. Hence, in the rest of the thesis we have chosen to continue with the methodology described in section 3.2.1.

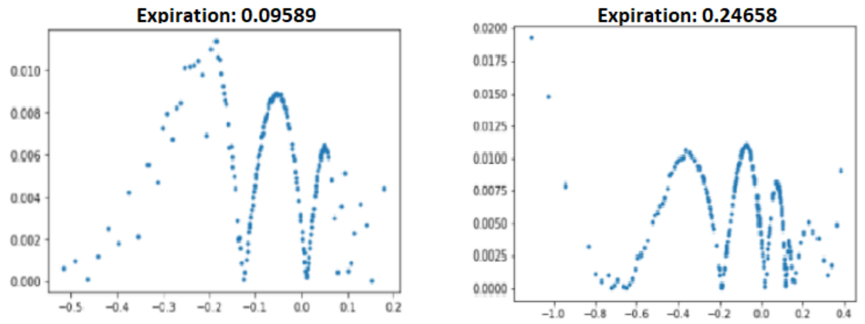


Figure 3.9: Absolute error plots for 2 chosen expirations corresponding to October 2nd. The error plots for the other expirations look similar.

3.3 Feedforward neural network fitting

In this section we will examine the degree to which feedforward neural networks are capable of constructing volatility smiles and volatility surfaces. We will study the application of feedforward neural networks in two different phases. In the first phase, we will adopt a 1-dimensional neural network that attempts to generate a volatility smile, and in the second phase we will design a 2-dimensional neural network to construct volatility surfaces. Furthermore, the significance of certain hyperparameters of a feedforward neural network will be investigated.

3.3.1 Experimental setup

In [4] a framework is presented with which an attempt is made to construct an implied volatility surface based on a multiplication of the feedforward neural network and the prior function. This prior ensures that the model's generalization adheres to a set of desired conditions. In [4], the prior is multiplied by a feedforward neural network and the presented model is described as follows:

$$\hat{\sigma}_{BS}(k,T) = \hat{\sigma}_{\text{NN}}(k,T) \times \omega_{\text{prior}}(k,T), \tag{3.5}$$

where $\hat{\sigma}_{\text{NN}}(k,T)$ and $\omega_{\text{prior}}(k,T)$ represent the feedforward neural network and the prior, respectively.

Possible choices for this prior could be, for example, the SSVI method. However, in this research the adopted prior is the function:

$$\omega_{\text{prior}}(k, T) = 1.$$

Note that since we did not specify a prior and this prior can help in setting up the structure of a volatility surface, it is more likely that the fits tend to deviate from the desired output. To avoid this, the learning process is repeated 5 times and the network with the smallest MSE is then chosen as the 'winning' feedforward neural network. Note that if this repetition is not used, the risk of ending up in a local minimum is much greater. Before moving on to a feedforward neural network that takes two dimensions as inputs, we first look at the 1-dimensional case, where each expiration is studied separately and only the strike serves as an input to the feedforward neural network. The chosen cost function associated with this network is the MSE. Furthermore, the SGD is the learning procedure that is utilized to train the neural network. L2-regularization is utilized with regularization term λ 0.1. The hyperparameter configuration is chosen by using 5 random configurations using table 3.6 and afterwards choosing the best performing configuration out of these 5 configurations. A more in-depth analysis of how to choose the best fitting configuration will be discussed in the remainder of the thesis. The adopted configuration can be found in table 3.5.

Epochs	**LR**	**Batch Size**	**Hidden Layers**	**Hidden Neurons**	**Activation Function**
5000	0.1	32	8	5	Sigmoid

Table 3.5: Settings for the 1-dimensional feedforward neural network.

Compared to the SSVI model, the feedforward neural network fits, that can be found in figure 3.10 appear to be more accurate, especially when looking at deep OTM predictions and to what extent the fits lie in the bid-ask spreads. Nonetheless, for some points with log-moneyness value around 0, we have some issues regarding the accuracy of the fits. Because for a trader it is most important to have a model that performs well for log-moneyness values around 0, it is important to pay some more attention to this. One possible way to achieve this is to setup the cost function in order to weight the points, for which the log-moneyness value is around 0, higher.

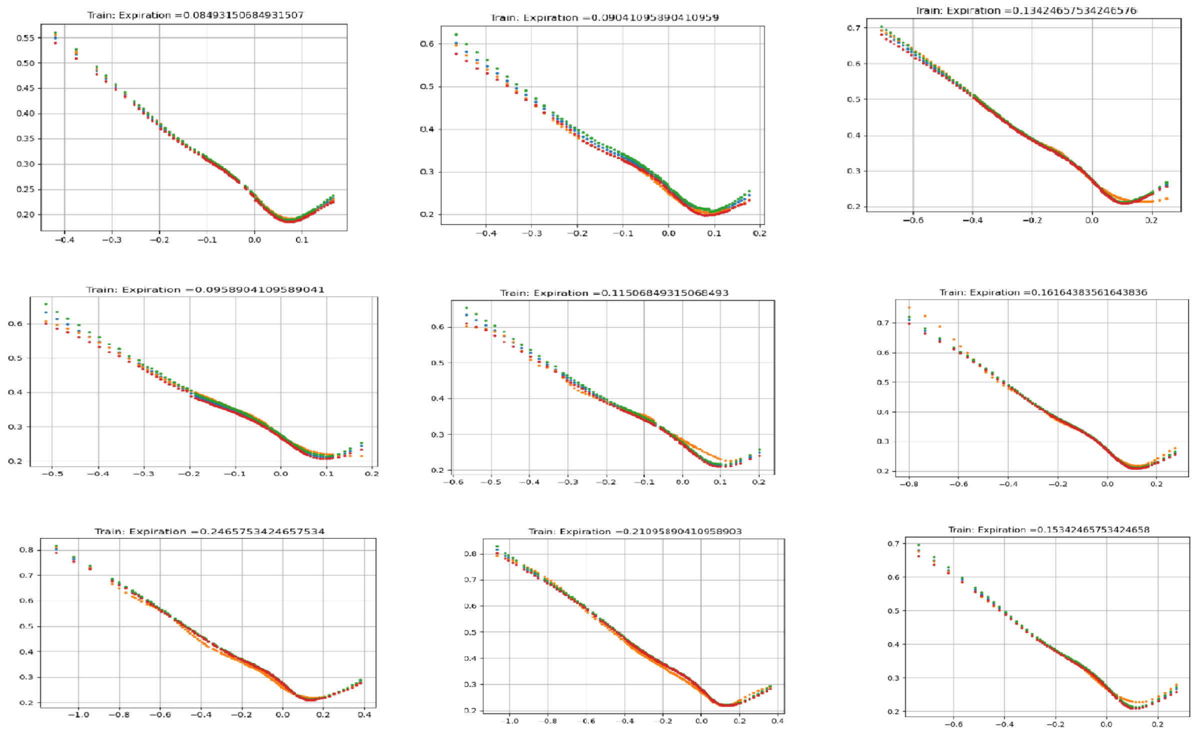


Figure 3.10: 1D feedforward neural network fits; green dots represent the ask-volatilities, the red dots represent the bid-volatilities and the orange dots represent the fitted values.

Then, for a 2D feedforward neural network, an attempt was made to use a similar network. However, picking a configuration at random and to obtain accurate results is still difficult at this stage. This is a problem that will be discussed and tackled in chapter 4.

3.3.2 Hyperparameter importance

Ideally, depending on the data set we are using to generate fits, it would be convenient to apply an optimization to find appropriate neural network hyperparameters. Before we can allow this optimization to take place, we should be able to quantify the importance of certain hyperparameters. So that we are able to know which hyperparameters deserve more attention. A possible framework that we could apply regarding hyperparameter importance is the *functional ANOVA Framework* [49]. This framework is explained in detail in [49]. The following is an overview of this explanation:

Let f be a neural network with n input hyperparameters with domains $\Theta_1, \ldots, \Theta_n$ and let the output of this function f the value of the loss function after training the neural network with the input hyperparameters.

- In this section, we will use positive numbers to denote the hyperparameters, and A will refer to the set $\{1, \ldots, n\}$ of all parameters. The parameter space is $\mathbf{\Theta} = \Theta_1 \times \cdots \times \Theta_n$. A parameter configuration is a vector $\boldsymbol{\theta} = \langle \theta_1, \ldots, \theta_n \rangle$ with $\theta_i \in \Theta_i$.

- A partial instantiation of a subset $U = \{u_1, \ldots, u_m\} \subseteq A$ is a vector $\theta_U = \langle \theta_{u_1}, \ldots, \theta_{u_m} \rangle$ with $\theta_{u_i} \in \Theta_{u_i}$.
- Let $\boldsymbol{\theta}_U = \langle \theta_{u_1}, \ldots, \theta_{u_m} \rangle$ be a partial instantiation of the parameters $U = \{u_1, \ldots, u_m\} \subseteq A; X(\boldsymbol{\theta}_U)$ is then the set of parameter configurations $\boldsymbol{\theta}_{A|U} = \langle \theta'_1, \ldots, \theta'_n \rangle$ such that $\forall j : j = u_k \Rightarrow \theta'_j = \theta_{u_k}$.
- Let $\hat{f} : \boldsymbol{\Theta} \to \mathbb{R}$ be a function, $U \subseteq A$, and $T = A \backslash U$. The marginal $\hat{a}_U(\boldsymbol{\theta}_U)$ of $\hat{f}$ over T is then defined as

$$\hat{a}_U(\boldsymbol{\theta}_U) = \mathbb{E}\left[\hat{f}\left(\boldsymbol{\theta}_{A|U}\right) \mid \boldsymbol{\theta}_{A|U} \in X(\boldsymbol{\theta}_U)\right] = \int \hat{f}\left(\boldsymbol{\theta}_{A|U}\right) \cdot p(\boldsymbol{\theta}_T)\, d\boldsymbol{\theta}_T,$$

where $p(\boldsymbol{\theta}_T)$ represents the uniform distribution for any T.

It may be clear that for each of the data sets we need to sample a configuration from $\boldsymbol{\Theta}$; so some realistic ranges should be chosen for every hyperparameter we need to tune. In table 3.6, an overview of the chosen ranges can be found for each of the hyperparameters. The choice to take the ranges quite wide is a conscious one, because of the uncertainty that still exists at this stage about the importance of each of the hyperparameters.

