UNIVERSITY OF KONSTANZ

MASTER OF SCIENCE IN MATHEMATICAL FINANCE

A deep learning approach to pricing and calibration of discrete-time volatility models

Author:
Henrik BRAUTMEIER

Supervisor:
Dr. Lyudmila GRIGORYEVA

A thesis submitted in fulfilment of the requirements for the degree of Master of Science in Mathematical Finance

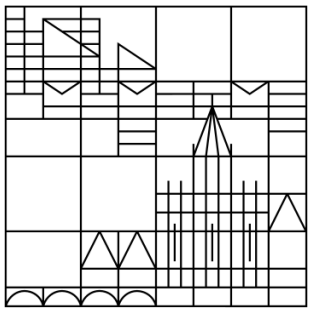
in the

Graduate School of Decision Sciences
Department of Mathematics and Statistics

at the

University of Konstanz

June 15, 2020



Declaration of Authorship

I, Henrik BRAUTMEIER, declare that this thesis titled, "A deep learning approach to pricing and calibration of discrete-time volatility models" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: 

Date: June 15, 2020

"Building advanced AI is like launching a rocket. The first challenge is to maximise acceleration, but once it starts picking up speed, you also need to focus on steering."

Jaan Tallinn

UNIVERSITY OF KONSTANZ

Abstract

Graduate School of Decision Sciences
Department of Mathematics and Statistics

Master of Science in Mathematical Finance

A deep learning approach to pricing and calibration of discrete-time volatility models

by Henrik BRAUTMEIER

We present a first deep learning based calibration and option pricing method for discrete-time volatility models using the Heston-Nandi-GARCH(1,1) model. We present specific CNN architectures for price approximation and parameter calibration. The purpose of neural networks in this context is approximation of pricing functions and their inverse mapping, which are difficult to evaluate by other approaches. Especially the issue of slow calibration and pricing is lifted by the usage of neural networks and might allow for application in industry practice. The calibration task can be performed in a few microseconds for a full implied volatility surface. We conduct several numerical experiments to evaluate the performance with respect to accuracy, computation time and robustness.

Keywords: Volatility Modelling, Heston Nandi GARCH, Deep Learning, Model Calibration, Derivative Pricing

Acknowledgements

I cannot express enough thanks to my supervisor, Dr. Lyudmila Grigoryeva, for her continued support and encouragement. I offer my sincere appreciation for the learning opportunities provided by her, Dr. Juan-Pablo Ortega and Dr. Alexandru Badescu. I hope that your future collaboration will be as enlightening as this one.

My completion of this project could not have been accomplished without the support of my fellow student Lukas.

I want to thank my parents, for supporting me through out my whole academic career.

Finally, to my caring, loving, and supportive girlfriend, Alena: my deepest gratitude. Your encouragement when the times got rough are much appreciated and duly noted. My heartfelt thanks.

Contents

List of Figures

List of Tables

List of Abbreviations

Model	
BS	**B**lack, **S**choles & Merton
GARCH	**G**eneralised **A**uto**R**egressive **C**onditional **H**eteroskedasticity
HNG	**H**eston **N**andi **G**ARCH

Neural Networks	
CNN	**C**onvolutional **N**eural **N**etwork
FFNN	**F**eed-**F**orward **N**eural **N**etwork
GRU	**G**ated **R**ecurrent **U**nit
LSTM	**L**ong **S**hort-**T**erm **M**emory Neural Network

Performance Measures	
AIC	**A**kaike **I**nformation **C**riterion
AICc	Small Sample **A**kaike **I**nformation **C**riterion
BIC	**B**ayes **I**nformation **C**riterion
MAPE	**M**ean **A**bsolute **P**ercentage **E**rror
MSE	**M**ean **S**quared **E**rror
CMSE	**C**onstrained **M**ean **S**quared **E**rror
RMSE	**R**oot **M**ean **S**quared **E**rror
RMSRE	**R**oot **M**ean **S**quared **R**elative **E**rror

