

**Imperial College  
London**

IMPERIAL COLLEGE LONDON

DEPARTMENT OF MATHEMATICS

---

**Artificial Neural Networks  
for SABR model  
calibration & hedging**

---

*Author:* Hugues Thorin (CID: 01805999)

A thesis submitted for the degree of  
*MSc in Mathematics and Finance, 2019-2020*

## **Abstract**

SABR model is a reference in financial industry to price fixed income derivatives, thanks to its ability to capture the volatility smile. The goal of this thesis is to develop a technique based on SABR model which captures all non-linearities, be more robust, gives tradable parameters instantaneously and gives also instantaneously robust hedging parameters.

**Keywords:** SABR, Neural Network, Volatility Smile, Calibration, Universal Approximation Theorem, Hagan, Hernandez

*To my mother, Doriane.*  
*To my fiancée, Gaëtane.*

## Acknowledgements

I would like to thank Deutsche Bank for having offered me the opportunity to write my MSc Thesis with them. I am particularly grateful to Fixed Income Emerging Market desk which welcomed me. I thank my colleagues: Tarik Hsaini, Anas Elkaddouri and Damien Hughes.

My warmest thanks are for Romano Trabalzini, my supervisor who has been very pedagogical and patient towards me. It has always been a pleasure to exchange views about mathematics, machine learning, finance,  $\dots$  and Italy with Romano. Whatever the market turmoils, Romano is Latin, always smiling and in a good mood. Thank you Romano. It has been a pleasure to write my MSc Thesis with you !

I thank Dr. Eyal Neuman, my thesis supervisor at Imperial College, and excellent teacher in Stochastic Processes.

Finally, I would like to thanks my mother and my fiancée for their support and encouragement.

I won't be there now without you.

# Contents

<b>Introduction</b>	<b>1</b>
<b>1 Basic concepts</b>	<b>3</b>
1.1 Bank account	3
1.2 Zero-coupon bond	3
1.3 Spot Interest Rates	4
1.4 Forward rates	4
1.5 Interest-Rate Swaps	5
1.6 Swaption	6
1.7 No-arbitrage Pricing	6
1.8 Numeraire change	7
1.8.1 T-forward measure	7
1.8.2 Swap measure	7
<b>2 Interest Rates &amp; SABR models</b>	<b>9</b>
2.1 The Bachelier (Normal) Model	9
2.2 The Black (Log-Normal) Model	10
2.3 Discussions	10
2.4 SABR dynamic	11
2.5 SABR asymptotic solution	11
2.6 SABR for negative interest rates	12
2.6.1 Shifted SABR	12
2.6.2 Absolute SABR	12
2.7 SABR calibration	12
2.8 SABR parameters	13
2.8.1 $\alpha$ parameter	13
2.8.2 $\beta$ parameter	14
2.8.3 $\rho$ parameter	14
2.8.4 $\nu$ parameter	15
2.9 SABR complements	15
2.9.1 The Backbone	15
2.9.2 The Skew	16
2.9.3 The convexity	16
<b>3 Artificial Neural Networks</b>	<b>17</b>
3.1 Introduction and history of Deep Learning	17
3.2 Architecture of Neural Networks	19
3.2.1 Brief Description	19
3.2.2 Detail Construction	19

3.2.3	Activation Functions	21
3.3	Training of Neural Networks	21
3.3.1	Loss Function	21
3.3.2	Minibatch and batch size	23
3.3.3	Epochs	23
3.3.4	Iterations	24
3.3.5	Summary example	24
3.3.6	Stochastic Gradient Descent: SGD	24
3.3.7	Backpropagation	25
3.4	Universal Approximation Theorem	27
3.4.1	Arbitrary Width	27
3.4.2	Arbitrary Depth	27
<b>4</b>	<b>Methodology</b>	<b>28</b>
4.1	Step 1: Data Presentation	29
4.2	Step 2: Sequential calibration on time-series	29
4.3	Step 3: Preparing Training Dataset	32
4.3.1	Step 3.1	33
4.3.2	Step 3.2	33
4.3.3	Step 3.3	34
4.3.4	Step 3.4	34
4.3.5	Step 3.5	35
4.3.6	Step 3.6	35
4.3.7	Step 3.7	35
4.3.8	Step 3.8	35
4.4	Step 4: Neural Networks Calibration	36
4.4.1	Architecture choice	37
4.4.2	Activation Function	38
4.4.3	Loss function	40
4.4.4	Optimizer choice	40
4.4.5	Epoch and Batch	40
4.4.6	Kernel Initializer	40
4.4.7	Early Stopping	40
4.4.8	Dynamic Learning Rate	41
4.5	Step 5: Out-of-Sample test	41
<b>5</b>	<b>Results</b>	<b>42</b>
5.1	Step 1: Data Presentation	42
5.2	Step 2: Sequential calibration on time-series	44
5.3	Step 3: Preparing Training Dataset	47
5.4	Step 4: Neural Networks calibration	48
5.5	Step 5: Out-Sample test	50
5.6	Discussions	52
5.6.1	Time to deliver Output	52
5.6.2	Time to calibrate the model	52
5.6.3	Robustness	52
5.6.4	Improvement possibilities	52

<b>6 Hedging</b>	<b>54</b>
6.1 Hedging under SABR model . . . . .	54
6.1.1 Hagan's formula . . . . .	55
6.1.2 Barlett's formula . . . . .	55
6.2 Hedging with Neural Networks . . . . .	56
<b>Conclusion</b>	<b>58</b>
<b>Appendix</b>	<b>59</b>
<b>Bibliography</b>	<b>65</b>

# List of Figures

2.1	alpha parameter influence . . . . .	13
2.2	beta parameter influence . . . . .	14
2.3	positive rho parameter influence . . . . .	14
2.4	negative rho parameter influence . . . . .	14
2.5	nu parameter influence . . . . .	15
2.6	Backbone effect with $\beta = 0$ . . . . .	16
2.7	Backbone effect with $\beta = 1$ . . . . .	16
3.1	History of AI . . . . .	17
3.2	ImageNet LSVRC . . . . .	18
3.3	Neural Network illustration . . . . .	19
3.4	Architecture of Neural Network . . . . .	20
3.5	Activation Function Illustration . . . . .	21
3.6	Loss Function List . . . . .	22
4.1	Grid vs Random search . . . . .	37
4.2	Relu graph . . . . .	38
4.3	Elu Graphs . . . . .	39
4.4	Learning Rate Illustration . . . . .	41
5.1	ZAR yield curve on 2 <sup>nd</sup> of September 2020 . . . . .	42
5.2	Time-series of SABR calibrated parameters . . . . .	45
5.3	Time-series of error of SABR calibrated parameters . . . . .	46
5.4	Step 4 Neural Network . . . . .	48
5.5	Time-series of SABR calibrated parameters with Neural Networks . . . . .	50
5.6	Time-series of error of SABR calibrated parameters with Neural Networks . . . . .	51
1	The mostly complete chart of Neural Networks . . . . .	59
2	One-dimensional Activation Function List . . . . .	60
3	Multi-dimensional Activation Function List . . . . .	60



# List of Tables

4.1	Step 2 parameters . . . . .	31
4.2	Step 2 summary . . . . .	31
4.3	Step 3 summary . . . . .	33
4.4	Step 4 summary . . . . .	36
5.1	South African Forward rate in percentage on 2 <sup>nd</sup> of September 2020 . . . . .	42
5.2	Black's implied volatility in percentage of South African at-the-money swaptions on 2 <sup>nd</sup> of September 2020 . . . . .	43
5.3	Black's implied volatility in percentage of South African at-the-money swaptions on 5 <sup>th</sup> of March 2020 . . . . .	44
5.4	Hagan's implied volatility in percentage of South African at-the-money swaptions on 5 <sup>th</sup> of March 2020 . . . . .	44
5.5	Step 4 parameters . . . . .	49
5.6	SABR Parameter values for both methods on an in-sample date: 5 <sup>th</sup> of March 2020 . . . . .	49
5.7	Comparison of Hagan/Hagan NN implied volatility in percentage of South African at-the-money swaptions on a in-sample date: 5 <sup>th</sup> of March 2020 . . . . .	49
5.8	SABR Parameter values for both methods on an out-of-sample date: 2 <sup>nd</sup> of September 2020 . . . . .	50
5.9	Comparison of Hagan/Hagan NN implied volatility in percentage of South African at-the-money swaptions on a out-of-sample date: 2 <sup>nd</sup> of September 2020 . . . . .	50

# Introduction

Investment Banks profit is mainly realised thanks to fees they charge to clients. However, banks sit on a treasure: 20 years of financial data, of time series of human-powered trading strategies, human operators that have traded and hedged securities for a long time, which is stored but not always exploited. JPM's quant Hans Buehler <sup>1</sup> recently explained that it is not any more the time of parametric models, it is the era of data-driven semi/non-parametric models.

Against this background, the purpose of this thesis is to provide a calibration method exploiting these large dataset. Thanks to machine learning technology, we are also attempting to create a recipe that could give results instantaneously. To achieve this goal, we will see how the bulk of the calibration needs to be done offline and Artificial Neural Networks (ANN) offer just the right technique.

In the choice of a pricing model, it is important for it to calibrate precisely and quickly. The main interest of using Neural Networks is the possibility to do the calibration (which in any model is often the time-consuming process) completely offline. This internship has been conducted in the Fixed Income team, and from the beginning the idea of the project has been formalized as: to use ANN to project swaption matrices into space of tradable parameters (SABR), directly in-line at limited computational cost and quickly.

Among the possible currencies that Emerging Market desk would trade, we settled for South African Rand (ZAR) and the project idea was "Artificial Neural Networks for SABR model calibration & hedging" in South African Fixed Income market, which is one of the most liquid markets operated by the team where I did the internship.

This project is organised as follows. Chapter 1 will introduce some standard Fixed Income concepts and some relevant financial products. We will review fundamental mathematical concepts that will allow us to model the financial products in a rigorous manner. Chapter 2 will describe some very common Interest Rate Models, see their limits to capture the volatility smile and will focus on the SABR model to try to solve this issue. Chapter 3 will offer a broad review of Artificial Neural Network, a specific machine learning technique that we will use in the main part of this work. Chapter 4 is the core chapter of this thesis. Here, we will first obtain a daily calibrated time-series of SABR parameters stretching two years. Starting from this daily time-series, we will create additional randomised samples in order

---

<sup>1</sup>[www.risk.net/derivatives/6705012/podcast-hans-buehler-on-deep-hedging-and-the-advantages-of-data-driven-approaches](http://www.risk.net/derivatives/6705012/podcast-hans-buehler-on-deep-hedging-and-the-advantages-of-data-driven-approaches)

to have enough data to train our Neural Network. After that we will perform the calibration with it. Chapter 5 will analyse the results of this exercise. Chapter 6 will show possibilities of hedging thanks to the model we have created.

# Chapter 1

## Basic concepts

In this chapter, we will introduce basic notions in interest rates theory that will be useful in the following sections. This chapter relies on notation in Brigo and Mercurio (2006) [7].

### 1.1 Bank account

**Definition 1.1.1** (Bank account). *We define  $B(t)$  to be the value of a bank account at time  $t \geq 0$ . We assume  $B(0) = 1$  and that the bank account evolves according to the following differential equation:*

$$dB(t) = r_t B(t) dt, \quad B(0) = 1 \quad (1.1)$$

where  $r_t$  is the instantaneous rate at which the bank account accrues. This instantaneous rate is usually referred to as instantaneous spot rate, or briefly as short rate. We get:

$$B(t) = \exp\left(\int_0^t r_s ds\right) \quad (1.2)$$

**Definition 1.1.2** (Discount factor). *The (stochastic) discount factor  $D(t, T)$  between two time instants  $t$  and  $T$  is the amount at time  $t$  that is “equivalent” to one unit of currency payable at time  $T$ , and is given by*

$$D(t, T) = \frac{B(t)}{B(T)} = \exp\left(-\int_t^T r_s ds\right) \quad (1.3)$$

### 1.2 Zero-coupon bond

**Definition 1.2.1** (Zero-coupon bond). *A  $T$ -maturity zero-coupon bond (pure discount bond) is a contract that guarantees its holder the payment of one unit of currency at time  $T$ , with no intermediate payments. The contract value at time  $t$ ,  $t < T$  is denoted by  $P(t, T)$ .*

$$P(T, T) = 1$$
$$P(t, T) = \mathbb{E}_t^{\mathbb{Q}}[D(t, T) \times 1] = \mathbb{E}_t^{\mathbb{Q}}\left[\exp\left(-\int_t^T r_s ds\right)\right] \quad (1.4)$$

where  $\mathbb{Q}$  is the risk neutral measure.

**Definition 1.2.2** (Time to maturity). *The time to maturity  $T - t$  is the amount of time (in years) from the present time  $t$  to the maturity time  $T > t$*

### 1.3 Spot Interest Rates

**Definition 1.3.1** (Simply-compounded spot interest rate). *The simply-compounded spot interest rate prevailing at time  $t$  for the maturity  $T$  is denoted by  $L(t, T)$  and is the constant rate at which an investment has to be made to produce an amount of one unit of currency at maturity, starting from  $P(t, T)$  units of currency at time  $t$ , when accruing occurs proportionally to the investment time.*

$$\begin{aligned} P(t, T) (1 + \tau(t, T)L(t, T)) &= 1 \\ L(t, T) &:= \frac{1 - P(t, T)}{\tau(t, T)P(t, T)} \end{aligned} \quad (1.5)$$

**Definition 1.3.2** (Continuously-compounded interest rate). *The continuously-compounded spot interest rate prevailing at time  $t$  for the maturity  $T$  is denoted by  $R(t, T)$  and is the constant rate at which an investment of  $P(t, T)$  units of currency at time  $t$  accrues continuously to yield a unit amount of currency at maturity  $T$ .*

$$\begin{aligned} P(t, T) \exp(R(t, T) \times (T - t)) &= 1 \\ R(t, T) &:= -\frac{\ln(P(t, T))}{\tau(t, T)} \end{aligned} \quad (1.6)$$

**Definition 1.3.3** (Annually-compounded spot interest rate). *The annually-compounded spot interest rate prevailing at time  $t$  for the maturity  $T$  is denoted by  $Y(t, T)$  and is the constant rate at which an investment has to be made to produce an amount of one unit of currency at maturity, starting from  $P(t, T)$  units of currency at time  $t$ , when reinvesting the obtained amounts once a year.*

$$\begin{aligned} P(t, T) (1 + Y(t, T))^{\tau(t, T)} &= 1 \\ Y(t, T) &:= \frac{1}{P(t, T)^{1/\tau(t, T)}} - 1 \end{aligned} \quad (1.7)$$

### 1.4 Forward rates

Forward rates are contract characterised by three instants:

- the contract valuation: time  $t$
- the contract expiry: time  $T$
- the contract maturity: time  $S$

We get the relationship among different instants:  $t \leq T \leq S$ .

Forward rates enables an investor to lock now an interest rate for a future investment. The investor can lock at time  $t$  an interest rate  $K$  for period  $[T, S]$ . The locked interest rate  $K$  is determined in relation to the current interest rate structure.

The value of the contract is dependant of previous factors as well as the notional of the contract  $N$ .

$$FRA(t, T, S, \tau(t, T), K, N) = N [P(t, T)\tau(t, T)K - P(t, T) + P(t, S)] \quad (1.8)$$

The contract is fair if it has an expected value of zero. Only one value of  $K$  makes the value of the contract fair at time  $t$ .

**Definition 1.4.1** (Simply-compounded forward interest rate). *The simply-compounded forward interest rate prevailing at time  $t$  for the expiry  $T > t$  and maturity  $S > T$  is denoted by  $F(t; T, S)$*

$$F(t; T, S) = \frac{1}{\tau(T, S)} \left( \frac{P(t, T)}{P(t, S)} - 1 \right) \quad (1.9)$$

We can define the instantaneous forward interest rate, even if not a tradable contract, it will be useful for mathematics modelisation. We take  $S \rightarrow T^+$ .

**Definition 1.4.2** (Instantaneous forward interest rate). *The instantaneous forward interest rate prevailing at time  $t$  for the maturity  $T > t$  is denoted by  $f(t, T)$*

$$f(t, T) = \lim_{S \rightarrow T^+} F(t, T, S) = -\frac{\partial \log P(t, T)}{\partial T} \quad (1.10)$$

To determine instantaneous forward interest rates from the market, it must be extrapolate from interest-rates curve.

In the European markets, the fundamental interbank interest rates are:

- LIBOR (London InterBank Offered Rate) is an interest rate which is the average of London biggest banks interbank rate, which means the rate at which these banks would lend to each others. These rates are on seven lending periods.
- EURIBOR (Euro InterBank Offered Rate) is a daily interest rate which is the average of Eurozone interbank rates.
- OIS (Overnight Indexed Swap) is an interest rate swap over some fixed term where the periodic floating payment is generally based on a return calculated from a daily compound interest investment

## 1.5 Interest-Rate Swaps

A generalisation of FRA is Interest Rates Swap (IRS). Interest-Rate Swap is a contract that exchanges payments between two differently indexed legs, starting from a future time instant. At every instant  $T_i$  in a pre-specified set of dates  $T_{\alpha+1}, \dots, T_\beta$  the fixed leg pays out the amount. The cash flows come from a fixed rate  $K$  or a floating rate, depending on the side: 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  $t < T_\alpha$  for a payer IRS is

$$IRS_{payer} = \sum_{i=\alpha+1}^{\beta} ND(T_{i-1}, T_i)\tau_i (F(t, T_i) - K) \quad (1.11)$$

## 1.6 Swaption

Swaptions are options on an IRS. 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  $T_\alpha$ . The discounted payoff of a payer swaption at time  $t < T_\alpha$  is:

$$SWAPTION_{payer} = ND(t, T_\alpha) \left( \sum_{i=\alpha+1}^{\beta} P(T_\alpha, T_i) \tau_i [F(T_\alpha, T_{i-1}, T_i) - K] \right)^+ \quad (1.12)$$

An importance consideration is to determine the strike of the the swaption, where it is at the money, it can be determine:

$$\begin{aligned} K_{ATM} = S_{T_\alpha, T_\beta}(t) &= \frac{\sum_{i=\alpha}^{\beta} P(t, T_i) \tau_i F(t, T_{i-1}, T_i)}{\sum_{i=\alpha}^{\beta} P(t, T_i) \tau_i} \\ &= \frac{P(t, T_\alpha) - P(t, T_\beta)}{C_{T_\alpha, T_\beta}(t)} \end{aligned} \quad (1.13)$$

Hence we can re-write the payoff of  $SWAPTION_{payer}$  as

$$SWAPTION_{payer} = NC_{T_\alpha, T_\beta}(t) (S_{T_\alpha, T_\beta}(T_\alpha) - K)^+ \quad (1.14)$$

Using the swap measure, we can determine the valuation  $V(t)$  of the  $SWAPTION_{payer}$  at time  $t < T_\alpha$

$$V(t) = N \times C_{T_\alpha, T_\beta}(t) \times \mathbb{E}^{\mathbb{Q}_{\alpha, \beta}} \left[ (S_{T_\alpha, T_\beta}(T_\alpha) - K)^+ | \mathcal{F}_t \right] \quad (1.15)$$

## 1.7 No-arbitrage Pricing

In this section, we will introduce basic notions in mathematical financial theory. This section relies on Brigo and Mercurio (2006) [8] and Tan (2012) [58].

For 50 years, the key assumption in financial mathematics and financial literature has been the absence of arbitrage opportunities in the financial market. Black and Scholes (1973) [6] are the key initiators. For the purpose of this work, we will not delve into the issues of funding and multi-curves frameworks, which have become very topical after the 2008 GFC.

Two fundamental theorems results of this hypothesis and are names the fundamental theorems of asset pricing (FTAP). We assume  $(\Omega, \mathcal{F}, \mathbb{P})$  is a probability space with the natural right-continuous filtration  $\mathcal{F} = \{\mathcal{F}_t : 0 \leq t \leq T\}$ .

**Theorem 1.7.1** (FTAP 1). *If a market model has a risk-neutral probability measure, then it does not admit arbitrage.*

**Theorem 1.7.2** (FTAP 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.*

**Definition 1.7.3.** *A financial market is complete if and only if every contingent claim is attainable.*

Harrison and Pliska (1983) demonstrates the following results:

**Theorem 1.7.4.** *A financial market is (arbitrage free and) complete if and only if there exists a unique equivalent martingale measure  $\mathbb{Q}$ . The existence of a unique equivalent martingale measure, therefore, not only makes the markets arbitrage free, but also allows the derivation of a unique price associated with any contingent claim.*

**Definition 1.7.5.** *A equivalent martingale measure  $\mathbb{Q}$  is a probability measure such that each share price is exactly equal to the discounted expectation of the share price under this measure.*

Hence, we obtain the unique price of a contingent claim  $V$  at time  $t$

$$\begin{aligned} V(t) &= B(t) \times \mathbb{E}^{\mathbb{Q}} \left( \frac{V(T)}{B(T)} \middle| \mathcal{F}_t \right) \\ &= \mathbb{E}^{\mathbb{Q}} \left( \exp \left[ - \int_t^T r(s) ds \right] V(T) \middle| \mathcal{F}_t \right) \end{aligned} \quad (1.16)$$

where the numeraire used is the bank account.