Hyperparameter	**Range**
Epochs	[1000, 2000, 3000, 4000, 5000, 6000, 7000]
Learning rate	[0.1, 0.01, 0.001, 0.0001, 0.00001]
Hidden Layers	[2, 4, 6, 8, 10]
Hidden Neurons	[2, 4, 6, 8, 10]
Batch Size	[32, 64, 128]

Table 3.6: Ranges for the hyperparameters

Functional ANOVA decomposes the variance of a function $\hat{f}$ of the form $\hat{f} : \boldsymbol{\Theta} \to \mathbb{R}$ into additive components that only depend on subset of the hyperparameters:

$$\hat{f}(\boldsymbol{\theta}) = \sum_{U \subseteq N} \hat{f}_U(\boldsymbol{\theta}_U).$$

The components $\hat{f}_U(\boldsymbol{\theta}_U)$ are defined as follows:

$$\hat{f}_U(\boldsymbol{\theta}_U) = \begin{cases} \int \hat{f}(\boldsymbol{\theta}) \cdot p(\boldsymbol{\theta}) d\boldsymbol{\theta} & \text{if } U = \emptyset \\ \hat{a}_U(\boldsymbol{\theta}_U) - \sum_{W \subsetneq U} \hat{f}_W(\boldsymbol{\theta}_W) & \text{otherwise} \end{cases}$$

The variance $\mathbb{I}(\hat{f})$ across its domain Θ can be defined by:

$$\mathbb{I}(\hat{f}) = \int \left(\hat{f}(\boldsymbol{\theta}) - \hat{f}_{\emptyset}\right)^2 \cdot p(\boldsymbol{\theta}) d\boldsymbol{\theta}.$$

If we assume a uniform prior, it is shown in [50] that the functional ANOVA can be described as follows:

$$\mathbb{I} = \sum_{U \subset N} \mathbb{I}_U, \text{ where } \mathbb{I}_U = \int \hat{f}_U \left(\boldsymbol{\theta}_U\right)^2 \cdot p\left(\boldsymbol{\Theta}_U\right) d\boldsymbol{\theta}_U.$$

As mentioned, f will be the 1-dimensional feedforward neural network that takes as an input a hyperparameter configuration and its output represents the value of the cost function corresponding to this configuration. The experiment will be set up as follows: our data set will be grouped by the the time to expiration T and each of these grouped data sets will be used as one of our data sets. Subsequently, random sampling is used to generate 40 random configurations belonging to each data set. The expectation is that an important hyperparameter j will result in a high variance contribution $\mathbb{I}_j/\mathbb{I}$. This experiment will be applied on each of the data sets, so we will end up with different variance contributions belonging to a hyperparameter. In conclusion, this experiment will indicate which hyperparameters should be tuned and which are not sensitive. In the end, this will make it easier to apply an optimization only on important hyperparameters.

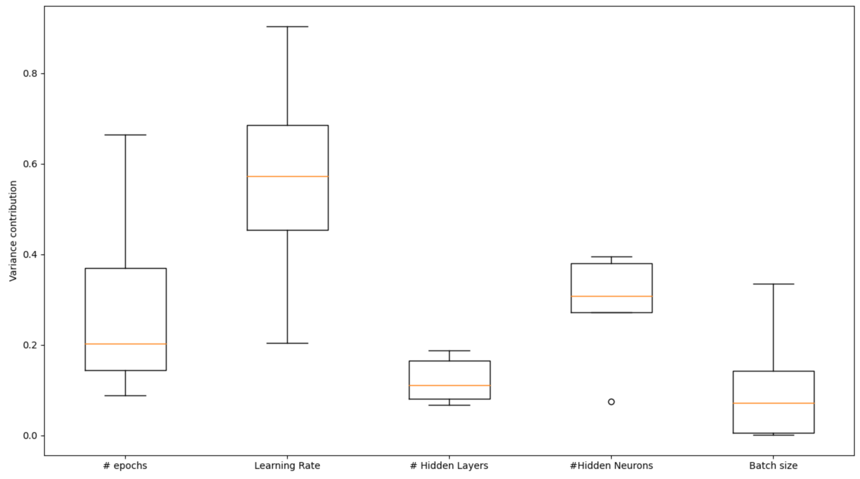


Figure 3.11: Variance contribution of each hyperparameter based on functional ANOVA analysis

It becomes clear from figure 3.11 that the number of epochs, the learning rate and the number of hidden neurons can be considered as important hyperparameters. It is therefore important to consider these parameters in a possible next step for optimization. To confirm this analysis, we want to perform two experiments. The first experiment is structured as follows: for a hyperparameter j, we randomly pick an element A from the range belonging to this hyperparameter j, which can be found in table 3.6. Then, we run an experiment in which we randomly generate 40 different hyperparameter configurations under the assumption that hyperparameter j is always equal to A. Given 40 configurations, we train these models on one randomly chosen data set and from there we engage with the best performing configuration. We then repeat this process for 30 iterations and after the i'th iteration we plot the average MSE value up to the i'th iteration. These plots can be found in figure 3.12. The purpose of this is to see to what extent a random draw for our hyperparameter will affect the MSE value. Note that in case of a significant hyperparameter, we expect a high MSE value when picking a random value for the hyperparameter that is considered. However, it is crucial to note that the number of different configurations and number of iterations will definitely affect the performance of this methodology. We have deliberately chosen to repeat the process 30 times and choose 40 configurations, because otherwise it would not be feasible from a computational perspective. Further research should be conducted to assess this choice.

From figure 3.12 it becomes clear that the learning rate and the number of epochs come out as significant hyperparameters, as the average MSE values in the end are relatively high. So we can simply conclude that picking a random learning rate and a random number of epochs will not per se lead to a low MSE.

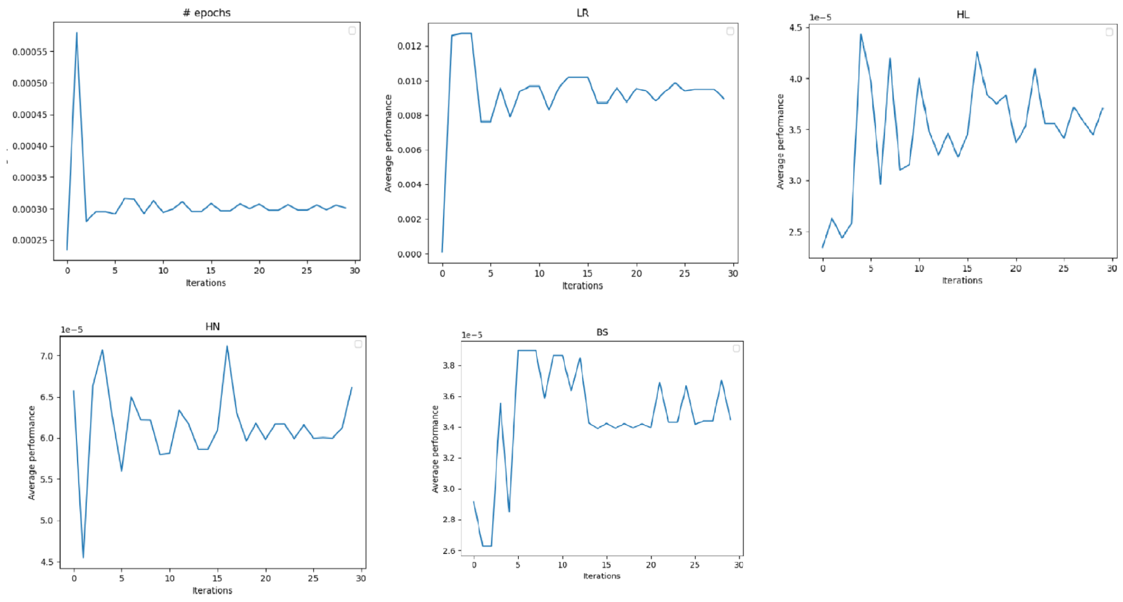


Figure 3.12: Verification of functional ANOVA Analysis using random optimization

Another method for assessing the functional ANOVA performance is as follows; for each hyperparameter j, we first look at all the possible values that this j can take using table 3.6; then each value from this range is taken fixed and the other hyperparameters are chosen 40 times using a random draw. So we will end up with 40 random configurations. The feedforward neural network will be trained for these 40 different configurations and the average RMSE value belonging to this value that j can take will be stored. This process should be repeated for every possible value of each of the hyperparameters. In this way, an impression can be obtained which hyperparameter values are likely to provide good prediction. This process is also known as the process of obtaining marginal predictions.

From tables 3.7, 3.8, 3.9, 3.10 and 3.11, it is obvious that no single hyperparameter value offers a significant improvement on its own. However, we may argue that a slightly higher learning rate represents better results.

Epochs	500	1000	2000	3000	4000	5000	6000	7000
Average MSE	0.068	0.042	0.035	0.045	0.035	0.022	0.060	0.050

Table 3.7: Marginal predictions (Number of epochs)

Learning Rate	0.1	0.01	0.001	10^{-4}	10^{-5}	10^{-6}	10^{-7}
Average MSE	0.011	0.012	0.017	0.038	0.075	0.090	0.092

Table 3.8: Marginal predictions (Learning Rate)

Hidden Layers	2	4	6	8	10
Average MSE	0.032	0.037	0.050	0.037	0.068

Table 3.9: Marginal predictions (Hidden Layers)

Hidden Layers	2	4	6	8	10
Average MSE	0.049	0.034	0.041	0.048	0.062

Table 3.10: Marginal predictions (Hidden Neurons)

Batch Size	32	64	128
Average MSE	0.039	0.050	0.049

Table 3.11: Marginal predictions (Batch Size)

3.4 Gated neural network fitting

As seen earlier, we have two forms of gated neural networks that are worth looking at separately. The single gated neural network and the multi gated neural network, where the latter uses a combination of multiple single gated neural networks. The purpose of this section will therefore be to dive deeper into the performance of these models.

3.4.1 Single gated neural network fitting

Two forms of the single gated neural network will be studied. One focuses on predicting volatility smiles, while the other aims at constructing volatility surfaces. This means that in the first case we aim to fit a model, where the expiration is fixed in advance and thus we are essentially dealing with a model that takes only 1 dimension as input; in the second case we aim to train a single 2-dimensional gated neural network, where the expiration and strike serve as input to the 2-dimensional multi gated neural network. Recall that the loss function that we introduced as (2.19) is defined as:

$$\ell = \ell_0 + \gamma\ell_1 + \delta\ell_2 + \eta\ell_3 + \rho\ell_4 + \omega\ell_5.$$

This loss function will not be usable for the 1-dimensional case, because the individual loss terms ℓ_1, ℓ_2, ℓ_3 and ℓ_4 all depend on the expiration in some way and for the 1-dimensional single gated neural network, the expiration is fixed in advance. For this reason, we restrict ourselves to ℓ_0 and ℓ_5. The new loss function is described as follows:

$$\ell = \ell_0 + \omega\ell_5.$$

For each expiration, the adopted configuration is borrowed from [5] and can be found in table 3.12 and the learning algorithm is the SGD. Please note that at this stage there is no valid analysis conducted to use these hyperparameters with a certain confidence; therefore, they also do not give any guarantee that they will be favorable for other data sets. Recall that J represents the number of hidden neurons.

Number of epochs	**Initial LR**	J	α	β	ω
3000	0.1	32	1	1	0.00005

Table 3.12: Settings for the 1D single gated neural network architecture

Figure 3.13 shows for two chosen expirations on the left side the error propagation and on the right side the predictions (in orange) and the values to be predicted (in blue). For the other expirations, the plots look quite similar. As can be seen clearly, for the chosen expirations, the error clearly converges smoothly to a very small number. From 500 epochs onwards, the changes seem very minimal and for this reason we have tried to adjust the learning rate by decreasing it with a factor 10 to influence the performance somewhat. All in all, we can conclude that from the graphs the 1-dimensional single gated neural network looks accurate.

