ANALYSIS OF SABR CALIBRATION WITH NEURAL NETWORKS. AN APPLICATION TO THE SWAPTION SMILE

Eugenio Martín Gallego

Trabajo de investigación 014/019

Master en Banca y Finanzas Cuantitativas

Directores: Luis Manuel García Muñoz
Fernando de Lope Contreras
Javier García Siles

Universidad Complutense de Madrid

Universidad del País Vasco

Universidad de Valencia

Universidad de Castilla-La Mancha

Analysis of SABR Calibration with Neural Networks

An Application to the Swaption Smile

Eugenio Martín Gallego

Msc in Banking and Quantitative Finance

Supervisors:
Luis Manuel García Muñoz
Javier García Siles
Fernando de Lope Contreras

Contents

Abstract

The objective of this thesis is to analyze the properties of feedforward neural networks as functional approximators of financial models. In particular, we aim to calibrate the swaption smile through the SABR model. Two different calibration schemes based on Hagan's analytical approximation are developed and analyzed. One scheme proposes to use the network as an inverse map, while the other makes use of the neural network as a substitute to the functional in the traditional calibration method.

Keywords: SABR, neural networks, feedforward, fixed income, derivatives, volatility smile, calibration, functional approximator, Hagan, Universal Aproximation Theorem.

Introduction

- So, what should we do tomorrow?
- There's a world of possibilities!

Since its inception in the early 40s, artificial neural networks have been developed with the aim to mimic the connectivity of the human brain. Starting as simple connections of neuron-like functions that receive an input and produce an output, neural networks have evolved to be an active applied to many different fields, from medical diagnosis through image recognition to speech translation, passing through the famous AlphaZero program.
Neural networks *learn* to perform tasks by observing examples without being programmed to follow any specific rule. They take some inputs to which they perform non-linear transformations at the artificial neurons and return an output. Along the learning process the weights on these transformations are updated so that the output of the network matches the output of the training data.
Feedforward neural networks are acyclic, fully connected neural networks that can act as a mathematical function that maps a set of inputs to some output values, as described by the Universal Approximation Theorem. Following this result, we aim to apply this functional approximation feature to the financial field. In quantitative finance models without closed formula are usually avoided, as their calibration process has a high computational cost, favoring simplicity over accuracy. If the neural network approximation is an accurate representation of the model it can be used as a substitute, decreasing drastically the model's computational cost.
The objective of this thesis is to calibrate the SABR model through a neural network. The SABR is a stochastic volatility model widely used by practitioners that aims to capture the volatility smile of financial instruments, specially those on interest rates. Although there is no closed-form solution to the model, an accurate and fast approximate solution exists as an asymptotic expansion of the parameters. Our objective is to substitute this expansion during the calibration process with a neural network, showing that a good degree of accuracy can be reached. If it proves to be accurate, this procedure could then be applied to unused model whose calibration time is computationally prohibitive.
The thesis is divided in five different sections. The first chapter is an introduction to interest rate derivatives and its valuation, while the second contains all the necessary information to understand feedforward neural networks. The third chapter corresponds to an introduction to the SABR model and the modifications that it admits to accept negative forward rates. Chapter 4 explains the different methodology followed to calibrate the model through neural networks is described, while in chapter 5 these calibration methods are developed and analysed.

Chapter 1

Theoretical background

Interest rate topics covered in the research are reviewed in this chapter, following principally the scope presented in Brigo and Mercurio (2007). For a more thorough introduction into any of these topics the reader should resort to this reference.

1.1 Basic definitions

- **Zero-coupon bond.** Contract that guarantees the payment of one unit of currency at time T without intermediate payments. At time $t < T$ the value of this contract is $P(t,T)$ and denotes the present value of one unit of currency to be paid at time T.
- **Year fraction and day-count convention.** Time to maturity, $\tau(t,T) = T - t$ is understood as the amount of time in years between the dates t and T, following the appropriate day-count convention, which defines how to measure the amount of time between both dates. There are several possible conventions, detailed in Miron and Swannell (1991) [1].
- **Tenor.** Amount of time left until the financial contract expires or, if the contract is an interest rate swap or similar, the length of time during this payments, expressed in the appropriate day-count convention.

For an investment started at time t of $P(t,T)$ units of currency until maturity T, the constant interest rate at which this investment has to be made to produce one unit of currency at time T, according to the accruing, is:

- **Simply-compounded rate.** If accruing is proportional to investment time

$$L(t,T) = \frac{1 - P(t,T)}{\tau(t,T)P(t,T)} \tag{1.1}$$

- **Anually-compounded rate.** If reinvesting the obtained amount once a year.

$$Y(t,T) = \frac{1}{P(t,T)^{\frac{1}{\tau(t,T)}}} - 1 \tag{1.2}$$

[1] An up-to-date description can be found at http://www.deltaquants.com/day-count-conventions

- **Continuously-compounded rate.** If accruing happens continuously

$$R(t,T) = -\frac{\ln(P(t,T))}{\tau(t,T)} \tag{1.3}$$

Next we shall look to some mathematical definitions that apply to finance. The reader should be accustomed to some basic stochastic calculus concepts, otherwise they can be reviewed at Shreve (2004).

- **Bank account (Money-market account)**: it represents a riskless investment where money is accrued continuously at the risk-free rate present in the market at every instant t. The money kept in this account accrues continuously (at every instant t) the instantaneous spot rate $r(t)$, therefore this process follows the equation:

$$dB(t) = r(t)B(t)dt \tag{1.4}$$

  where $B(0)$ is the amount invested at time $t = 0$. Consequently,

$$B(t) = B(0)\exp\left(\int_0^t r(s)ds\right) \tag{1.5}$$

- **Stochastic discount factor**: the continuously-compounded (stochastic) discount factor between time instants t and T is the amount at time t equivalent to one unit of currency at time T $(t < T)$. It is defined by

$$D(t,T) = \frac{B(t)}{B(T)} = \exp\left(\int_t^T r(s)ds\right) \tag{1.6}$$

1.2 Forward rates

A forward rate is typically defined from a *forward-rate agreement* (FRA). This contract involves three time instants, the contract (valuation) time t, the expiry time T and the maturity time S where $t < T < S$, and it gives its holder an interest rate payment between for the period $[T,S]$. A fixed payment is exchanged against a floating payment based on the spot rate $L(T,S)$ at maturity, therefore the contract allows to assure an interest rate *(K)* for the time period $[T,S]$. The value of this contract in t (for a nominal value N) can be derived to obtain the expression

$$\mathbf{FRA}(t,T,S,\tau(T,S),K,N) = N[P(t,S)\tau(T,S)K - P(t,T) + P(t,S)] \tag{1.7}$$

There is only one value of K that turns this contract fair at t:

- **Simply-compounded forward interest rate**: the most likely simply-compounded interest rate expected at time t for the investment period $[T,S]$ is defined by

$$F(t;T,S) = \frac{1}{\tau(T,S)}\left(\frac{P(t,T)}{P(t,S)} - 1\right) \tag{1.8}$$

Taking the limit when $S \to T^+$ of this definition

- **Instantaneous forward interest rate**: the instantaneous forward interest rate prevailing at time t for maturity $T > t$ is defined as

$$f(t,T) = \lim_{S \to T^+} F(t;T,S) = -\frac{\partial \log P(t,T)}{\partial T} \tag{1.9}$$

Instantaneous forward rates, although fundamental to the interest rates theory, cannot be observed directly in the markets, and they have to be bootstrapped from interest-rate curves. There are several important *interbank* interest rates used in the European markets:

- **LIBOR** (***L****ondon* ***I****nter****b****ank Offered Rate*) is an interest rate average calculated by the major banks in London where each bank estimates how much they would be charged it they borrowed from other banks. These rates are calculated for five currencies and seven borrowing periods (overnight to one year).
- **EURIBOR** (***Eur****o* ***I****nter****b****ank* ***O****ffered* ***R****ate*) is a daily reference rate that represents the average rate at which Eurozone banks would offer to lend unsecured funds to other banks in the interbank market. Generally interest rate derivatives are referenced to this index
- **OIS** (***O****vernight* ***I****ndexed* ***S****wap*) it is a daily reference based on an interest rate swap where the periodic floating payment is a geometric mean of a specific daily overnight index. It is the common choice to be used as the risk-free rate proxy.

1.3 Mathematical framework

The fundamental assumption applied through financial literature since Black and Scholes (1973) is the absence of arbitrage opportunities in the financial market under analysis, model that has been developed in Harrison and Pliska (1981) and Harrison and Pliska (1983). Under the typical framework [2] (which can also be found in Shreve (2004)), the main contributions are the two fundamental theorems of asset pricing.

Theorem 1.3.1. *If a market model has a risk-neutral probability measure, then it does not admit arbitrage.*

Theorem 1.3.2. *Consider a market model that has a risk-neutral probability measure. The market is complete if and only if the risk-neutral probability measure is unique.*

Where a market is said to be complete if and only if every contingent claim is attainable. Therefore given the *equivalent martingale measure* Q[3], the unique price associated to an attainable contingent claim V at any instant of time t is obtained through the conditional expectation under the Q measure of the discounted value at maturity of the contingent claim

$$V(t) = B(t)\,\mathbb{E}^{\mathbb{Q}}\left[\frac{V(T)}{B(T)}\middle|\mathscr{L}_t\right] = \mathbb{E}^{\mathbb{Q}}\left[\exp\left(-\int_t^T r(s)ds\right)V(T)\middle|\mathscr{L}_t\right] \tag{1.10}$$

[2] A probability space $(\Omega, \mathscr{L}, P)$ with the natural right-continuous filtration $\mathscr{L} = \{\mathscr{L}_t : 0 \leq t \leq T\}$

[3] It is a probability measure defined on the probability space, that follows certain properties: the *equivalence* to the original probability measure of the space, the existence of the Radon-Nikodym derivative and the "discounted asset price" process is a martingale under this measure (all the properties can be found in Brigo and Mercurio (2007))

where we have used as numeraire[4] the bank account. The presence of the discount factor inside the conditional expectation can hinder the derivative valuation if interest rates are considered stochastic, therefore the use of other possible numeraires under their relative measure can be helpful towards the valuation:

- **T-forward measure**: uses a zero-coupon bond with maturity T as numeraire. A contingent claim will have an unique value of

$$V(t) = P(t,T)\,\mathbb{E}^{\mathbb{F}^T}\left[\frac{V(T)}{P(T,T)}\middle|\mathscr{L}_t\right] = P(t,T)\,\mathbb{E}^{\mathbb{F}^T}\left[V(T)|\mathscr{L}_t\right] \tag{1.11}$$

- **Swap measure**: denoted as $\mathbb{Q}^{\alpha,\beta}$, this measure takes as numeraire a portfolio of zero coupon bonds with maturity $T_1,...,T_n$, whose value at $t < T_i$ is

$$C_{T_0,T_n}(t) = \sum_{i=1}^{n}(T_i - T_{i-1})P(t,T_i) \tag{1.12}$$

  Therefore the unique value of a contigent claim at time $t < T_0, T_1, ...T_n$ is

$$V(t) = C_{T_0,T_n}(t)\mathbb{E}^{\mathbb{Q}_{\alpha,\beta}}\left[\frac{V(T_0)}{C_{T_0,T_n}(T_0)}\middle|\mathscr{L}_t\right] \tag{1.13}$$

To change between these measures the Radon-Nikodym derivative is used, as explained in Geman *et al.* (1995).