**Definition 1.7.6.** *A numeraire is a positive, non-dividend paying asset by which value is computed. It is a reference for all other assets*

## 1.8 Numeraire change

It is very helpful to use other numeraires under different respective measures which will make valuation computation easier.

We introduce two very helpful numeraire changes. See *Changes of numéraire, changes of probability measure and option pricing* [26] and *A Change of Numeraire Toolkit* (Brigo & Mercurio) [9].

### 1.8.1 T-forward measure

This measure uses as numeraire a zero coupon bond with maturity  $T$ . The measure will be denoted:  $F^T$ . Under this numeraire, a contingent claim  $V$  will have the unique value:

$$V(t) = P(t, T) \times \mathbb{E}^{F^T} \left[ \frac{V(T)}{P(T, T)} \middle| \mathcal{F}_t \right] = P(t, T) \times \mathbb{E}^{F^T} [V(T) | \mathcal{F}_t] \quad (1.17)$$

### 1.8.2 Swap measure

This measure uses as numeraire portfolio of many zero-coupon bonds with maturity  $T_{\alpha+1}, \dots, T_{\beta}$ . The value of this portfolio at time  $t < T_{\alpha+1}$  is:

$$C_{T_{\alpha}, T_{\beta}}(t) = \sum_{i=\alpha+1}^{\beta} (T_i - T_{i-1}) P(t, T_i) \quad (1.18)$$



The measure will be denoted:  $\mathbb{Q}^{\alpha,\beta}$ . Under this numeraire, a contingent claim  $V$  will have the unique value:

$$V(t) = C_{T_\alpha, T_\beta}(t) \mathbb{E}^{\mathbb{Q}^{\alpha,\beta}} \left[ \frac{V(T_\alpha)}{C_{T_\alpha, T_\beta}(T_\alpha)} \middle| \mathcal{F}_t \right] \quad (1.19)$$

# Chapter 2

## Interest Rates & SABR models

As our aim is to be able to price swaptions, we will review here some of the main models used in the financial market to price swaptions and to determine implied volatility. This chapter was written thanks to *The Volatility Surface* (2006) [24] and *Option Pricing Model comparing Louis Bachelier with Black-Scholes Merton* (2016) [59].

### 2.1 The Bachelier (Normal) Model

Louis Bachelier was a french mathematician (1870-1946) who introduced the Brownian motion in his PhD thesis (*Théorie de la spéculation*, 1900) [3]. Bachelier is considered as the forefather of mathematical finance and a pioneer in the study of stochastic processes.

The Bachelier Model describes the instantaneous forward rate  $F_t$  with the dynamic:

$$dF_t = \sigma_{Bachelier} dW_t \quad (2.1)$$

where  $\sigma_{Bachelier}$  is a constant volatility and  $W_t$  is a standard brownian motion (under the forward measure). The solution is:

$$F(t) = F(0) + \sigma_{Bachelier} W_t, \quad W_t \sim \mathcal{N}(0, \sqrt{t}) \quad (2.2)$$

This model is popular because it allows negative interest rates (particularly interesting in these period of low rates). But it has a big drawback, i.e. due to the normal distribution of interest rates it assumes there is no lower bound and interest rates can go infinitely low, which is economically not realistic.

Under this model, the value of a swaption payer is:

$$SWAPTION_{Payer} = NC_{T_\alpha, T_\beta}(t) \left( [f - K] \Phi(d) + \sigma_{Bachelier} \sqrt{T_\alpha} \phi(d) \right) \quad (2.3)$$

with  $d = \frac{f-K}{\sigma_{Bachelier} \sqrt{T_\alpha}}$  and  $f = S_{T_\alpha, T_\beta}(t)$

$T_\alpha$  is the instant the swap starts,  $\Phi$  is the normal cumulative probability function,  $\phi$  is the normal probability density function.

## 2.2 The Black (Log-Normal) Model

Fischer Black (1938-1995) was an American economist, Nobel Prize in economy, known for the Black–Scholes equation.

The Black Model describes the instantaneous forward rate  $F_t$  with the dynamic:

$$dF_t = \sigma_{Black} dW_t \quad (2.4)$$

where  $\sigma_{Black}$  is a constant volatility and  $W_t$  is a standard Brownian motion (under the forward measure). The solution is:

$$F(t) = F(0) \exp\left(\sigma_{Black} W_t - \frac{1}{2} \sigma_{Black}^2 t\right), \quad W_t \sim \mathcal{N}(0, \sqrt{t}) \quad (2.5)$$

This model follows a log normal distribution, which forbids negative forward interest rate. Due to the current economic context, this propriety is not so relevant anymore and it can become an issue for the model itself.

Under this model, the value of a payer swaption is:

$$SWAPTION_{Payer} = NC_{T_\alpha, T_\beta}(t) [f\Phi(d_+) - K\Phi(d_-)] \quad (2.6)$$

$$\text{where } d_\pm = \frac{\log(\frac{f}{K}) \pm \frac{1}{2} \sigma_{Black}^2 T}{\sigma_{Black} \sqrt{T}}$$

## 2.3 Discussions

In the financial industry, these two formulas are used to price and hedge European options. The price and volatility are directly linked. The volatility which matches the market price is referred as the implied volatility  $\sigma_{implied}$ .

However, in these two models, we assume the volatility to be constant. When looking at the market, different volatilities are needed for different strikes. This is usually known as the volatility smile.

In order to price and hedge correctly it is very important to understand the dynamic of the volatility. Models where volatility is not constant are of two types: local volatility models and stochastic volatility models.

Local volatility models have been developed by Dupire (1994) [22], Derman & Kani (1994, 1998) [19] [13]. Stochastic volatility models are in many flavours, and a few relevant examples can be: Hull & White (1987) [40], Heston (1993) [37], Lipton (2002) [45], Hagan et al (2002) [30].

A great summary is provided in Lecture Notes: *Stochastic Volatility and Local Volatility* by Jim Gatheral(2012) [25]

We will review specifically SABR model, which was developed by Patrick S. Hagan, Deep Kumar, Andrew Lesniewski, and Diana Woodward to solve this main

issue [30]. This section was written thanks to: *Explicit SABR Calibration Through Simple Expansions* (2014) [43], *Implied Volatilities for Mean Reverting SABR Models* (2020) [32], *Managing Smile Risk* (2002) [29], *Managing Vol Surfaces* (2018) [33] and *The SABR Chronicles* [28].

## 2.4 SABR dynamic

The SABR model is a popular model in finance and compared to models presented in chapter 2, the volatility is not assumed to be constant. It is a stochastic volatility model which tries to capture the volatility smile. It is widely used in interest rates derivative market. SABR means Stochastic Alpha Beta Rho, where alpha beta and rho are the parameters of the model.

We work as usual on a probability space:  $(\Omega, \mathcal{F}, \mathbb{P})$ . We assume the forward follows the below 2 factors dynamics:

$$\begin{aligned} dF_t &= \sigma_t(F_t)^\beta dW_t & F_0 &= f \\ d\sigma_t &= \nu\sigma_t dZ_t & \sigma_0 &= \alpha \\ dW_t dZ_t &= \rho dt \end{aligned} \quad (2.7)$$

The initial values are the net present value of the forward rate  $F_0$  (discount by the tenor) and  $\sigma_0$ .  $W_t$  and  $Z_t$  are two Wiener processes under the forward measure, their correlation is  $-1 \leq \rho \leq 1$ .  $\rho$  is the slope of the implied skew,  $\nu$  is called the volvol (volatility of volatility) and satisfies  $\nu \geq 0$ .  $\alpha$  is the level of the implied volatility of ATM options and satisfies  $\alpha \geq 0$ . Obviously with  $\alpha = 0$ , we get a CEV (constant elasticity of variance) model.

$\beta$  satisfies  $0 \leq \beta \leq 1$ . If  $\beta = 1$ , the SABR dynamics correspond to a stochastic lognormal model ; if  $\beta = 0$ , the SABR dynamics correspond to a stochastic normal model; if  $\beta = 1/2$ , the dynamics correspond to a square-root process, and example of which is CIR model (here with no drift) as explained in *A theory of the term structure of interest rates* (1985) [16].

## 2.5 SABR asymptotic solution

We consider an option on the forward  $F$  with a strike  $K$  and a maturity  $T$  and a payoff  $P$ . The value of the option is the discounted expected payoff.

An asymptotic expansion in  $\epsilon := T\nu^2$  gives a solution of the SABR equations for the implied volatility:

$$\begin{aligned} \sigma_{\text{impl}} &= \frac{\alpha}{(fK)^{(1-\beta)/2}} \left\{ 1 + \left[ \frac{(1-\beta^2)}{24} \frac{\alpha^2}{(fK)^{1-\beta}} + \frac{1}{4} \frac{\alpha\beta\nu\rho}{(fK)^{(1-\beta)/2}} + \frac{2-3\rho^2}{24} \nu^2 \right] T \right\} \\ &\quad \times \frac{1}{1 + \frac{1}{24}(1-\beta)^2(\ln f/K)^2 + \frac{1}{1920}(1-\beta)^4(\ln f/K)^4} \times \frac{\zeta}{D(\zeta)} \end{aligned} \quad (2.8)$$

where:

- $\zeta = \frac{\nu}{\alpha}(fK)^{(1-\beta)/2} \ln\left(\frac{f}{K}\right)$

$$\bullet D(\zeta) = \log \left( \frac{\sqrt{1-2\rho\zeta+\zeta^2+\zeta-\rho}}{1-\rho} \right)$$

When the option is at the money, i.e. when  $f = K$ , this approximation can be written:

$$\sigma_{\text{impl}} = \alpha \frac{1}{f^{1-\beta}} \left\{ 1 + \left[ \frac{(1-\beta)^2}{24} \frac{\alpha^2}{f^{2-2\beta}} + \frac{1}{4} \frac{\rho\beta\alpha\nu}{f^{1-\beta}} + \frac{2-3\rho^2}{24} \nu^2 \right] T \right\} \quad (2.9)$$

The term  $\alpha \frac{1}{f^{1-\beta}} = \alpha \frac{f^\beta}{f}$  is very important, as for  $T = 0$  the implied volatility is expressed in function of the initial volatility and the forward rate percentage.

## 2.6 SABR for negative interest rates

Due to the popularity of the SABR model which has become a workhorse of the financial industry, after the GFC and the low interest rates regime that ensued, the SABR model has been adapted to negative interest rates, as in *The Free Boundary SABR: Natural Extension to Negative Rates* (2015) [1], *From arbitrage to arbitrage-free implied volatilities* (2016) [27] and *Finite Difference Techniques for Arbitrage Free SABR* (2014) [44].

### 2.6.1 Shifted SABR

The Shifted SABR method translates the forward rate of a constant  $s$ . The purpose is to shift the lower bound of the forward rate to force it to be always positive. The value of this shift is dependant of how low is the economic environment.

$$\begin{aligned} dF_t &= \sigma_t(F_t + s)^\beta dW_t & F_0 &= f \\ d\sigma_t &= \alpha\sigma_t dZ_t & \sigma_0 &= \alpha \\ dW_t dZ_t &= \rho dt \end{aligned} \quad (2.10)$$

where  $s$  is a positive shift.

### 2.6.2 Absolute SABR

The Absolute SABR method forces the forward rate to stay positive thanks to taking the absolute value of the forward rate. Whatever the economic environment, the forward will be positive.

$$\begin{aligned} dF_t &= \sigma_t |F_t|^\beta dW_t & F_0 &= f \\ d\sigma_t &= \alpha\sigma_t dZ_t & \sigma_0 &= \alpha \\ dW_t dZ_t &= \rho dt \end{aligned} \quad (2.11)$$

## 2.7 SABR calibration

To calibrate the SABR model we would need to determine four parameters:  $\alpha, \beta, \rho, \nu$ .  $F_0 = f$  is observable directly in the market.

To proceed into this calibration, it is common to fix  $\beta$ . This procedure is allowed, because  $\rho$  and  $\beta$  have similar influence, as proved by Tan (2012) [58]. Normally it is the desk that assign the parameter by their understanding of the market dynamics.

To calibrate  $(\alpha, \rho, \nu)$ , we solve the following equation:

$$(\hat{\alpha}, \hat{\rho}, \hat{\nu}) = \arg \min_{(\alpha, \rho, \nu)} \sum_i d[\sigma^{market, ATM}, \sigma^{impl}(f_i, T_i, \alpha, \rho, \nu)] \quad (2.12)$$

where we loop on the different volatilities observed in the market and  $d(x, y)$  is an error metric. It is common to use  $d(x, y) = (y - x)^2$ .

An other common method is to fix  $\alpha$  and to minimise only on  $(\rho, \nu)$ . Then  $\alpha$  is found by solving the cubic polynomial for ATM market volatilities:

$$\begin{aligned} & \left( \frac{\beta(\beta - 2)}{24} T f^{3\beta - 2} \right) \alpha^3 + \left( \frac{1}{4} \rho \beta \nu T f^{2\beta - 1} \right) \alpha^2 \\ & + \left( 1 + \frac{2 - 3\rho^2}{24} \nu^2 T \right) f^\beta \alpha - \sigma^{market, ATM} = 0 \end{aligned} \quad (2.13)$$

However, three roots will be determined. It has been proved by West (2005) [61], that the lowest positive one must be chosen. Due to the real coefficients and the oddness of the polynomial degree, there will be at least one real solution.

## 2.8 SABR parameters

It is interesting to understand the influence of each parameters on the volatility.

### 2.8.1 $\alpha$ parameter

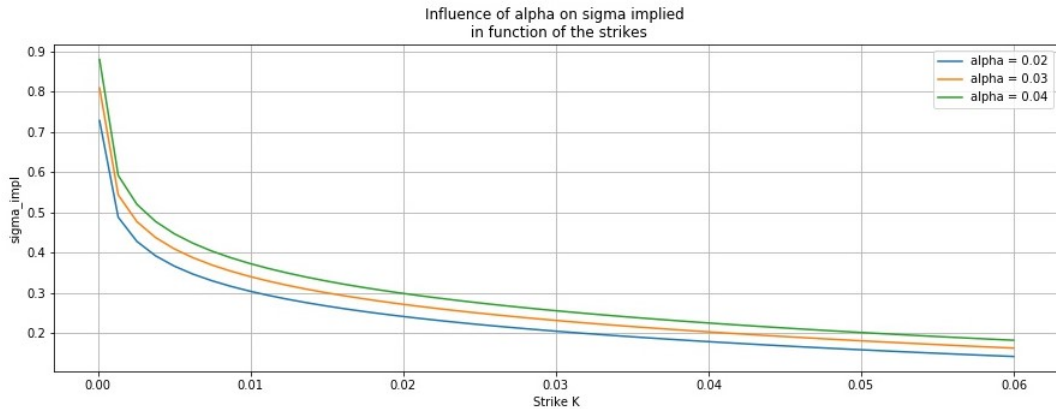


Figure 2.1: alpha parameter influence

It is important to recall that  $\sigma_0 = \alpha$ , it is the first instant of the stochastic volatility. Hence the higher is  $\alpha$ , the more the curve is shifted upward.

### 2.8.2 $\beta$ parameter

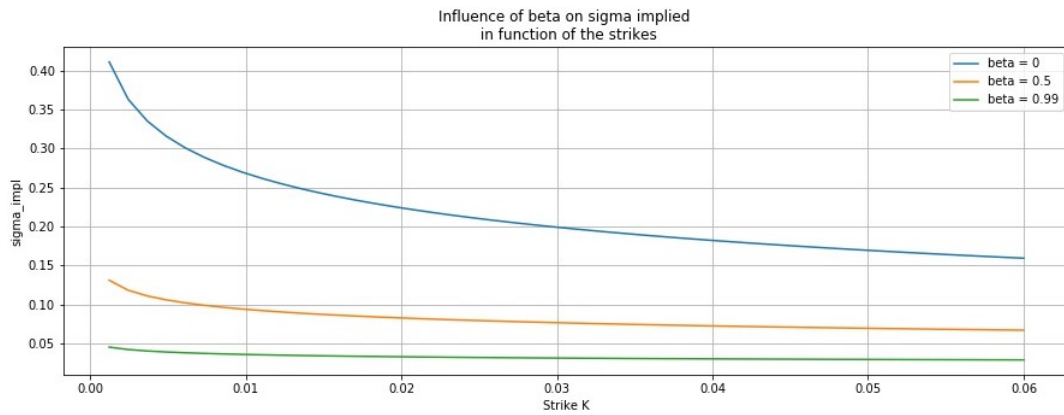


Figure 2.2: beta parameter influence

We observe  $\beta$  is responsible of the volatility smile of the curve. This influence on the volatility smile is higher for lower strikes.

We also notice that the higher is  $\beta$ , the lower is the curve. This is normal because we recall  $\beta$  is the power of the forward rate:

$$dF_t = \sigma_t(F_t)^\beta dW_t$$

### 2.8.3 $\rho$ parameter

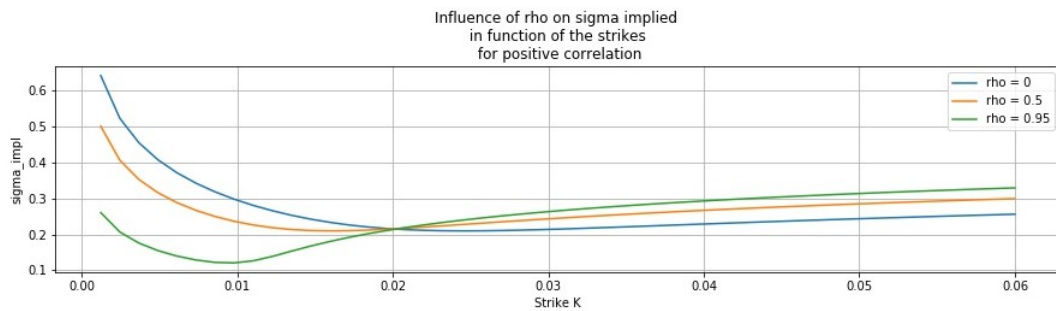


Figure 2.3: positive rho parameter influence

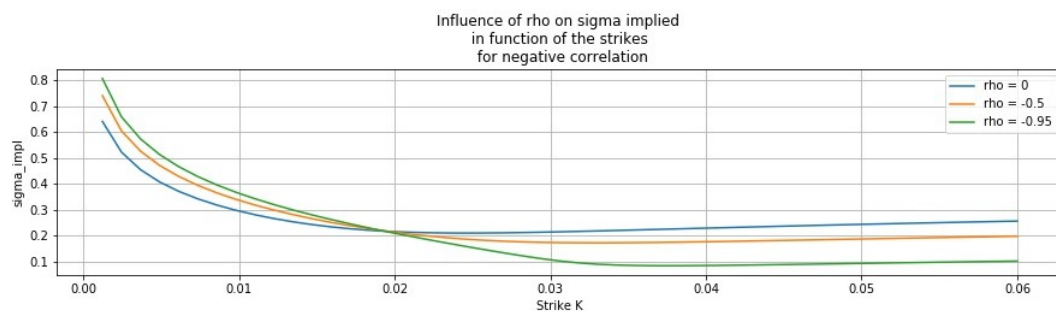


Figure 2.4: negative rho parameter influence

We observe that for positive  $\rho$ , increasing  $\rho$  creates a steeper volatility smile. On the contrary, for negative  $\rho$ , decreasing  $\rho$  creates a steeper volatility smile. The effect of  $\rho$  is similar to the effect of  $\beta$ , which justifies (as we will see in next section) fixing one parameter for the calibration.

### 2.8.4 $\nu$ parameter

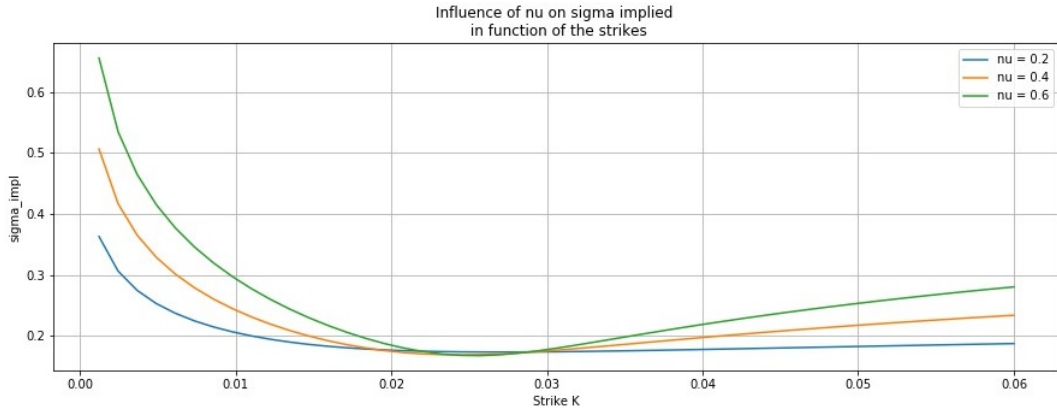


Figure 2.5:  $\nu$  parameter influence

We notice the higher is  $\nu$ , the more convex is the curve.