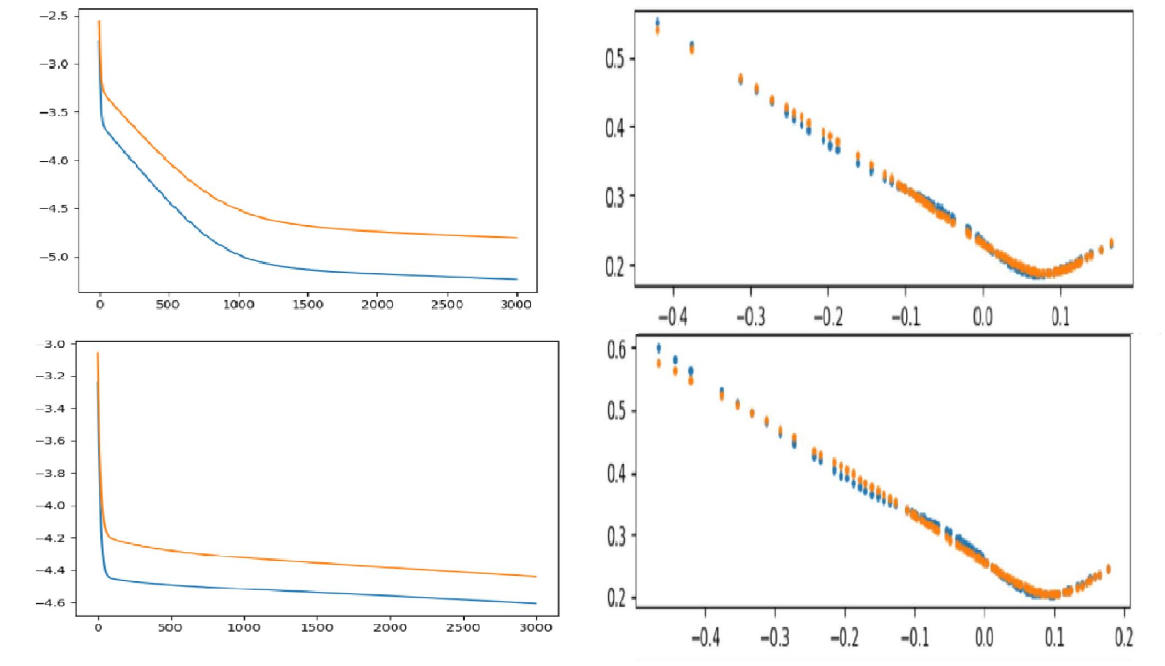


Figure 3.13: Left plots: Logarithm of the error as the number of epochs grows, orange plots denote the errors based on the validation set and the blue graph corresponds to the error based on the training set; Right plots: Fitted smiles obtained from 1D single gated neural network in blue and the desired values in orange; The first row corresponds to plots for expiration 0.085 and the second row corresponds to the plots for expiration 0.090.

Regarding the 2-dimensional single gated neural network fits that can be found in figure 3.14, we can conclude that the learning procedure struggles to find an accurate fit for the shorter expirations. A possible reason may be that the behavior of the predictions of the somewhat higher expirations predominate. This could have to do with the fact that for higher expirations more data is available. The hyperparameters that are used to setup the training process are described in table 3.13. Recall that J represents the number of hidden neurons.

Number of epochs	**Initial LR**	J	α	β	η	ρ	γ	δ	ω
3000	0.1	32	1	1	1	10	10	1	0.00005

Table 3.13: Settings for the 2-dimensional single gated neural network architecture

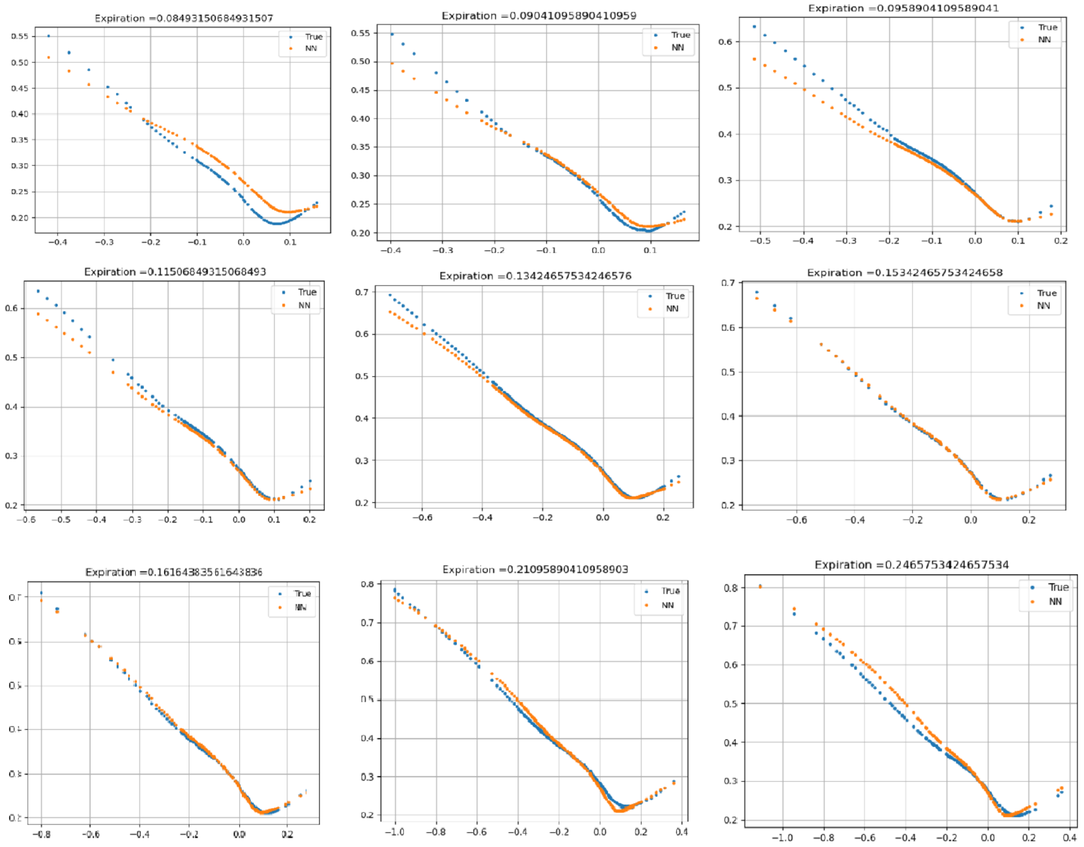


Figure 3.14: The 2-dimensional single gated neural network fits. The blue dots correspond to the desired values and the orange dots represent the fitted values.

3.4.2 Multi gated neural network

For the multi gated neural network, we ran the algorithm using the following hyperparameters and the loss function (2.19). The hyperparameters choices are based on [5]. Recall that J represents the number of hidden neurons, I the number of single models and K serves as an important parameter for (2.18).

Number of epochs	**Initial LR**	J	I	α	β	η	ρ	γ	δ	ω	K
3000	0.1	8	4	1	1	1	10	10	1	0.00005	5

Table 3.14: Settings for the 2-dimensional multi gated neural network architecture

As can be clearly observed in figure 3.15, except from the first expiration, the performance appears that of the models in scope based on the graphs this model looks to get the best fit. A quantitative comparison will follow chapter 4. However, there is a caveat to this; the number of parameters to be trained has increased drastically compared to the single gated neural network, thus affecting the runtime of the training phase. However, it should be noted that in this phase there has not yet been an attempt to apply a hyperparameter optimization procedure.

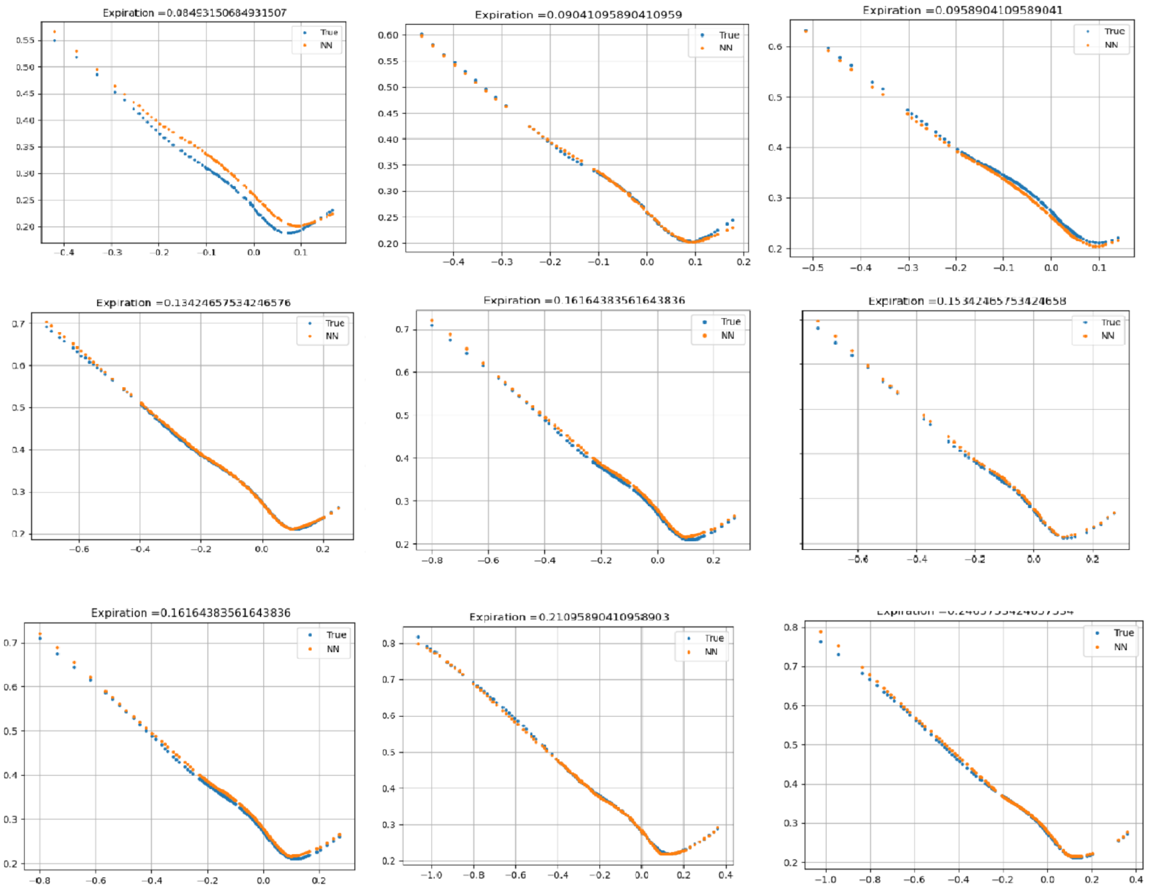


Figure 3.15: 2D multi gated neural network; orange dots represent fit and blue dots represent the mid volatilities

To get insight in the composition of a multi gated neural network, it is important to do experiments to see how certain components of such a complex architecture contribute to the overall performance of the network. As mentioned earlier, a multi gated neural network consists of a number of single gated neural networks. One may wonder to what extent these single networks add value to the overall performance and how many of them we may need. A small experiment has been conducted to see if there is a significant improvement when the number of single gated neural networks increases within the multi gated neural network setting. In order to do this experiment, we used the first three trading days of October 2020 and trained a multi gated neural network using 2, 4 and 6 single models for each trading day. The other hyperparameters are chosen according to table 3.14. Note that adding even more single gated neural networks caused the number of parameters to be learned to increase drastically which will definitely not improve the runtime performance. Hence the choice was made to set the upper limit at 6. Furthermore, each training procedure is repeated three times in order to get rid of local optimum issues.