Miscellaneous	
AI	**A**rtifical **I**ntelligence
ERM	**E**mpirical **R**isk **M**inimisation
MLE	**M**aximum **L**ikelihood **E**stimation
SDE	**S**tochastic **D**ifferential **E**quation
SGD	**S**tochastic **G**radient **D**escent Algorithm
T-Bill	US-**T**reasury **B**ill

List of Symbols

Sets and Spaces

$\mathbb{N}, \mathbb{Z}, \mathbb{R}$	set of natural numbers, integers and real numbers
$\mathbb{R}^+$	set of positive reals
Ω	set of events
$\mathcal{F}$	σ-algebra on a set Ω
$\sigma(M)$	σ-algebra generated by a set M
$(\mathcal{F}_t)_t$	Filtration on a probability space $(\Omega, \mathcal{F}, \mathbb{P})$
$L^p(\Omega), L^p(\mathbb{P})$	stochastic L^p-space on a probability space $(\Omega, \mathcal{F}, \mathbb{P})$
$\theta, \hat{\theta}$	parameter or statistic with corresponding estimator
$\mathcal{F}(A, B)$	space of functions $f : A \to B$

Stochastic Processes

i.i.d.	identically and mutually independent distributed
$\mathcal{N}(\mu, \Sigma)$	multivariate normal distribution, with expectation vector μ and covariance matrix Σ
$\mathbb{P}$-a.s.	equation holds almost sure with respect to measure $\mathbb{P}$

Mathematical Operators

$\mathbb{V}[X]$	variance of a random variable X
$\mathbb{E}[X]$	expected value of a random variable X
$\mathbb{E}\left[X \middle\vert \mathcal{F}\right]$	conditional expectation of X with respect to $\mathcal{F}$
$\mathbb{V}\left[X \middle\vert \mathcal{F}\right]$	conditional variance of X with respect to $\mathcal{F}$
$\lVert\cdot\rVert_2$	Euclidean norm on $\mathbb{R}^d$
$\lVert\cdot\rVert_\infty$	maximum norm
$\mathbb{1}_C(\cdot)$	indicator function
$\ln(\cdot)$	natural logarithm
$\mathrm{id}_X(\cdot)$	identity function of a space X
$\mathrm{Re}(\cdot)$	real part of a complex number
ι	vector of constant 1 in each entry
$(\cdot)^+$	positive part of a function
$\mathrm{Vec}(X)$	vectorisation of a matrix X

Chapter 1

Introduction

1.1 Context of Research

The appearance of financial markets shaped the modern economy and contributed to economic growth by providing investment opportunities to a wide class of investors. Among those investments, derivative contracts play a huge role in today's risk management and investment decisions. Since the first occurrence of modern derivatives, during the end of the last century, their importance has increased significantly. In 2019, the total volume of futures and options contracts traded on exchanges worldwide, reaching an all-time high with a record of 34.47 billion contracts (FIA (2019)). According to the most recent data from the Bank for International Settlements (2019), the total notional amounts outstanding for contracts in the derivatives market is at an all-time high of an estimated $640 trillion. The gross market value of all contracts sums up to approximately $12.1 trillion. This roughly equals the nominal GDP of China in 2019 with $14.1 trillion (IMF (2019)). In light of such numbers, it is essential that practitioners and researchers thoroughly understand modelling and hedging of derivatives. Consequently, researchers have spent years developing sufficiently realistic models for pricing such financial products. Starting with the work of Black and Scholes (1973) and Merton (1973), a plethora of market models were introduced and analysed, each with its own upsides and flaws. Today's stochastic pricing models are able to capture most empirically observed properties as leptokurtosis and skewness of returns or even advanced characteristics such as market shocks (see for example Runggaldier (2003) on jump-diffusion processes). However, such sufficiency comes at a significant expense in computational load and memory. Hence, the Black-Scholes-Merton (BS) model is still widely applied, due to its simplicity. This over-reliance on the same formula, and more generally the deviance in pricing of complex derivative products, has been highly criticised during the recent financial crises. Especially during the subprime