1.4 Interest-rate derivatives

Several of the most used financial instruments are related to interest rates:

- **Interest-Rate Swap (IRS)**: it is a portfolio of the **FRA** presented in equation 1.7. This contract is an agreement between two parties to exchange a given number of cash flows during a fixed period of time (the time between resets is called *tenor*). The cash flows can come from a *fixed* rate (K) or a *floating* rate [5], indexed to a forward interest rate, where the *payer swap* receives the floating and pays the fixed rate, and the *receiver swap* pays the floating rate and receives the fixed one. The discounted payoff at time $t < T_i$ for pay dates $T_1, ..., T_n$ [6] of a payer IRS is:

$$\mathbf{IRS}_{\text{PAYER}} = N\sum_{i=1}^{n} DF(t,T_i)\tau_i(F(T_{i-1},T_i) - K) \tag{1.14}$$

- **Swaption**: a European payer swaption is a contract that gives its holder the right to enter a payer IRS at a given time, the swaption maturity (T_0). The payoff of the swaption with underlying described in equation 1.14, observed at time $t < T_0$ is

$$\mathbf{SWPT}_{\text{PAYER}} = DF(t,T_0)N\left(\sum_{i=1}^{n} P(T_0,T_i)\tau_i(F(T_0;T_{i-1},T_i) - K)\right)^{+} \tag{1.15}$$

[4]A numeraire is a positive, non-dividend paying asset that acts as a reference to normalize all other assets with respect to it. Equation 1.10 will still hold under the new numeraire, since as shown in the paper self-financing strategies hold this property under a numeraire change.

[5]A thorough description of IRS types can be found in Hull (2003). Note that usually tenors for different legs are of different length.

[6]Here fixed-rate and floating-rate payments occur at the same dates to simplify notation.

The swaption is At-the-money if (using equation 1.8 and telescoping property):

$$K_{ATM} = S_{T_0,T_n}(t) = \frac{\sum_{i=1}^{n} P(t,T_i)\tau_i F(t;T_{i-1},T_i)}{\sum_{i=1}^{n} \tau_i P(t,T_i)} = \frac{P(t,T_0) - P(t,T_n)}{C_{T_0,T_n}(t)} \tag{1.16}$$

where $S_{T_0,T_n}(t)$ is known as the forward swap rate for period $[T_0, T_n]$ evaluated at time t. Consequently, the payoff of a payer swaption can be expressed in function of this rate as:

$$\mathbf{SWPT}_{\text{PAYER}} = N\left(S_{T_0,T_n}(T_0) - K\right)^+ C_{T_0,T_n}(T_0) \tag{1.17}$$

The fair value of a payer swaption given the payoff at the option's expiry date (T_0) under the swap measure is, at time $t < T_0$,

$$V(t) = C_{T_0,T_n}(t) N\, \mathbb{E}^{\mathbb{Q}_{\alpha,\beta}}\left[\left(S_{T_0,T_n}(T_0) - K\right)^+ \Big| \mathscr{L}_t\right] \tag{1.18}$$

1.5 Interest-rate models

There have been several simple models that have prevailed to quote prices in the market. In this section the most popular models are presented, while valuing a swaption under them.

1.5.1 Bachelier (Normal) model

Introduced by Bachelier (1900), it is the simplest model that allows for negative forward rates, and is therefore widely used in the industry to quote implied normal volatilities from instrument prices. The instantaneous forward rate $f(t)$ follows the equation:

$$dF_t = \sigma_N\, dW_t \tag{1.19}$$

where W_t is a standard brownian motion under the T-forward measure and σ_N is the constant Normal volatility. The solution to this stochastic differential equation (SDE) is

$$F(t) = F(0) + \sigma_N\, W_t \qquad\qquad W_t \sim \mathcal{N}(0, \sqrt{t}) \tag{1.20}$$

Allowing for negative forward rates is a property that should follow every model that expects to adjust to current market quotes, although it has to be dealt with carefully. Even though forward rates rarely fall below a certain level, in this model they follow a normal distribution, which allows forward rates to become arbitrarily negative.
Under the Bachelier model the value of a swaption is, from equation 1.17:

$$V_{\text{PAYER}}(T, C_{T_0,T_n}, f, K, \sigma_N) = N C_{T_0,T_n}(t)\left[(f - K)\Phi(d) + \sigma_N \sqrt{T_0}\phi(d)\right] \tag{1.21}$$

where f refers to the forward swap rate $S_{T_0,T_n}(t)(T_0)$, T_0 is the expiry date (when the option expires and the swap begins), $\Phi(\cdot)$ is the normal cumulative distribution function, $\phi(\cdot)$ is the normal probability density function and

$$d = \frac{f - K}{\sigma_N \sqrt{T_0}} \tag{1.22}$$

1.5.2 Black (lognormal) model

Introduced in the expansion to commodity contracts of the Black-Scholes model in Black (1976), the instantaneous forward rate follows the stochastic differential equation

$$dF_t = \sigma_B F_t \, dW_t \tag{1.23}$$

where σ_B is the lognormal constant volatility and W_t is a brownian motion under the T-forward measure. The solution to this SDE is

$$F_t = F_0 \, \exp\left(\sigma_B W_t - \frac{1}{2}\sigma_B^2 t\right) \tag{1.24}$$

Following this model, the value of a swaption at time t with expiry date T, strike K and notional N is, from equation 1.17:

$$V_{\text{PAYER}}(T, C_{T_0,T_n}, f, K, \sigma_N) = N C_{T_0,T_n}(t)\,[f\Phi(d) - K\Phi(d)] \tag{1.25}$$

where $\Phi(\cdot)$ is the normal distribution function, and

$$d_{\pm} = \frac{\log\left(\frac{f}{K}\right) \pm \frac{1}{2}\sigma_B^2 T}{\sigma_B \sqrt{T}} \tag{1.26}$$

As it can be observed the future's process follows a lognormal distribution, therefore it does not allow for negative forward rates, which is an undesired property in the current state of the market. The option's value, shown in equation 1.25, exposes another drawback of this model. The value depends on the logarithm of the quotient of the forward rate and the strike, therefore if one of them is negative the logarithm is not defined, and even it both of them are negative, rendering the logarithm positive, the model was not designed to be applied under those circumstances.

1.5.3 Shifted Black model

Although forward rates can be negative, they usually present a lower limit close to zero, therefore a slight modification to the Black model can be made so it can be applied to the current financial environment. In this model both the forward rate and the strike are shifted upwards by a constant parameter s, so the instantaneous forward rate follows the process:

$$dF_t = \sigma_B(F_t + s)\, dW_t \tag{1.27}$$

The new shift is introduced in both the forward rate and the strike. Defining $\tilde{F_t} = F_t + s$ and $\tilde{K} = K + s$, the valuation formula in equation 1.25 of any derivative in the shifted Black model is equivalent to Black's model.
This parameter needs to be chosen a priori and quoted along the volatility of the financial instrument, and its value will be high enough to surpass negative rates but low enough so the forward rate process and the value of the financial instrument are not distorted. This approach has drawbacks, since the model might need to be recalibrated if the value of the forward rate decreases below the chosen shift value.

Chapter 2

Deep Feedforward Networks

Artificial neural networks (ANN) are computational schemes originally created with the objective of emulating the behaviour of biological neural networks, which are formed by thousands of connected neurons organized in several consecutive layers, receiving signals (inputs) that are sent and transformed through all the network to produce outputs. Similarly, ANNs are organized in multiple layers, made up by neurons (or units) where at each neuron a function is applied to the inputs received from previous layers, with the objective of mapping a set of initial values $\vec{\boldsymbol{x}}$ to their outputs $\vec{\boldsymbol{y}} = f(\vec{\boldsymbol{x}}; \vec{\boldsymbol{\theta}})$ while *learning* the parameters $\vec{\boldsymbol{\theta}}$ that give a better function (f) approximation for the output.
Feedforward networks are the most popular neural networks used, and the backbone to many other architectures such as convolutional networks (used for image recognition) or recurrent networks (when feedback connections are included, and the model learns from itself). In *feedforward* networks information flows from the set of inputs $\vec{\boldsymbol{x}}$ through the intermediate computations that define f to the output $\vec{\boldsymbol{y}}$ directly and acyclically, and can be seen as a mapping $f : \mathbb{R}^m \to \mathbb{R}^n$.
They are *networks* due to these intermediate computations, which consist in the composition of several functions $f(\vec{\boldsymbol{x}}) = \left(f^{(3)} \circ f^{(2)} \circ f^{(1)}\right)(\vec{\boldsymbol{x}})$ where the superscript represent the order of the layer, being $f^{(1)}(\vec{\boldsymbol{x}})$ the input layer and $f^{(3)}(\cdot)$ the output layer. The total length of the function composition is the *depth* of the model, while the intermediate layers where the output values are not interpretable are called *hidden* layers (layers 1 and 2). Each layer consists of a given number of neurons or units (the *width* of the layer), where a function transforms the input from the previous layer's units to a scalar.
The process in which the parameters $\vec{\boldsymbol{\theta}}$ of these layers are learnt is called *training*. In this step, the training data (a set of inputs $\vec{\boldsymbol{x}}$ and outputs $\vec{\boldsymbol{y}} = f^*(\vec{\boldsymbol{x}})$) define the output layer, which must return a value the closest possible to $\vec{\boldsymbol{y}}$ for each input $\vec{\boldsymbol{x}}$. The behaviour of the hidden layers will not be specified by this data but by the learning algorithm applied, which will define the values of the parameters for each unit in these layers.
In the next sections the information about these features is expanded, looking mostly at their applications within feedforward networks.

2.1 Architecture design

One of the main decisions to take when building a neural network is its structure: the number of hidden layers the network will have, the number of units per layer and how these units will be connected between themselves, and the activation function for each layer. In 2.1 a prototypical (one hidden layer) feedforward neural network is presented, where all the layers are fully connected.

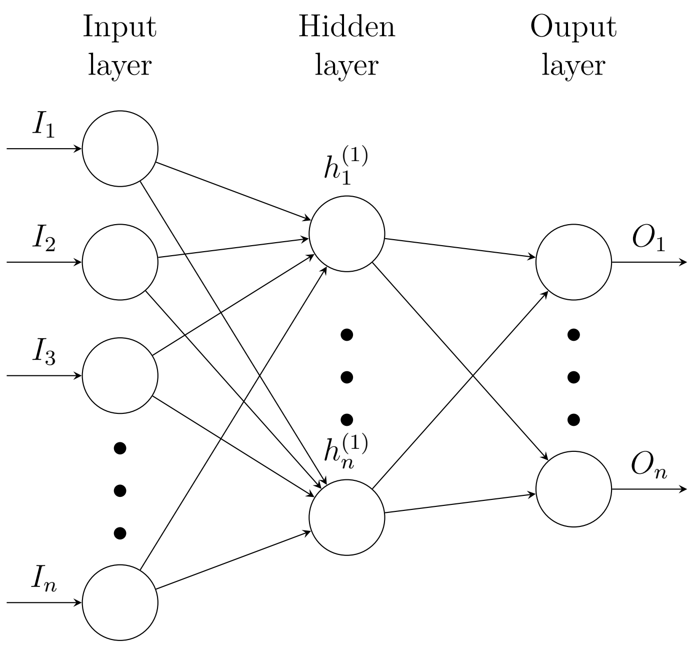

Figure 2.1: Architecture of a feedforward neural network

The input for each layer is a function of the preceding layer's outputs weighted by a factor and modified by a bias term

$$\boldsymbol{h}^{(i)} = g^{(i)}\left(\boldsymbol{W}^{(i)}\vec{\boldsymbol{h}}^{(i-1)} + \vec{\boldsymbol{b}}^{(i)}\right) \tag{2.1}$$

where $\boldsymbol{W}^{(i)}$ is the weight matrix with dimension $m \times n$, being m the number of neurons in the i^{th} layer and n the number of neurons in the $(i-1)^{th}$ layer, and $\vec{\boldsymbol{b}}$ is the vector of bias terms of the i^{th} layer. To obtain an appropriate method to approximate non-linear functions, a function $g(\cdot)$ is applied to this affine transformation, and its choice will be discussed in section 2.2 Although there is not a given set of rules to define the structure of the network, experimentation throws some insight. Regarding the number of hidden layers a single hidden layer is enough in theory to approximate a continuous function (as shown in section) given it has enough neurons, however the ability to approximate the function with a reduced number of parameters to estimate drastically improves with the use of multiple layers. We can motivate theoretically this observation through the result obtained by Eldan and Shamir (2015) regarding the power of depth of neural networks:

Theorem 2.1.1. *There is a simple (approximately radial) function on $\mathcal{R}^d$, expressible by a small 3-layer feedforward neural networks, which cannot be approximated by any 2-layer network, to more than a certain constant accuracy, unless its width is exponential in the dimension.*

Even though the type of function that this theorem refers to is restrictive, it can give an idea about the motivation towards increasing the number of hidden layers.