## 2.9 SABR complements

Precious help and illustrations for this section were found in *The SABR Model* by F. Rouah [20] and BSIC: Bocconi Students Investment Club <sup>1</sup>.

For strikes close to the forward rate, we can approximate the implied volatility formula by:

$$\sigma_{impl}(f, K) = \frac{\alpha}{f^{1-\beta}} \left\{ 1 - \frac{1}{2} (1 - \beta - \rho\lambda) \ln \left( \frac{K}{f} \right) + \frac{1}{12} [(1 - \beta)^2 + (2 - 3\rho^2)\lambda^2] \ln^2 \left( \frac{K}{f} \right) \right\} \quad (2.14)$$

where  $\lambda = \frac{\nu}{\alpha} f^{1-\beta}$  is the strenght of  $\nu$ , the volvol compared to the local volatility of the forward rate.

We can now study the implied volatility deeper.

### 2.9.1 The Backbone

The factor  $\frac{\alpha}{f^{1-\beta}}$  in the ATM  $\sigma_{impl}$  is very important, it is the change in ATM vol for a change in the forward rate.

The backbone is characterised completely by  $\beta$ .

<sup>1</sup>[www.bsic.it/sabr-stochastic-volatility-model-volatility-smile](http://www.bsic.it/sabr-stochastic-volatility-model-volatility-smile)



- $\beta = 0$  (stochastic volatility normal model) gives a steeply downward sloping backbone
- $\beta = 1$  (stochastic volatility log-normal model) gives a nearly flat backbone

We plot the backbone for fixed  $\alpha, \beta, \rho, \nu$ , for different values of  $f$ , and ATM strikes i.e.  $f = K$ .

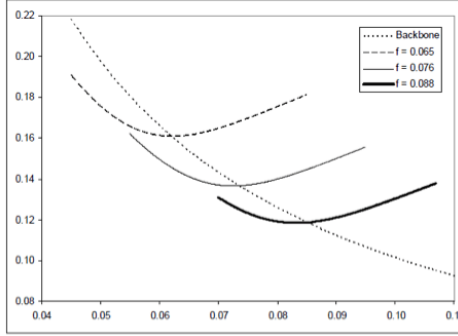


Figure 2.6: Backbone effect with  $\beta = 0$

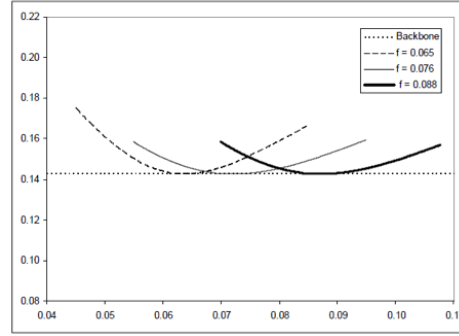


Figure 2.7: Backbone effect with  $\beta = 1$

## 2.9.2 The Skew

The skew is determined by  $-\frac{1}{2}(1 - \beta - \rho\lambda) \log\left(\frac{K}{f}\right)$ . We decompose the skew in two terms:

- **The Vanna Skew**  $\frac{1}{2}\rho\lambda \log\left(\frac{K}{f}\right)$ . This term is the correlation between instantaneous volatility and  $f$ .
- **The Beta Skew**  $-\frac{1}{2}(1 - \beta) \log\left(\frac{K}{f}\right)$ . Because  $0 \leq (1 - \beta) \leq 1$ , the implied vol is decreasing in  $K$ .

## 2.9.3 The convexity

The convexity is determined by  $\frac{1}{12}[(1 - \beta)^2 + (2 - 3\rho^2)\lambda^2] \ln^2\left(\frac{K}{f}\right)$ . We decompose the convexity into two terms which induce the smile:

- **The Skew effect**  $\frac{1}{12}(1 - \beta)^2 \ln^2\left(\frac{K}{f}\right)$
- **The Volga effect**  $\frac{1}{12}(2 - 3\rho^2)\lambda^2 \ln^2\left(\frac{K}{f}\right)$

# Chapter 3

## Artificial Neural Networks

In this chapter, we introduce Artificial Neural Networks or just Neural Networks. This chapter is inspired of *Deep Learning: An Introduction for Applied Mathematicians* (2019) [38] and *Deep Learning Lecture notes* by Mikko Pakkanen (2019) [52]. Illustrations in this chapter without mentioned sources are extract from *Deep Learning Lectures Notes*.

### 3.1 Introduction and history of Deep Learning

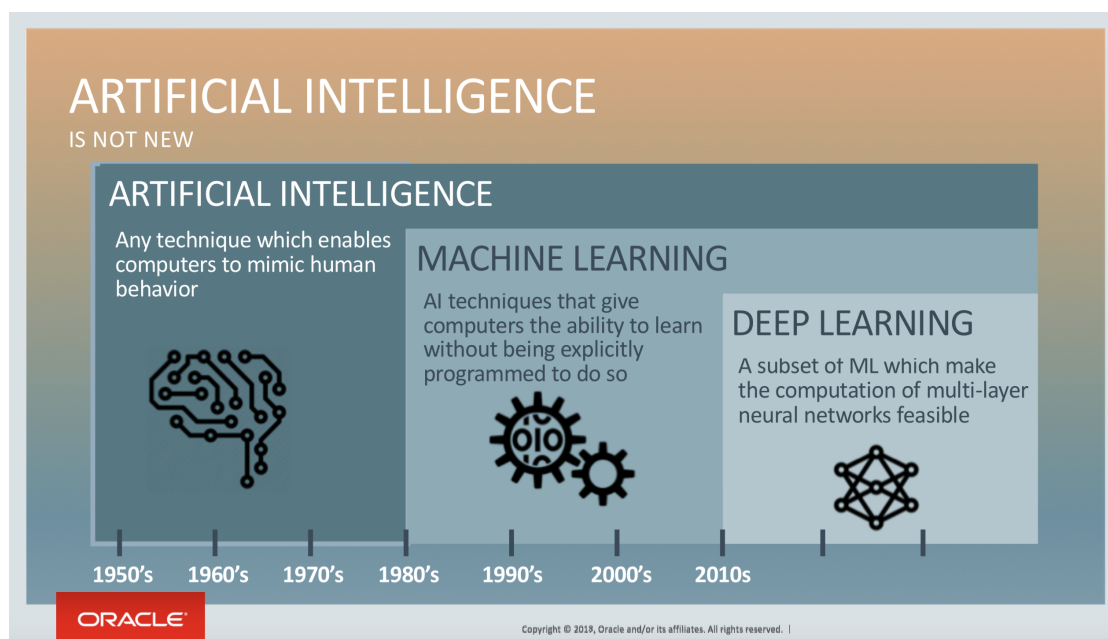


Figure 3.1: History of AI

Deep Learning is part of Machine Learning that employs Neural Networks to manage problem with high complexity such as speech or image recognition, optimal decision making. Yann Le Cun <sup>1</sup> - together with Geoffrey Hinton and Yoshua Bengio are named the "Godfathers of AI" and "Godfathers of Deep Learning" <sup>2</sup>. It also ex-

<sup>1</sup>[yann.lecun.com](http://yann.lecun.com)

<sup>2</sup>[www.forbes.com/sites/samshead/2019/03/27/the-3-godfathers-of-ai-have-won-the-prestigious-lm-turing-prize](http://www.forbes.com/sites/samshead/2019/03/27/the-3-godfathers-of-ai-have-won-the-prestigious-lm-turing-prize)

plained because for two decades (1990-2010) only the three of them in IA community believed in Neural Networks.

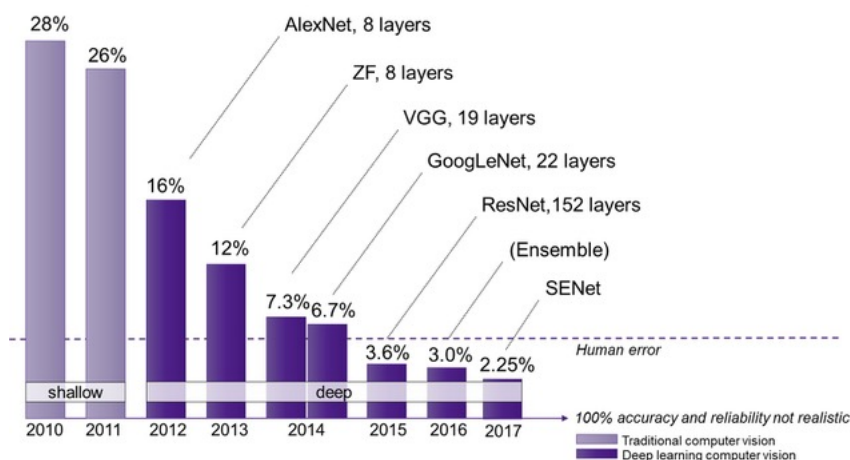


Figure 3.2: ImageNet LSVRC

In 2010's, Deep Learning entered a new era. Every year, ImageNet Large Scale Visual Recognition Challenge (LSVRC) is organized. In 2012, at general surprise, a deep learning algorithm improved drastically the performances. This for two reasons. First, data amount available became big enough to train Neural Network. Secondly, GPU processors did big progresses.

In 2015, the general audience discovered Deep Learning, Google DeepMind created a code which defeated best human players in the Chinese Game: AlphaGo which was considered too complex for computers (like with IBM Deep Blue for chess against Garry Kasparov in 1996).

The financial industry rapidly got interest in this new field, plenty of promises. As explained by Hans Buehler <sup>3</sup>, parametric models are always an approximation and data-driven models can "do much heavier, much more precise calculations". As mentioned in the introduction, banks rely on two decades of stored financial data. They also can afford the best cutting-edge technologies. Naturally, they are front-runners in this new field. Equivalent competitors are Oil extraction companies. Main applications are Pricing, hedging, model calibration, algorithmic-trading.

<sup>3</sup>[www.risk.net/derivatives/6705012/podcast-hans-buehler-on-deep-hedging-and-the-advantages-of-data-driven-approaches](http://www.risk.net/derivatives/6705012/podcast-hans-buehler-on-deep-hedging-and-the-advantages-of-data-driven-approaches)

## 3.2 Architecture of Neural Networks

### 3.2.1 Brief Description

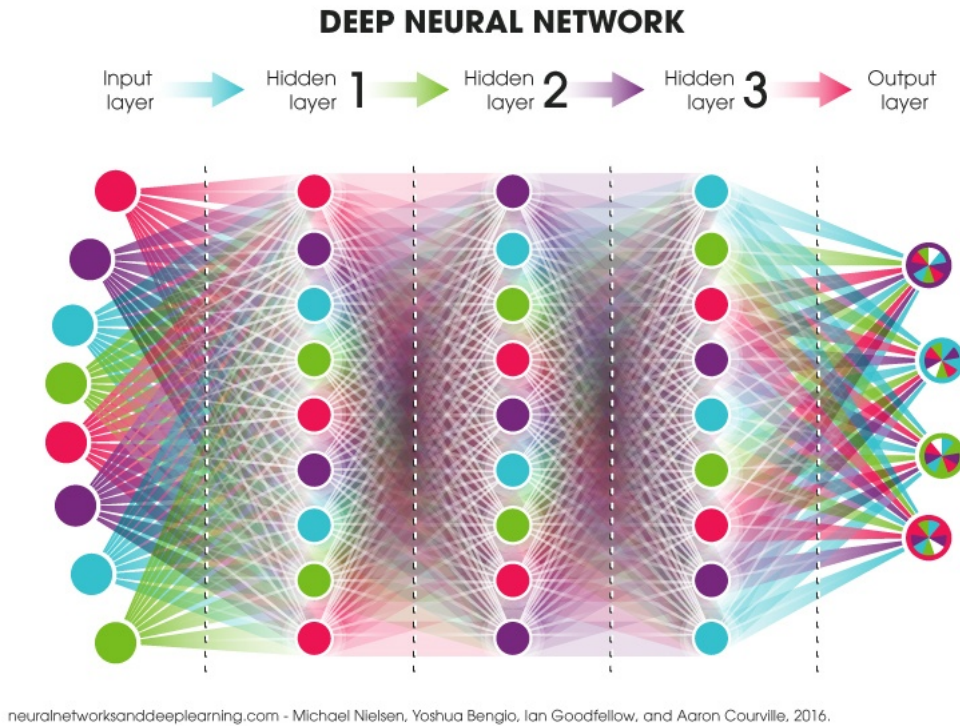


Figure 3.3: Neural Network illustration

Neural Networks imitates the behaviour of biological neural networks. However, Medicine has now a better understanding and this modelisation of biological Neural Network is contested. Hence the comparison with human brain is for historical reasons <sup>4</sup>.

It is formed by thousands of neurons (or units) connected together through consecutive layers. The first layer is called input layer, the last is the output layer and intermediary layers are called hidden layers. Only the Feed Forward Neural Network will be studied in this project, it is the most popular Neural Network architecture. However plenty of other models exist. A complete graph "The mostly complete chart of Neural Networks" from [www.towardsdatascience.com](http://www.towardsdatascience.com) is provided in appendix 1.

### 3.2.2 Detail Construction

The purpose of Neural Networks is to approximate a function:

$$f = (f_1, \dots, f_O) : \mathbb{R}^I \rightarrow \mathbb{R}^O \quad (3.1)$$

with  $I \in \mathbb{N}$  inputs and  $O \in \mathbb{N}$  outputs.

<sup>4</sup>[www.analyticsindiamag.com/neural-networks-not-work-like-human-brains-lets-debunk-myth](http://www.analyticsindiamag.com/neural-networks-not-work-like-human-brains-lets-debunk-myth)

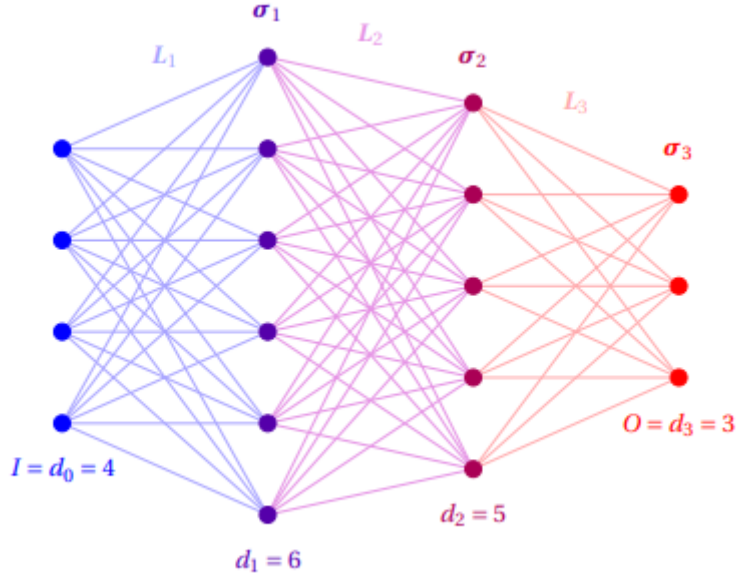


Figure 3.4: Architecture of Neural Network

**Definition 3.2.1.** Let  $I, O, r \in \mathbb{N}$ .

Function  $f : \mathbb{R}^I \rightarrow \mathbb{R}^O$  is a feedforward neural network with  $r - 1 \in \{0, 1, \dots\}$  hidden layers, where there are  $d_i \in \mathbb{N}$  units in the  $i$ -th hidden layer for any  $i = 1, \dots, r - 1$ , and activation functions  $\sigma_i : \mathbb{R}^{d_i} \rightarrow \mathbb{R}^{d_i}$ ,  $i = 1, \dots, r$ , where  $d_r := O$ , if

$$\mathbf{f} = \sigma_r \circ \mathbf{L}_r \circ \dots \circ \sigma_1 \circ \mathbf{L}_1 \quad (3.2)$$

where  $\mathbf{L}_i : \mathbb{R}^{d_{i-1}} \rightarrow \mathbb{R}^{d_i}$ , for any  $i = 1, \dots, r$ , is an affine function

$$\mathbf{L}_i(\mathbf{x}) := W^i \mathbf{x} + \mathbf{b}^i, \quad \mathbf{x} \in \mathbb{R}^{d_{i-1}} \quad (3.3)$$

parametrised by weight matrix  $W^i = [W_{j,k}^i]_{j=1, \dots, d_i, k=1, \dots, d_{i-1}} \in \mathbb{R}^{d_i \times d_{i-1}}$  and bias vector

$\mathbf{b}^i = (b_1^i, \dots, b_{d_i}^i) \in \mathbb{R}^{d_i}$ , with  $d_0 := I$ . We shall denote the class of such functions  $\mathbf{f}$  by

$$\mathcal{N}_r(I, d_1, \dots, d_{r-1}, O; \sigma_1, \dots, \sigma_r) \quad (3.4)$$

If  $\sigma_i(\mathbf{x}) = (g(x_1), \dots, g(x_{d_i}))$ ,  $\mathbf{x} = (x_1, \dots, x_{d_i}) \in \mathbb{R}^{d_i}$ , for some  $g : \mathbb{R} \rightarrow \mathbb{R}$ , we write will  $g$  in place of  $\sigma_i$ .

The architecture of the Neural Network is fully characterised by:

- $W^1, \dots, W^r$  and  $b^1, \dots, b^r$  are the parameters
- $\sigma^1, \dots, \sigma^r$  are the activation functions
- $r$  and  $d_1, \dots, d_{r-1}$  are the hyper-parameters

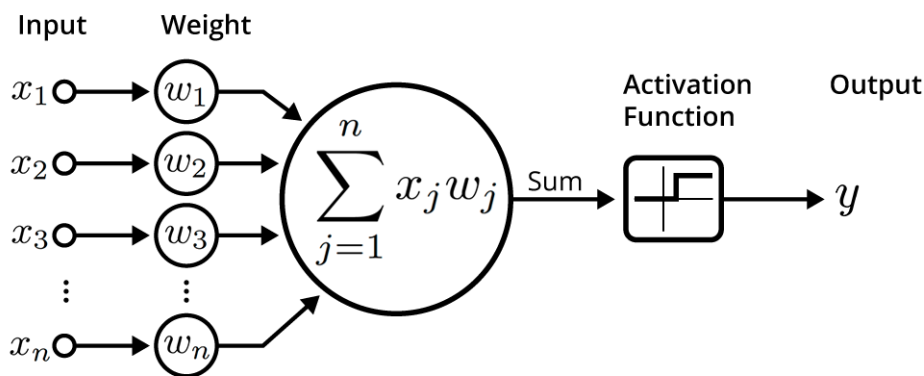
The composition of linear function is very important in Neural Network to capture non linearities. By composing affine functions, it produces high-order polynomials exponentially quickly.

It is important to consider the number of parameters of the architecture. Each parameter will have to be trained later.

**Proposition 3.2.2.** *The number of parameters that characterise  $f \in \mathcal{N}_r(I, d_1, \dots, d_{r-1}, O; \sigma_1, \dots, \sigma_r)$  is*

$$\sum_{i=1}^r (d_{i-1} + 1) d_i \tag{3.5}$$

### 3.2.3 Activation Functions



An illustration of an artificial neuron. Source: Becoming Human.

Figure 3.5: Activation Function Illustration

After each affine transformation, an activation function is applied to the output which becomes the input of the next layer. Activation Functions have a crucial role in the architecture of the Neural Network.

We can study some of them.

There are two types of Activation Functions:

- one-dimensional Activation Functions
- multi-dimensional Activation Functions

Detailed illustrations can be found in appendix for one-dimensional Activation Functions 2 and multi-dimensional Activation Functions 3.

## 3.3 Training of Neural Networks

In this section, we describe how to train the Neural Network.

### 3.3.1 Loss Function

The first thing needed to calibrate a Neural Network is to get a metric to quantify the error of calibration. The goal obviously will be to minimise this error.