crisis of 2007 and the following global financial crisis the weaknesses of the used rating and pricing models were uncovered. Currently, regulatory (e.g. Basel IV, MiFID, IFRS9) and environmental pressures are existent and pushing banks to upgrade their risk management.
Simultaneously, society and economy are disrupted by digitalisation, which fundamentally transforms major business sectors as the automobile, communication or commerce. It drives entrepreneurial innovation, productivity, and also has implications for the development of labor markets, and political participation. Frey and Osborne (2017) forecast that "around 47% of total US employment is in the high risk category" which refers to working environments that they expect to be automated over the next decade. The increase of computational power, the linked rise of big data applications and advances in artificial intelligence (AI) and machine learning algorithm are the main drivers of those developments. Hence it is no surprise that this transformation does not stop at the financial markets. According to the European Financial Management Association (2018), roughly 35% of European financial organisations have deployed at least one machine learning solution at the end of 2018. Further, they state that of those organisations that are yet to deploy a solution, 23% believed they would have an AI solution in place within in a year. Besides huge improvements in automating customer experience, e.g. detecting customer behaviour patterns or automatised chatbot programs, financial risk management and trading departments can profit from such solutions. The availability of intraday data as well as the achievements in the area of data driven algorithms provide a good framework to combine financial theory and machine learning techniques. Lately, many practitioners and researchers focus on such non-parametric approaches as LSTM or GRU architectures to forecast financial time-series (see for example Kim and Won (2018) or Cao, Z. Li, and J. Li (2019)). One major advantage of these approaches is that they are highly data-driven and allow to incorporate non-standard financial data or time-series, like meteorological or social media data, with no additional effort. However, statistical inference and analysis of the given estimators and forecasts is difficult for such approaches (see for example Geman, Bienenstock, and Doursat (1992)). Furthermore, recent work by Liu, Oosterlee, and Bohte (2019) implies that those approaches do not outperform stochastic volatility models in out-of-sample comparison, but it indicates that deep learning techniques can be used to significantly decrease computation time.

1.2 Research Questions and Motivation

This thesis focuses on the efficient pricing and calibration of stochastic volatility models. With increasing complexity of models, it is not possible to ensure the existence of closed-form pricing formulas. Therefore, computation of option prices and calibration of model parameter lead to the necessity of very computation-intensive simulations, which themselves are a potential source of error. Additionally, the computation time of the Monte-Carlo based approaches scales at least linear with number of paths while accuracy often scales sublinear. Hence, such methods become unattractive with higher needs for accuracy. This explains the attractiveness of the Black-Scholes-Merton model as well as non-parametric approaches even after more realistic stochastic pricing models have been developed.

To tackle these problems, Horvath, Muguruza, and Tomas (2019) introduce a new approach. The authors provide a method for pricing and calibration of continuous volatility models (namely Rough Bergomi Model) using feed-forward neural networks (FFNN). In a first step, the authors train a FFNN to map model parameters directly to a volatility surface. By doing so computational-intensive simulations are needed only once for training. Afterwards the trained neural network can be used as a fast and accurate pricing tool. Secondly, the calibration task can be done by back-propagation through the trained net. Nonetheless, they were not able to provide an approach on training a neural networks for the calibration task. The usage of continuous-time stochastic volatility models also often results in problems when one is asked not only to calibrate a model to observable market data, but also to construct forecasts.

This thesis uses an approach similar to Horvath, Muguruza, and Tomas (2019), but focuses on discrete-time models, namely the model introduced by Heston and Nandi (2000). Moving from a continuous-time to a discrete-time framework allows to incorporate observed data with no additional effort and to calibrate parameters solely on the basis from historic observations. Another advantage of using the Heston-Nandi-GARCH (HNG) model is the existence of a closed-form solution for option pricing. Nonetheless, our approach is easily generalisable to any other discrete-time model and not exclusively usable for the HNG model. With the increased model complexity and additional parameter constraints, FFNN architectures are not able to fully capture the properties of the model, but using convolutional neural networks (CNN) resolves this issue. However, CNN are not widely deployed in a financial

setting yet and this thesis presents a first approach to utilising such a network architecture for the pricing of derivatives. Traditionally, CNN have been applied in image recognition tasks with huge success. As we transform the option pricing exercise to an image recognition problem, we are able to exploit the beneficial properties of such an architecture. Furthermore this procedure allows to simultaneously price full volatility surfaces.
It is crucial to point out that this thesis neither aims to find a well performing volatility model nor to compare existing volatility models. The interested reader is referred to relevant publications as Hansen and Lunde (2005) or Ahmend, Atiya, El Gayar, and El-Shishiny (2010). This thesis focuses on the ability of CNN to approximate discrete-time volatility models and their suitability as an efficient approximation tool for option pricing and model calibration.