2.2 Activation function

As explained in the previous section, the input for each unit is a function of the previous units. There are three different types of layers, each with different types of activation functions:

- **Input layer**: it is comprised of the input vector $\boldsymbol{X}$

$$h^{(0)} = \underbrace{X}_{\left(n^{[0]}\times 1\right)} \tag{2.2}$$

- **Hidden layers**: Layers that connect the input and the output. Each hidden neuron in these layers is generated from a function of an affine transformation of the previous layer values, as described in 2.1. Each next hidden layer will be a function of the previous layer outputs.

$$\underbrace{h^{(i)}}_{(n^{[i]}\times 1)} = g^{(i)}\left(\underbrace{W^{(i)}}_{(n^{[i]}\times n^{[i-1]})}\underbrace{h^{(i-1)}}_{(n^{[i-1]}\times 1)} + \underbrace{b^{(i)}}_{(n^{[i]}\times 1)}\right) \tag{2.3}$$

- **Output layer**: this layer returns the output $\boldsymbol{Y}$ of the classification or regression problem. Its activation function is different from the hidden layer's functions: the softmax function is used for multiclass classification problems, the sigmoid for binary classification and the linear function for regression problems.

$$Y = \underbrace{h^{(n)}}_{(n^{[n]}\times 1)} = g^{(n)}\left(\underbrace{W^{(n)}}_{(n^{[n]}\times n^{[n-1]})}\underbrace{h^{(n-1)}}_{(n^{[n-1]}\times 1)} + \underbrace{b^{(n)}}_{(n^{[n]}\times 1)}\right) \tag{2.4}$$

The training of the neural network is done by the backpropagation algorithm, who works by propagating the error gradient from the outputs to the inputs, through each hidden neuron. Consequently, to facilitate these computations, the activation function $g^{(i)}(\cdot)$ in the hidden layers should be a non-linear function whose gradient does not vanish or explode when updating the parameters. When the gradient descent algorithm progresses to the lower layers the gradient becomes smaller, and it may vanish. If the input is large (either positive or negative) the value of the function may saturate at 0 or 1, as it is shown in Glorot and Bengio (2010) that happens to the sigmoid function. The most used non-linear activation functions, together with their gradients, are shown in figure 2.1.

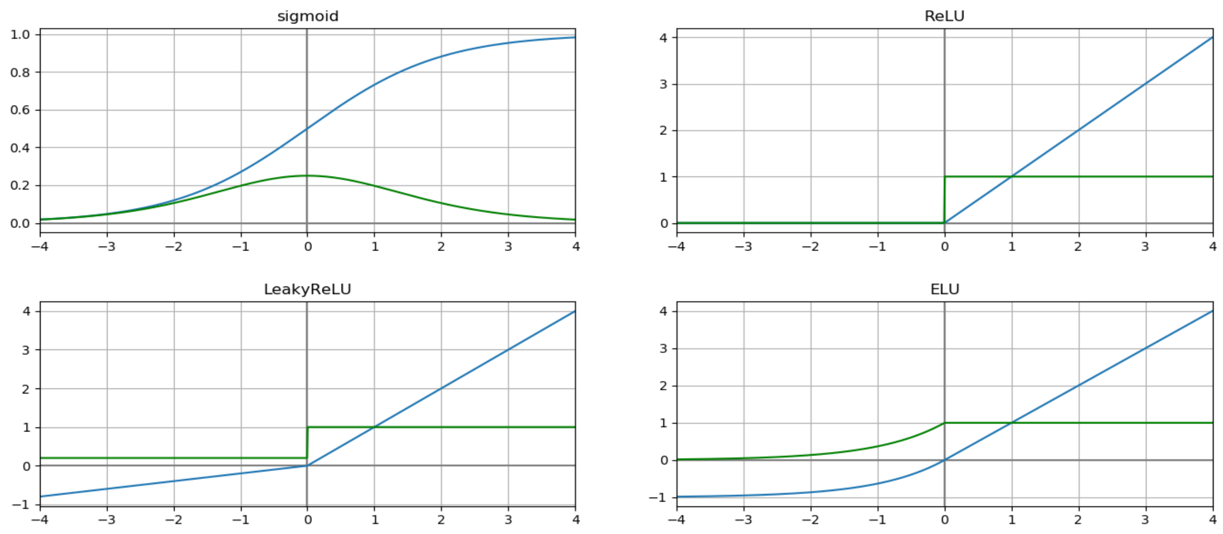

Figure 2.2: Most used non-linear activation functions at the hidden units. The blue line denotes the output of the function, and the green line its derivative

First, we are going to motivate why linear activation cannot be used in the hidden layers. If we consider a one hidden layer with linear activation function, then the output layer would be:

$$Y = h^{(2)} = W^{(2)}h^{(1)} + b^{(2)} = W^{(2)}\left(W^{(1)}X + b^{(1)}\right) + b^{(2)} = WX + b \tag{2.5}$$

which is linear to the inputs, rendering the neural network into a linear regression.
The sigmoid function takes a real number as input and outputs a value in $S(z) \in (0, 1)$. It has some of the desired properties, such as being smooth, monotonic and differentiable, however the sigmoid saturates the values, rendering the gradient null and giving place to the vanishing gradient problem. This problem causes the network to stop learning, since the gradient has become null and cannot update further the parameters.

$$h_{\text{sigmoid}}(z) = \frac{1}{1 + e^{-z}} \tag{2.6}$$

Also its output its not centered around 0, making optimization harder. To solve this issue the hyperbolic tangent started being used, but this function still presents the vanishing gradient problem.
The Rectified Linear Units (ReLU) function $h_{\text{ReLU}}(z) = max(0, z)$, introduced in Glorot *et al.* (2011), solves the vanishing gradient problem (at least for positive values). Since this function is simpler the training will be faster than with the other activation functions. However if $z < 0$ the gradient is also 0, and will not update any further with the network, causing the neuron to die. To solve this problem the Leaky ReLU function appeared, defined as $h_{\text{LReLU}}(z) = max(\alpha z, z)$. The hyper-parameter α defines how much the function "leaks" when $z < 0$, solving the dying neuron problem by introducing a small slope for the negative values. There are several possible values for α and usually it is fixed at $\alpha = 0.01$, however there are other possible solutions that can be found at Xu *et al.* (2015).
Another kind of activation function is the Exponential Linear Unit (ELU), proposed in Clevert *et al.* (2015). There it was observed that it outperformed all variants of ReLU, reducing the training time despite being more computationally costly, since it converges faster. The typical value of the parameter is $\alpha = 1$, and the function has all the desired properties. It is smooth everywhere, which improves the results from gradient descent, at $z < 0$ the gradient is not zero and it takes negative values in this region, causing the mean output of the unit to be closer to zero and consequently reducing the vanishing gradients problem.

$$h_{\text{ELU}}(z) = \begin{cases} \alpha(e^z - 1) & \text{if } z < 0 \\ z & \text{if } z \geq 0 \end{cases} \tag{2.7}$$

Therefore the ELU activation unit should return the best results when used as activation unit for the hidden layers.

2.3 Parameters initialization

The first idea when initializing the parameters is to set them as 0 and let the learning algorithm do its job. However, if every weight is initialized at the same value, the output will be symmetrical to the value of all units in the same layer, which will turn the gradient to be also the same for every weight in the same layer. The gradient descent algorithm will fail to converge to a good model since it will update each weight to the same amount, rendering useless the existence of multiple neurons in the same hidden layer. Consequently, the weights will be initialized randomly, and usually the bias terms are set to 0 (there is one for each neuron). However, if the initial values are too big we could incur in the vanishing or exploding gradients problem, and if they are too small we will return to the original problem. Following He *et al.* (2015), their objective is that the variance of the random sampling is the same for every layer. For all variants of the ReLU function (and ELU) they derive that the initial values for the weights have to be drawn from the distribution $\mathcal{N}\left(0, \dfrac{4}{n_{\text{input}} + n_{\text{output}}}\right)$. This setup is usually called *He_normal* initialization.

2.4 Gradient-Based Learning

For each hidden layer there are $n^{[i-1]} \times n^{[i]} + n^{[i]}$ parameters to optimize, so in order to train the neural network it is not computationally feasible to use second order methods (based on the Hessian) to optimize the cost function, and we must fall back to gradient descent algorithms. These are first order iterative algorithms used to find the minimum of a function by taking steps that are proportional to the negative of the gradient (as will be explained in section 2.6). The computation of the gradient of this error function (in regression problems this function is the mean squared error) is done through backward algorithmic differentiation. When applied to neural networks this algorithm consists of two steps:

- **Forward propagation**: given the input $\mathbf{X}$ we compute the different layer activations until we reach the output layer, where the error metric is calculated for each output neuron. For k training samples, the algorithm is computed as follows:

$$\underbrace{h^{(1)}}_{n^{[1]}\times k} = g^{(1)}\left(\underbrace{W^{[1]}}_{n^{[1]}\times n^{[0]}} X \oplus \underbrace{b^{[1]}}_{n^{[1]}\times 1}\right)\Bigg|_{\text{k samples}}$$

$$\vdots$$

$$\underbrace{h^{(i)}}_{n^{[i]}\times k} = g^{(i)}\left(\underbrace{W^{[i]}}_{n^{[i]}\times n^{[i-1]}} h^{[i-1]} \oplus \underbrace{b^{[i]}}_{n^{il]}\times 1}\right)\Bigg|_{\text{k samples}}$$

$$\vdots$$

$$Y' = \underbrace{h^{(n)}}_{n^{[n]}\times k} = g^{(n)}\left(\underbrace{W^{[n]}}_{n^{[n]}\times n^{[n-1]}} A^{[n-1]} \oplus \underbrace{b^{[n]}}_{n^{[n]}\times 1}\right)\Bigg|_{\text{k samples}}$$

$$\vdots$$

$$J(\epsilon) = J\left(Y_{\text{training data}}, Y'_{\text{NN output}}\right) = \frac{1}{2k}\sum_{i=1}^{k}(Y - Y')^2$$

  where $\oplus$ represents the sum term by term and the function g^i is applied term by term. The cost function corresponds to a regression problem, and will be different if it is a classification problem.