It becomes clear from table 3.15 that increasing the number of single gated neural networks in a multi gated neural network has a quite positive impact on the performance in terms of the MSE. In conclusion, adding single networks will generally result in a better fit. However, the fit is already accurate with 4 single models and the improvement with more single models is not impressive anymore.

Number of single models	2	4	6
1 October	$1.21 \cdot 10^{-5}$	$6.49 \cdot 10^{-6}$	$4.43 \cdot 10^{-6}$
2 October	$1.78 \cdot 10^{-5}$	$8.63 \cdot 10^{-6}$	$8.30 \cdot 10^{-6}$
5 October	$3.00 \cdot 10^{-5}$	$1.88 \cdot 10^{-5}$	$6.73 \cdot 10^{-6}$

Table 3.15: Importance of the number of single gated neural networks in the multi gated neural network using the MSE.

Chapter 4

Multiple trading days analysis

In this chapter, the focus will be on the use of certain weights in combination with the MSE as a cost/loss function for all considered neural networks. The models will be applied on multiple trading days. In addition, robustness will be an important part; where previously we have chosen configurations without conducting an analysis, in this chapter we will search for appropriate hyperparameters by conducting an analysis.

4.1 Robustness

It is perhaps obvious to use the previous configurations for our neural networks and thus obtain the performance of the various models; however, it is apparent that a certain configuration which works well for one data set belonging to a trading day does not necessarily guarantee the performance of a data set belonging to another trading day. Add to this the fact that a change in the loss function also requires a change in the hyperparameter settings. For this reason it is important to regularly carry out a hyperparameter optimization which, with some certainty, will give us a certain configuration in which confidence can be kept. To analyze hyperparameter configurations, we will do the following: first, we will perform a hyperparameter optimization based on the first 5 trading days of October for the expirations between 0.1 and 0.4 and then check the error propagation to see if the error terms decrease smoothly as the number of epochs grows. The data sets belonging to the trading days will be split in training sets and validation sets by randomly assigning 4/5 of the data points to the former and the remaining 1/5 to the latter. In this chapter, we will restrict ourselves to the 2-dimensional feedforward neural network and the 2-dimensional multi gated neural network, where the expiration and strike price serve as inputs. The cost function used for these models is the MSE.

With respect to the 2-dimensional feedforward neural network, a Bayesian hyperparameter optimization has been applied to the learning rate, hidden layers, hidden neurons and the regularization term. The first 5 trading days of October are used to conduct our analysis. The number of epochs that are chosen is equal to 15000, the SGD is used as the learning algorithm and 36 iterations are used to perform the optimization. Due to limited computational resources and research time, we chose 36 iterations. In the future, it could be further investigated whether this choice is sufficient. The adopted activation function is the Tanh activation function. Moreover, each configuration was run 3 times on each of the 5 trading days and the average validation set MSE among these 15 runs is then used as the MSE corresponding to this configuration. Table 4.1 gives us an overview of the 8 top performing configurations. The best performing configuration in terms of the MSE in this table is chosen as our 'winning' hyperparameter set. Not only because of the great performance in terms of the MSE, but also because of the error propagation plots in figure 4.1 which show smooth convergence for two randomly chosen trading days.

Learning rate	**Hidden Layers**	**Hidden Neurons**	**Regularization term** λ	**MSE**
0.006	10	10	$3.01 \cdot 10^{-6}$	$4.35 \cdot 10^{-5}$
0.0004	7	9	0.0027	$4.64 \cdot 10^{-5}$
0.007	6	7	$2.87 \cdot 10^{-6}$	$4.85 \cdot 10^{-5}$
0.09	10	6	$7.41 \cdot 10^{-6}$	$5.11 \cdot 10^{-5}$
0.0004	9	7	0.00029	$5.21 \cdot 10^{-5}$
0.007	8	6	0.00018	$5.52 \cdot 10^{-5}$
0.01	7	9	$1.35 \cdot 10^{-5}$	$6.05 \cdot 10^{-5}$
0.08	7	7	$4.17 \cdot 10^{-6}$	$6.20 \cdot 10^{-5}$

Table 4.1: Best 8 performing configurations Bayesian Hyperparameter optimization for a 2-dimensional feedforward neural network

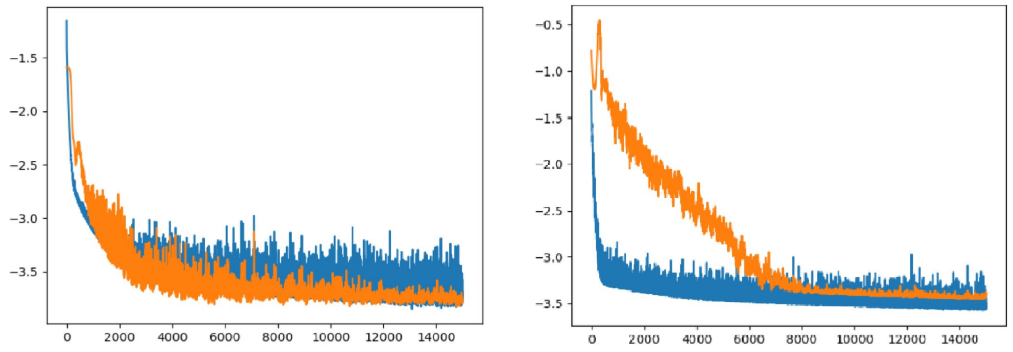


Figure 4.1: Error propagation plots 2-dimensional neural network: Epochs vs logarithm of the MSE. The orange graph represents the error propagation using the validation data set. The blue graph represent the error propagation using the train data set. The plots are based on the first and third trading day of October 2020.

The same optimization procedure is applied to the 2-dimensional multi gated neural network. 10000 training epochs were used and the SGD is utilized as the training algorithm. The choice is made to optimize based on three hyperparameters, namely the number of single models, regularization term and the learning rate. The other hyperparameters J and K are taken fixed as in table 3.14. We also used L2-regularization to avoid overfitting. Again, for each configuration the training procedure is applied 3 times for each of the 5 trading days and the average validation set MSE of these 15 procedures is used as the MSE corresponding to a configuration. Table 4.2 can be referenced to observe the 8 best performing configurations. Again, the top performing configuration is utilized because of the MSE performance and the smooth error propagation as can be observed in figure 4.2.

Learning rate	**Single models**	**Regularization term λ**	**MSE**
0.091	4	$1.07 \cdot 10^{-6}$	$5.53 \cdot 10^{-5}$
0.100	3	$1.00 \cdot 10^{-6}$	$6.06 \cdot 10^{-5}$
0.100	7	$5.81 \cdot 10^{-6}$	$6.72 \cdot 10^{-5}$
0.057	4	$2.21 \cdot 10^{-5}$	0.00010
0.033	7	$3.13 \cdot 10^{-6}$	0.00011
0.009	3	$5.04 \cdot 10^{-6}$	0.00012
0.098	7	0.00012	0.00013
0.014	4	$1.00 \cdot 10^{-6}$	0.00013

Table 4.2: Best 8 performing configurations outputted from the Bayesian hyperparameter optimization using the 2-dimensional multi gated neural network

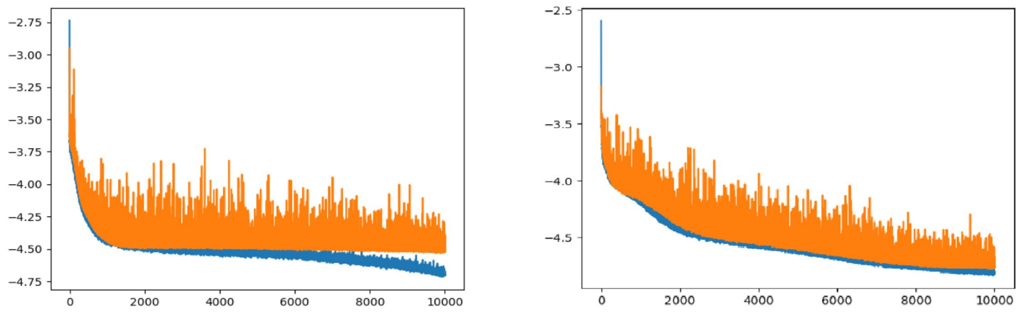


Figure 4.2: Error propagation plots 2-dimensional multi gated neural network: Epochs vs logarithm of the MSE. The orange graph represents the error propagation using the validation data set. The blue graph represent the error propagation using the train data set. The plots are based on the first and third trading day of October 2020.

4.2 Performance based on 5 consecutive trading days

In this section, we will use the found hyperparameter configuration from the performed Bayesian hyperparameter optimization to observe the performance of the 2-dimensional feedforward neural network and the 2-dimensional gated neural network. The SSVI procedure, which was introduced in section 3.2.1, is also applied for each of the 5 trading days in October. The first trading day results can be found in figure 4.3, the other trading day fits can be found in appendix B. The different MSE values for different strikes for each of the models can be found in table 4.3.

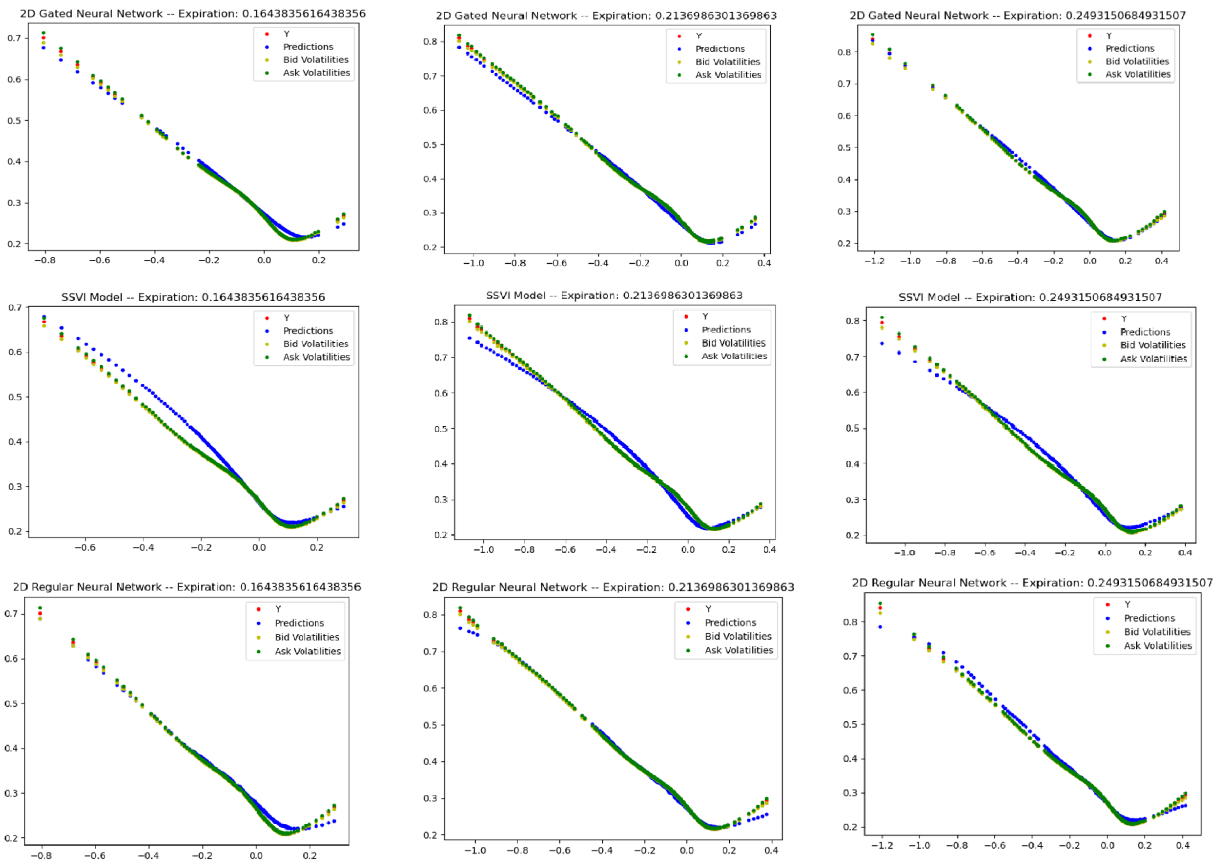


Figure 4.3: Plots corresponding to the first trading day of October. The red dots represent the mid-volatilities, the blue plots represent the fits, the yellow dots denote the bid volatilities and the green dots represent the ask volatilities.