**Definition 3.3.1.** A loss function is

$$\ell : \mathbb{R}^O \times \mathbb{R}^O \rightarrow \mathbb{R} \tag{3.6}$$

If  $X$  is the random vector input and  $Y$  is the random vector output, the goal is to make match  $Y$  and  $f(X)$ , i.e. to minimise  $\ell(f(X), Y)$ . If we knew the distribution of  $(X, Y)$ , we could search optimal  $f$  by minimising:

$$\mathbb{E} [\ell(f(X), Y)]$$

However practically, the distribution is not know, so we will minimise on samples i.e.

$$\mathcal{L}(f) = \frac{1}{N} \sum_{i=1}^N \ell (f(x^i), y^i)$$

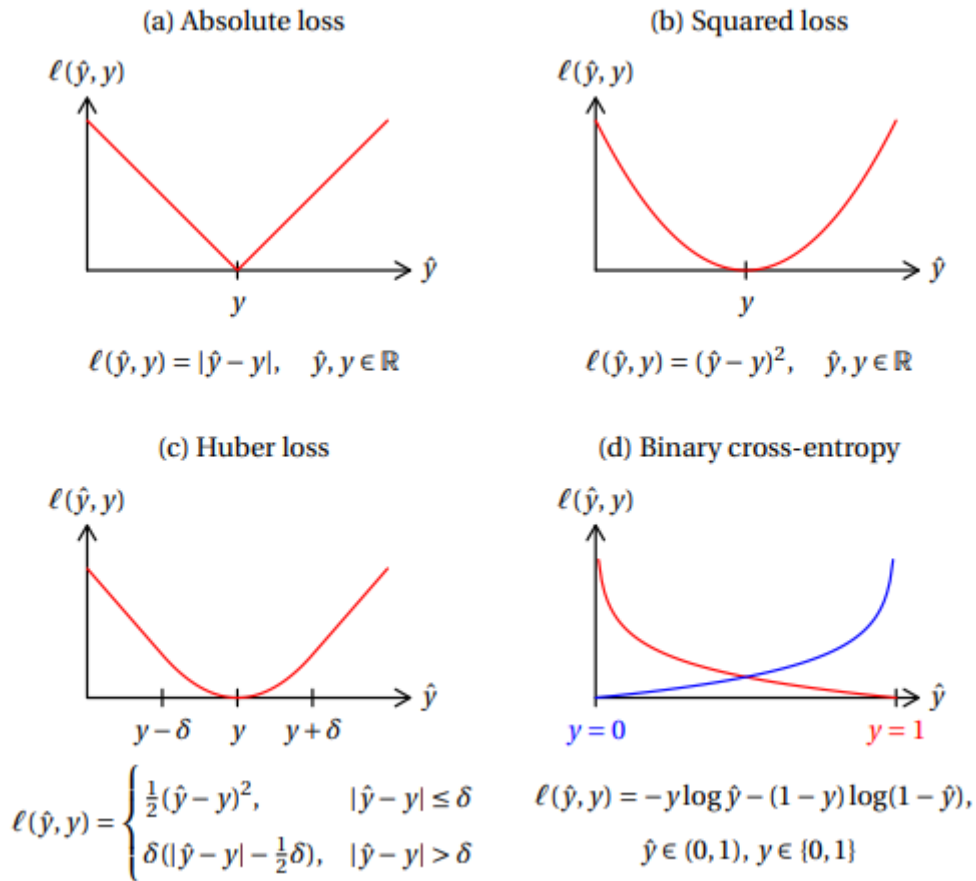


Figure 3.6: Loss Function List

The loss function is minimised for:

$$\hat{y} = \arg \min_{\hat{y} \in \mathbb{R}} \mathbb{E} [\ell(\hat{y}, Y)] \tag{3.7}$$

We can study some of the most famous loss functions:

- Absolute Loss

Equation 3.7 is minimised for  $\hat{y}$  equals the median. So Absolute Loss targets the median.

- Squared Loss

Equation 3.7 is minimised for  $\hat{y}$  equals the mean. So Squared Loss targets the mean.

A potential issue with Squared Loss is that because of the square, far value from the target will be more penalised than close values. Outliers can be over-emphasised.

- Huber Loss

Huber Loss tries to mitigate this issue. It behaves as a Squared Loss function close to zero values and as an Absolute Loss further.

- Binary cross-entropy

Binary cross-entropy is a loss function for binary output.

For multidimensional outputs, the loss function can be a sum of one-dimension loss functions seen previously.

### 3.3.2 Minibatch and batch size

#### Minibatch

Minibatch is an important concept in loss function minimising.

The error is average on different subset called minibatch:  $B \subset \{1, \dots, N\}$  of samples.

The function to minimise is rewritten:

$$\mathcal{L}_B(f) = \frac{1}{\#B} \sum_{i \in B}^N \ell(f(x^i), y^i)$$

#### Batch size

The batch size is the number of samples in a minibatch.

### 3.3.3 Epochs

One epoch is the passage of the whole data set through the Neural Network.

One epoch is divided in smaller subset (minibatches).

*What it means more than one epoch?*

Neural Networks need lots of samples to train. Hence the same dataset will go



through the same Neural Network many times. The number of times is the number of epochs. This number must be chosen carefully, a small number of epoch will produce under-fitting, a large number of epochs will produce over-fitting.

*What is the right numbers of epochs?*

As we will see in later part, 4.4, parameters tuning is a subtle art. There is no rules pre-established, it is specific to each datasets.

### 3.3.4 Iterations

Iteration is an easy concept after batch size and epochs.

Iterations is the number of batch at each epoch. Hence the number of batches is equal to number of iterations for one epoch.

**Example 3.3.1.** *Let's say we have 10,000 samples.*

*If we divided this set into batches of 1000. It will takes 10 iterations to complete on epoch.*

*So batch size: 1000, iterations: 10, epoch: 1.*

### 3.3.5 Summary example

**Example 3.3.2.** *Let's say, we have a 20,000 samples dataset.*

*Let's set epochs to 500 and batch size to 100.*

*The dataset will divided in  $\frac{20,000}{100} = 200$  minibatch, each with 100 samples.*

*The model weights will be updated after each minibatch of 100 samples. One epoch will involve 200 updates to the model.*

*With 500 epochs, the model will be trained 500 times with the all dataset, hence  $200 \times 500 = 100,000$  minibatches.*

### 3.3.6 Stochastic Gradient Descent: SGD

We have seen how to quantify the error of the Neural Network. *But how to minimise this error?*

The common method is called Stochastic Gradient Descent: SGD.

SGD is an iterative method used to find minima of a curve. The gradient is the rate of inclination of the curve. The gradient descent iteratively goes to the minima by going into the decreasing way of the curve. The speed it converges to the minima is determined by the learning rate  $\eta$ .

#### Ordinary Gradient Descent

We wish to find the minimum of a function  $F : \mathbb{R}^d \rightarrow \mathbb{R}$ . We immediately solved  $\nabla F = 0$  where  $\nabla F$  is the gradient of  $F$ . However, often the explicit expression of the derivative is unknown.

$$\frac{d\mathbf{x}(t)}{dt} = -\nabla F(\mathbf{x}(t)), \quad t > 0 \quad (3.8)$$

This expression, with initial condition  $x_0 \in \mathbb{R}^d$  defines the gradient flow  $(x(t))_{t \geq 0}$  of  $F$  defined in *Gradient Flows: an overview* (2016) [56]. With  $t \rightarrow \infty$ , the minima is reached. However we need to discretise this equation in:

$$\frac{x(t + \eta) - x(t)}{\eta} \approx -\nabla F(\mathbf{x}(t)), \quad t > 0 \quad (3.9)$$

where  $\eta$  is the learning rate. The equation can be rewritten:

$$x(t + \eta) \approx x(t) - \eta \nabla F(\mathbf{x}(t)), \quad t > 0 \quad (3.10)$$

Hence we get an ordinary gradient descent process. We can converge iteratively to the minima by:

$$x_{\text{new}} \approx x_{\text{old}} - \eta \nabla F(\mathbf{x}_{\text{old}}), \quad t > 0 \quad (3.11)$$

### Stochastic Gradient Descent

The computation of gradient can be costly for Neural Networks, hence the Stochastic Gradient Descent is applied.

We start by dividing the dataset into minibatches, where the disjoint union of all minibatches is the dataset. The parameter of the Neural Network are updated after each minibatch. The output is the input of the next minibatch.

This procedure is repeated on multiple epochs. For each epoch, the split of the data into minibatches is changed.

The first initialisation of the parameters can be specified manually or be random. Parameters mustn't be all initialised to zero. The gradient descent would be symmetric and no updated would be realised after each minibatch.

SGD present some disadvantages. As we will see later, the choice of the learning rate is tricky. Variants have been proposed.

### 3.3.7 Backpropagation

We know how to reach the minima of a complex function.

*However how to update all parameters of a Neural Network?.*

The technic is called backpropagation.

But before, *How to compute the gradient?*

The gradient is not always explicitly known.

The trick is to use the finite differences method:

$$F'(x) \approx \frac{F(x + \frac{1}{2}\Delta) - F(x - \frac{1}{2}\Delta)}{\Delta} \quad (3.12)$$

where  $\Delta$  is real value close to zero.

The finite differentiation doesn't work for highly non linear functions. Other methods can solve this issue, such as symbolic differentiation or algorithmic differentiation.

Backpropagation is a special case, it is the computation of the gradient for Forward Neural Network. We look closer how it works.

We consider a function  $f$  approximated by a Neural Network with parameters  $\theta = (W^1, \dots, W^r; \mathbf{b}^1, \dots, \mathbf{b}^r)$ . The function becomes  $f_\theta$ . The hyper-parameters are:  $\mathcal{N}_r(I, d_1, \dots, d_{r-1}, O, \sigma_1, \dots, \sigma_r)$ .

The activation function  $\sigma_i = (g_i, \dots, g_i)$ .

We need to minimise:

$$\nabla_{\theta} \mathcal{L}_B(\theta) = \frac{1}{\#B} \sum_{i \in B} \nabla \ell(\mathbf{f}_\theta(\mathbf{x}^i), \mathbf{y}^i) \quad (3.13)$$

Hence we will only study the minimisation of:

$$\nabla \ell(\mathbf{f}_\theta(\mathbf{x}^i), \mathbf{y}^i) \quad (3.14)$$

We introduce common notations, extracted from Deep Learning Lecture Notes:

$$\begin{aligned} \mathbf{z}^i &= (z_1^i, \dots, z_{d_i}^i) := \mathbf{L}_i(\mathbf{a}^{i-1}) = W^i \mathbf{a}^{i-1} + \mathbf{b}^i, & i = 1, \dots, r \\ \mathbf{a}^i &= (a_1^i, \dots, a_{d_i}^i) := \mathbf{g}_i(\mathbf{z}^i), & i = 1, \dots, r \\ \mathbf{a}^0 &:= \mathbf{x} \in \mathbb{R}^I, \end{aligned} \quad (3.15)$$

hence:  $\mathbf{f}_\theta(\mathbf{x}) = \mathbf{a}^r$  and  $\ell(\mathbf{f}_\theta(\mathbf{x}), \mathbf{y}) = \ell(\mathbf{a}^r, \mathbf{y})$ .

We introduce the adjoint:  $\boldsymbol{\delta}^i = (\delta_1^i, \dots, \delta_{d_i}^i) \in \mathbb{R}^{d_i}$ .

$$\delta_j^i := \frac{\partial \ell}{\partial z_j^i}, \quad j = 1, \dots, d_i, \quad i = 1, \dots, r, \quad (3.16)$$

It is important to recall first the chain rule for  $G : \mathbb{R}^d \rightarrow \mathbb{R}$  and  $\mathbf{F} = (F_1, \dots, F_d) : \mathbb{R}^d \rightarrow \mathbb{R}^d$

$$\frac{\partial H}{\partial x_i}(x) = \sum_{j=1}^d \frac{\partial G}{\partial y_j}(y) \frac{\partial F_j}{\partial x_i}(x), \quad x = (x_1, \dots, x_d) \quad (3.17)$$

Hence:

### Proposition 3.3.2.

$$\begin{aligned} \boldsymbol{\delta}^r &= \mathbf{g}'_r(\mathbf{z}^r) \odot \nabla_{\hat{\mathbf{y}}} \ell(\mathbf{a}^r, \mathbf{y}) & (3.18) \\ \boldsymbol{\delta}^i &= \mathbf{g}'_i(\mathbf{z}^i) \odot (W^{i+1})' \boldsymbol{\delta}^{i+1}, & i = 1, \dots, r-1 \\ \frac{\partial \ell}{\partial b_j^i} &= \delta_j^i, & i = 1, \dots, r, j = 1, \dots, d_i \\ \frac{\partial \ell}{\partial W_{j,k}^i} &= \delta_j^i a_k^{i-1}, & i = 1, \dots, r, j = 1, \dots, d_i, k = 1, \dots, d_{i-1} \end{aligned}$$

where  $\odot$  stands for the component-wise Hadamard product of vectors.

## 3.4 Universal Approximation Theorem

Neural networks became incredibly popular in the financial industry, but *what justify them mathematically?* It is Universal Approximation Theorems.

Universal Approximation Theorems are of two forms:

- Theorems that prove the approximation ability of Neural Networks with an arbitrary number of artificial neurons (“arbitrary width” case).
- Theorems that prove the approximation ability of Neural Networks with arbitrary number of hidden layers, each containing a limited number of artificial neurons (“arbitrary depth” case).

These theorems, in summary tell us Neural Networks can represent plenty of functions if they have the good weights, but give no clues on how to build the architecture and hyper-parameters on the Neural Network.

### 3.4.1 Arbitrary Width

This theorem from Allan Pinkus [53] is an extension of the results of George Cybenko and Kurt Hornik [39]. It was studied too in *Universal Approximation with Deep Narrow Network* (2019) [42].

**Theorem 3.4.1.** *We set a continuous function  $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ , positive integers:  $d$  and  $D$ . The function  $\sigma$  is not a polynomial if and only if, for every continuous function  $f : \mathbb{R}^d \rightarrow \mathbb{R}^d$ , every subset compact  $K \in \mathbb{R}^d$ , and every  $\epsilon > 0$ , there exists a continuous function  $f_\epsilon : \mathbb{R}^d \rightarrow \mathbb{R}^d$  with representation  $f_\epsilon = W_2 \circ \sigma \bullet W_1$ , where  $W_1, W_2$  are composable affine maps and  $\bullet$  denotes component-wise composition, such that the approximation bound  $\sup_{x \in K} \|f(x) - f_\epsilon(x)\| < \epsilon$  holds.*

This theorem can be immediately applied to neural networks with any fixed number of hidden layers. It assumes the first layer can approximate any desired function and the latter approximate the identity function. In conclusion any fixed-depth network can approximate any continuous function with bounded depth and arbitrary width.

### 3.4.2 Arbitrary Depth

The Universal Approximation Theorem was proved by Zhou Lu et al (2017) [50]. They proved Neural Networks of width  $n + 4$  can approximate any Lebesgue integrable function on  $n$ -dimensional input space with respect to  $L^1$ -distance if depth can be arbitrary.

**Theorem 3.4.2.** *For any Lebesgue-integrable function  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  and any  $\epsilon > 0$ , there exists a fully-connected ReLU network  $\mathcal{A}$  with width  $d_m < N + 4$  such that the function  $F_{\mathcal{A}}$  represented by this network satisfies  $\int_{\mathbb{R}^n} |f(x) - F_{\mathcal{A}}(x)| dx < \epsilon$*

In conclusion any fixed-width network can approximate any continuous function with bounded width and arbitrary depth.

# Chapter 4

## Methodology

This chapter tries to generalise and apply to a different dataset ideas pioneered in a couple of papers by the researcher A. Hernandez: *Model Calibration with Neural Networks* [35] and *Model Calibration: Global Optimizer vs. Neural Network* [36].

The methodology that we will present in this chapter can be summarised in five steps:

1. **Step 1: Data Presentation**

Market Data extraction, preprocessing and explanations

2. **Step 2: Sequential calibration on time-series**

For each day, we wish to get SABR parameters given implied volatility swaption and forward matrix. This calibration will be accomplished thanks to standard minimisation techniques, i.e. no Neural Networks. We will then obtain some hundreds market samples.

3. **Step 3: Preparing Training Dataset**

Previous step results will be the seed for generation of additional new samples. Samples created will come from shocking and introducing noise on observed market samples.

4. **Step 4: Neural Network Calibration**

Once enough samples will be reached, Neural Network will be trained.

5. **Step 5: Out-of-Sample test**

We will test the Neural Network calibrated on real market out-of-sample data.

SABR is a good model to capture the volatility smile. SABR parameters are tradable parameters which operates a reduction of cardinality, hence they are of great interest for traders. Even if it presents some issues, which need to be solved such as the rapid degradation of the volatility prediction (more than 1%) for larger than 10y maturities ATM or larger products [2].

However the model is time-consuming to calibrate. With traditional methods, the calibration needs to be done every day. The goal is to push offline the calibration process where the calibration will only need to be done every month and SABR parameters will be obtained "instantaneously".

Neural Networks are excellent candidate for this goal, thanks to their ability to capture non-linearities, their instantaneous prediction once calibrated and their ability to give derivatives. But they need an important number of samples to be trained and as the market does not provide enough data, new data will be created.

## 4.1 Step 1: Data Presentation

The idea behind this project is to apply ANN in calibrating the SABR parameters for IR Swaptions in a particular EM market, South Africa Rand (ZAR). Deutsche Bank has allowed me to use time series of historical data - stretching back almost 2 years. For this project, we will be particularly interested for requesting yield curves, forward curves and implied volatilities of swaption.

This internship has been realized in Emerging Market department. It had been chosen to apply the SABR calibration to South Africa's market (ZAR currency). Traders consider it as the more stable market (even if market stability became relative with COVID-19) of emerging markets.

The challenges are:

- liquidity
- noisy data

It is important in all the project to recall South Africa is an emerging market with poor liquidity. This crucial point will have many consequences. It can't be assumed that the market is perfectly efficient, presents no arbitrage. Every standard assumptions in financial mathematics will have to be checked previously to see if it is not violated.

If the techniques provided in the following are successful, then it is our aim to extend the scope of this project to other Emerging Market such as Mexico (MXN currency), Russia (RUB), Israel (ILS) and Turkey (TRY).

All the code has been done in Python. Two important libraries for Neural Networks will be used: Tensorflow <sup>1</sup> and Keras <sup>2</sup>.

## 4.2 Step 2: Sequential calibration on time-series

This step has been realised thanks to the help of *Calibration of the SABR Model in Illiquid Market* (2005) [61]. It is important not to forget we are working on South African market which is illiquid. West did the study of SABR calibration on South African market too. More precisely, he studied the SAFEX (South African Futures EXchanges).

---

<sup>1</sup>[www.tensorflow.org](http://www.tensorflow.org)

<sup>2</sup><https://keras.io>

In the following we are using the notation convention in Hernandez. Let us first denote the SABR model with  $\mathbf{M}$  and an instrument theoretical quote by  $\mathbf{Q}$ :

$$\mathbf{Q}(\tau) = \mathbf{M}(\theta; \tau, \phi) \quad (4.1)$$

where:

- $\theta$  are the model parameters,  $(\alpha, \rho, \nu)$  here
- $\tau$  are the identifying properties such as list of tenors and maturities
- $\phi$  are the exogenous parameters, the forward rate matrix here

The model  $\mathbf{M}$  has  $n$  parameters, some of them have constraints:

$$\theta \in S \subseteq \mathbb{R}^n \quad (4.2)$$

As mentioned in 2.7, it needs to determine four parameters:  $(\alpha, \beta, \rho, \nu)$ .  $F_0 = f$  is observable directly in the market.

Thanks to trader's intuition, we fix  $\beta = 1$ .

To calibrate  $(\alpha, \rho, \nu)$ , we solve the following equation:

$$(\hat{\alpha}, \hat{\rho}, \hat{\nu}) = \arg \min_{(\alpha, \rho, \nu)} \sum_i d[\sigma^{market, ATM}, \sigma^{impl}(f_i, T_i, \alpha, \rho, \nu)] \quad (4.3)$$

where we loop on the different volatilities observed from the market with  $d(x, y) = (y - x)^2$  metric <sup>3</sup>.

The minimisation in Python is done thanks to the Method L-BFGS-B <sup>4</sup> which uses the L-BFGS-B algorithm very appropriated for bound constrained minimization problems. This method was developed in *L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization* (1997) [62] and *A limited memory algorithm for bound constrained optimization* (1995) [12].

As we were saying in Section 4.1, ZAR is illiquid market and the swaption matrices in the time series are not always very accurate. In order to denoise them, we were suggested by the desk to do the following: to replace some implied volatilities either by the historical volatility i.e. the realised volatility or either a constant.

The following two tables describe the structure of the calibration done in Step 4.2. It is worth remarking that all the calibration has happened with traditional optimisation methods and no ANN is present at this stage.

---

<sup>3</sup>This step was realised with advices of *SABR calibration in Python* (2016) [60].

<sup>4</sup>[www.docs.scipy.org/doc/scipy/reference/optimize.minimize-lbfgsb.html#optimize-minimize-lbfgsb](http://www.docs.scipy.org/doc/scipy/reference/optimize.minimize-lbfgsb.html#optimize-minimize-lbfgsb)

Parameter	Value
$(\alpha_0, \rho_0, \nu_0)$	(15%, 85%, 20%)
$(\alpha, \rho, \nu)$	$((0, \infty), (0, 1], (0, \infty))$
$\beta$	1
Method	'L-BFGS-B'
Tolerance	1E-16

Table 4.1: Step 2 parameters

The following table summarises inputs and outputs.

	Inputs	Outputs
For each day	<p style="text-align: center;"><u>Swaption</u></p> $\begin{pmatrix} \sigma_{T_1, M_1} & \cdots & \sigma_{T_1, M_m} \\ \vdots & \ddots & \vdots \\ \sigma_{T_t, M_1} & \cdots & \sigma_{T_t, M_m} \end{pmatrix}$ <p style="text-align: center;"><u>Forward matrix</u></p> $\begin{pmatrix} f_{T_1, M_1} & \cdots & f_{T_1, M_m} \\ \vdots & \ddots & \vdots \\ f_{T_t, M_1} & \cdots & f_{T_t, M_m} \end{pmatrix}$ <p style="text-align: center;"><u>Time List</u></p> <p style="text-align: center;">Maturities = <math>(M_1, \dots, M_m)</math> Tenors = <math>(T_1, \dots, T_t)</math></p>	<p style="text-align: center;"><u>Calibrated parameters</u></p> <p style="text-align: center;"><math>(\hat{\alpha}, \hat{\rho}, \hat{\nu}, \text{Error})</math></p>

Table 4.2: Step 2 summary

The stretch of time for this exercise, i.e. the time series extension of the data is from February 2019 to August 2020. So far, all the calibration has happened with traditional optimisation methods and no ANN was ever present.