- **Backpropagation**: once the cost function is computed through forward propagation we have to compute the derivative of the errors to update the weights according to the gradient descent algorithm used. We want to compute the derivative of the cost function for every weight and bias in the neural network, and this can be done efficiently applying the chain rule, that allows us to compute unknown derivatives from previously computed derivatives. If $Z^{[i]} = W^{[i]}Z^{[i-1]} + b^{[i]}$, then the gradient of the cost function for the parameters is:

$$\frac{\partial \epsilon}{\partial Y'}$$

$$\frac{\partial \epsilon}{\partial Z^{[i]}} = \frac{\partial Y'}{\partial Z^{[i]}}\frac{\partial \epsilon}{\partial Y'} \qquad , \qquad Y' = \sigma^{[i]}\left(Z^{[i]}\right)$$

$$\frac{\partial \epsilon}{\partial h^{[i-1]}} = \frac{\partial Z^{[i]}}{\partial h^{[i-1]}}\frac{\partial \epsilon}{\partial Z^{[i]}}, \quad \frac{\partial \epsilon}{\partial W^{[i]}} = \frac{\partial Z^{[i]}}{\partial W^{[i]}}\frac{\partial \epsilon}{\partial Z^{[i]}}, \quad \frac{\partial \epsilon}{\partial b^{[i]}} = \frac{\partial Z^{[i]}}{\partial b^{[i]}}\frac{\partial \epsilon}{\partial Z^{[i]}}$$

$$\vdots$$

We can observe that as we go down in the hidden layers the partial derivatives depend from previously calculated derivatives, and this feature is the one that this algorithm tries to exploit to improve the efficiency of the neural network, as is described in Hecht-Nielsen (1992)

2.5 Minibatch gradient descent

Usually a large number of samples are used to train the neural network, and with the large number of operations that are carried out in the aforementioned algorithms, if we used all the available samples to compute the gradient the computations would take too much time and the learning process would be too slow. To solve this problem, the training data is divided into minibatches, which will act as the whole training sample when computing the gradient. If the minibatch size is high enough (usually 32,64,128... samples) it will act as a good approximation of the whole training sample, with the benefit of reducing the computing time. Obviously the larger the size the better the approximation is, but the network will take longer to be trained. The model parameters δ (weights and biases) are updated according to the gradient descent formula, which will be described in section 2.6.
The training data is split into subsets of that size and the gradient is computed only with the data in that subset, updating the parameters only with this information. This process is repeated for each subset until all the training data is used (the process is called an epoch), and this process is repeated again with all the data until the learning is finished.

2.6 Adam algorithm

ADAptive Moment estimation or Adam algorithm, first defined in Kingma and Ba (2014), is an optimization algorithm that incorporates several improvements over the classical (minibatch) gradient descent method, which updates the parameters following the equation:

$$\theta_n := \theta_{n-1} - \eta\frac{\partial \epsilon}{\partial \theta} \tag{2.8}$$

Adam algorithm incorporates the following features:

- **Variance reduction**: due to the noise introduced by the minibatch gradient when the gradient is low we can estimate the gradient variance by an exponential weighted moving average. The usual value for the parameter is $\beta_2 = 0.999$

$$S_n = \beta_2 S_{n-1} + (1-\beta_2)\left(\left.\frac{\partial \epsilon}{\partial \theta}\right|_n\right)^2 \tag{2.9}$$

- **Momentum**: it estimates the gradient as an exponentially weighted moving average of the previous gradients, with the aim of reducing the noise of the network. It allows to go past local optima by using the gradient as an accelerator. This feature introduces an additional parameter called momentum, and its usual value is $\beta_1 = 0.9$

$$V_n = \beta_1 V_{n-1} + (1 - \beta_1) \frac{\partial \epsilon}{\partial \theta}\bigg|_n \tag{2.10}$$

Introducing these changes, the (minibatch) gradient descent formula is

$$\theta_n := \theta_{n-1} - \eta \frac{V_n}{\sqrt{S_n + \epsilon}} \tag{2.11}$$

The smoothing term ϵ is initialized close to 0, and its objective is that the denominator never falls reaches 0.
The η parameter, called *learning rate* can be reduced with the number of epochs. It is useful when the algorithm has reached a plateau and the learning rate is too big and does not allow the gradient to descend to the minimum.

2.7 Universal Approximation Theorem

This result, first introduced by Hornik *et al.* (1989), is the theoretical basis that motivates the investigation carried out through this thesis. This theorem guarantees that there is a feedforward network with a linear output layer and at least one hidden layer with a sigmoid-like activation function that can approximate any Borel-measurable function (any continuous function on a closed and bounded set of $\mathbb{R}^n$), given it has enough hidden units. This result guarantees that the function can be approximated, however, it does not guarantee that the neural network can learn this approximation. Also, it says that there is a neural network large enough, but does not define how large it needs to be (which may turn the network unlearnable).
This theorem has also been proven for width-bounded networks for ReLU activation functions, as in Lu *et al.* (2017).

Theorem 2.7.1. *For any Lebesgue-integrable function* $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}$ *and any* $\epsilon > 0$ *there exists a fully connected ReLU network* $\mathcal{A}$ *with width* $d_m \leq n + 4$ *such that the function* $\mathcal{F}_{\mathcal{A}}$ *represented by the network satisfies*

$$\int_{\mathbb{R}^n} |f(x) - F_{\mathcal{A}}(x)|\, dx < \epsilon \tag{2.12}$$

It can also be shown in a similar fashion for convex functions and ReLU activation functions (Hanin (2017)). Finally, in a recent article by Ohn and Kim (2019) where it is analysed the approximation ability of deep neural network for a wide range of activation functions (including the elu function), the authors show that all of them are valid activation functions to approximate any Hölder smooth function and give a limit for the depth and width of the neural network.

Chapter 3

SABR Model

3.1 Introduction to the SABR model

Models described in section 1.5 are widely used accross the industry to price financial interest rate derivatives due to its simplicity. However, as explained in the aforementioned section, they also present major drawbacks, and the most important one is the assumption of a constant volatility across different strikes, since market observation shows that options for different strikes need different implied volatilities to agree with market prices, a phenomena often called *volatility smile*.
There have been multiple proposals to model this smile, and one of the most used ones is the *SABR model* (which stands for *Stochastic Alpha Beta Rho*), presented in Hagan *et al.* (2002) with the objective of parametrizing the volatility *smile* observed in markets. Its fast adoption as an industry standard has been mainly due to the fast and accurate approximate closed-form expression for the implied volatility obtained in the original paper for both Black and Bachelier models.
The SABR model is a 2-factor stochastic model whose state variables are F_t, the forward interest rate at maturity T and σ_t, its volatility parameter, and it follows the system of SDEs:

$$dF_t = \sigma_t \, F_t^\beta \, dW_t \qquad F_0 = f \tag{3.1}$$
$$d\sigma_t = \nu \, \sigma_t \, dZ_t \qquad \sigma_0 = \alpha \tag{3.2}$$
$$d[W, Z]_t = \rho \, dt \tag{3.3}$$

where $\nu > 0$, $-1 \leq \rho \leq 1$ and $0 \leq \beta \leq 1$
The parameter α is the initial value of the volatility, ν is the volatility of the volatility of the forward rate and β is the power parameter. The parameter ρ defines the correlation between the Brownian Motions W_t and Z_t under the T-forward measure. The forward rate is specified for a single forward rate, so it does not allow forward rates of different maturity to interact among each other. Consequently, this model is not valid if we want to describe the movements of the yield curve, but it is valid for fitting and interpolating the implied volatilities observed for a single exercise date. As parameter α increases its value the level of the smile increases. When parameter ρ increases the steepness of the curve increases, and the curvature of the smile is proportional to ν.
In the next sections we present the implied volatility approximation and several paths to overcome the negative interest rates problem.

3.2 Analytical approximation to the SABR model

As explained in the SABR introduction one of the reasons for the popularity of this model is the existence of an approximated analytical solution for the stochastic differential equations that returns the implied volatility for either the Black or Bachelier (normal) models. This volatility should later be introduced in the corresponding formula to obtain the price of the derivative option. Both expressions can be found in Hagan *et al.* (2002) but here only the normal volatility scheme is presented, since it will be the one used through the empirical sections.
The implied volatility for an option with maturity at time T, with strike K and present value of the forward rate f is, if f_{av} is the arithmetic (or geometric) mean of f and K

$$\sigma_N(f,K,T) = a_N(f,K,T)\frac{\alpha(f-K)}{b_N(f,K)}\frac{z(f,K)}{g(z)} \tag{3.4a}$$

$$a_N(f,K,T) = 1 + \left(\frac{2\gamma_2-\gamma_1^2}{24}\alpha^2 f_{av}^{2\beta} + \frac{1}{4}\alpha\rho\nu\gamma_1 f_{av}^{\beta} + \frac{2-3\rho^2}{24}\nu^2\right)T \tag{3.4b}$$

$$b_N(f,K) = \frac{1}{1-\beta}\left(f^{1-\beta} - K^{1-\beta}\right) \tag{3.4c}$$

$$\gamma_1(f,K) = \frac{\beta}{f_{av}}, \qquad \gamma_2(f,K) = \frac{-\beta(1-\beta)}{f_{av}^2} \tag{3.4d}$$

$$z(f,K) = \frac{\nu}{\alpha}\frac{f-K}{f_{av}^{\beta}} \tag{3.4e}$$

$$g(z) = \log\left(\frac{\sqrt{1-2\rho z+z^2}+z-\rho}{1-\rho}\right) \tag{3.4f}$$

When the option is at-the-money $(f=K)$ then the approximation reduces to:

$$\sigma_N(f,K,T) = \alpha f^{\beta} a_N(f,K,T) \tag{3.5}$$

This asymptotic solution is developed in the original paper under the assumption that $\alpha^2 T \ll 1$.

3.3 Model calibration

The SABR model presents four parameters (α,β,ρ,ν) that need to be calibrated to market data. A standard approach is to prefix the β parameter, since its influence on the volatility smile is similar to the influence of ρ (β if we look at the model a local volatility perspective and ρ from a stochastic volatility perspective). and calibrating them together might not yield stable results (Tan (2012)). Since β also controls the movement of the smile under movements of the forward rate f, this value is fixed externally according to the dynamics of the forward rate.
The simplest method is to calibrate the three parameters directly by minimizing the sum of quadratic errors:

$$\hat{\alpha},\hat{\rho},\hat{\nu} = \arg\min_{\alpha,\rho,\nu}\sum_i\left(\sigma_i^{\text{mkt}} - \sigma(f,K_i,T;\alpha,\rho,\nu)\right)^2 \tag{3.6}$$

An extension to this method is to only fit parameters ρ and ν, and let α be implied from inverting the equation (3.5) for the ATM market volatility:

$$\left(\frac{-\beta(2-\beta)}{24f^{2(1-\beta)}}Tf^{\beta}\right)\alpha^3 + \left(\frac{f^{\beta}\rho\beta\nu T}{4f^{1-\beta}}\right)\alpha^2 + \left(1+\frac{2-3\rho^2}{24}\nu^2 T\right)f^{\beta}\alpha - \sigma^{\text{ATM}} = 0 \tag{3.7}$$

At every iteration this method defines α in terms of ρ and ν by solving the cubic polynomial root for $\alpha(\rho,\nu)$. This polynomial root might have more than one real root, of which it is suggested in West (2005) to choose the the lowest positive root.