Day + Model	**MSE**: $k < -0.2$	**MSE**: $-0.2 < k < 0$	**MSE**: $0 < k < 0.2$	**MSE**: $k > 0.2$	**MSE**
Day 1: GaNN	0.00014	$6.91 \cdot 10^{-5}$	0.00010	0.00025	0.00010
Day 2: GaNN	0.00016	$7.09 \cdot 10^{-5}$	0.00011	0.00019	0.00011
Day 3: GaNN	0.00017	$7.41 \cdot 10^{-5}$	0.00012	0.00019	0.00012
Day 4: GaNN	0.00018	$7.46 \cdot 10^{-5}$	0.00013	0.00016	0.00012
Day 5: GaNN	0.00018	$7.34 \cdot 10^{-5}$	0.00013	0.00016	0.00012
Day 1: SSVI	0.00073	0.00017	$8.14 \cdot 10^{-5}$	$7.58 \cdot 10^{-5}$	0.00030
Day 2: SSVI	0.00074	0.00018	$8.40 \cdot 10^{-5}$	$7.39 \cdot 10^{-5}$	0.00030
Day 3: SSVI	0.00074	0.00018	$9.36 \cdot 10^{-5}$	$9.08 \cdot 10^{-5}$	0.00030
Day 4: SSVI	0.00073	0.00018	$9.57 \cdot 10^{-5}$	$9.27 \cdot 10^{-5}$	0.00031
Day 5: SSVI	0.00072	0.00018	0.00010	0.00011	0.00030
Day 1: NN	0.00035	$2.84 \cdot 10^{-5}$	0.00015	0.00048	0.00017
Day 2: NN	0.00039	$5.30 \cdot 10^{-5}$	0.00013	0.00043	0.00019
Day 3: NN	0.00042	$5.11 \cdot 10^{-5}$	0.00012	0.00043	0.00019
Day 4: NN	0.00043	$4.76 \cdot 10^{-5}$	0.00015	0.00017	0.00019
Day 5: NN	0.00045	$6.69 \cdot 10^{-5}$	0.00015	0.00042	0.00021

Table 4.3: MSE performance for the SSVI, 2-dimensional gated neural network and feedforward neural network for different regions based on the log-moneyness k.

What can be clearly observed is that for log-moneyness values around 0, we do not yet see a perfect fit for the 2-dimensional feedforward neural network and the 2-dimensional multi gated neural network. Here we could improve by slightly adjusting the cost function, such that more attention will be paid to this region. The next section will look at this in more detail. In addition, we clearly see that for log-moneyness values smaller than -0.2, the 2-dimensional gated neural network comes out as the best performing one. For log-moneyness values that are greater than -0.2 and less than 0, the feedforward neural network is the best performing one. For log-moneyness values that are greater than 0, the SSVI is the best performing model. If we consider the overall MSE, we see that the 2-dimensional multi gated neural network is the best performing one.

4.3 Weighted MSE and expiration splitting

As mentioned earlier, traders may have a specific opinion of what a fit should look like to be used in practice. The part for which the log-moneyness values are around 0, should be close to perfect. For this reason we will replace the MSE from the previous section with a weighted Mean Squared Error ($wMSE$), which is defined as follows:

$$wMSE = \frac{1}{n} \frac{\sum_{i=0}^{n} w_i \cdot (y_i - \hat{y}_i)^2}{\sum_{i=0}^{n} w_i},$$

where y_i represents the desired fit, $\hat{y}_i$ the predicted value and w_i denotes the weight corresponding to point i.

This allows us to give certain volatility points a higher weight during training, so that while training a neural network extra attention will be given to making the points with log-moneyness value around 0 perfect. Specifically, all points for which the log-moneyness values are between -0.2 and 0.2 will get a higher weight. The weight we will use for these points will be equal to 10.

In addition, it is important to consider the distribution of the data, as a distribution can have a huge impact in how well a neural network is able to learn an underlying structure. This has to do with the fact that if there is much more data for certain areas, a neural network, will put too much focus on these areas. Hence, we aim to cover this potential issue by splitting the data sets into two parts. So to ensure a more accurate fit, we suggest the following: for volatility points with log-moneyness values around 0, we use higher weights in the wMSE, while at the same time we split the data set corresponding to a trading day into two parts. We will use a gated neural network and a feedforward neural network to capture the underlying structure of the volatility surface and we will apply these adjustments on the first 3 trading days of October for all expirations between 0.1 and 0.4. The distribution of the data for these trading days can be found in figure 4.4.

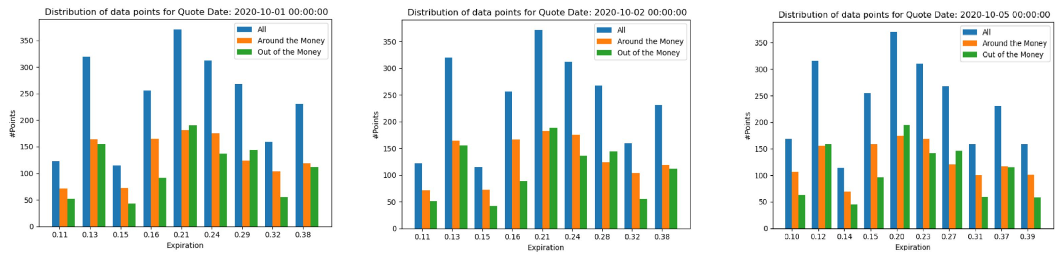


Figure 4.4: Distributions for the first three trading days of October. Around the money points correspond to points for which the absolute value of the log-moneyness value is smaller than 0.2. Out of the money points correspond to points for which the absolute value of the log-moneyness value is greater than 0.2.

The hyperparameters that are found in the previous section by applying the Bayesian hyperparameter optimization procedure, will be used in this section as well. For splitting the data sets, we have decided to examine expirations between 0.1 and 0.2 separately from those between 0.2 and 0.4. As a result, we will use two neural networks to determine a trading day's volatility surface.

For the feedforward neural network, in tables 4.4 and 4.5, the MSE values can be found for the adopted approach, where we use two feedforward neural networks to construct volatility surfaces. We observe that for log-moneyness values smaller than -0.2 some great improvements in the MSE are visible for the considered trading days compared to the procedure from the previous section where the MSE served as a cost function. For points where the log-moneyness value is between -0.2 and 0, we see equivalent MSE values for the feedforward neural network that targets expirations between 0.1 and 0.2 and slightly higher MSE values for the feedforward neural network which focuses on expirations between 0.2 and 0.4. For log-moneyness values greater than 0 and smaller than 0.2, we can find a strong improvement over the neural network with the MSE as a cost function. Looking at the overall MSE, we can conclude that this new approach is much better able to construct volatility surfaces for the considered trading days. This can also be seen in figure 4.5, where we see that for 3 chosen expirations for each of the first 3 trading days of October, we have constructed very accurate smiles, especially for the points where the log-moneyness value is around 0.

Day	**MSE**: $k < -0.2$	**MSE**: $-0.2 < k < 0$	**MSE**: $0 < k < 0.2$	**MSE**: $k > 0.2$	**MSE**
Day 1	0.00013	$4.63 \cdot 10^{-5}$	$6.94 \cdot 10^{-6}$	$8.99 \cdot 10^{-5}$	$5.49 \cdot 10^{-5}$
Day 2	0.00016	$4.36 \cdot 10^{-5}$	$9.51 \cdot 10^{-6}$	$7.89 \cdot 10^{-5}$	$5.91 \cdot 10^{-5}$
Day 3	0.00013	$3.30 \cdot 10^{-5}$	$9.39 \cdot 10^{-6}$	$7.03 \cdot 10^{-5}$	$4.95 \cdot 10^{-5}$

Table 4.4: MSE performance of the 2-dimensional feedforward neural network with wMSE for the first 3 trading days of October applied on expirations between 0.1 and 0.2. Different log-moneyness regions are studied.

Day	**MSE**: $k < -0.2$	**MSE**: $-0.2 < k < 0$	**MSE**: $0 < k > 0.2$	**MSE**: $k > 0.2$	**MSE**
Day 1	0.00020	$3.33 \cdot 10^{-5}$	$2.17 \cdot 10^{-5}$	$6.87 \cdot 10^{-5}$	$8.80 \cdot 10^{-5}$
Day 2	0.00030	0.00012	$6.38 \cdot 10^{-5}$	0.00012	0.00017
Day 3	0.00025	0.00012	$8.19 \cdot 10^{-5}$	0.00016	$7.00 \cdot 10^{-5}$

Table 4.5: MSE performance of the 2-dimensional feedforward neural network with wMSE for the first 3 trading days of October applied on expirations between 0.2 and 0.4. Different log-moneyness regions are studied.