The calibration problem is now a function of  $N$  parameters (the number of identifying properties  $\tau$ ) to  $n$  output (numbers of model parameters, 3 here plus the error).

$$\Theta : \mathbb{R}^N \mapsto S \in \mathbb{R}^n \tag{4.4}$$

We will see in Section 4.4 that in reality  $\Theta$  can be approximated (and it will be) via an ad hoc constructed ANN.



### 4.3 Step 3: Preparing Training Dataset

As mentioned previously, we have a small stretch of data. Even if we had time-series on longer time-windows, that would still not be sufficient for the purpose of training the Artificial Neural Networks. We need to generate additional samples. In the following, we will use Hernandez's notations.

We know a very good approximation of the inverse of  $\Theta$ , it is the asymptotic formula of the implied volatility 2.8. We use the model itself which will give us the training dataset.

$$\Theta^{-1}(\theta; \{\tau\}, \phi) \approx \mathbf{M}(\theta; \{\tau\}, \phi) = \{Q\} \quad (4.5)$$

where  $\{Q\}$  is the future inputs of the Neural Networks, it is the market quote of relevant instruments.

This idea allows to generate plenty of new samples by generating random parameters  $\theta$  and exogenous parameters  $\phi$  and keeping the structure between them.

This step is the core idea of Hernandez and of this project. Neural Networks need millions of samples to be trained. With one sample a day for SABR model, we would need 4000 years to reach this number.

The idea is to apply statistical transformation to reduce the noise and keep core information, then apply noise again. The main goal is not to shock data randomly but to keep structure of the data and the distribution of the features.

The steps are in summary the following:

1. Collect errors for each calibration instrument for each day
2. As parameters are positive, take the natural logarithm on the calibrated parameters
3. Rescale forward curves, parameters, and errors to have zero mean and variance 1
4. Apply dimensional reduction via PCA to forward curve, and keep parameters for given explained variance (99.5%)
5. Calculate covariance of rescaled log-parameters, PCA forward curve values, and errors
6. Generate random normally distributed vectors consistent with given covariance
7. Apply inverse transformations: rescale to original mean, variance, and dimensionality, and take exponential of parameters
8. Obtain implied volatility for all calibration instruments and apply random errors to results

The following table summarises inputs and outputs.

Inputs	Outputs
<p><b>On each day calibrated</b></p> <p><u>Forward matrix</u></p> $\begin{pmatrix} f_{T_1, M_1} & \cdots & f_{T_1, M_m} \\ \vdots & \ddots & \vdots \\ f_{T_t, M_1} & \cdots & f_{T_t, M_m} \end{pmatrix}$ <p><u>Calibrated parameters</u></p> $(\hat{\alpha}, \hat{\rho}, \hat{\nu}, \text{Error})$ <p><u>Time List</u></p> <p>Maturities = <math>(M_1, \dots, M_m)</math></p> <p>Tenors = <math>(T_1, \dots, T_t)</math></p> <p>number of samples desired</p>	<p><b>For number of samples desired</b></p> <p><u>Forward matrix</u></p> $\begin{pmatrix} f_{T_1, M_1} & \cdots & f_{T_1, M_m} \\ \vdots & \ddots & \vdots \\ f_{T_t, M_1} & \cdots & f_{T_t, M_m} \end{pmatrix}$ <p><u>Swaption</u></p> $\begin{pmatrix} \sigma_{T_1, M_1} & \cdots & \sigma_{T_1, M_m} \\ \vdots & \ddots & \vdots \\ \sigma_{T_t, M_1} & \cdots & \sigma_{T_t, M_m} \end{pmatrix}$ <p><u>Calibrated parameters</u></p> $(\hat{\alpha}, \hat{\rho}, \hat{\nu}, \text{Error})$

Table 4.3: Step 3 summary

Forward matrix in inputs and outputs are not the same. Outputs are shocked version of real market inputs.

Now, we look into more details each step.

### 4.3.1 Step 3.1

#### Collect errors for each calibration instrument for each day

We need to collect result from 4.2. For each day, we get the swaption implied volatilities matrix and swaption rates matrix for the same list of tenors and maturities. For each day, SABR parameters  $\alpha, \rho, \nu$  have been calibrated by minimising the error. We collect these calibrated parameters and these errors

### 4.3.2 Step 3.2

#### As parameters are positive, take the natural logarithm on the calibrated parameters

Lots of parameters collected in previous sections have values close to zero, with few volatilities. The application of log function enables to spread these values.

Unfortunately, with COVID-19,  $\nu$ , the volatility of volatility dived to zero at the start of the epidemic. We applied this transformation only on  $\alpha$  and  $\rho$ .

### 4.3.3 Step 3.3

#### Rescale forward curves, parameters, and errors to have zero mean and variance 1

It is important to recall South Africa is not a full efficient market.

Normally the forward rate of a contract with tenor  $T_a$  and maturity  $T_b - T_a$  is:

$$\frac{(1 + r_b)^{T_b}}{(1 + r_a)^{T_a}} - 1 \quad (4.6)$$

However, by looking at the South African forward rates for special tenors and maturities, this assumption is violated. Each forward curve, i.e for a fixed tenor and a series of maturities carry its own lot of information.

In Hernandez example, the yield curve is used. From the yield curve, every forward curves can be extrapolated.

However in this project, for each day, there will be the number of tenors of forward curves. For each tenors, the forward curves will be rescaled to have zero mean and error one.

The same data standardisation process will be applied to SABR parameters and to the error.

This standardisation is common for machine learning estimators <sup>5</sup>, <sup>6</sup>. All the features need to be transformed into standard normally distributed data, else data can be difficult to model. This standardisation forces data, which can be heterogeneous to behave in the same way.

Furthermore, numerous data science function take as default parameters, a feature with mean zero and variance one. To process them at the beginning decreases the risk of errors later.

Standardising data is crucial in Principal Component Analysis (PCA) because it projects your data onto orthogonal directions, where each direction maximises the variance ; and PCA is the next step.

### 4.3.4 Step 3.4

#### Apply dimensional reduction via PCA to forward curve, and keep parameters for given explained variance (99.5%)

The idea to generate new samples is to reduce noise and keep core data in the exogenous parameters, here the forward curves. Applying PCA on yield curves is common and has been studied in *Common Factors Affecting Bond Returns* (1991) [46] and in *Deconstructing the Yield Curve* (2019) [17].

---

<sup>5</sup>[communities.sas.com/t5/SAS-Communities-Library/To-standardize-data-or-not-to-standardize-data-that-is-the/ta-p/361726#](https://communities.sas.com/t5/SAS-Communities-Library/To-standardize-data-or-not-to-standardize-data-that-is-the/ta-p/361726#)

<sup>6</sup><https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html>

In a general way, data is made of the signal and the noise. The use of Principal Component Analysis allows to extract the signal, which reduces the dimensionality and cardinality of the dataset. It only keeps the features with the greatest explanatory power. The idea of PCA is to use the covariance matrix and project it onto a smaller subspace in term of dimensions, where all explanatory variables are orthogonal, hence deletes multicollinearity (repetition of information). Unfortunately, it is often hard to interpret the economic meaning of news axes.

However, on the yield curve it is possible. Three principal components only can be used to model the yield curve <sup>7</sup>.

The first component is a constant and expresses the long term interest rates.

The second is the slope, approximately the term premium.

The third is the curvature.

As verified on our data, these three components explained more than 99.5% of the information as desired in Hernandez procedure.

### 4.3.5 Step 3.5

#### Calculate covariance of rescaled log-parameters, PCA forward curve values, and errors

Magic operates at this step. We calculate covariance among model parameters  $\theta$ , and exogenous parameters,  $\phi$ , which are in a reduced space summarised by PCA eigenvalues. The covariance matrix preserves the structure and multicollinearity of the parameters.

### 4.3.6 Step 3.6

#### Generate random normally distributed vectors consistent with given covariance

Now we know and relies the structure of the parameters among themselves. We can generate new model parameters  $\theta$  and exogenous parameters  $\phi$ .

### 4.3.7 Step 3.7

#### Apply inverse transformations: rescale to original mean, variance, and dimensionality, and take exponential of parameters

We apply inverse transformation to project again our obtained values in the original space.

### 4.3.8 Step 3.8

#### Obtain implied volatility for all calibration instruments and apply random errors to results

We can apply  $\mathbf{M}$  to dates previously generated. Hence we have generated a new sample  $\mathbf{Q}$

---

<sup>7</sup>[www.towardsdatascience.com/applying-pca-to-the-yield-curve-4d2023e555b3](http://www.towardsdatascience.com/applying-pca-to-the-yield-curve-4d2023e555b3)

## 4.4 Step 4: Neural Networks Calibration

The use of Neural Networks requires a very precise, almost surgical tuning of parameters and model hyper-parameters. This tuning is sometimes called "black art". No rules are pre-established. It requires expertise and brute-force search. Help was found in *A neural network-based framework for financial model calibration* (2019) [47], in *An Artificial Neural Network Representation of the SABR Stochastic Volatility Model* (2018) [51], in *Machine learning SABR model of stochastic volatility with lookup table* (2020) [48]. Help was also found in *Practical Bayesian Optimization of Machine Learning Algorithms* (2012) [57], which idea is to use Gaussian process for sample and kernel to obtain "a good optimizer that can achieve expert-level performance".

Helps to code in Python was found in *Deep Learning with Python* (2017) [14].

The following table summarises inputs and outputs.

Inputs		Outputs
<b>For number of samples</b>		<u>NN parameters</u>  Weights and biases of the NN
<u>NN hyper-parameters</u>	number of hidden layers number of units activation function epoch batch size loss function	
<u>Data inputs</u>	<u>Swaption</u> $\begin{pmatrix} \sigma_{T_1, M_1} & \cdots & \sigma_{T_1, M_m} \\ \vdots & \ddots & \vdots \\ \sigma_{T_t, M_1} & \cdots & \sigma_{T_t, M_m} \end{pmatrix}$ <u>Forward matrix</u> $\begin{pmatrix} f_{T_1, M_1} & \cdots & f_{T_1, M_m} \\ \vdots & \ddots & \vdots \\ f_{T_t, M_1} & \cdots & f_{T_t, M_m} \end{pmatrix}$	
<u>Targets</u>	<u>Calibrated parameters</u> $(\hat{\alpha}, \hat{\rho}, \hat{\nu})$	

Table 4.4: Step 4 summary

### 4.4.1 Architecture choice

The choice of hyper-parameters is crucial for the result of the calibration. However, no standard and accepted method exist. It is a matter of mathematical art.

The input layer and output layer number of units is easy. It is determined by input and output's model. The first hidden layer is often half of the input layer.

A formula for the number of neurons was proposed in *Neural Network Design* (2014) [18].

$$N_h = \frac{N_s}{\alpha * (N_I + N_O)} \quad (4.7)$$

where:

- $N_h$ : optimal number of neurons
- $N_I$ : Input number of neurons
- $N_O$ : Output number of neurons
- $N_s$ : Output number of samples
- $\alpha$ : Scaling Factor between 2 and 10

Precious help found on StackExchange website forum <sup>8</sup> and TowardsDataScience website <sup>9</sup>.

We also proceeded to grid and random search. New studies proved that random search is more efficient for hyper-parameters optimisation as explained and illustrated in *Random search for hyper-parameter optimization* (2012) [5].

BERGSTRA AND BENGIO

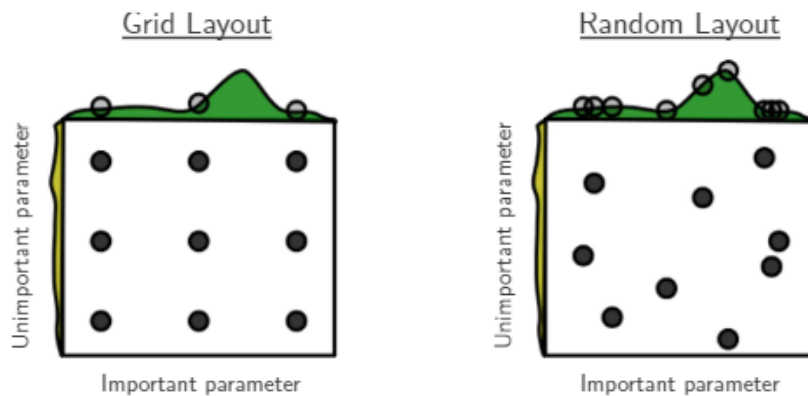


Figure 4.1: Grid vs Random search

<sup>8</sup>[www.stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layer-s-and-nodes-in-a-feedforward-neural-netw](http://www.stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layer-s-and-nodes-in-a-feedforward-neural-netw)

<sup>9</sup>[www.towardsdatascience.com/17-rules-of-thumb-for-building-a-neural-network-93356f9930af](http://www.towardsdatascience.com/17-rules-of-thumb-for-building-a-neural-network-93356f9930af)

## 4.4.2 Activation Function

We chose 'elu' as activation function, which means 'Exponential Linear Unit' <sup>10</sup>.

'elu' is a recent activation function which solves issues of 'ReLU' and improves activation functions which already tried to solve this issue such as 'Leaky ReLU' or 'PReLU'.

Some recalls about 'Relu', Rectified Linear Unit:

Relu is a very popular activation function. It is used in many machine learning applications. The output follows a simple structure, such as th derivative:

$$Relu(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{else} \end{cases} \quad (4.8)$$

$$Relu'(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{else} \end{cases} \quad (4.9)$$

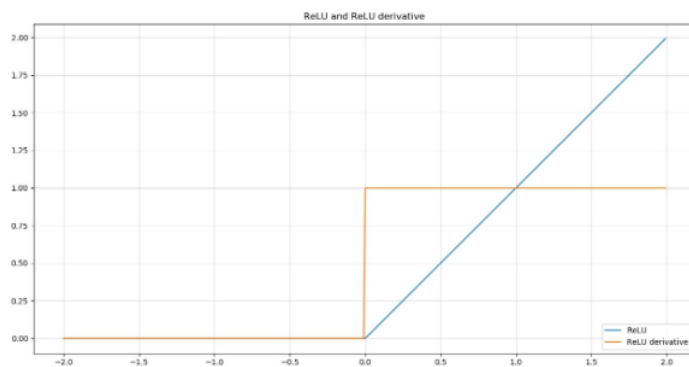


Figure 4.2: Relu graph

The advantage of this function is that thanks to the zero term, many neurons will be "deactivate".

An other advantage is that the gradient is easy, it is not sensitive to the *vanishing gradients problem* <sup>11</sup> (when activation function reduces the input into a smaller subspace, which can makes the gradient very small and backpropagation impossible, because it cancels the chain rule), which is either 0, either 1. This ease of computation of 'Relu' function ( $\max(x, 0)$  and 0 or 1 for the derivative) is the main reason of its popularity today.

Unfortunately, even if 'Relu' is very popular, it has some disadvantages.

Gradient is either 0 or 1, which can cause the dying ReLU problem studied in *ReLU and Initialization: Theory and Numerical Examples* (2019) [49]. The Dying Relu problem is when the entire layer is deactivated, because of one zero gradient in the chain of gradients. The neuron can't improve and is considered as 'dead'. If too many neurons are dead, the learning process will be very slow. However the mean

<sup>10</sup>[www.machinecurve.com/index.php/2019/12/09/how-to-use-elu-with-keras](http://www.machinecurve.com/index.php/2019/12/09/how-to-use-elu-with-keras)

<sup>11</sup>[www.towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484](http://www.towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484)

activation of any ReLU neuron is nonzero. Hence the leaning process is slowed down but not "killed", compared to activation functions that activates far from zero as studied in *Fast and Accurate Deep Network Learning by Exponential Linear Units* (2015) [15].

Hence new activation functions were created to solve this issue such as Leaky Relu or PRelu.

- Leaky Relu prevents dying Relu by making negative part really small but non zero by multiplying by a close to zero constant  $\alpha$

$$\text{LeakyRelu}(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{else} \end{cases} \quad (4.10)$$

- PReLU generalizes Leaky Rely by not specifying  $\alpha$  in advance, which means to make assumptions.  $\alpha$  is dependant of inputs and is trainable. It also avoids the dying ReLU problem.

However these two "upgraded" activation functions doesn't solve all issues. Clevert et al in *Fast and accurate deep network learning by exponential linear units*, (2015) [15] even tells that new issues were introduced. They solved the vanishing gradients problem and the dying ReLU problem, however they have no "noise-robust deactivation state" .

Hence the authors proposed the 'Elu': Exponential Linear Unit activation function <sup>12</sup>.

$$\text{Elu}(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha(e^x - 1) & \text{else} \end{cases} \quad (4.11)$$

This new activation function

- deletes the vanishing gradients issue and the dying ReLU problem.
- a quicker learning process
- it saturates to  $-\alpha$ , which makes it robust to noise

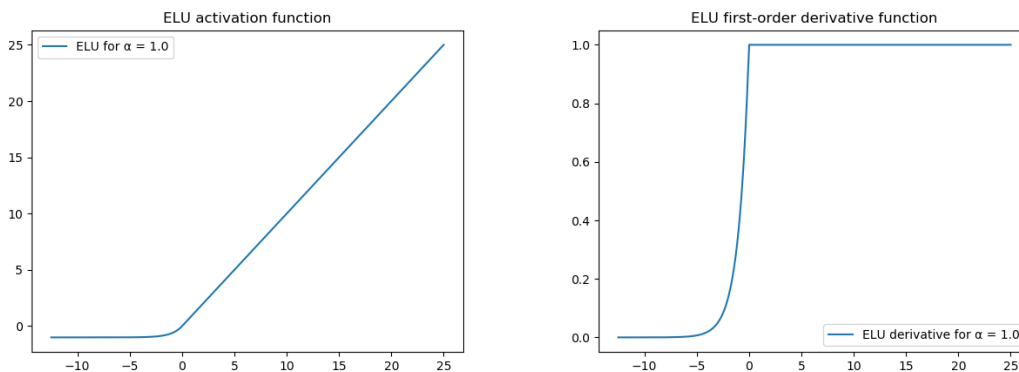


Figure 4.3: Elu Graphs

<sup>12</sup>[https://keras.io/api/layers/activation\\_layers/elu](https://keras.io/api/layers/activation_layers/elu)



### 4.4.3 Loss function

We needed a regression loss function <sup>13</sup> for our problem. The use of Mean Absolute Loss Function or Mean Square Loss function didn't change our parameters and hyper-parameters calibration results. We finally used Mean Square Loss function <sup>14</sup>.

### 4.4.4 Optimizer choice

For the Neural Network, Nadam optimizer <sup>15</sup> was used. It means Nesterov ADaptive Moment estimation. It is a mix of RMSprop <sup>16</sup> and Momentum Nesterov optimizers. The idea of RMSprop optimizer is to "Maintain a moving (discounted) average of the square of gradients" and "Divide the gradient by the root of this average". The idea of Nesterov Momentum optimizer is to use previous gradients to make the gradient descent smoother as explained in *Incorporating Nesterov Momentum into Adam* (2016) [21].

We chose Adam because it requires low memory needs and is efficient for little change in hyper-parameters. <sup>17</sup>

### 4.4.5 Epoch and Batch

We chose epoch of 100 and batch of 200 thanks to a manual hyper-parameters research.

### 4.4.6 Kernel Initializer

The Neural Network starts with some random weights and biases, it updates them iteratively by SGD with better values. However these weights and biases follow a statistical distribution.

The kernel initializer <sup>18</sup> will initialise these weights and biases with the chosen distribution, which should be an idea or an estimation of the real unknown distribution. The two main distributions are the uniform distribution and the normal distribution.

Due to the step 4.3, sample generation process, the Gaussian kernel will be chosen. The distribution will be  $\mathcal{N}\left(0, \frac{4}{n_{\text{input}} + n_{\text{output}}}\right)$ .

### 4.4.7 Early Stopping

When there is no more enough improvement at each iteration of the loss metric. It can be useful to stop the calibration process earlier <sup>19</sup>. We decided to stop training the Neural Network after 5 epochs where the metric was not improved by more than  $0.2E - 5$

---

<sup>13</sup><https://keras.io/api/losses>

<sup>14</sup><https://keras.io/api/losses/regression-losses/#mean-squared-error-function>

<sup>15</sup><https://keras.io/api/optimizers/Nadam>

<sup>16</sup><https://keras.io/api/optimizers/rmsprop>

<sup>17</sup>[www.dlology.com/blog/quick-notes-on-how-to-choose-optimizer-in-keras](http://www.dlology.com/blog/quick-notes-on-how-to-choose-optimizer-in-keras)

<sup>18</sup><https://keras.io/api/layers/initializers>

<sup>19</sup>[https://keras.io/api/callbacks/early\\_stopping](https://keras.io/api/callbacks/early_stopping)

### 4.4.8 Dynamic Learning Rate

The choice of the learning rate is crucial for the precision and time calibration of the model. A low learning rate will calibrate precisely but very slowly. A high learning rate calibrates quickly but risk to bounce indefinitely close to the minima and never reaches it.