1.3 Summary of Results

We present specific CNN-architectures for price approximation and calibration. We conduct several numerical experiments to evaluate the performance with respect to accuracy, computation time and robustness.
The architecture of the pricing network allows to generate full implied volatility surfaces instead of a computation of single values. We compare the pricing approach to two benchmarks - a classic Monte-Carlo simulation and the FFNN-based pricing method introduced by Horvath, Muguruza, and Tomas (2019). Hereby, the performance is measured in terms of accuracy and speed. We also check the significance of our results by testing for stochastic dominance in the error distributions. Our approach reaches an average 0.6298% relative deviation to the closed-form solution in the implied volatility pricing task, significantly outperforming both benchmarks for all given confidence levels. Our approach prices implied volatility surfaces significantly faster than the closed-form solution or the Monte-Carlo simulation. It prices at a comparable rate as the FFNN-based method.
To measure the performance of the proposed calibration approach, we compare it to two benchmarks - calibration via standard constraint optimisations algorithms as well as the FFNN-calibration approach of Horvath, Muguruza, and Tomas (2019). Our approach leads to a 99.97% lower average computation time in comparison to the closest benchmark. Simultaneously, the calibration accuracy in each parameter increases by at least 50%. As the HNG(1,1) is extremely sensitive in the parameters, it is important to test for the robustness

of the calibrated parameters. We show that the calibrated parameters lead to accurate surfaces, when applied to the pricing network. This findings pave the way for potential usage in industry applications.
Furthermore, we analyse the performance of the proposed neural networks architecture when used on different datasets. The results indicate that our propose is indeed able to generalise and is not bounded to the introduced HNG(1,1) model.

1.4 Structure

This master's thesis is structured along the following chapters:

Chapter 2 contains the concepts that are necessary to grasp the remaining theory of the thesis. We briefly introduce the mathematical foundation of market models and focus on risk-neutral valuation, the important concepts of no-arbitrage and market completeness. We review the Black-Scholes-Merton model as well as the closely linked concept of implied volatility.

Chapter 3 covers the discrete-time volatility model developed by Heston and Nandi (2000), which is utilised as a central model in this thesis. We introduce the model specifications, derive the closed-form solution and discuss the most important properties as well as its limitations. Finally, we illustrate the empirical performance of Heston-Nandi-GARCH(1,1). For that purpose the historical estimation as well as option price based calibration on S&P 500 data is analysed.

Chapter 4 provides details of the machine learning methodology that we adopt to answer our research questions. We introduce the concepts of statistical learning and artificial neural networks. We introduce a neural network approach to price and calibrate option prices in the Heston-Nandi-GARCH(1,1) model and describe the benchmarks used to assess the performance of the models.

Chapter 5 concludes the thesis. We provide details on the conducted numerical experiments and discuss the results. We point out implications of our work along with the limitations and finalise the thesis with suggestions for future research.

Chapter 2

Options, Pricing and Finance Theory

The structure and notation of this chapter is based on Chapter 1 of Föllmer and Schied (2016) and the lecture notes of the course on Mathematical Finance offered by Prof. Michael Kupper at the University of Konstanz (see Kupper (2018)). We provide a mathematical structure for financial market models. Since we are dealing with the HNG model later on, this section focuses on discrete-time models. Especially when dealing with high frequency trading and the corresponding analysis of connected events e.g. Flash Crashes (see for example Kirilenko, Kyle, Samadi, and Tuzun, 2017), such models are of great usefulness. We point out that ultimately every real financial market is of discrete nature independent of its size and liquidity.