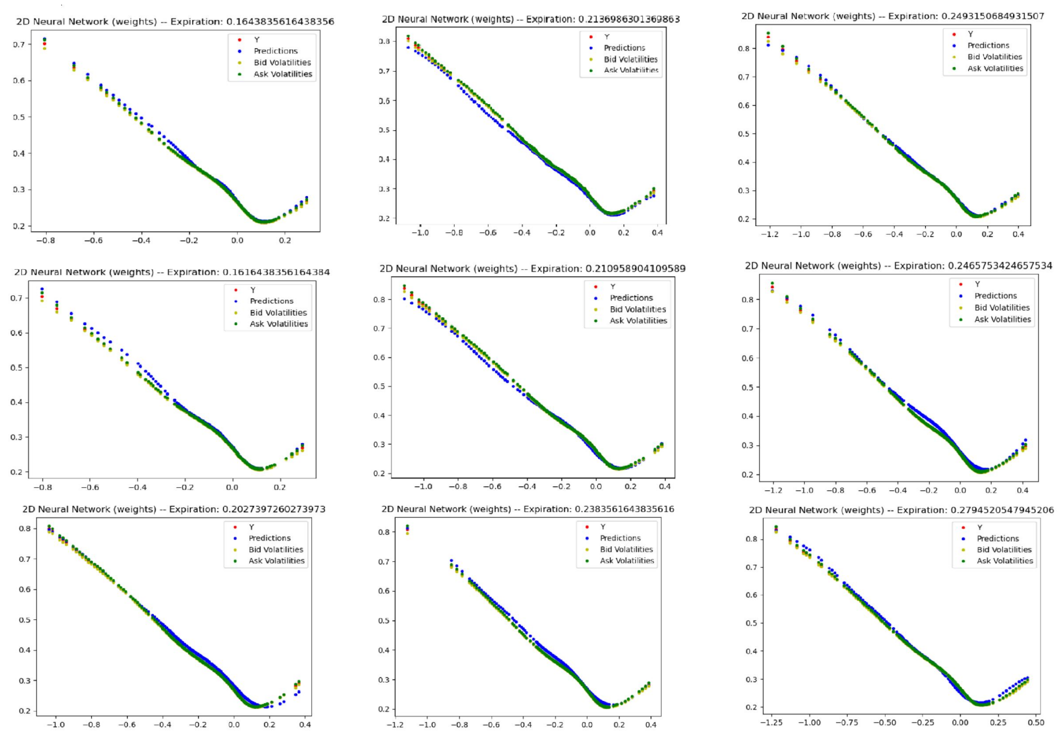


Figure 4.5: Each row corresponds to the feedforward neural network fit using the wMSE as a cost function for a different trading day. Row 1 represents the fits for 3 expirations from the first trading day, row 2 represents 3 expirations from the second trading day and row 3 represents again 3 expirations from the third trading day. The red dots correspond to the mid-volatilities, the blue dots correspond to the fits, the yellow dots correspond to the bid-volatilities and the green dots denote the ask-volatilities.

The same concept of a dual neural network approach is then also applied in exactly the same way for the 2-dimensional multi gated neural network. The results of applying this concept can be found in the tables 4.6 and 4.7. Again, we see strong improvements over the approach where we used a gated neural network network with the MSE as a loss function. For the log-moneyness values smaller than -0.2 we see great improvements, for the log-moneyness values between -0.2 and 0 we also see good improvements and also for the points for which the log-moneyness values are larger than 0 and smaller than 0.2, we see very good accuracy. So it is also no surprise that the overall MSE is also strongly improved. All in all, we can conclude that we have improved both models by the proposed modifications and we observe that even after these modifications, the 2-dimensional gated neural network is the best performing one.

Day	**MSE**: $k < -0.2$	**MSE**: $-0.2 < k < 0$	**MSE**: $0 < k < 0.2$	**MSE**: $k > 0.2$	**MSE**
Day 1	$2.21 \cdot 10^{-5}$	$3.17 \cdot 10^{-6}$	$7.82 \cdot 10^{-6}$	$4.81 \cdot 10^{-5}$	$9.87 \cdot 10^{-6}$
Day 2	$2.40 \cdot 10^{-5}$	$3.18 \cdot 10^{-6}$	$9.11 \cdot 10^{-6}$	$4.16 \cdot 10^{-5}$	$1.05 \cdot 10^{-5}$
Day 3	$2.12 \cdot 10^{-5}$	$2.89 \cdot 10^{-6}$	$9.43 \cdot 10^{-6}$	$3.81 \cdot 10^{-5}$	$9.61 \cdot 10^{-6}$

Table 4.6: MSE performance of the 2-dimensional gated neural network with wMSE for the first 3 trading days of October applied on expirations between 0.1 and 0.2. Different log-moneyness regions are studied.

Day	**MSE**: $k < -0.2$	**MSE**: $-0.2 < k < 0$	**MSE**: $0 < k < 0.2$	**MSE:** $k > 0.2$	**MSE**
Day 1	$3.25 \cdot 10^{-5}$	$1.17 \cdot 10^{-5}$	$1.26 \cdot 10^{-5}$	0.00011	$2.32 \cdot 10^{-5}$
Day 2	$2.31 \cdot 10^{-5}$	$9.56 \cdot 10^{-6}$	$1.09 \cdot 10^{-5}$	0.00015	$2.02 \cdot 10^{-5}$
Day 3	$2.14 \cdot 10^{-5}$	$8.12 \cdot 10^{-6}$	$1.08 \cdot 10^{-5}$	0.00016	$2.02 \cdot 10^{-5}$

Table 4.7: MSE performance of the 2-dimensional gated neural network with wMSE for the first 3 trading days of October applied on expirations between 0.2 and 0.4. Different log-moneyness regions are studied.

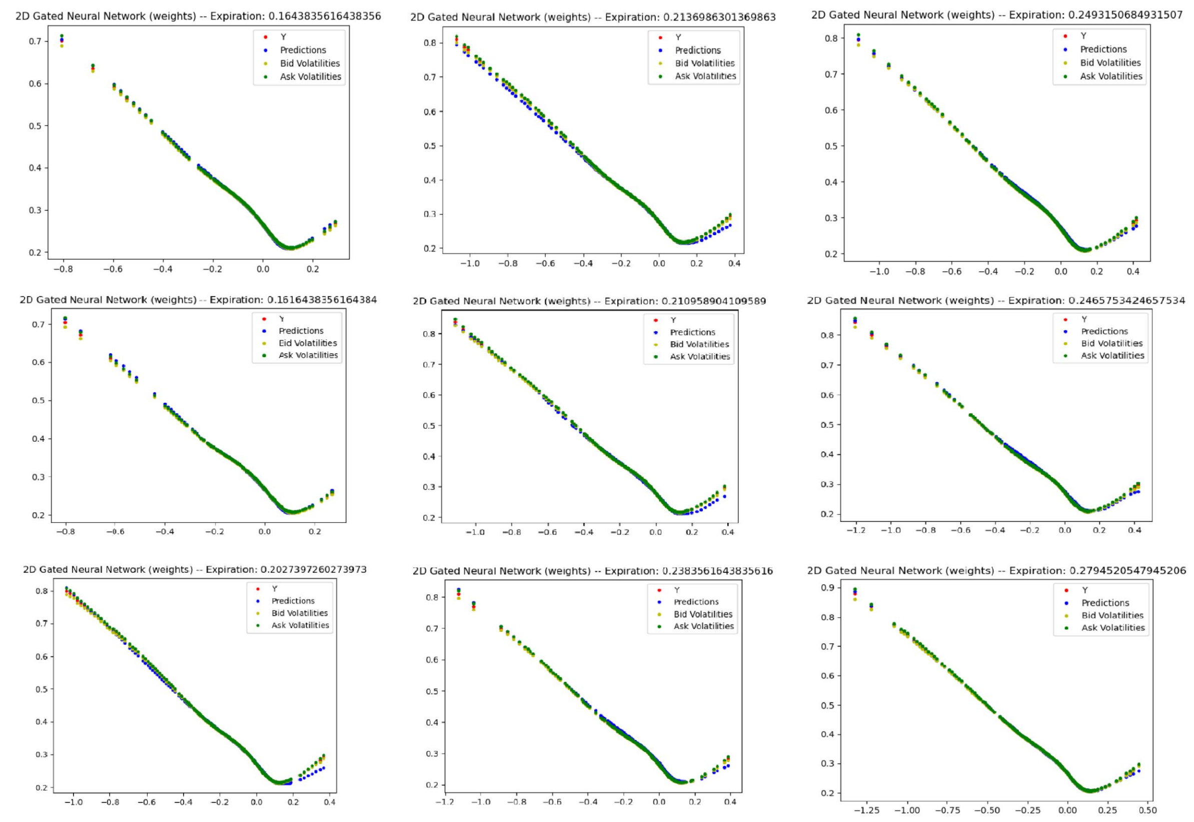


Figure 4.6: Each row corresponds to the multi gated neural network fit using the wMSE as a cost function for a different trading day. Row 1 represents the fits for 3 expirations from the first trading day, row 2 represents 3 expirations from the second trading day and row 3 represents again 3 expirations from the third trading day. The red dots correspond to the mid-volatilities, the blue dots correspond to the fits, the yellow dots correspond to the bid-volatilities and the green dots denote the ask-volatilities.

Chapter 5

Conclusions and future recommendations

In this thesis we considered different models to construct volatility surfaces, namely the SSVI, the feedforward neural network and the gated neural network, starting with an approach where we extracted the different configurations from the literature or in a rather random way, and then tested them on data for 1 trading day. In this phase, we aimed for getting familiar with the implementation and getting an impression of the performance of the different models. At this stage it has to be said that the different models deal with different objective functions or loss function. In this case the multi gated neural network gave the best fit for the volatility surface.

Next, we looked at the hyperparameter choice for the neural networks with the MSE as loss function. An optimization method called Bayesian optimization was performed to obtain appropriate hyperparameters for the gated neural network and the feedforward neural network. Subsequently, the different models with these found hyperparameters were applied to multiple trading days and it turned out that the multi gated neural network came out as the best performing model and outperformed the feedforward neural network and the SSVI model. However, more trading days are needed to strengthen this conclusion.

Finally, we considered a more advanced method of constructing a volatility surface using neural networks. This arose from two ideas: traders desire to have an accurate fit for log-moneyness values around 0, and at the same time, neural networks are more difficult to fit regions with less data. A dual neural network approach was proposed where one neural network serves for shorter expirations and the other serves for higher expirations. At the same time, the weighted mean squared error has been used as a loss function so that we can assign higher weights to points around the money. We have seen that this approach outperforms all the other approaches.