3.4 Normal SABR model

This version of the SABR model is the only one that allows to model negative interest rates without any further modifications. It is usually denominated normal SABR in accordance with the Bachelier model 1.5.1, since the parameter β is fixed to 0. However, it is important to remember that despite the name, the forward rate does not follow a normal distribution due to the correlation between the two brownian motions.. The expression for the implicit volatilities (obtained from the Bachelier model) is the equivalent of equation 3.4 for $\beta = 0$:

$$\sigma_N(f,K,T) = \frac{(f-K)}{g(z)}\left(1+\frac{2-3\rho^2}{24}\nu^2 T\right)\nu \tag{3.8a}$$

$$z(f,K) = \frac{\nu}{\alpha}(f-K) \qquad g(z) = \log\left(\frac{\sqrt{1-2\rho z+z^2}+z-\rho}{1-\rho}\right) \tag{3.8b}$$

When the option is at-the-money $(f = K)$ then this function reduces to:

$$\sigma_N(T) = \alpha\left(1+\frac{2-3\rho^2}{24}\nu^2 T\right) \tag{3.9}$$

3.5 Shifted SABR model

In this version both the forward rate and the strike are shifted by an amount s, similarly to the shifted Black model in 1.5.3. This allows the model to change the lower bound on the forward rate from 0 to -s at the expense of adding an additional parameter to be set in advance. The SDEs that drive this modification, under the T-forward measure, are

$$dF_t = \sigma_t\,(F_t+s)^\beta\,dW_t \qquad F_0 = f \tag{3.10}$$
$$d\sigma_t = \nu\,\sigma_t\,dZ_t \qquad \sigma_0 = \alpha \tag{3.11}$$
$$d[W,Z]_t = \rho\,dt \tag{3.12}$$

and the equation for the implied volatilities obtained through Hagan's approximation (equation 3.4) is equivalent but displacing the strike and forward rates.

Chapter 4

Methodology

The objective of this thesis is to propose an alternative to the calibration step described in section 3.3 through the implementation of a neural network, whose properties and basics of use are described in chapter 2. The SABR model is a stochastic model that has become a market standard to interpolate the implied volatility surface of options due to its fast and accurate asymptotic expansion 3.4, which maps model parameters to approximations of the implied volatility, but is limited at the core by this expansion. When fit to market data, Hagan's formula can yield errors in the implied volatility greater than 1% for large maturities or extreme strikes, following the author's analysis in Antonov *et al.* (2013). In this paper an alternative based on integration is developed at the cost of increasing computation time. Several other alternatives based in different techniques have been proposed to solve the SABR equation in 3.1, such as Montecarlo simulations (for example Chen *et al.* (2011) and Leitao *et al.* (2017)) or finite element methods (Horvath and Reichmann (2018)). All this alternative methods lack computational speed, so there is room to explore for faster alternatives that could at least keep up or even improve Hagan's formula accuracy.
Following this consideration and looking at the nature of neural networks, it is a possibility worth considering trying to apply this technique to map these parameters to obtain implied volatilities. This process could reduce the computing time needed to obtain these smiles, reducing the calibration speed by translating the process to an off-line preprocessing step while improving the accuracy. Additionally, this procedure could be applied to other complex models that are too costly from a computational standpoint. Another benefit, although not contemplated in this thesis, is that neural networks are fully differentiable, and can therefore provide simple access to parameter sensitivities and hedging.
Taking a look at the SABR structure needed to fit negative interest rates (normal and shifted SABR), there are five parameters that could be subject to fitting if they are not set fixed, α, β, ρ, ν and the shift. However, throughout this thesis we have decided to use the normal SABR model, which allows negative rates without the need to shift the forward rate, reducing therefore the number of free parameters in the model from five to three. From a financial point of view this choice is supported by the current dynamic of the forward rate and has already been adopted by some analysts. Meanwhile, from a computational point of view, reducing the number of free parameters to use in the neural network reduces the number of degrees of freedom, which as a result could improve the predictions calculated by the neural network, since there will be one less output where the neural network error could be propagated.
Ideally the two objectives that we would like to obtain and that would allow this approach to start being evaluated as an alternative path are

- To obtain a fast functional approximator, reducing computational bottlenecks.
- To increase the accuracy when fitting to data compared to standard methods.

Whether these objectives are attained or not, if the results are within reasonable bound errors the thesis would show that this approach is valid and that it could be applied to computationally heavy schemes.
We propose two alternatives based on neural networks, one where we calibrate the parameters directly to market data and other where we follow a two-step approach, where first the neural network learns the model and then it is used to calibrate the data. In the following sections these methods are described, developed and analysed.

4.1 Direct calibration to market data

The most direct approach towards creating a neural network to calibrate the SABR parameters that best fit any smile is to obtain as outputs of this network these three parameters, while having as inputs the different strikes that form the curve along with their corresponding volatilities and the maturity of the swaption. We would therefore have a network that takes all the available data about the smile and predicts the parameters that could have produced this curve. Mathematically, our aim is to calibrate the SABR model by considering the inverse map

$$\mathcal{F}^{-1}\left(\left\{\sigma_i^{MKT}\right\}_{i=1}^{n}, \{K_i\}_{i=1}^{n}, T_j\right) \rightarrow \hat{\theta} \tag{4.1}$$

that performs the calibration task directly through the neural network.
Although this approach is simple it has drawbacks, since we cannot control the function $\mathcal{P}^{-1}$ and we cannot guarantee that this learned mapping will satisfactorily return the correct parameters when exposed to new data (market data). As stated above, the neural network takes as inputs:

- The forward rate, since we are working with swaptions it corresponds to forward swap rate
- Eleven strikes, defined as differences with respect to the forward (ATM) rate in basic points in an evenly spaced interval, $K \in [-250, 400]$. The input is given to the network in absolute values.
- Eleven implied normal volatilities, corresponding to the aforementioned strikes.
- Time to maturity of the swaption, where T is an integer number in the interval $T \in [5, 30]$

While the output of the neural network will be, since we have decided to define the SABR model with no shift and $\beta = 0$:

- The three SABR parameters, α, ρ, ν

The main type of error committed through this approach is the error that the neural network commits when mapping the aforementioned inputs to the three SABR parameters, and can be observed from samples contained in the test set.

4.2 Two step calibration approach

This method divides into two steps the neural network defined above, being a method more similar to traditional procedures, where there is a function that depends on some parameters that needs to be calibrated to data. First we create a neural network that learns the map from parameters to implied volatilities (offline procedure) and then we calibrate the outputs of this map to market data. This procedure overcomes some problems from the previous method since even if the model is faced by out of sample data, it will solve the calibration problem if the mapping is accurate.
The main characteristic and also the main advantage of this calibration method is that the neural network completely substitutes the function mapping (in this case Hagan's formula). This can prove to be a great step towards starting to explore more complex models that have typically been depreciated due to its heavy computational cost even if they return better results. If we had chosen to solve numerically the true distribution of the SABR model by any of the methods described in the introduction of this chapter we would have been interested in reducing the time cost of computing at every iteration the time-consuming operations needed to calculate the map. Training a neural network with simulations from this models translates to an off-line training step (when we do not need to use the model) all the time that we would spend computing the model online, since if we are able to obtain an accurate enough approximation, the neural network is almost instantaneous predicting the output.
This procedure can be described formally in two steps:

1. Learn the map $\mathcal{F}(\theta, \xi) \to \{\sigma\}_{i=1}^{n}$ through a fully connected NN
2. Calibrate the model $\hat{\theta} = \arg\min_{\theta} \left(\mathcal{F}(\theta, \xi), \mathcal{F}^{MKT}(\xi)\right)$

where ξ denotes the input parameters of the function approximator, and will vary along different representations. In the case of Hagan's formula, ξ would represent the forward swap rate and the different option strikes.
The second step is the same as in traditional procedures, but now the mapping is not done by a function but by a neural network. This step has traditionally been a computational bottleneck, since all optimizers used for calibrating share that they require a high number of evaluations of the mapping function in order to arrive to the optimal value. There are also additional advantages towards the two-steps calibration method compared to other procedures. Since the neural network is only used as a computational improvement, the interpretability of the traditional models remains unaltered, while network parameters do not need interpretation, leaving freedom to the choice of architecture as long as it returns an accurate approximation. Also this method can be implemented when MonteCarlo simulations are needed to compute the pricing map, relocating the time-consuming simulation into an off-line preprocessing part and speeding up the online calibration, extending the aim of this calibration method to almost any numerical procedure. If a consistent pricer exists, It can be approximated and sped up by a neural network. The numerical calibration algorithm will therefore take the general steps shown in algorithm 4.1.
It is important to realize that the parameters obtained through this algorithm are influenced by two types of errors. The first type of error that will be analysed is the one made by the neural network when approximating Hagan's formula, and will be assessed by observing the difference between the neural network volatility output obtained from a random sampling of the parameters and the corresponding implied volatility obtained through Hagan's formula for those parameters. The second type of error corresponds to the one induced by the calibration algorithm, which can be observed when the model is calibrated to a test set smile.

Algorithm 4.1 SABR calibration with neural networks

```
Neural network training with simulated data
  1: **Input**: Combination of the strikes, forward rate, SABR parameters and time to maturity
  2: **Output**: Set of implied normal volatilities
  3: **Training step:** neural network supervised learning, as described before
Model calibration
  1: **Input**: market swaption premium in basic points
  2: Obtain implied normal volatility from these premiums
  3: Define a cost function $J(\theta) = \left[\sigma^{MKT} - \sigma^{NN}\right]\left[\sigma^{MKT} - \sigma^{NN}\right]'$
  4: Minimise $J(\theta)$ by iterating over the parameters $\alpha, \rho, \nu$
  5: Return the optimal parameters $\hat{\alpha}, \hat{\rho}, \hat{\nu}$
```

We have developed two similar alternatives within this approach with the objective of exploring what happens with different inputs.

4.2.1 Calibration through strikes

This alternative proposes to train the network with the strikes defined within a given range, therefore although they remain equal if expressed as ATM differences, if we express them as absolute strikes they are different from smile to smile, and as they are relevant to the model since these are the corresponding strikes for each implied volatilities, they can be used as inputs into the model. Consequently, the neural network will have as inputs:

- The three SABR parameters α, ρ, ν
- 11 absolute strikes obtained from a uniformly spaced sampling of 11 strikes expressed in ATM difference in the range $K_{bp} \in [-250, 400]$
- Time to maturity of the swaption, being T an integer so $T \in [5, 30]$

Therefore this neural network will have 15 inputs. Since there are 11 strikes we can get 11 implied volatilities corresponding to each strike, obtained as output predictions of the neural network.

4.2.2 Calibration through forward rate

In this alternative we have decided to extract from market data all the strikes in ATM differences quoted (in basic points) for all the market swaptions. Most of these strikes are shared across different maturities and tenors, broadening its range as the swaption maturity increases. For short maturities the difference from the lowest to highest reference strikes will be around a few basic points, while for large maturities it will be of a few hundred basic points. Nonetheless, the union of all these strikes will be a finite number, and we will train the neural network with the implied volatilities calculated through Hagan's formula for all these strikes. Since all the strikes (in ATM differences) are the same for every sample they do not deliver new information, and we can remove them from the input. The network will therefore take as inputs:

- The three SABR parameters α, ρ, ν
- The forward swap rate

- Time to maturity of the swaption, being T an integer so $T \in [5, 30]$

Meanwhile the outputs of the neuronal network will be the implied normal volatilities that correspond to each of these unique strikes.