The following Kaggle graph shows this issue.

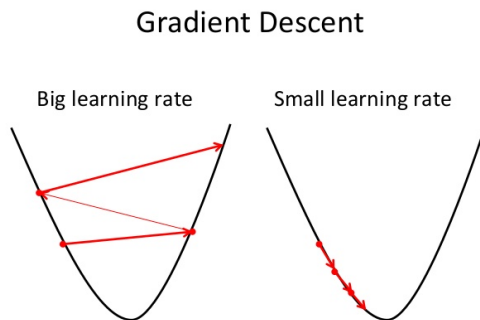


Figure 4.4: Learning Rate Illustration

To solve this issue and find a compromise, the idea is to have a dynamic learning rate <sup>20</sup>, starting from a high and decreasing it progressively.

## 4.5 Step 5: Out-of-Sample test

When considering the training of a model, it is important to consider the repeatability of the calibration. However, ANN are random, when a Neural Network with the exact same architecture is trained on the same data, we get different results.

This randomness is critical, it produces a unstable and unreliable model. It can cause issue when code is shared and results differ.

*But why is there randomness in artificial neural networks?*

Multiple reasons:

- Randomness in optimisation techniques like Stochastic Gradient Descent, which is stochastic and not deterministic as mentioned in the name
- Random initialisation of weights and biases
- Randomness in regularisation techniques

*How do we fix the randomness of an ANN?*

We need to fix the randomness, hence randomness will be predictable. The results will be the same hence at each training. To do that, we fixed the seed in keras <sup>21</sup> in order to get the same result every time.

<sup>20</sup>[https://keras.io/api/callbacks/reduce\\_lr\\_on\\_plateau](https://keras.io/api/callbacks/reduce_lr_on_plateau)

<sup>21</sup>[www.tensorflow.org/api-docs/python/tf/random/set\\_seed](http://www.tensorflow.org/api-docs/python/tf/random/set_seed)

# Chapter 5

## Results

### 5.1 Step 1: Data Presentation

The main issue in this step was to familiarise with Deutsche Bank database. Below we reproduce the most recent input data for our exercise.

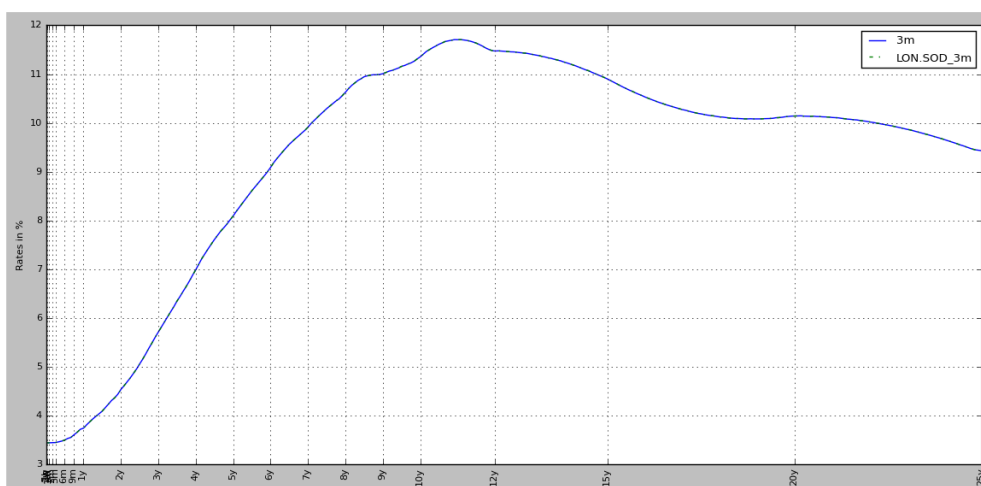


Figure 5.1: ZAR yield curve on 2<sup>nd</sup> of September 2020

		Tenors													
		1y	2y	3y	4y	5y	6y	7y	8y	9y	10y	15y	20y	25y	30y
Maturities	1m	3.53	3.80	4.19	4.68	5.19	5.66	6.10	6.49	6.83	7.12	7.95	8.19	8.29	8.32
	2m	3.55	3.84	4.25	4.75	5.26	5.74	6.17	6.56	6.90	7.18	7.99	8.23	8.33	8.35
	3m	3.58	3.89	4.32	4.82	5.34	5.81	6.25	6.63	6.97	7.24	8.04	8.27	8.37	8.39
	6m	3.69	4.05	4.53	5.06	5.57	6.04	6.47	6.85	7.17	7.43	8.18	8.39	8.48	8.50
	9m	3.84	4.25	4.76	5.30	5.82	6.28	6.70	7.07	7.37	7.63	8.33	8.52	8.60	8.61
	1y	4.02	4.47	5.01	5.57	6.07	6.53	6.94	7.29	7.58	7.83	8.48	8.65	8.72	8.73
	18m	4.42	4.97	5.55	6.10	6.59	7.03	7.41	7.73	8.00	8.23	8.77	8.91	8.96	8.96
	2y	4.94	5.54	6.13	6.66	7.13	7.54	7.89	8.18	8.42	8.62	9.07	9.17	9.20	9.19
	3y	6.19	6.80	7.31	7.78	8.18	8.53	8.80	9.04	9.22	9.36	9.62	9.67	9.65	9.63
	4y	7.44	7.94	8.39	8.78	9.12	9.37	9.60	9.77	9.89	9.97	10.08	10.08	10.02	9.99
	5y	8.47	8.93	9.31	9.64	9.88	10.09	10.24	10.34	10.41	10.45	10.44	10.39	10.30	10.26
	7y	10.19	10.50	10.68	10.86	10.99	11.04	11.07	11.08	11.05	11.02	10.86	10.73	10.62	10.57
	10y	11.53	11.59	11.55	11.50	11.44	11.35	11.26	11.17	11.10	11.04	10.83	10.64	10.55	10.50
	15y	10.76	10.60	10.48	10.40	10.35	10.32	10.30	10.27	10.24	10.19	9.95	9.88	9.84	9.82
	20y	10.14	10.11	10.07	10.01	9.94	9.86	9.78	9.71	9.63	9.56	9.50	9.48	9.47	9.46
30y	9.29	9.29	9.29	9.29	9.29	9.29	9.29	9.29	9.29	9.29	9.29	9.29	9.29	9.29	

Table 5.1: South African Forward rate in percentage on 2<sup>nd</sup> of September 2020

We observe that forward rates for maturities of 30 years are all the same, which means these forwards are illiquid and no good quote is available in the market. Furthermore, rates increase with the tenor and forward rate are in accordance with the yield curve, rates increase up to 15 years maturity and then decrease.

		Tenors													
		1y	2y	3y	4y	5y	6y	7y	8y	9y	10y	15y	20y	25y	30y
Maturities	1m	9.82	15.49	17.58	19.67	21.76	22.05	22.33	22.62	22.91	23.19	24.25	25.30	25.30	25.30
	2m	10.02	15.83	17.91	19.99	22.07	22.40	22.73	23.06	23.39	23.72	24.77	25.82	25.82	25.82
	3m	10.20	16.17	18.23	20.30	22.36	22.73	23.11	23.49	23.86	24.24	25.28	26.32	26.32	26.32
	6m	11.18	18.05	19.50	20.95	22.40	22.81	23.22	23.63	24.04	24.45	25.24	26.02	26.02	26.02
	9m	12.70	18.84	20.04	21.24	22.44	22.83	23.22	23.61	23.99	24.38	25.06	25.74	25.74	25.74
	1y	14.21	19.63	20.58	21.53	22.48	22.85	23.21	23.58	23.94	24.31	24.89	25.47	25.47	25.47
	18m	16.61	21.32	22.14	22.95	23.77	24.13	24.49	24.85	25.22	25.58	26.01	26.44	26.44	26.44
	2y	19.06	22.78	23.57	24.35	25.14	25.39	25.65	25.90	26.16	26.41	26.78	27.15	27.15	27.15
	3y	24.60	25.88	26.36	26.84	27.32	27.42	27.52	27.62	27.72	27.82	28.04	28.27	28.27	28.27
	4y	25.25	26.44	26.81	27.18	27.56	27.61	27.66	27.71	27.76	27.81	27.94	28.08	28.08	28.08
	5y	25.90	27.00	27.27	27.53	27.79	27.79	27.79	27.80	27.80	27.80	27.84	27.89	27.89	27.89
	7y	25.20	26.12	26.09	26.06	26.03	26.04	26.06	26.08	26.09	26.11	26.05	25.99	25.99	25.99
	10y	24.16	24.78	24.32	23.85	23.38	23.42	23.46	23.50	23.53	23.57	23.36	23.14	23.14	23.14
	15y	24.16	24.78	24.32	23.85	23.38	23.42	23.46	23.50	23.53	23.57	23.36	23.14	23.14	23.14
	20y	24.16	24.78	24.32	23.85	23.38	23.42	23.46	23.50	23.53	23.57	23.36	23.14	23.14	23.14
	30y	24.16	24.78	24.32	23.85	23.38	23.42	23.46	23.50	23.53	23.57	23.36	23.14	23.14	23.14

Table 5.2: Black's implied volatility in percentage of South African at-the-money swaptions on 2<sup>nd</sup> of September 2020

We observe that implied volatility for maturities longer than 10 years maturity are all the same, which means these swaption are illiquid and no good quote is available in the market.

Furthermore, forward rate and implied volatilities are in accordance with the yield curve, volatilities increase with the tenor and increase up to 10 years maturity and then decrease.

## 5.2 Step 2: Sequential calibration on time-series

We will now look at this step on a particular date: 5<sup>th</sup> of March 2020. Equation 4.3 is minimised for  $\alpha = 17.82\%$  ,  $\rho = 85.62\%$  and  $\nu = 26.36\%$ .

The Black’s implied volatilities this day for at-the-money swaptions are:

		Tenors													
		1y	2y	3y	4y	5y	6y	7y	8y	9y	10y	15y	20y	25y	30y
Maturities	1m	12.93	14.86	15.67	16.48	17.28	17.28	17.27	17.26	17.26	17.25	16.16	15.06	15.06	15.06
	2m	12.64	14.84	15.70	16.57	17.43	17.45	17.47	17.49	17.52	17.54	17.00	16.46	16.46	16.46
	3m	12.35	14.82	15.74	16.66	17.58	17.63	17.68	17.73	17.78	17.84	17.87	17.90	17.90	17.90
	6m	12.61	15.95	16.49	17.02	17.56	17.64	17.73	17.81	17.89	17.98	18.06	18.15	18.15	18.15
	9m	13.98	16.19	16.65	17.11	17.57	17.66	17.75	17.84	17.93	18.02	18.15	18.28	18.28	18.28
	1y	15.33	16.43	16.82	17.20	17.58	17.68	17.78	17.87	17.97	18.07	18.24	18.40	18.40	18.40
	18m	16.36	16.70	16.95	17.19	17.44	17.64	17.83	18.03	18.23	18.42	18.46	18.49	18.49	18.49
	2y	17.13	17.36	17.52	17.67	17.82	17.95	18.07	18.19	18.32	18.44	18.64	18.84	18.84	18.84
	3y	18.19	18.59	18.78	18.98	19.18	19.29	19.41	19.53	19.65	19.77	19.67	19.56	19.56	19.56
	4y	19.21	19.21	19.40	19.58	19.77	19.84	19.91	19.97	20.04	20.11	19.97	19.82	19.82	19.82
	5y	20.21	19.82	20.00	20.19	20.37	20.38	20.40	20.41	20.43	20.44	20.26	20.08	20.08	20.08
	7y	21.41	20.75	20.68	20.61	20.54	20.57	20.60	20.63	20.66	20.69	20.47	20.26	20.26	20.26
	10y	23.20	22.15	21.70	21.25	20.80	20.85	20.90	20.95	21.00	21.05	20.79	20.52	20.52	20.52
	15y	23.20	22.15	21.70	21.25	20.80	20.85	20.90	20.95	21.00	21.05	20.79	20.52	20.52	20.52
	20y	23.20	22.15	21.70	21.25	20.80	20.85	20.90	20.95	21.00	21.05	20.79	20.52	20.52	20.52
	30y	23.20	22.15	21.70	21.25	20.80	20.85	20.90	20.95	21.00	21.05	20.79	20.52	20.52	20.52

Table 5.3: Black’s implied volatility in percentage of South African at-the-money swaptions on 5<sup>th</sup> of March 2020

For this day, implied volatility has the same characteristic than previously. Implied volatility increase with tenors and with maturity. No quotes are available for more than 10 years maturity due to lack of liquidity of these products.

Hagan’s volatility is the same for a fixed tenor, which is normal because we fixed  $\beta = 1$ , hence Hagan’s implied volatility formula 2.9 has no more dependency in  $F_0 = f$ .

The Hagan’s implied volatilities this day for at-the-money swaptions and previously mentioned SABR parameters is:

	Maturities															
	1m	2m	3m	6m	9m	1y	18m	2y	3y	4y	5y	7y	10y	20y	25y	30y
Hagan	17.84	17.85	17.86	17.91	17.95	17.99	18.08	18.16	18.33	18.50	18.67	19.00	19.51	20.36	21.20	22.89

Table 5.4: Hagan’s implied volatility in percentage of South African at-the-money swaptions on 5<sup>th</sup> of March 2020

The RMSE: Root Mean Square Error for this day is 1.45%. It is some days before the lockdown.

The following figure gives results on the parameters calibration in time:

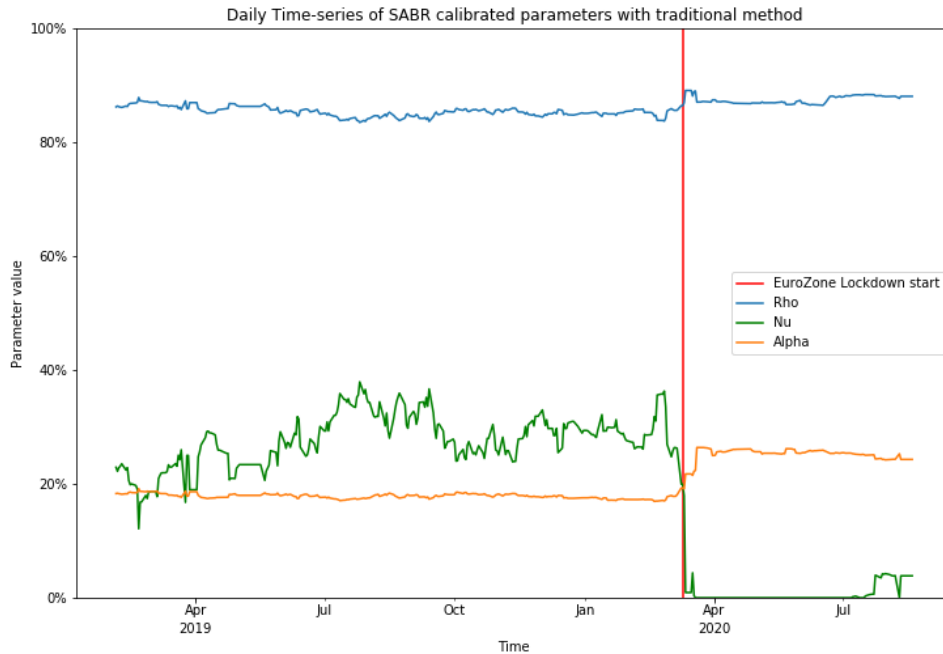


Figure 5.2: Time-series of SABR calibrated parameters

We can observe from this calibration:

- a stable  $\alpha$ ,
- a slightly increasing  $\rho$  at the start of COVID-19, which is normal, because in period of crisis, asset correlation tends to increase
- a stable  $\nu$  up to the COVID-19, where it shift from a stochastic volatility model to a constant volatility model

These parameters are the ones that minimise equation 4.3.

The following figure gives RMSE of implied volatility with SABR calibrated parameters.

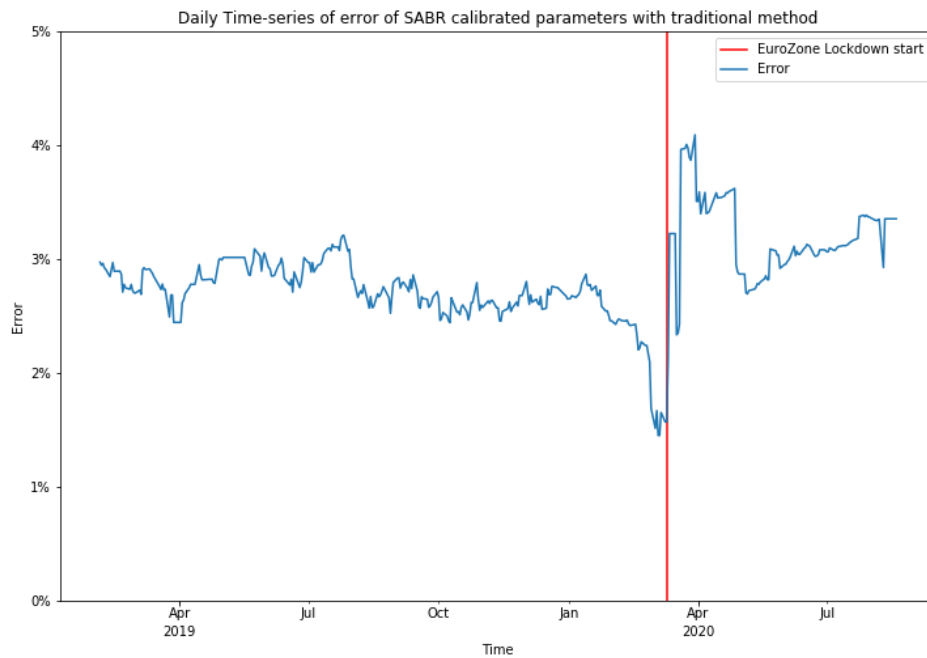


Figure 5.3: Time-series of error of SABR calibrated parameters

We can observe a net increase in the error at the beginning of the lockdown due to COVID-19.

### 5.3 Step 3: Preparing Training Dataset

In his paper, Hernandez generates 150,000 samples, however Hernandez Neural Network has 200 inputs: 154 swaption implied volatilities (12 maturities  $\times$  13 tenors) and 44 points on yield curves and 2 outputs.

In our project, we have 448 inputs: a matrix of 224 swaption's implied volatility (16 maturities  $\times$  14 tenors) and the forward matrix corresponding to these tenors and maturities and four outputs: 3 parameters ( $\alpha, \rho, \nu$ ) and the error of calibration.

The number of parameters to train in a Neural Network increases exponentially, hence we will generate one million sample compared to 150,000 Hernandez's one. We try first with only 150,000 samples, however the loss function when we calibrated the Neural Network was not stable through epoch.

Thanks to this step, we generated one million new samples. It is the longest step in time of this project. It takes 4 hours to generate 100,000 samples and one day and a half for one million samples.

We chose a learning ratio of 30%, so 700,000 samples were used to train the Neural Network and 300,000 to validate it.

This length of generation: 0.15 second per sample is mainly due to step 4.3.6. We generate a random normal variable thanks the covariance matrix. But we generate a 46 dimension vector, because we have 4 parameters ( $\alpha, \rho, \nu, \text{error}$ ) and three eigenvalues from the PCA times the number of time-series of forward curves, i.e. the number of tenors: 14. Hence the vector dimension is  $4 + 3 \times 14 = 46$ .



### 5.4 Step 4: Neural Networks calibration

Thanks to equation 4.7, we get with a scaling factor of 10:

$$N_h = \frac{1,000,000}{5 \times (448 + 4)} \approx 200 \tag{5.1}$$

It helped us to fix the first layer to 200 neurons, which was in accordance with rule that first layer must be less than half of the input layer.

The second hidden layer was one out of fourth of the first hidden layer and 10 times the output layer. 50 neurons was a good compromise.

These configuration, with two hidden layers gave the best results.

We reached the following architecture for our Neural Network.