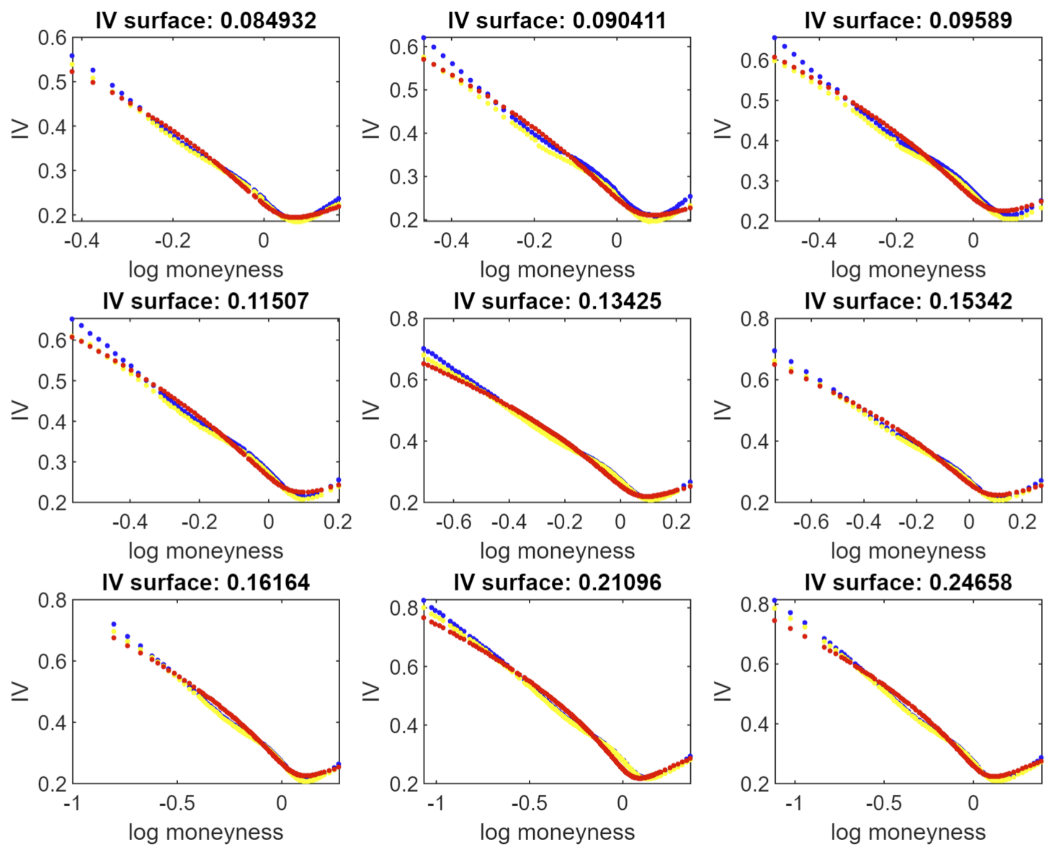
The use of neural networks has enormous potential when it comes to the construction of volatility surfaces. As a follow-up to this research, a number of directions could be taken. For example, from a theoretical perspective, it could be worth investigating why the structure of a gated neural network appears to be so capable of creating volatility surfaces For example, the structure of gated neural network could be able to automatically meet certain conditions that a volatility surface must fulfill. In addition, a hyperparameter significance analysis might be performed on the gated neural network's hyperparameters to determine which hyperparameters appear to be critical.

Finally, we proposed a dual network approach by splitting the data set in two based on the expiration date and using a weighted mean squared error to ensure higher accuracy for certain regions. We chose 0.2 as the separation point; in the future, this might be investigated further to choose a suitable approach to assign weights, which would improve accuracy. Furthermore, the assigning of weights for points around the money is an area that requires special attention and that could possibly be linked to the distribution of data, so that we are able to assign reliable weights for points around the money. In general, this thesis serves as a starting point for this approach and some research should be performed to asses the applicability of it.

Appendices

Appendix A

IV fits by SSVI



Appendix B

Performance multiple trading days

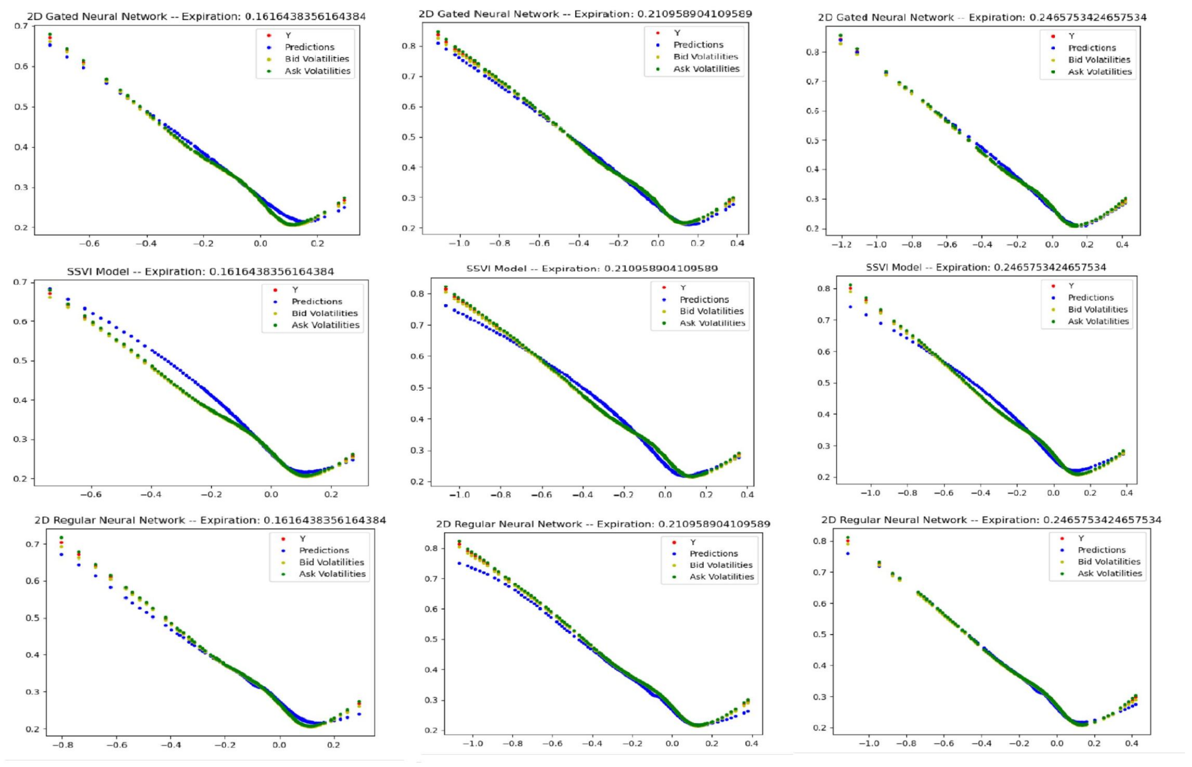


Figure B.1: Plots corresponding to the second trading day of October. The red dots represent the mid-volatilities, the blue plots represent the fits, the yellow dots denote the bid volatilities and the green dots represent the ask volatilities.

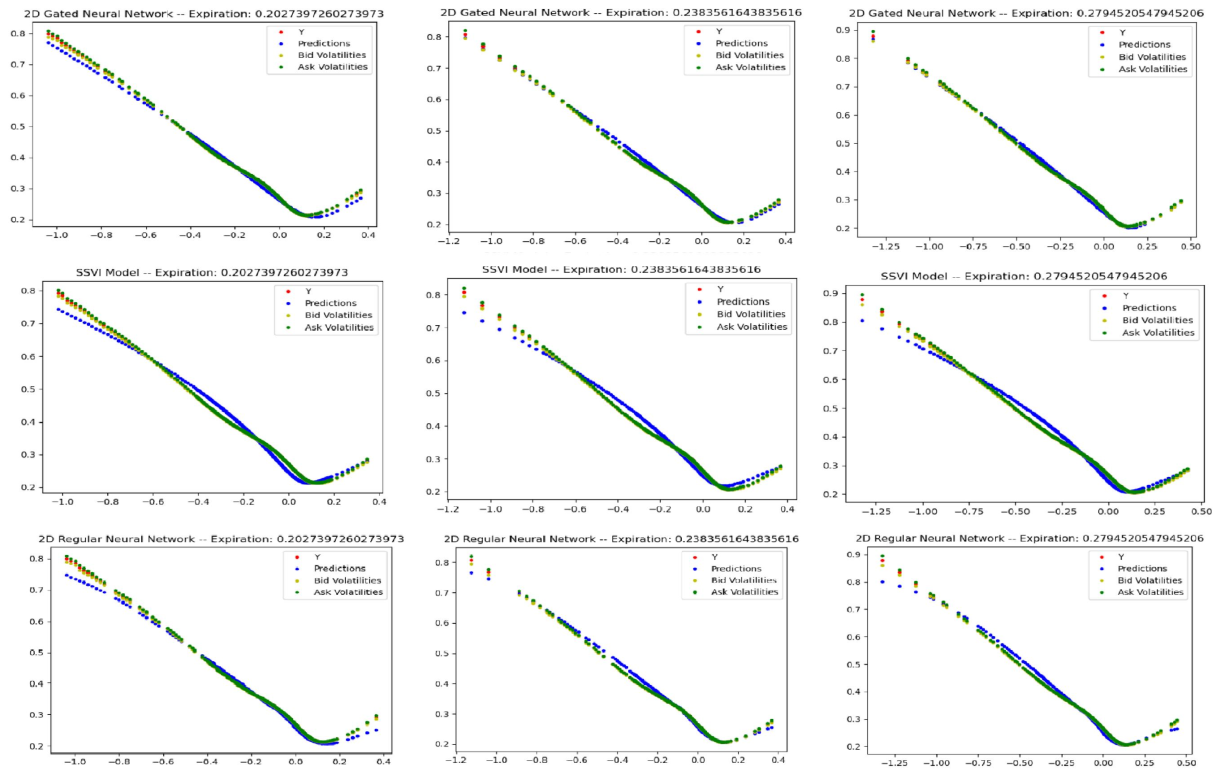


Figure B.2: Plots corresponding to the third trading day of October. The red dots represent the mid-volatilities, the blue plots represent the fits, the yellow dots denote the bid volatilities and the green dots represent the ask volatilities.

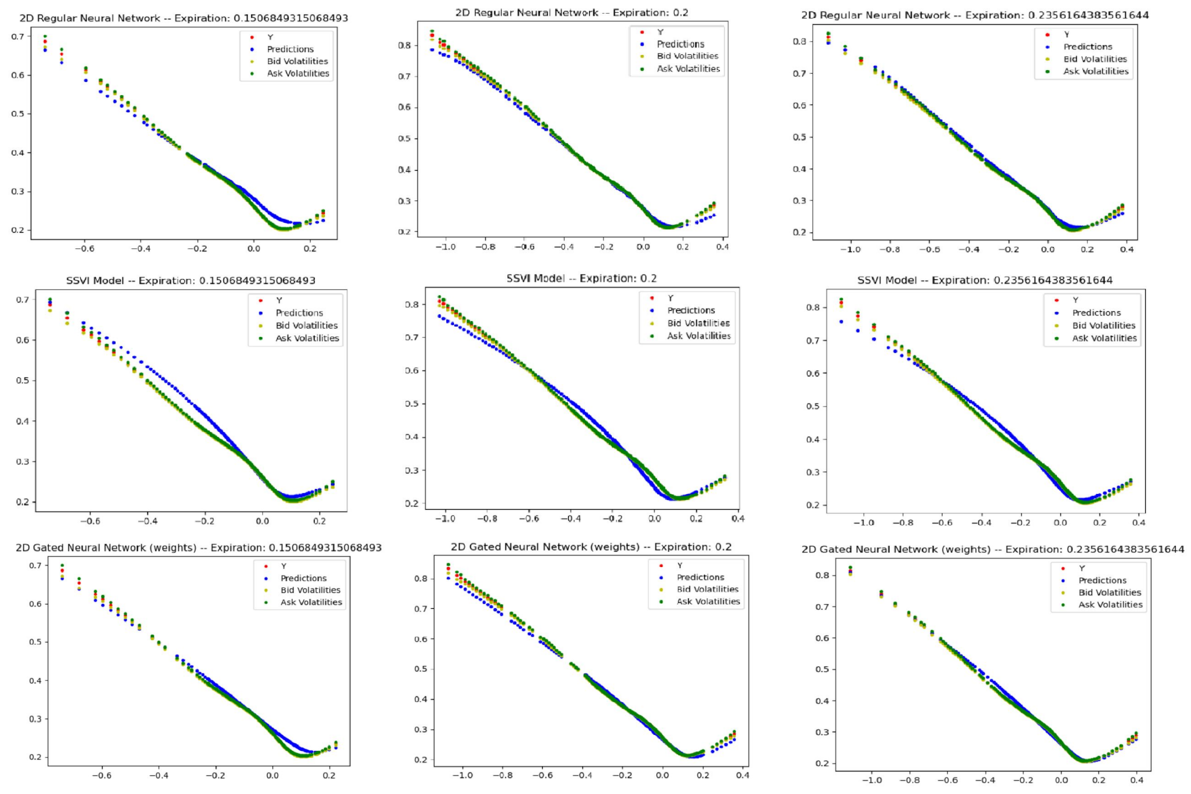


Figure B.3: Plots corresponding to the fourth trading day of October. The red dots represent the mid-volatilities, the blue plots represent the fits, the yellow dots denote the bid volatilities and the green dots represent the ask volatilities.