4.2.3 Cost function minimization algorithm

There are multiple optimization algorithms available to solve a nonlinear optimization problem (minimizing the cost function defined at step 4 in algorithm 4.1) and its choice can completely determine the output. These algorithms can be divided into two categories, unconstrained, when there are no boundaries in the variables needed to optimize, and constrained optimization, when there are constrains in the variables. Looking at the nature of the cost function, we want to find the zero (root) of a multivariate function, therefore an iterative algorithm similar to Newton's method can be applied. The original method devised by Newton needs to compute the full Jacobian and Hessian matrices and has convergence problems, therefore more advanced algorithms like the quasi-Newton method is used, where the convergence is more easily obtained and the computational burden is reduced by computing an estimate of the Hessian matrix. Since our model has boundaries on the variables we should be looking at bound constrained optimization algorithms like the *L-BFGS-B*, described Zhu *et al.* (1997), which was used when minimizing the cost function for the standard SABR calibration with Hagan's formula. However the algorithm did not converge to a solution for some smiles when substituting Hagan's formula with the neural network, so we had to revert to the more general *L-BFGS* method, developed in Liu and Nocedal (1989), which lacks the ability to define simple boundaries for the variables but is able to converge to a solution.

Chapter 5

Numerical procedure and results

5.1 Data preprocessing

In this section we are going to discuss the characteristics of the market data to which we desire to calibrate the SABR model. We aim to calibrate the complete set of swaption smiles available from all swaptions listed in the market for a given currency (EUR). The available data is within the range of

- Tenors: {1Y, 2Y, 3Y, 4Y, 5Y, 6Y, 7Y, 8Y, 9Y, 10Y, 15Y, 20Y, 25Y, 30Y}
- Maturity: {1M, 3M, 6M, 1Y, 2Y, 3Y, 4Y, 5Y, 6Y, 7Y, 8Y, 9Y, 10Y, 12Y, 15Y, 20Y, 25Y, 30Y}

For a given forward swap rate there are available 11 different strikes quoted in ATM differences, each one with its corresponding premium defined in basic points (bp) from which we will calculate the implied normal volatilities. Consequently, there are available 252 different market smiles, that we will use to compare different calibration procedures.
The swaption maturities range from 1 month to 30 years and at first all neural networks were trained within this range. However we could not find any hyper-parameter combination for one network that returned results for the test data that reached the degree of accuracy desired. When looking at the output we observed that for longer maturities the results were in the expected order of magnitude, however for shorter maturities the results were not, as the neural network was not capturing correctly the shape of the smile. This behaviour could somewhat have been expected, since when calibrating the smiles through Hagan's formula it can be observed that as the time to maturity increases the slope and curvature of the smile decreases, which relates directly to the values of ρ (indicates the slope) and ν (curvature). Since the values for large maturities are in a smaller range than for short ones, the neural network predicts better in that range, and it is the one that we have chosen to train the neural network. Furthermore the objective of this thesis is to explore all the possibilities towards calibrating the SABR model through neural networks rather than carrying out a full implementation of a SABR calibration method, consequently we chose to train and predict within the range $T \in [5, 30]$.
Training data generation: The simulations needed for the supervised learning step are generated through random samplings of the inputs in Hagan's implicit formula for normal volatilities in 3.4. Through all the analysis conducted we have sampled several input parameters following a uniform distribution between a defined range of values. For the three SABR parameters (since as we have already discussed we have set $\beta = 0$) and the forward swap rate the chosen ranges have been

- $\alpha \in \mathcal{U}\,(0.001\%, 1.5\%)$
- $\rho \in \mathcal{U}\,(-98\%, 98\%)$
- $\nu \in \mathcal{U}\,(0.001\%, 70\%)$
- $f \in \mathcal{U}\,(-0.5\%, 3.5\%)$

These are the values that these variables have historically been limited to in the current economic environment. The time to maturity will change across different models, as well as the number of strikes and their range (in ATM difference). Through Hagan's approximation we will only obtain normal implied volatilities for the strikes defined as inputs. All the variables obtained above can either be inputs or outputs of the neural network, and will change depending on the model analysed.
Since in market data there are 11 strikes available for each smile, we have decided that each smile is generated by at least 11 implied volatilities obtained through Hagan's approximation. To create the training and test set a total of $1e6$ smiles have been simulated through this approximation, carrying out a typical train/test separation, with 80% of the simulations used to train the neural network and 20% used to test the accuracy of the training phase.

5.2 Cross validation and hyper-parameter selection

There are several hyper-parameters that need to be tuned in a neural network. Ideally a grid search over all the different options could be computed however this approach is computationally too expensive, therefore some parameters have to be set previously to cross validation. In these neural network implementations we have decided to set:

- **Activation function**: the activation function applied in the nodes from all the available options is the elu function. Although the other functions are computationally less expensive, this function is the one that should perform best
- **Weight initialization**: as motivated by section (dnde lo explique) the weights are initialized through the *he_normal* initializer defined in the keras library and explained in section 2.3.
- **Minibatch size**: the higher its value the more accurate the gradient is to the true gradient of all the training sample, however the model will take more time to converge to the local minimum. We have chosen a size of 128 samples.
- **Epochs**: the number of epochs chosen is ideally a number large enough so the cost function has reached a minimum but short enough to prevent overfitting. To avoid this dichotomy *early stopping* is implemented, so if during a predefined consecutive number of epochs the cost function has not reduced its value then the training is stopped.
- **Learning rate**: we will be using Adam's optimization algorithm which is an industry standard, and after some experimentation we decided to set the learning rate to $\alpha = 0.001$. Additionally an adaptive learning rate was implemented, so when the error of the cost function reaches a plateau (it does not improve during a number of epochs) the learning rate decreases.

- **Regularization schemes**: a number of regularization procedures can be adopted to prevent overfitting. However when approximating functionals we do not want the network to adapt to new input data, but to act as a function replicator, therefore there is no need for the neural network to generalize its results.

There are two additional hyper-parameters that are arguably the most important towards a correct neural network architecture, the **depth** and the number of neurons per hidden layer (**width**). To choose the best combination we decide to carry through a cross validated grid search (computed with the GridSearchCV class in the sklearn library, available in Python). This model selection algorithm will test for the best width and depth combination for the neural network, for a number of hidden layers in $[4, 5, 6, 7, 8]$ and a number of neurons per layer between $[32, 64, 128, 256]$ by training the network for each possible combination.*K-fold* Cross-validation is used in machine learning to use a limited sample to asses how the model will perform when predicting inputs that were not used during training, reducing the bias. The model is trained with $k-1$ folds, and the remaining fold is used to validate the model. This process is repeated over the folds, and the results averaged, returning the hyper-parameter combination that has given better results.

5.3 One-step calibration

As explained in section 4.1 this implementation seeks to obtain the SABR parameters for a given smile from the information available in the market.This information is the forward swap rate, the the maturity of the swaption, the strikes and the corresponding implied normal volatility for each strike. To train the neural network 10^6 samples have been generated following the process described in section 5.1 and divided into train and test set accordingly. The strikes chosen to train the network were the 11 most frequent strikes relative to the forward rate (in basic points) accross all smiles with maturity greater than 5 years. These are:

- Strikes: $\{-225, -200, -150, -100, -50, 0, 50, 100, 175, 250, 400\}$

As described in section 5.2 trough cross validation the best combination of hyper-parameters is 6 hidden layers with 64 neurons each layer, applying as activation function the elu functions. The election of the other hyper-parameters is described in the aforementioned section, with the number of epochs of the training epochs being 200 (mini-batch size is 128).
We have also decided to perform a transformation to rescale the α parameter, since we discovered at first that the network failed at predicting the level of the smile (α parameter). Since we have defined as limits for training $\rho \in [-1, 1]$ and $\nu \in [10^{-4}, 0.7]$ while $\alpha \in [10^{-4}, 0.015]$, we rescale the value of α in the range $\alpha \in [0, 1]$, eliminating the scaling of the outputs from the set of features that the neural network has to learn. Obviously this transformation has to be reverted when testing the accuracy of the learnt network.
The computation process that we follow to measure the goodness of fit of the neural network is described in algorithm 5.1.

First we are going to analyse the accuracy of the neural network, therefore we want only to measure the error with smiles generated with the same procedure as the one employed in the training set. We follow the procedure described in the test set error measurement section in algorithm 5.1. We are using the metric Root Mean Squared Error (RMSE) to measure the error of the prediction, hence we need to generate the same number of volatilities from the predicted parameters than the volatilities given as input to the neural network. From the data generated

Algorithm 5.1 One-step SABR calibration

```
Neural network training with simulated data ($f = K_6$)
1: Input: $K_1, K_2, ..., K_{11}, \sigma_1, \sigma_2, ..., \sigma_{11}, T$
2: Output: $\alpha, \rho, \nu$
Test set error measurement:
1: Given a set of inputs, predict with the neural network
2: Rescale the $\alpha$ parameter
3: Through Hagan's normal implicit formula obtain the smile for the predicted parameters
4: Calculate the RMSE for each strike comparing to the volatilities from each input
           $RMSE = \sqrt{N^{-1} \sum \left(\sigma^{MKT} - \sigma^{(\alpha,\rho,\nu)}\right)^2}$
Error measurement on market data
1: Calibrate each market smile and obtain $\alpha, \rho, \nu$
2: Obtain, through Hagan's approximation, the implied volatilities for each curve
3: Show the error with the market smile for each strike: {ATM - 5, ..., ATM, ATM + 5}
4: For each positional strike and smile, compute the RMSE of the calibrated volatility
```

for the test set we predict its parameters through the neural network and then proceed to obtain applying Hagan's formula their implied volatilities. Consequently, we have two smiles, the one generated to create the test data and the one generated from applying Hagan's formula to the neural network prediction, the maturity and the strikes that generated the prediction (and were part of the input data). To obtain a precise enough estimation we repeated this process for 1000 samples and computed the RMSE for each strike. The results are shown in table 5.1.

Strikes (bp)	-225	-200	-150	-100	-50	0	50	100	175	250	400
RMSE	2.281	1.924	1.564	1.202	0.878	0.678	0.860	1.183	1.532	1.894	2.251

Table 5.1: One-step calibration RMSE for 1000 samples in basic points (relative to ATM)

A sample of four predicted smiles is shown in 5.1, where we have compared the smile simulated to train the model following the process described in section 5.1 and the volatility smile predicted with the neural network. Observing table 5.1, we can conclude that even though there are slight differences between the simulated and the predicted smiles, this neural network is, for in-sample data, a good calibration method, with errors for the central strikes lower than one basic point and around two basic points for the extreme strikes.

We now procede to compute the out-of-sample analysis of the neural network, and try to calibrate the set of market swaption smiles described at the beginning of section 5.1. As it has been explained there are 11 different strikes for each smile, and the relative distance between them will vary with the maturity of the swaption and the swap's maturity, and will grow wider as these maturities increase. We will repeat the analysis carried out for the in-sample set to this market set, and repeat the metric applied to measure the calibration error. Now there are two types of errors, the one made by the neural network when predicting the parameters and the calibration error. If this second error is bigger than the prediction error it would mean that the network has either not obtained enough flexibility to adapt to new data or the new data is too different from the training data for the neural network to be able to predict correctly the parameters.