2.1 Financial Markets in Discrete-Time

Definition 2.1 (Financial Market)**.** We consider a finite number $(d+1)$ of financial assets, those assets can be stocks, bonds, or any other tradable good. Those assets are priced at times $t \in \{0, \dots, T\}$. Consider a filtered probability space $(\Omega, \mathcal{F}, (\mathcal{F}_t)_{t\in\{0,\dots,T\}}, \mathbb{P})$ and a $d+1$-dimensional non-negative price process $\overline{S} = (S^0, S) = (S^0_t, S^1_t, \dots, S^d_t)_{t\in\{0,\dots,T\}}$, which is adapted to $(\mathcal{F}_t)_{t\in\{0,\dots,T\}}$. Even though today's markets allow for negative prices, especially in the energy spot markets, positivity is assumed for the sake of simplicity. The interested reader is referred to Alaton, Djehiche, and Stillberger (2002) or Deng (2000). As for every stochastic model the filtration represents the information which is available at every time step. Again, for simplicity it is useful to assume that $\mathcal{F}_0 = \{\emptyset, \Omega\}$ and $\mathcal{F}_T = \mathcal{F}$. This corresponds to the assumption that no information is given before trading and all relevant information is given at the end of observation at time T. In this notation S^0 will always denote the used Numéraire, e.g. the value of money stored in a transaction account

and therefore we will assume $S_t^0 > 0$ for all $t \in \{0, \dots, T\}$. Later on, further assumption on the distribution of S^0 will be made (see Section 2.3).

To trade assets on this market requires the construction of a trading strategy, which determines the actions of a contributor on our market.

Definition 2.2 (Trading Strategy)**.** A trading strategy ξ is an $\mathbb{R}^{d+1}$-valued process on $(\Omega, \mathcal{F}, (\mathcal{F}_t)_{t\in\{0,\dots,T\}}, \mathbb{P})$, where ξ_{t+1} is measurable with respect to $\mathcal{F}_t$ for all $t \in \{0, \dots, T-1\}$. ξ_t^i can be interpreted as the quantity of shares held in the i-th asset between t and $t-1$. Therefore, a trading strategy can be interpreted as a portfolio of asset in a given market over the lifetime of this market.

It is useful to assume that no external capital is added to portfolio while trading.

Definition 2.3 (Self-financing)**.** A trading strategy ξ is called self-financing if $\xi_t S_t = \xi_{t+1} S_t$ holds for every $t \in \{1, \dots, T-1\}$.

Especially when replicating contingent claims with a trading strategy, it is necessary to assume self-financing. It would be otherwise difficult to fairly price such claims as the present value of added capital needs to be subtracted. $\xi_0 S_0$ corresponds to the initial endowment of the trader. It is easy to see that every trading strategy ξ can be changed to a self-financing strategy by adjusting the number of assets held in the Numéraire.

Definition 2.4 (Discounted Price)**.** Since all values are usually expressed in terms of the Numéraire S^0, it is easier to deal with the discounted price process

$$\overline{X} = (1, X) := \frac{\overline{S}}{S^0} = \left(1, \frac{S_t}{S_t^0}\right)_{t\in\{0,\dots,T\}}.$$

This also allows to reduce the dimension of the market by one and hence drastically simplifies all estimations and calculations in terms of complexity as well as computational power.
Finally, all tools are given to define the value of a portfolio.

Definition 2.5 (Value Process)**.** The value process V of a given trading strategy ξ is given by $V_0 = \xi_0 \overline{X}_0$ and $V_t = \xi_t \overline{X}_t$, where V_0 represents the initial endowment of an investor. The accumulated trading gains G of such strategy ξ are given by

$$G_0 = 0,\ G_t = \sum_{i=1}^{t} \xi_i (\overline{X}_i - \overline{X}_{i-1})$$

To ensure a market model, in which it is not possible to generate higher gains than the Numéraire at anytime without taking additional risk, we need additional assumptions on the set of trading strategies:

Definition 2.6 (Arbitrage)**.** A self-financing trading strategy is called an arbitrage opportunity, if its value process satisfies $V_0 \leq 0, V_T \geq 0$ $\mathbb{P}$-as and $\mathbb{P}(V_T > 0) > 0$. A market model which does not allow for such arbitrage opportunities is called arbitrage-free.