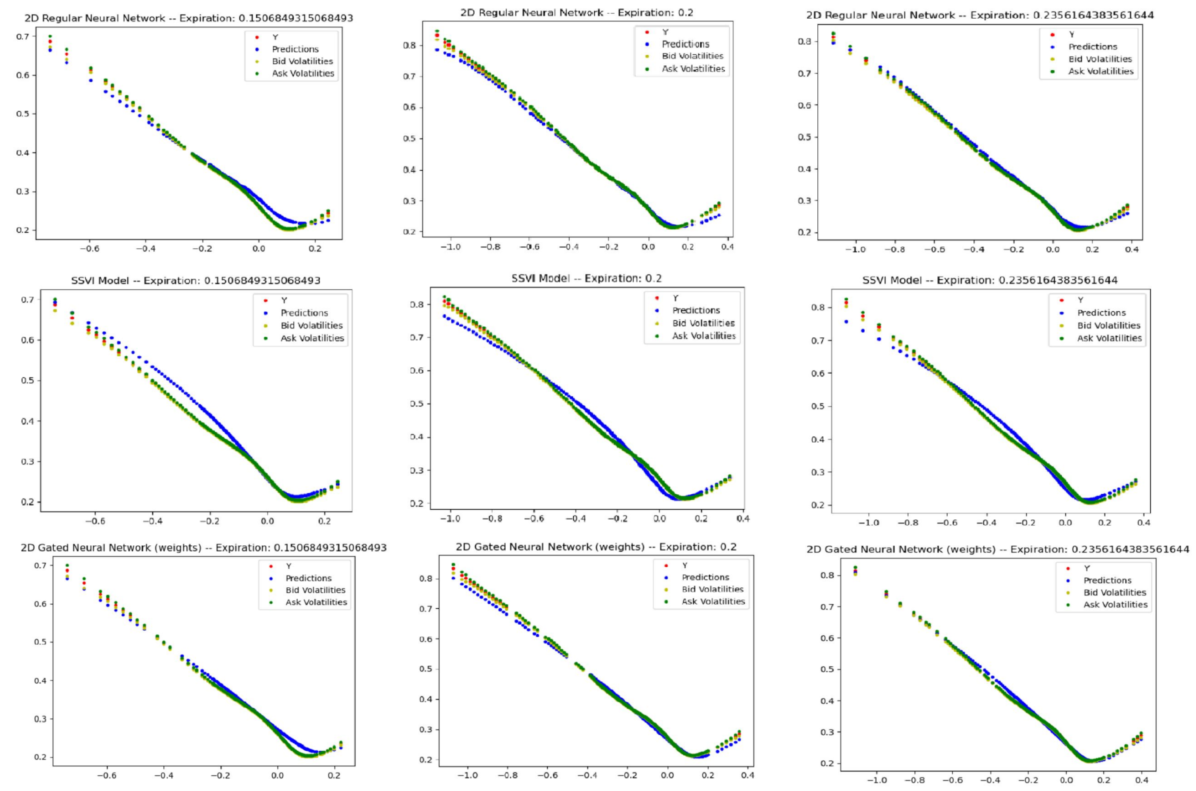


Figure B.4: Plots corresponding to the fifth trading day of October. The red dots represent the mid-volatilities, the blue plots represent the fits, the yellow dots denote the bid volatilities and the green dots represent the ask volatilities.

Bibliography

[1] Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. *Journal of political economy*, 81(3):637–654, 1973.

[2] Jim Gatheral. A parsimonious arbitrage-free implied volatility parameterization with application to the valuation of volatility derivatives. *Presentation at Global Derivatives & Risk Management, Madrid*, page 0, 2004.

[3] Jim Gatheral and Antoine Jacquier. Arbitrage-free svi volatility surfaces. *Quantitative Finance*, 14(1):59–71, 2014.

[4] Damien Ackerer, Natasa Tagasovska, and Thibault Vatter. Deep smoothing of the implied volatility surface. *Available at SSRN 3402942*, 2019.

[5] Yu Zheng, Yongxin Yang, and Bowei Chen. Gated deep neural networks for implied volatility surfaces. *arXiv preprint arXiv:1904.12834*, 2019.

[6] Cristian Homescu. Implied volatility surface: Construction methodologies and characteristics. *Available at SSRN 1882567*, 2011.

[7] Peter Carr, Hélyette Geman, Dilip B Madan 5, and Marc Yor. From local volatility to local lévy models. *Quantitative Finance*, 4(5):581–588, 2004.

[8] Wim Schoutens. *Lévy processes in finance: pricing financial derivatives.* 2003.

[9] Andreas Kyprianou, Wim Schoutens, and Paul Wilmott. *Exotic option pricing and advanced Lévy models.* John Wiley & Sons, 2006.

[10] Ole E Barndorff-Nielsen and Neil Shephard. Econometric analysis of realized volatility and its use in estimating stochastic volatility models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 64(2):253–280, 2002.

[11] Rama Cont and José Da Fonseca. Dynamics of implied volatility surfaces. *Quantitative finance*, 2:45–60, 2002.

[12] Andrey Itkin. To sigmoid-based functional description of the volatility smile. *The North American Journal of Economics and Finance*, 31:264–291, 2015.

[13] M Avellaneda, A Carelli, and F Stella. A bayesian approach for constructing implied volatility surfaces through neural networks. *Journal of Computational Finance*, 4(1):83–107, 2000.

[14] Shaikh A Hamid and Zahid Iqbal. Using neural networks for forecasting volatility of s&p 500 index futures prices. *Journal of Business Research*, 57(10):1116–1125, 2004.

[15] Thomas F Coleman, Yuying Li, and Cheng Wang. Stable local volatility function calibration using spline kernel. *Computational optimization and applications*, 55(3):675–702, 2013.

[16] Kyoko Suzuki, Tetsuya Shimokawa, and Tadanobu Misawa. Agent-based approach to option pricing anomalies. *IEEE transactions on evolutionary computation*, 13(5):959–972, 2009.

[17] Francesco Audrino and Dominik Colangelo. Semi-parametric forecasts of the implied volatility surface using regression trees. *Statistics and Computing*, 20(4):421–434, 2010.

[18] Antonio Gulli and Sujit Pal. *Deep learning with Keras*. Packt Publishing Ltd, 2017.

[19] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[20] Rahul R Marathe and Sarah M Ryan. On the validity of the geometric brownian motion assumption. *The Engineering Economist*, 50(2):159–192, 2005.

[21] Takeaki Kariya and Regina Y Liu. Options, futures and other derivatives. In *Asset Pricing*, pages 9–26. Springer, 2003.

[22] Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. In *World Scientific Reference on Contingent Claims Analysis in Corporate Finance: Volume 1: Foundations of CCA and Equity Valuation*, pages 3–21. World Scientific, 2019.

[23] Volatility smile. https://en.wikipedia.org/wiki/Volatility$_{s}mile/media/File : Volatility_{s}mile.svg. Accessed : 2020-05-22$.

[24] Richard L Burden and J Douglas Faires. Numerical analysis, 2011.

[25] Sebas Hendriks and Claude Martini. The extended ssvi volatility surface. *Available at SSRN 2971502*, 2017.

[26] Douglas T Breeden and Robert H Litzenberger. Prices of state-contingent claims implicit in option prices. *Journal of business*, pages 621–651, 1978.

[27] LCG Rogers and MR Tehranchi. Can the implied volatility surface move by parallel shifts? *Finance and Stochastics*, 14(2):235–248, 2010.

[28] Jim Gatheral and Antoine Jacquier. Convergence of heston to svi. *Quantitative Finance*, 11(8):1129–1132, 2011.

[29] Peter K Clark. A subordinated stochastic process model with finite variance for speculative prices. *Econometrica: journal of the Econometric Society*, pages 135–155, 1973.

[30] Kenji Suzuki, Hiroyuki Abe, Heber MacMahon, and Kunio Doi. Image-processing technique for suppressing ribs in chest radiographs by means of massive training artificial neural network (mtann). *IEEE Transactions on medical imaging*, 25(4):406–416, 2006.

[31] Li Deng, Geoffrey Hinton, and Brian Kingsbury. New types of deep neural network learning for speech recognition and related applications: An overview. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 8599–8603. IEEE, 2013.

[32] Deep neural networks theory. https://medium.com/pharos-production/deep-neural-networks-theory-part-1-7af6fcfdd59f. Accessed: 2020-05-22.

[33] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

[34] David E Rumelhart, Richard Durbin, Richard Golden, and Yves Chauvin. Backpropagation: The basic theory. *Backpropagation: Theory, architectures and applications*, pages 1–34, 1995.

[35] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[36] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[37] Agnes Lydia and Sagayaraj Francis. Adagrad—an optimizer for stochastic gradient descent. *Int. J. Inf. Comput. Sci.*, 6(5), 2019.

[38] Luke Taylor and Geoff Nitschke. Improving deep learning using generic data augmentation. *arXiv preprint arXiv:1708.06020*, 2017.

[39] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):1–48, 2019.

[40] MS Windows NT what is underfitting and overfitting in machine learning and how to deal with it. https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803a989c76. Accessed: 2020-05-22.

[41] Rainer Storn and Kenneth Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997.

[42] David JC MacKay. Introduction to gaussian processes. *NATO ASI series F computer and systems sciences*, 168:133–166, 1998.

[43] Bayesian optimization. http://krasserm.github.io/2018/03/21/bayesian-optimization/. Accessed: 2020-05-22.

[44] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

[45] Donald R Jones, Cary D Perttunen, and Bruce E Stuckman. Lipschitzian optimization without the lipschitz constant. *Journal of optimization Theory and Applications*, 79(1):157–181, 1993.

[46] Sudarshan K Valluru, Rajul Kumar, and Rahul Kumar. Design and real time implementation of fmincon, moga tuned io-pid and fo-piλdμ controllers for stabilization of trms. *Procedia Computer Science*, 171:1241–1250, 2020.

[47] Uday K Chakraborty. *Advances in differential evolution*, volume 143. Springer, 2008.

[48] Vitaliy Feoktistov. *Differential evolution*. Springer, 2006.

[49] Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. An efficient approach for assessing hyperparameter importance. In *International conference on machine learning*, pages 754–762. PMLR, 2014.

[50] Giles Hooker. Generalized functional anova diagnostics for high-dimensional functions of dependent variables. *Journal of Computational and Graphical Statistics*, 16(3):709–732, 2007.