The summary of the analysis is shown in table 5.2. As it can be seen, the neural network is not able to correctly predict the parameters for the market smiles. Consequently, this neural

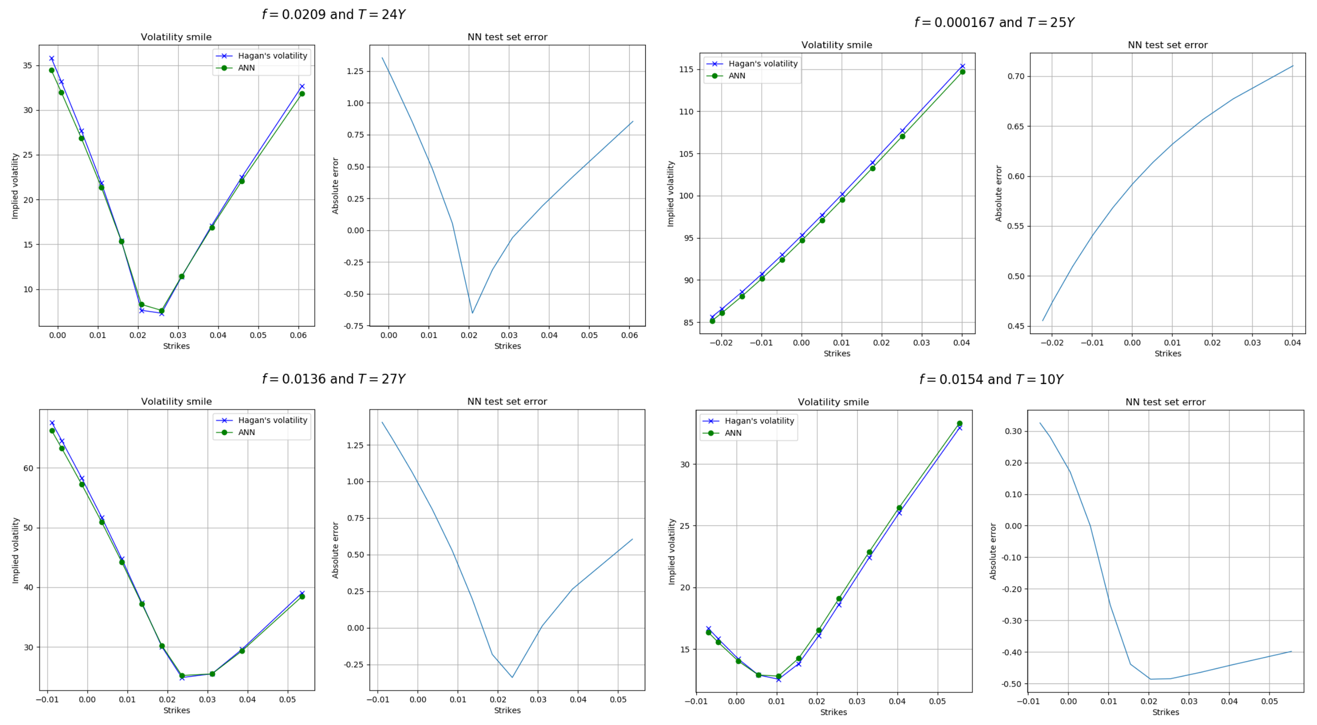

Figure 5.1: Simulated (in-sample) vs predicted smiles, and the error committed by the NN, in basic points

Strikes	-5	-4	-3	-2	-1	ATM	+1	+2	+3	+4	+5
RMSE market	31.81	31.29	30.60	29.96	29.45	29.19	28.98	29.13	29.58	30.88	33.16

Table 5.2: One-step calibration RMSE for the market swaption smiles in basic points. The Strikes row denotes the relative position to the ATM strike

network, with these hyper-parameters, although it is somewhat accurate for in-sample data, it is not able to predict correctly out-of-sample data, and cannot be used as a calibration method for the SABR model.

5.4 Two-steps calibration: calibration through strikes

In this section we take as inputs the absolute strikes of the smile instead of the forward rate, hopefully giving the neural network more information to work with to obtain a greater precision degree. Through experimentation and observing the results in simple implementations we have pre-defined some hyper-parameters. The optimization algorithm is of course Adam's algorithm, where we have decided to implement an adapative learning rate, starting by lr $= 0.001$ and reducing it by half if the MSE has not improved in 40 epochs. We have chosen not to introduce regularization techniques (L1, L2, dropout), to choose as mini-batch size 128 and to train the model for 300 epochs, while the activation function is the elu function, with linear outputs. Through 3-fold cross validation we have found that the best width and size combination for the fully connected feed forward neural network is 5 layers with 32 neurons in each layer.
The choice of the optimization algorithm is limited by the effectiveness of each method. Although the natural choice would have been to apply an algorithm that allowed us to introduce boundaries

to the three SABR parameters to optimize, such as L-BFGS-B or TNC, when calibrating market data we observed that these algorithms were not able to reach a global optima for many smiles (L-BFGS-B) or reached a local optima but not a global one (TNC). Therefore we had to consider unconstrained minimization methods, choosing the *BFGS* algorithm.
As previously described in section 4.3, the calibration algorithm used to obtain the optimal combination of parameters is:
 In the first place we are going to measure the neural network error, that is the error that the

Algorithm 5.2 Calibration through strikes

```
Neural network training with simulated data ($f = K_6$)
  1: **Input**: $K_1, K_2, ..., K_{11}, \alpha, \rho, \nu, T$
  2: **Output**: $\sigma_1, \sigma_2, ..., \sigma_{11}$
Model calibration:
  1: **Input**: market swaption premium in basic points, $K_1, K_2, ..., K_{11}, T$
  2: Obtain implied normal volatility from these premiums
  3: Define the cost function $J(\theta) = \sum_{i=1}^{11} \left(\sigma^{MKT} - \sigma^{NN}\right)^2$
  4: Iterate over $\alpha, \rho, \nu$ until the BFGS algorithm arrives to an optimal value
  5: Return the optimal parameters
Error measurement on market data
  1: Calibrate each market smile and obtain $\alpha, \rho, \nu$
  2: Obtain, through Hagan's approximation, the implied volatilities for each curve
  3: Measure the error between the approximation and the market smile
  4: Represent the error for each strike as positional: {ATM - 5, ..., ATM, ATM + 5}
  5: For each positional strike, compute the RMSE of the calibrated volatility
         $RMSE = \sqrt{N^{-1} \sum (\sigma^{MKT} - \sigma^{NN})^2}$
  6: Compare the error metric to Hagan and Hagan ATM
```

network commits when predicting the implied volatilities. Again we resort to the RMSE metric and follow a procedure similar to section 5.3, where we the test set to predict the volatilities and compared them to the ones obtained through Hagan's formula. The results are shown in table 5.3, quoted in basic points: Although these results are several times better than those obtained in

Strikes (bp)	-225	-200	-150	-100	-50	0	50	100	175	250	400
RMSE test	0.76	0.63	0.65	0.62	0.67	0.53	0.59	0.69	0.68	0.72	0.73

Table 5.3: Two-step calibration (with strikes) RMSE for the test set in basic points. The Strikes row denotes the relative position to the ATM strike

the one-step calibration method, we have to note that these strikes are fixed, as opposed to what happens for market data, which will probably result in a more accurate prediction that if their relative position varied. The results are uniform across strikes, which means that the prediction error is also uniform across them and there are no biases in the prediction error towards larger strikes, as happened in the one-step method.
Now we move on to measure the calibration error of the neural network for out-of-sample data. We are going to compare it to the market standard, calibrating the whole market swaption smiles using as functional Hagan's formula, both directly and through the ATM implementation described in 3.7. The metric used to measure the goodness of fit is again the root mean squared error (RMSE). There are eleven different strikes for each smile, therefore we are going to measure

the accuracy of the model taking into account the strike's relative positions, as described in 5.2. Even though we are approximating the mapping function through the neural network, we still want to preserve the spirit of the underlying model. This way we maintain its interpretability instead of converting the calibration into a black box algorithm. After calibrating the model through the neural network we want to apply Hagan's approximation to the calibrated parameters to obtain the smile instead of predicting the implied volatilities through the neural network with the obtained parameters. The calibration step is highly susceptible to the initial values of the parameters, just as it happens for the other calibration method. For this neural network the seeds that return the best fit for the market data are $[\alpha, \rho, \nu] = [0.001, 0.01, 0.4]$, and the RMSE for each relative strike is shown in table 5.4

Strikes	-5	-4	-3	-2	-1	ATM	+1	+2	+3	+4	+5
NN	1.208	0.917	0.819	0.793	0.920	0.667	0.618	0.822	1.007	1.655	2.520
Hagan	2.770	2.069	1.494	0.907	0.376	0.799	1.391	1.534	1.528	1.314	1.024
Hagan ATM	0.617	0.338	0.272	0.232	0.237	0.000	0.385	0.243	0.206	0.205	0.371

Table 5.4: Two-step calibration RMSE for the market swaption smiles in basic points. The Strikes row denotes the relative position to the ATM strike

We can observe that the neural network has the same order of accuracy than the calibration computed through Hagan's formula, although it lacks the accuracy of the calibration through the ATM volatility, which is more than twice as accurate as the neural network. If we compare the error of the network when calibrating to market data and when predicting with test data we can observe the part of the total error that belongs to the calibration step. The summary is presented in table 5.5.

Strikes	-5	-4	-3	-2	-1	ATM	+1	+2	+3	+4	+5
Prediction error	0.762	0.638	0.653	0.622	0.678	0.531	0.590	0.691	0.684	0.729	0.734
Market error	1.208	0.917	0.819	0.793	0.920	0.667	0.618	0.822	1.007	1.655	2.520
Calibration error	0.446	0.279	0.166	0.171	0.242	0.136	0.028	0.131	0.323	0.926	1.786

Table 5.5: Two-step calibration RMSE for the market swaption smiles in basic points, showing the two types of errors committed during the calibration

The results show that more than half of the total error is committed by the neural network when predicting the implied volatilities, and the rest is due to the calibration algorithm and to the market smiles, since they do not follow exactly Hagan's formula. This feature can be observed in the calibration error, since for most strikes the error committed while fitting to market data is less than 0.4 basic points, and it decreases even for the central strikes, while for the farther strikes (specially in the upper part of the domain) the error increases drastically. Hagan's formula is an approximation, and as such it stops being accurate when the terminal conditions are reached, leading the formula to not capture correctly the curvature for strikes in the wings of the smile, being probably the main source of error when calibrating to the market smiles.
We could change substep 2 in the error measurement on market data step in algorithm 5.2 if once we calibrated the model we obtained the implied volatilities through the neural network instead of Hagan's formula. This alternative would transform the calibration method into a black box where the underlying SABR model would only be present in the training step, since its parameters would no longer be relevant towards the calibration algorithm. Despite improving slightly the capacity of the model to approximate correctly the volatility smile, the algorithm

would lack interpretability, and we could drop the SABR model altogether. Since we do not want to drop the information that the SABR parameters return about the curve this approach will not be followed, although its results are reproduced in table 5.6.

Strikes	-5	-4	-3	-2	-1	ATM	+1	+2	+3	+4	+5
NN & SABR	1.208	0.917	0.819	0.793	0.920	0.667	0.618	0.822	1.007	1.655	2.520
NN alone	0.865	0.698	0.541	0.358	0.354	0.361	0.606	0.689	0.537	0.783	0.784
Hagan ATM	0.617	0.338	0.272	0.232	0.237	0.000	0.385	0.243	0.206	0.205	0.371

Table 5.6: Comparison between calibrating the smile and applying Hagan's formula vs predicting the implied volatilities through the NN, in basic points

Next we compare some of the calibrated swaption smiles. We show the market swaption, the calibration through neural network and then calibration through Hagan's ATM formula. The plot 5.2 aim to show the calibration obtained for different swaption maturities and durations of the underlying IRS, and the error committed by each method in basic points.