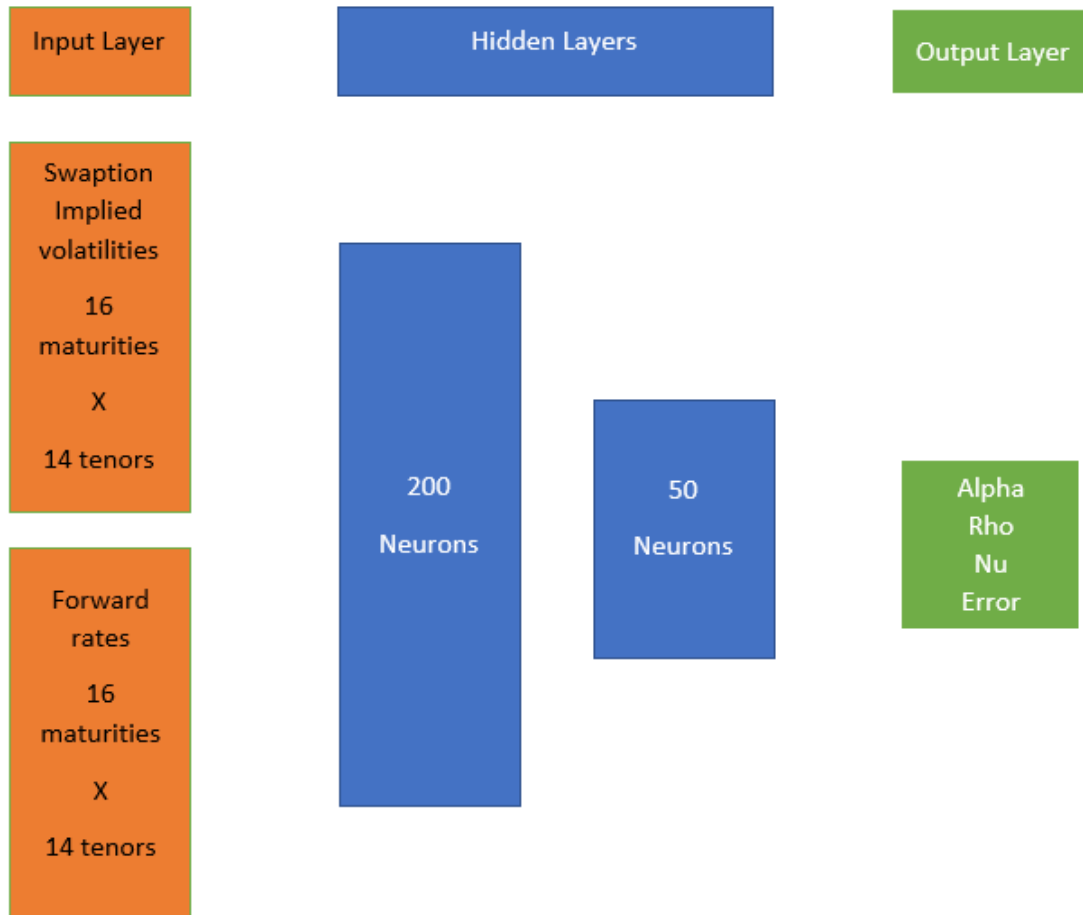


Figure 5.4: Step 4 Neural Network

Parameter	Value
Activation Function	elu
Loss Function	Mean Square Loss
Optimiser	Adam
Epoch	100
Batch size	200
Kernel Initialisation	Gaussian distribution
Early Stop	5 epochs without error 0.2E-5 improvement
Dynamic Learning rate	Keras default
Training Ratio	30%

Table 5.5: Step 4 parameters

The final error for this Neural Network realised on the validation dataset is 0.18%, where the error is:

$$error = \frac{1}{3 \times n} \times \sum_{i=1}^n [|\alpha_i - \hat{\alpha}_i| + |\rho_i - \hat{\rho}_i| + |\nu_i - \hat{\nu}_i|] \quad (5.2)$$

with  $n = 300000$ .

The final Mean Square Loss on the training dataset is 0.00082%. 100 epochs were realised on the training data of  $70\% \times 1,000,000 = 700,000$  samples. Each batch has a size of 200, hence each epoch will realise 3,500 iterations.

We are going to look results first on time-windows from February 2019 to August 2020. The Neural Network has not been trained with data from step 4.2, but with data from step 4.3 which have been generated thanks to data from step 4.2.

We now look at this step again on: 5<sup>th</sup> of March 2020.

	Parameters		
	alpha	rho	nu
Traditional Method	17.82	85.62	26.36
NN method	17.72	91.58	3.76

Table 5.6: SABR Parameter values for both methods on an in-sample date: 5<sup>th</sup> of March 2020

The Hagan’s implied volatilities this day for at-the-money swaptions and previously mentioned SABR parameters is:

	Maturities															RMSE	
	1m	2m	3m	6m	9m	1y	18m	2y	3y	4y	5y	7y	10y	20y	25y		30y
Hagan	17.84	17.85	17.86	17.91	17.95	17.99	18.08	18.16	18.33	18.50	18.67	19.00	19.51	20.36	21.20	22.89	1.45
Hagan NN	17.72	17.72	17.73	17.73	17.74	17.75	17.76	17.77	17.80	17.83	17.85	17.91	17.98	18.12	18.25	18.51	2.12

Table 5.7: Comparison of Hagan/Hagan NN implied volatility in percentage of South African at-the-money swaptions on a in-sample date: 5<sup>th</sup> of March 2020

## 5.5 Step 5: Out-Sample test

We now look at this step again on: 2<sup>nd</sup> of September 2020.

	Parameters		
	alpha	rho	nu
Traditional Method	23.92	88.05	1.56
NN method	22.51	89.12	10.15

Table 5.8: SABR Parameter values for both methods on an out-of-sample date: 2<sup>nd</sup> of September 2020

	Maturities															RMSE	
	1m	2m	3m	6m	9m	1y	18m	2y	3y	4y	5y	7y	10y	20y	25y		30y
Hagan	23.92	23.92	23.92	23.92	23.93	23.93	23.94	23.95	23.97	23.99	24.01	24.05	24.11	24.21	24.31	24.50	3.25
Hagan NN	22.52	22.53	22.54	22.57	22.59	22.62	22.68	22.73	22.84	22.95	23.07	23.29	23.62	24.17	24.73	25.84	3.44

Table 5.9: Comparison of Hagan/Hagan NN implied volatility in percentage of South African at-the-money swaptions on a out-of-sample date: 2<sup>nd</sup> of September 2020

The following figure gives results on the parameters calibration with Neural Networks in time:

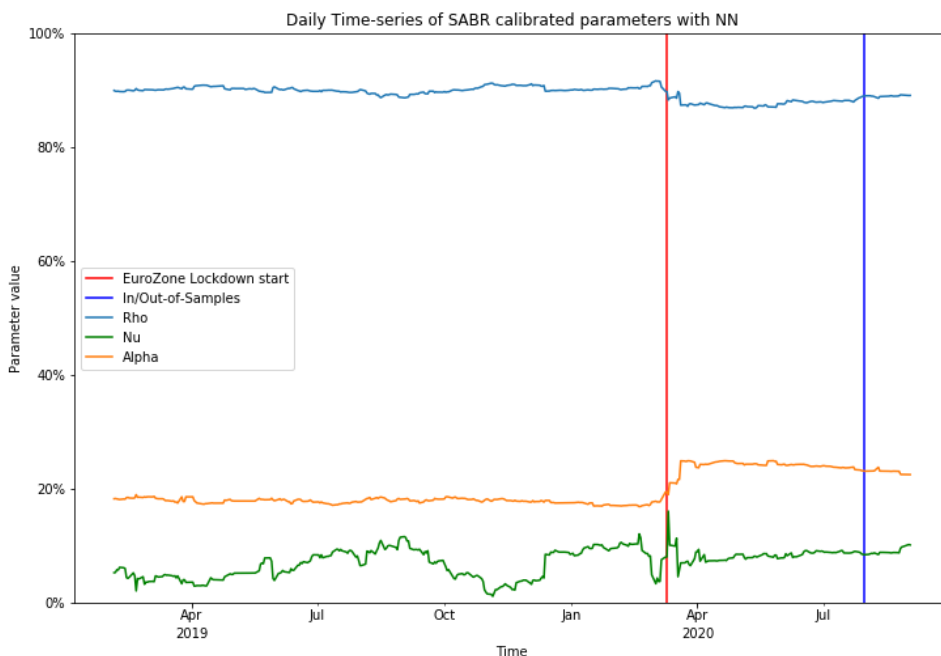


Figure 5.5: Time-series of SABR calibrated parameters with Neural Networks

The following figure gives RMSE of implied volatility with SABR parameters given by Neural Networks.

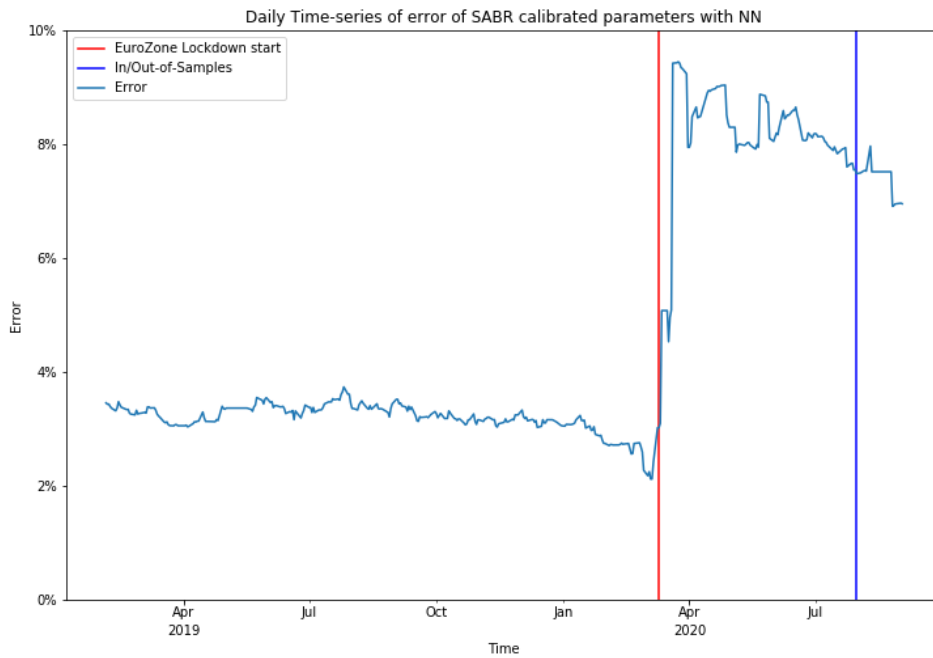


Figure 5.6: Time-series of error of SABR calibrated parameters with Neural Networks

We can see on figures 5.5 and 5.6 a sharp blue line which is the distinction between the in-sample and out-of-sample data.

In next section, we discuss these results.

## 5.6 Discussions

To judge a model, we need to look time needed and difficulty of calibration, robustness of the model and time to deliver outputs.

### 5.6.1 Time to deliver Output

This goal was fully reached. With traditional SABR method calibration, 30 seconds were needed to obtain SABR parameters. With Neural Network, SABR parameters are given instantaneously.

### 5.6.2 Time to calibrate the model

With traditional method, 30 seconds times the number of days (on which the calibration was done) were needed to get SABR parameters. With Neural Networks, the length is determined by Step 4.3: Preparing Training Data-set, which took for one million sample one day and a half. However, each sample generation is independent, this time can be reduced thanks to parallelisation. Furthermore, it mustn't be forgotten that this step will need to be conducted every one or two months, depending on the degradation of the inline calibration. As this will be offline, its cost will not be significant in any way.

### 5.6.3 Robustness

We noticed similar results in SABR parameters and errors between traditional method and NN method before COVID-19.

SABR parameter and error given by Neural Network are very stable on the out-of-sample data.

$\alpha$  and  $\rho$  values are conclusive in time across both models.

$\nu$  is not conclusive among models, however  $\nu = 0$  is surprising in standard method, furthermore it is the lower terminal of constraint, so this value must be wrong.  $\nu$  in NN method is more stable and seems more realistic. Neural Networks enable to smooth regime changes.

### 5.6.4 Improvement possibilities

Some improvement possible:

- Due to time constraints to write this thesis, higher volatility on the market because of COVID-19, the choice was made in all this project to fix  $\beta = 1$ , however this assumption is debatable. This exercise will be pursued by calibrating also on  $\beta$ .
- More data would have been relevant to look further the out-of-sample dataset. However South-African data at Deutsche Bank were not available easily for longer period of time.

- COVID-19 created lost of instability, particularly in Emerging Market, it would be interesting to realise this study in the past, with no COVID-19.
- When minimising the error in Step 4.2, with traditional method and in Step 4.4, with Neural Network, it could be interesting to take the Net Present Value of the square of the difference between result and target. Discounting the error by the tenor will penalise more closer tenor. Longer tenor are not liquid, so even the observed market value can be unprecise.
- In Step 4.3: Preparing Training Dataset, new samples were generated by introducing sample found in Step 4.2: Sequential Calibration on time-series. Some value of  $\nu$  in Step 4.2 were equal to zero during the lockdown. When creating new sample in step 4.3, gaussian random noise was introduced creating sample with  $\nu < 0$ . The NN in Step 4.4: Neural Network calibration tried to minimise the error between zero  $\nu$  and negative  $\nu$ .  $\nu$  created in Step 4.4 should have floored to zero.

# Chapter 6

## Hedging

“The banks need to hedge their exposures, and they do a significant amount of this in the exchange market. Furthermore, as is usual, the relevant models of the skew, which will be applied equally to over-the-counter products as well as exchange traded products, will be parameterized via exchange traded information. Thus it is necessary to have a robust model of the derivative skew for mark to market and hedging of positions” according to West [61].

”Another benefit, ..., is that neural networks are fully differentiable, and can therefore provide simple access to parameter sensitivities and hedging” tells Eugenio Martin in his thesis [23].

*How Neural Networks can be used for hedging?*

This chapter will be written thanks to *Hedging under SABR Model* (2006) [4] (which is an improvement of Hagan’s work by Barlett), *Bartlett’s Delta in the SABR Model* (2019) [34] (which is Hagan answer paper to Barlett’s work about his own work, sort of mathematical version of a tennis game).

This chapter will also use specific papers for *Hedging through Neural Networks such as Deep Hedging* (2018) [11], *Neural networks for option pricing and hedging: a literature review* (2019) [54], *A Nonparametric Approach to Pricing and Hedging Derivative Securities Via Learning Networks* (2001) [41], *Hedging with Neural Networks* (2020) [55].

### 6.1 Hedging under SABR model

We consider an option with forward rate  $f$ , strike  $K$  and expiry  $T$ . The value of this option under Hagan’s model is a modified version of Black formula:

$$V = B(f, K, \sigma_B(K; f, \alpha, T), T) \quad (6.1)$$

$B(f, K, \sigma, T)$  is the Black formula and  $\sigma_B(K; f, \alpha, T)$  is the SABR implied volatility.

It is important to notice implied volatility depend of strike,  $\alpha$  and time before expiration. Furthermore  $\alpha$  and  $f$  are correlated. All Greeks need to be calculated again to have a precise hedging.

### 6.1.1 Hagan's formula

#### Delta term

Initially Hagan calculated the new delta by shifting  $f$  and keeping  $\alpha$  constant:

$$\begin{aligned} f &\rightarrow f + \Delta f \\ \alpha &\rightarrow \alpha \end{aligned} \quad (6.2)$$

which gives the new option value:

$$\Delta V = \left\{ \frac{\partial B}{\partial f} + \frac{\partial B}{\partial \sigma} \frac{\partial \sigma}{\partial f} \right\} \Delta f \quad (6.3)$$

and the new option delta:

$$\Delta = \frac{\partial B}{\partial f} + \frac{\partial B}{\partial \sigma} \frac{\partial \sigma}{\partial f} \quad (6.4)$$

We recognise the first term which is the Black formula term, which is completed by a change in implied volatility with respect to the underlying.

#### Vega term

Hagan considered  $f$  constant and  $\alpha$  shifted:

$$\begin{aligned} f &\rightarrow f \\ \alpha &\rightarrow \alpha + \Delta \alpha \end{aligned} \quad (6.5)$$

which leads to the new vega:

$$\Lambda = \frac{\partial B}{\partial \sigma} \frac{\partial \sigma}{\partial \alpha} \quad (6.6)$$

We recognise the classic Black term updated by the implied volatility dynamic.

### 6.1.2 Barlett's formula

Barlett's idea is to better consider change in  $\alpha$  and  $f$  which are not uncorrelated as assumed by Hagan. Barlett proposed the following dynamic:

$$\begin{aligned} f &\rightarrow f + \Delta f \\ \alpha &\rightarrow \alpha + \delta_f \alpha \end{aligned} \quad (6.7)$$

where  $\delta_f \alpha$  is how much  $\alpha$  change in average due to the change of the underlying  $f$ .

To make differentiation work easier, Barlett gets the idea to "de-correlate" the dynamic by writing it with independent Brownian motions:

$$\begin{aligned} df_t &= \alpha_t f_t^\beta dW_t \\ d\alpha_t &= v\alpha_t \left( \rho dW_t + \sqrt{1 - \rho^2} dZ_t \right) \end{aligned} \quad (6.8)$$

where  $W_t$  and  $Z_t$  are two independent brownian motions. It allows to easily get  $d\alpha_t$ :

$$\delta_f \alpha = \frac{\rho v}{f^\beta} \Delta f \quad (6.9)$$



We can now calculate new value's term:

$$\Delta V = \left[ \frac{\partial B}{\partial f} + \frac{\partial B}{\partial \sigma} \left( \frac{\partial \sigma}{\partial f} + \frac{\partial \sigma}{\partial \alpha} \frac{\rho v}{f^\beta} \right) \right] \Delta f \quad (6.10)$$

### New Delta

Hence Barlett's delta is given by:

$$\Delta = \frac{\partial B}{\partial f} + \frac{\partial B}{\partial \sigma} \left( \frac{\partial \sigma}{\partial f} + \frac{\partial \sigma}{\partial \alpha} \frac{\rho v}{f^\beta} \right) \quad (6.11)$$

The new term  $\frac{\partial B}{\partial \sigma} \frac{\partial \sigma}{\partial \alpha} \frac{\rho v}{f^\beta}$  in Barlett's delta compared to Hagan's delta is  $\frac{\rho v}{f^\beta}$  multiplied by Hagan's vega. If the portfolio is vega hedged, this additional term is zero, hence Hagan's delta and vega are hedged, hence Barlett's delta is also hedged.

### New vega

Again, Barlett proposed a new dynamic:

$$\begin{aligned} f &\rightarrow f + \delta_\alpha f \\ \alpha &\rightarrow \alpha + \Delta \alpha \end{aligned} \quad (6.12)$$

where  $\delta_\alpha f$  is how much  $f$  change in average due to the change of the underlying  $\alpha$ .

Hence the new option's value is:

$$\Delta V = \frac{\partial B}{\partial \sigma} \left( \frac{\partial \sigma}{\partial \alpha} + \frac{\partial \sigma}{\partial f} \frac{\rho f^\beta}{v} \right) \Delta \alpha \quad (6.13)$$

and the new vega is:

$$\Lambda = \frac{\partial B}{\partial \sigma} \left( \frac{\partial \sigma}{\partial \alpha} + \frac{\partial \sigma}{\partial f} \frac{\rho f^\beta}{v} \right) \quad (6.14)$$

### Example

Empirical Example with Barlett's formula gives better results to hedge a portfolio as proved in *Hedging under SABR Model*, part 4, [4].

Hagan replied 13 years later to Barlett's paper, i.e last year by giving a mathematical rigorous justification. He also studied time decay, gamma, vanna and volga. More details in *Barlett's delta in the SABR model* by Hagan and Lesniewski [34].

## 6.2 Hedging with Neural Networks

A Neural Network realised affine transformation at each layer composed by an activation function. The derivative of an affine function is immediate, such as the derivative of 'elu' function.

Hence immediately, the Neural Network can give the derivative of  $\alpha, \rho, \nu$  with respect to the forward rate or implied volatility. The function is pre-implemented in keras: `gradients`<sup>1</sup>.

Unfortunately, due to lack of time, this technic was not tested at the time this thesis was written. Furthermore, literature on the subject is quite poor. It is very difficult to get a Neural Network doing parameters calibration and hedging in the same time. There is no free lunch.

However, literature about Neural Networks and hedging is rich, but the goal of the Neural Network since its inception is to hedge, which will give a different architecture. For example, inputs will include contain additional information such as trading signals, news analytics, or past hedging decisions.

---

<sup>1</sup>[www.tensorflow.org/api\\_docs/python/tf/keras/backend/gradients](http://www.tensorflow.org/api_docs/python/tf/keras/backend/gradients)

# Conclusion

The goal of this thesis was to apply a Neural Network Approach to obtain in a faster and more robust way SABR parameters. To answer this question, a million of samples have been created by introducing noise in real market data, in keeping structure and distribution of original market data. The trained Neural Network tried to map Black's implied swaption volatilities and corresponding forward rates for different tenors and maturities.

Despite COVID-19 and lack of data for South African market, results are promising. Neural Networks allow an instantaneous delivery of SABR parameters. Neural Networks deliver stable parameter in time and are less subject to market turmoil than the traditional SABR calibration method. Furthermore Neural Network error is similar to the SABR calibration traditional method.

The future goal of this thesis is to apply this method to other Emerging Markets such as Brazil, Russia or Turkey, where similar issues of liquidity and highly noisy data will present similar challenges.

As a concluding note, two additional remarks are in place.

On the one hand, an alternative method to generate new samples can be the use of Variational Auto Encoders (VAE) <sup>2</sup>. We could generate in our model new samples through already existing samples and VAE are a very powerful tool for data generation thanks to existing data.