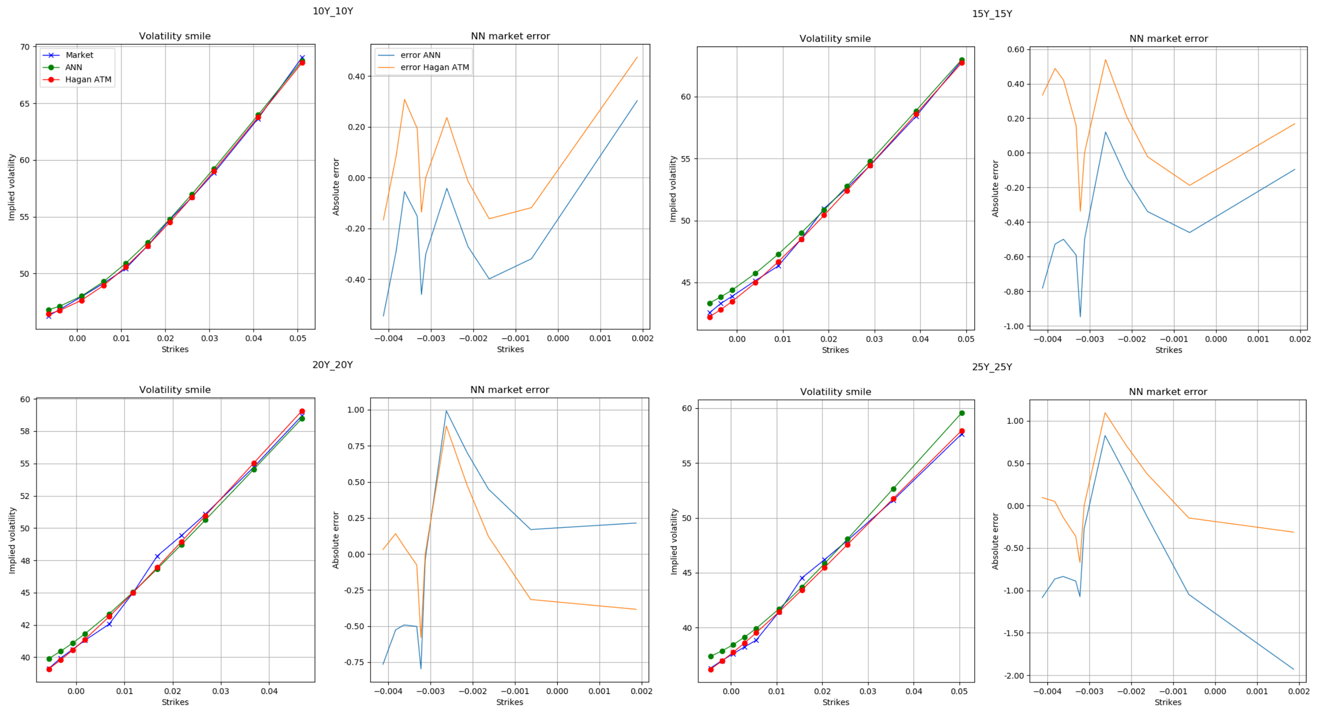

Figure 5.2: Market smiles vs calibration through NN vs calibration through Hagan's ATM formula. Plots and differences between market and calibration data are in basic points

5.4.1 Calibration through forward rate

This variant proposes to follow the same procedure as before but with the neuronal network taking different inputs and outputs. The choice for the inputs will be the three SABR parameters, the swaption maturity and the forward rate.

Analysing the market swaption smiles for maturities greater or equal to five years we observe that most strikes (relative to the forward rate) are repeated, so we can choose as output to the neural network the implied volatility that corresponds to each strike. In particular, for this specific set there are 19 different strikes, and the neural network output will be 19 implied volatilities. By

calculating every possible strike's volatility we can eliminate the information given by the strikes from the inputs, and only keep the relevant information defined above.
When substituting Hagan's formula for the neural network we will obtain volatilities for more strikes than the ones we have to calibrate to, however this is not a problem since we can just choose the volatilities that correspond to the ones in the smile to calibrate and throw away the rest. Algorithm 5.2 will therefore change only during training, as shown in 5.3

Algorithm 5.3 Calibration through forward rate

```
Neural network training with simulated data ($f = K_6$)
1: **Input**: $f, \alpha, \rho, \nu, T$
2: **Output**: $\sigma_1, \sigma_2, ..., \sigma_{19}$
                              ⋮
```

The results when calibrating to market data are shown in table 5.7.

Strikes	-5	-4	-3	-2	-1	ATM	+1	+2	+3	+4	+5
NN	3.574	2.738	2.050	1.245	0.464	0.931	1.754	1.957	1.007	1.655	1.352
Hagan	2.770	2.069	1.494	0.907	0.376	0.799	1.391	1.534	1.969	1.730	1.024
Hagan ATM	0.617	0.338	0.272	0.232	0.237	0.000	0.385	0.243	0.206	0.205	0.371

Table 5.7: Two-step calibration RMSE (through the forward rate) for the market swaption smiles in basic points. The Strikes row denotes the relative position to the ATM strike

The out-of-sample calibration with market data shows that although this network still calibrates properly, it is far less accurate than the network presented before, and consequently will not be analysed further.

All the software developed throughout this thesis can be found in this link.

Conclusion

The main objective of this thesis is to answer the question: Can a feedforward neural network accurately calibrate a complex financial model? To analyse this proposition we have proposed two different calibration methods based on neural network for the SABR model. The neural networks have been trained with data generated through Hagan's formula (an approximate mapping from parameters to implied volatility).
The first calibration method proposed (one-step calibration) maps the swaption maturity, strikes and implied volatilities to the SABR parameters. The results obtained show that the neural network obtains fairly accurate results for in-sample data (around 1 or 2 bp) but when calibrating to market swaption smiles the parameters predicted by the network are totally inaccurate. Therefore this method is not a good calibrator for out-of-sample data
The second calibration method follows two steps. In the first step the neural network is trained to map the SABR parameters, swaption maturity and strkes to the implied volatility obtained through Hagan's formula. With this approach we have created a replica of Hagan's formula which will be substituted inside the traditional calibration algorithm. We have tested this approach with in-sample data, and obtained that the neural network has a high degree of accuracy, since the difference between the trained and predicted volatilities is less that 0.7 basic points. Next we calibrated a complete set of swaption smiles through this approach, and obtained that the total error committed by the calibration method is less than 1 bp except for the strikes in the wings of the smile.
After calibration we would like to obtain the implied volatilities with Hagan's formula to keep the model interpretable, however if we use the neural network to predict the volatilities the bias in the wings disappears, leading to a calibration error of less than 1 bp. Either of the two solutions yields a better calibration accuracy than fitting the Hagan's formula directly, but worse than fitting Hagan's formula with the ATM volatilities.
Although the calibration process through neural networks is not as accurate as Hagan's ATM calibration it is a close candidate, since its error is below 1 basic point. However our objective is not to obtain the most accurate calibration method, but to show that it is possible to approximate a complex model like the SABR through a neural network while maintaining a high degree of accuracy.
This result can be extrapolated to other complex financial models that are not used due to its high computational cost (as in Bayer and Stemper (2018)) or to value derivatives, as in Ferguson and Green (2018). Also it could be of value trying to map a model that is solved through MonteCarlo simulations, as the neural network can reduce the noisiness from random draws.
Additionally we could opt to improve the current development of the neural network calibration. We have decided to map only swaptions with maturities greater that 5Y since for lower maturities the parameters are of different magnitudes (ν grows up) and the network did not capture it properly. If we wanted to implement this calibration method, it would be necessary to train the network by maturity blocks. The SABR can be also solved through a finite differences scheme or Montecarlo simulations to obtain a more accurate representation of the original model, and if we fed this data into the neural network the calibration would probably be more accurate.

Bibliography

Antonov, A., Konikov, M., and Spector, M. (2013). Sabr spreads its wings. *Risk*, **26**(8), 58. 18

Bachelier, L. (1900). Théorie de la spéculation. In *Annales scientifiques de l'École normale supérieure*, volume 17, pages 21–86. 6

Bayer, C. and Stemper, B. (2018). Deep calibration of rough stochastic volatility models. *arXiv preprint arXiv:1810.03399*. 32

Black, F. (1976). The pricing of commodity contracts. *Journal of financial economics*, **3**(1-2), 167–179. 7

Black, F. and Scholes, M. (1973). The pricing of options and corporate liabilities. *Journal of political economy*, **81**(3), 637–654. 4

Brigo, D. and Mercurio, F. (2007). *Interest rate models-theory and practice: with smile, inflation and credit*. Springer Science & Business Media. 2, 4

Chen, B., Oosterlee, C. W., and Van Der Weide, H. (2011). Efficient unbiased simulation scheme for the sabr stochastic volatility model. *Preprint, http://ta. twi. tudelft. nl/mf/users/oosterle/oosterlee/SABRMC. pdf*. 18

Clevert, D.-A., Unterthiner, T., and Hochreiter, S. (2015). Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*. 11

Eldan, R. and Shamir, O. (2015). The power of depth for feedforward neural networks. *CoRR*, **abs/1512.03965**. 9

Ferguson, R. and Green, A. (2018). Deeply learning derivatives. 32

Geman, H., El Karoui, N., and Rochet, J.-C. (1995). Changes of numeraire, changes of probability measure and option pricing. *Journal of Applied probability*, **32**(2), 443–458. 5

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. 10

Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. 11

Hagan, P. S., Kumar, D., Lesniewski, A. S., and Woodward, D. E. (2002). Managing smile risk. *The Best of Wilmott*, **1**, 249–296. 15, 16

Hanin, B. (2017). Universal function approximation by deep neural nets with bounded width and relu activations. *arXiv preprint arXiv:1708.02691*. 14

Harrison, J. M. and Pliska, S. R. (1981). Martingales and stochastic integrals in the theory of continuous trading. *Stochastic processes and their applications*, **11**(3), 215–260. 4

Harrison, J. M. and Pliska, S. R. (1983). A stochastic calculus model of continuous trading: complete markets. *Stochastic processes and their applications*, **15**(3), 313–316. 4

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034. 12

Hecht-Nielsen, R. (1992). Theory of the backpropagation neural network. In *Neural networks for perception*, pages 65–93. Elsevier. 13

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, **2**(5), 359–366. 14

Horvath, B. and Reichmann, O. (2018). Dirichlet forms and finite element methods for the sabr model. *SIAM Journal on Financial Mathematics*, **9**(2), 716–754. 18

Hull, J. C. (2003). *Options futures and other derivatives*. Pearson Education India. 5

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*. 13

Leitao, Á., Grzelak, L. A., and Oosterlee, C. W. (2017). On an efficient multiple time step monte carlo simulation of the sabr model. *Quantitative Finance*, **17**(10), 1549–1565. 18

Liu, D. C. and Nocedal, J. (1989). On the limited memory bfgs method for large scale optimization. *Mathematical programming*, **45**(1-3), 503–528. 22

Lu, Z., Pu, H., Wang, F., Hu, Z., and Wang, L. (2017). The expressive power of neural networks: A view from the width. In *Advances in Neural Information Processing Systems*, pages 6231–6239. 14

Miron, P. and Swannell, P. (1991). *Pricing and hedging swaps*. Euromoney Publ. 2

Ohn, I. and Kim, Y. (2019). Smooth function approximation by deep neural networks with general activation functions. *arXiv preprint arXiv:1906.06903*. 14

Shreve, S. E. (2004). *Stochastic calculus for finance II: Continuous-time models*, volume 11. Springer Science & Business Media. 3, 4

Tan, C. C. (2012). *Market Practice in Financial Modelling*. World Scientific Publishing Company. 16

West, G. (2005). Calibration of the sabr model in illiquid markets. *Applied Mathematical Finance*, **12**(4), 371–385. 17

Xu, B., Wang, N., Chen, T., and Li, M. (2015). Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*. 11

Zhu, C., Byrd, R. H., Lu, P., and Nocedal, J. (1997). Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Trans. Math. Softw.*, **23**(4), 550–560. 22