On the other hand, SABR model assume the dynamic of a single forward rate, which is one of its limitation compared to Libor Market Model. In Remark 11.4.1 of *Interest Rate Models — Theory and Practice: With Smile, Inflation and Credit* [10], Brigo and Mercurio mention "Hagan, Kumar, Lesniewski and Woodward postulate the evolution of a single forward asset, and show that their model accommodates quite well implied volatilities for a single maturity. [...] In a LIBOR market model, in fact, not only has one to specify the joint evolution of forward rates under a common measure, but also to clarify the relations among the volatility dynamics of each forward rate.". Neural Networks are excellent candidate to capture the joint distribution of different forward rate. SABR/LMM model is studied in *LIBOR market model with SABR style stochastic volatility* by Hagan and Lesniewski [31].

---

<sup>2</sup>[www.towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf](http://www.towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf)

# Appendix

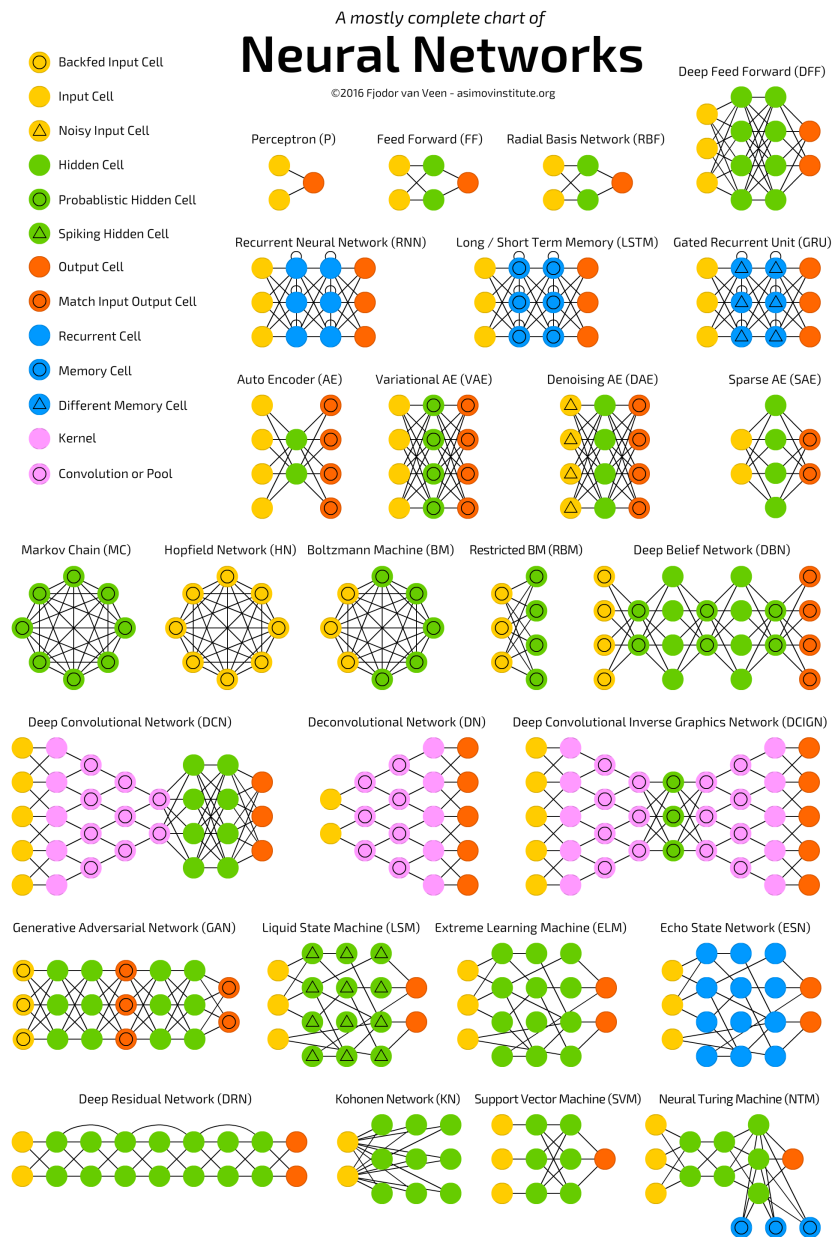


Figure 1: The mostly complete chart of Neural Networks










Activation	Definition	Derivative	Range	Smoothness
 Identity (Id)	$g(x) = x$	$g'(x) = 1$	$\mathbb{R}$	$C^\infty$
 Sigmoid (Logistic, $\sigma$ )	$g(x) = \frac{1}{1 + e^{-x}}$	$g'(x) = g(x)(1 - g(x))$	$(0, 1)$	$C^\infty$
 Heaviside ( $H$ )	$g(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$	$g'(x) = 0, x \neq 0$	$\{0, 1\}$	none
 Hyperbolic tangent (tanh)	$g(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$g'(x) = 1 - g(x)^2$	$(-1, 1)$	$C^\infty$
 Rectified linear unit (ReLU)	$g(x) = \max\{x, 0\}$	$g'(x) = \begin{cases} 0, & x < 0 \\ 1, & x > 0 \end{cases}$	$[0, \infty)$	$C$
 Parametric rec- tified linear unit (PReLU)	$g(x) = \begin{cases} \alpha x, & x < 0 \\ x, & x \geq 0 \end{cases}$ $\alpha > 0$	$g'(x) = \begin{cases} \alpha, & x < 0 \\ 1, & x > 0 \end{cases}$	$\mathbb{R}$	$C$
 Exponential linear unit (ELU)	$g(x) = \begin{cases} \alpha(e^x - 1), & x < 0 \\ x, & x \geq 0 \end{cases}$ $\alpha > 0$	$g'(x) = \begin{cases} g(x) + \alpha, & x < 0 \\ 1, & x > 0 \end{cases}$	$(-\alpha, \infty)$	$C^1, \alpha = 1$ $C, \alpha \neq 1$
 Softplus	$g(x) = \log(1 + e^x)$	$g'(x) = \frac{1}{1 + e^{-x}}$	$(0, \infty)$	$C^\infty$
 Gaussian	$g(x) = e^{-x^2}$	$g'(x) = -2xg(x)$	$(0, 1)$	$C^\infty$

Figure 2: One-dimensional Activation Function List

Activation	Definition	Derivative	Range	Smoothness
Softmax	$g_i(\mathbf{x}) = \frac{e^{x_i}}{\sum_{j=1}^d e^{x_j}}, i = 1, \dots, d$	$\frac{\partial g_i(\mathbf{x})}{\partial x_j} = \begin{cases} g_i(\mathbf{x})(1 - g_i(\mathbf{x})), & i = j \\ -g_i(\mathbf{x})g_j(\mathbf{x}), & i \neq j \end{cases}$	$(0, 1)^d$	$C^\infty$
Maxout	$g(\mathbf{x}) = \max\{x_1, \dots, x_d\}$	$\frac{\partial g(\mathbf{x})}{\partial x_i} = \begin{cases} 1, & x_i > \max_{j \neq i} x_j \\ 0, & x_i < \max_{j \neq i} x_j \end{cases}$	$\mathbb{R}$	$C$

Figure 3: Multi-dimensional Activation Function List

# Bibliography

- [1] Alexandre Antonov, Michael Konikov, and Michael Spector. The free boundary sabr: Natural extension to negative rates. *SSRN Electronic Journal*, January 2015. doi:10.2139/ssrn.2557046.
- [2] Alexandre Antonov and Michael Spector. Advanced analytics for the sabr model. *SSRN Electronic Journal*, page 2, March 2012. doi:10.2139/ssrn.2026350.
- [3] Louis Bachelier. *Théorie de la spéculation*, volume 3e série, 17, pages 21–86. Elsevier, 1900. doi:10.24033/asens.476.
- [4] Bruce Bartlett. Hedging under sabr model. *Wilmott*, pages 2–3, February 2006. URL: <http://lesniewski.us/papers/published/HedgingUnderSABRModel.pdf>.
- [5] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13:281–305, February 2012. URL: <https://jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf>.
- [6] Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. *Journal of Political Economy*, 81(3):637–654, 1973. URL: [www.cs.princeton.edu/courses/archive/fall109/cos323/papers/black\\_scholes73.pdf](http://www.cs.princeton.edu/courses/archive/fall109/cos323/papers/black_scholes73.pdf).
- [7] Damiano Brigo and Fabio Mercurio. *Interest Rate Models — Theory and Practice: With Smile, Inflation and Credit*, pages 1–22. Springer, January 2006. doi:10.1007/978-3-540-34604-3.
- [8] Damiano Brigo and Fabio Mercurio. *Interest Rate Models — Theory and Practice: With Smile, Inflation and Credit*, pages 23–47. Springer, January 2006. doi:10.1007/978-3-540-34604-3.
- [9] Damiano Brigo and Fabio Mercurio. *Interest Rate Models — Theory and Practice: With Smile, Inflation and Credit*, volume 2, pages 28–36. Springer, January 2006. doi:10.1007/978-3-540-34604-3.
- [10] Damiano Brigo and Fabio Mercurio. *Interest Rate Models — Theory and Practice: With Smile, Inflation and Credit*, volume 2, page 510. Springer, January 2006. doi:10.1007/978-3-540-34604-3.
- [11] Hans Bühler, Lukas Gonon, Josef Teichmann, and Ben Wood. Deep hedging. *Econometric Modeling: Derivatives eJournal*, pages 1–30, 2018. arXiv:1802.03042.

- 
- [12] Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal of Scientific Computing*, 16:1190–1208, September 1995. doi:[10.1137/0916069](https://doi.org/10.1137/0916069).
- [13] Peter Carr, Dilip Madan, Stephen Chung, Emanuel Derman, Raphael Douady, Bruno Dupire, Ognian Enchev, Chris Fern, Marvin Friedman, Iraj Kani, Keith Lewis, Harry Mendell, Lisa Polsky, John Ryan, and Murad Taqqu. Towards a theory of volatility trading. pages 1-21, December 1998. URL: [pdfs.semanticscholar.org/9781/c764ab2f09eafa9beb1fbf8d3b75124da6ed.pdf](https://pdfs.semanticscholar.org/9781/c764ab2f09eafa9beb1fbf8d3b75124da6ed.pdf).
- [14] Francois Chollet. *Deep Learning with Python*, pages 3–340. Manning Publications Co., USA, 1st edition, 2017.
- [15] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). *Under Review of ICLR2016*, pages 1–14, November 2015. arXiv:[1511.07289](https://arxiv.org/abs/1511.07289).
- [16] John Cox, Jonathan Ingersoll, and Stephen Ross. A theory of the term structure of interest rates. *Econometrica*, 53:385–407, February 1985. doi:[10.2307/1911242](https://doi.org/10.2307/1911242).
- [17] Richard K. Crump and Nikolay Gospodinov. Deconstructing the yield curve. Staff Reports 884, Federal Reserve Bank of New York, April 2019. URL: <http://ideas.repec.org/p/fip/fednsr/884.html>.
- [18] Howard B. Demuth, Mark H. Beale, Orlando De Jess, and Martin T. Hagan. *Neural Network Design*. Martin Hagan, Stillwater, OK, USA, 2nd edition, 2014. URL: <https://hagan.okstate.edu/NNDesign.pdf>.
- [19] Emanuel Derman and Iraj Kani. Riding on a smile. *Risk publications*, 7:277–284, January 1994.
- [20] Fabrice Douglas Rouah. The sabr model. URL: [laurent.jeanpaul.free.fr/Enseignement/The%20SABR%20Model.pdf](http://laurent.jeanpaul.free.fr/Enseignement/The%20SABR%20Model.pdf).
- [21] Timothy Dozat. Incorporating nesterov momentum into adam, 2016. URL: [http://cs229.stanford.edu/proj2015/054\\_report.pdf](http://cs229.stanford.edu/proj2015/054_report.pdf).
- [22] Bruno Dupire. Pricing with a smile. *Risk Magazine*, pages 18–20, 1994.
- [23] Eugenio Martín Gallego. Analysis of sabr calibration with neural networks. an application to the swaption smile. In *Universidad Complutense de Madrid*, 2019. URL: [www.uv.es/bfc/TFM2019/EugenioMartin](http://www.uv.es/bfc/TFM2019/EugenioMartin).
- [24] Jim Gatheral. *The Volatility Surface*. John Wiley & Sons, Ltd, 2006.
- [25] Jim Gatheral. Lecture 1: Stochastic volatility and local volatility. *Case Studies in Financial Modelling Course Notes*, Courant Institute of Mathematical Sciences, fall term 2012. URL: [www.cmap.polytechnique.fr/~rama/teaching/ea2003/gatheral.pdf](http://www.cmap.polytechnique.fr/~rama/teaching/ea2003/gatheral.pdf).
-

- 
- [26] Hélyette Geman, Nicole El Karoui, and Jean-Charles Rochet. Changes of numéraire, changes of probability measure and option pricing. *Journal of Applied Probability*, 32(2):443–458, 1995. doi:[10.2307/3215299](https://doi.org/10.2307/3215299).
- [27] Lech Grzelak and Cornelis Oosterlee. From arbitrage to arbitrage-free implied volatilities. *The Journal of Computational Finance*, 20:1–19, January 2016. doi:[10.21314/JCF.2016.316](https://doi.org/10.21314/JCF.2016.316).
- [28] Patrick Hagan. The sabr chronicles, November 2017. doi:[10.13140/RG.2.2.15143.44969](https://doi.org/10.13140/RG.2.2.15143.44969).
- [29] Patrick Hagan, Deep Kumar, Andrew Lesniewski, and Diana Woodward. Managing smile risk. *Wilmott Magazine*, 1:84–108, January 2002. URL: [https://www.next-finance.net/IMG/pdf/pdf\\_SABR.pdf](https://www.next-finance.net/IMG/pdf/pdf_SABR.pdf).
- [30] Patrick Hagan, Deep Kumar, Andrew Lesniewski, and Diana Woodward. Arbitrage-free sabr. *Wilmott*, 2014, January 2014. doi:[10.1002/wilm.10290](https://doi.org/10.1002/wilm.10290).
- [31] Patrick Hagan and Andrew Lesniewski. Libor market model with sabr style stochastic volatility, January 2006. URL: [pdfs.semanticscholar.org/b61a/a89e166eb65fdbf9dde3ffc742a623c6de22.pdf](https://pdfs.semanticscholar.org/b61a/a89e166eb65fdbf9dde3ffc742a623c6de22.pdf).
- [32] Patrick Hagan, Andrew Lesniewski, and Diana Woodward. Implied volatilities for mean reverting sabr models. *Wilmott*, 2020, October 2017. doi:[10.1002/wilm.10859](https://doi.org/10.1002/wilm.10859).
- [33] Patrick Hagan, Andrew Lesniewski, and Diana Woodward. Managing vol surfaces. *Wilmott*, 2018:24–43, January 2018. doi:[10.1002/wilm.10643](https://doi.org/10.1002/wilm.10643).
- [34] Patrick S. Hagan and Andrew Lesniewski. Bartlett’s delta in the SABR model. *Wilmott*, pages 54–61, April 2017. arXiv:[1704.03110](https://arxiv.org/abs/1704.03110).
- [35] Andres Hernandez. Model calibration with neural networks. *SSRN*, June 2016. doi:[10.2139/ssrn.2812140](https://doi.org/10.2139/ssrn.2812140).
- [36] Andres Hernandez. Model calibration: Global optimizer vs. neural network. *SSRN*, July 2017. doi:[10.2139/ssrn.2996930](https://doi.org/10.2139/ssrn.2996930).
- [37] Steven Heston. *The Heston (1993) Stochastic Volatility Model*, chapter 5, pages 136–162. John Wiley & Sons, Ltd, 2015. doi:[10.1002/9781119202097.ch5](https://doi.org/10.1002/9781119202097.ch5).
- [38] Catherine F. Higham and Desmond J. Higham. Deep Learning: An Introduction for Applied Mathematicians. *arXiv e-prints*, January 2018. arXiv:[1801.05894](https://arxiv.org/abs/1801.05894).
- [39] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359 – 366, 1989. doi:[10.1016/0893-6080\(89\)90020-8](https://doi.org/10.1016/0893-6080(89)90020-8).
- [40] John Hull and Alan White. The pricing of options on assets with stochastic volatilities. *The Journal of Finance*, 42(2):281–300, 1987. doi:[10.1111/j.1540-6261.1987.tb02568.x](https://doi.org/10.1111/j.1540-6261.1987.tb02568.x).
-



- 
- [41] James Hutchinson, Andrew Lo, and Tomaso Poggio. A nonparametric approach to pricing and hedging derivative securities via learning networks. *The Journal of Finance*, 49.3, January 2001. doi:[10.2307/2329209](https://doi.org/10.2307/2329209).
- [42] Patrick Kidger and Terry Lyons. Universal Approximation with Deep Narrow Networks. *arXiv e-prints*, page arXiv:1905.08539, May 2019. arXiv:[1905.08539](https://arxiv.org/abs/1905.08539).
- [43] Fabien Le Floc’h and Gary Kennedy. Explicit sabr calibration through simple expansions. *SSRN Electronic Journal*, January 2014. doi:[10.2139/ssrn.2467231](https://doi.org/10.2139/ssrn.2467231).
- [44] Fabien Le Floc’h and Gary Kennedy. Finite difference techniques for arbitrage free sabr. *SSRN Electronic Journal*, January 2014. doi:[10.2139/ssrn.2402001](https://doi.org/10.2139/ssrn.2402001).
- [45] Alexander Lipton. The vol smile problem. *Risk*, 15:61–65, January 2002. URL: [web.math.ku.dk/~rolf/LiptonVolSmileProblem.pdf](http://web.math.ku.dk/~rolf/LiptonVolSmileProblem.pdf).
- [46] Robert B. Litterman and José A. Scheinkman. Common factors affecting bond returns. In *Goldman Sachs Journal*, 1991. URL: [math.nyu.edu/faculty/avellane/Litterman1991.pdf](http://math.nyu.edu/faculty/avellane/Litterman1991.pdf).
- [47] Shuaiqiang Liu, Anastasia Borovykh, Lech A. Grzelak, and Cornelis W. Oosterlee. A neural network-based framework for financial model calibration. *Journal of Mathematics in Industry*, 9, April 2019. arXiv:[1904.10523](https://arxiv.org/abs/1904.10523).
- [48] Mahir Lokvancic. Machine learning sabr model of stochastic volatility with lookup table, April 2020. doi:[10.2139/ssrn.3589367](https://doi.org/10.2139/ssrn.3589367).
- [49] Lu Lu, Yeonjong Shin, Yanhui Su, and George Karniadakis. Dying relu and initialization: Theory and numerical examples. In *arxiv*, March 2019. arXiv:[1903.06733](https://arxiv.org/abs/1903.06733).
- [50] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The expressive power of neural networks: A view from the width. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6231–6239. Curran Associates, Inc., 2017. URL: [papers.nips.cc/paper/7203-the-expressive-power-of-neural-networks-a-view-from-the-width.pdf](http://papers.nips.cc/paper/7203-the-expressive-power-of-neural-networks-a-view-from-the-width.pdf).
- [51] William A McGhee. An artificial neural network representation of the sabr stochastic volatility model, November 2018. doi:[10.2139/ssrn.3288882](https://doi.org/10.2139/ssrn.3288882).
- [52] Mikko Pakkanen. Deep learning lecture notes. *Department of Mathematics, Imperial College London*, December 2019.
- [53] Allan Pinkus. Approximation theory of the MLP model in neural networks. *Acta Numerica*, 8:143–195, January 1999. doi:[10.1017/S096249290002919](https://doi.org/10.1017/S096249290002919).

- 
- [54] Johannes Ruf and Weiguan Wang. Neural networks for option pricing and hedging: a literature review. *arXiv e-prints*, November 2019. [arXiv:1911.05620](#).
- [55] Johannes Ruf and Weiguan Wang. Hedging with Neural Networks. *arXiv e-prints*, page arXiv:2004.08891, April 2020. [arXiv:2004.08891](#).
- [56] Filippo Santambrogio. { Euclidean, Metric, and Wasserstein } Gradient Flows: an overview. *Bulletin of Mathematical Sciences*, pages 87–154, September 2016. [arXiv:1609.03890](#).
- [57] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical Bayesian Optimization of Machine Learning Algorithms. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*, NIPS’12, page 2951–2959, Red Hook, NY, USA, June 2012. Curran Associates Inc. [arXiv:1206.2944](#).
- [58] Chi Chiang Tan. Market practice in financial modelling. *World Scientific Books*, January 2012. [doi:10.1142/8257](#).
- [59] Ian Thomson. Option pricing model comparing louis bachelier with black-scholes merton, March 2016. [doi:10.13140/RG.2.1.3896.7446](#).
- [60] Giovanni Travaglini. Sabr calibration in python. *Econometrics: Computer Programs & Software eJournal*, 2016. [doi:10.2139/ssrn.2725485](#).
- [61] Graeme West. *Calibration of the SABR Model in Illiquid Markets*, volume 12, pages 371–385. Routledge, December 2005. [doi:10.1080/13504860500148672](#).
- [62] Ciyou Zhu, Richard H. Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Trans. Math. Softw.*, 23(4):550–560, December 1997. [doi:10.1145/279232.279236](#).