Lemma 2.7. The following are equivalent

i. There exists an arbitrage opportunity

ii. There exists a bounded arbitrage opportunity with V_0

Proof. See Kupper (2018). □

This claim is remarkable given that an unbounded trading strategy would allow for an infinite amount of trades, which is highly unrealistic in any given market. Therefore, the possibility of solely unbounded arbitrage opportunities would undermine the economic reasoning of such a definition.
As the assumption of no-arbitrage builds the basis for every further pricing, it is of utmost importance to find characterisation of this property. In particular, connecting the structure of $\mathbb{P}$ with the concept of arbitrage allows for a deeper stochastic analysis of such markets:

Definition 2.8 (Equivalence)**.** Let $\mathbb{P}, \mathbb{Q}$ be measures on a measurable space $(\Omega, \mathcal{F})$. Define the set of all null set as

$$\mathcal{N}_{\mathbb{P}} = \{A \in \mathcal{F} \,|\mathbb{P}(A) = 0\}, \mathcal{N}_{\mathbb{Q}} = \{A \in \mathcal{F} \,|\, \mathbb{Q}(A) = 0\}$$

The measure $\mathbb{P}$ is said to be absolutely continuous in reference to $\mathbb{Q}$ if and only if $\mathcal{N}_{\mathbb{Q}} \subseteq \mathcal{N}_{\mathbb{P}}$. The two measures $\mathbb{P}, \mathbb{Q}$ are called equivalent, if they are mutually absolutely continuous ($\mathcal{N}_{\mathbb{Q}} = \mathcal{N}_{\mathbb{P}}$).

Definition 2.9 (Martingale Measure)**.** Let X be process on a probability space $(\Omega, \mathcal{F}, \mathbb{P})$. A probability measure $\mathbb{Q}$ on $(\Omega, \mathcal{F})$ is called martingale measure of X, if X is a $\mathbb{Q}$-martingale. Such a martingale measure $\mathbb{Q}$ is called an equivalent martingale measure, if $\mathbb{Q} \sim \mathbb{P}$.

Theorem 2.10 (First Fundamental Theorem of Asset Pricing)**.** The market model S is arbitrage-free if and only if the set $\mathcal{P}(\mathbb{P})$ of equivalent martingale measures of $\mathbb{P}$ is nonempty. In this case there is even a $\mathbb{Q} \in \mathcal{P}(\mathbb{P})$ which has a bounded Radon–Nikodym density $\frac{\mathrm{d}\mathbb{Q}}{\mathrm{d}\mathbb{P}}$.

Proof. See Kupper (2018). □

Even though the second part of the theorem might sound unimportant, it has a huge implication for any pricing: Once able to show or assume lack of arbitrage, one is able to move between the real/observed probability measure $\mathbb{P}$, which might have undesirable properties and a martingale measure $\mathbb{Q}$ using transformation theorems (e.g. Girsanov Theorem). Nonetheless, it is important to say that Theorem 2.10 does not hold for every continuous market or discrete markets with indefinite time horizon. Especially for continuous-time models, it does not exist a unique, universally accepted approach to pricing options by arguing with the existence of arbitrage.

2.2 Contingent Claims, Options and Completeness

From now on, we assume that for every market model, it holds $\mathcal{P} \neq \emptyset$, or to be more precise we assume arbitrage-free market models.

Definition 2.11 (Contingent Claims & Derivatives)**.** A non-negative random variable C on $(\Omega, \mathcal{F}, \mathbb{P})$ is called a European-style contingent claim. A European-style contingent claim is called a derivative of the underlying assets $\overline{S}$ if C is measurable with respect to $\sigma(\overline{S}_0, \ldots, \overline{S}_T)$.
Remark: Obviously the relation between a derivative and a specific underlying S_i can be trivial and C does not have to depend on every single asset. European-style derivatives usually depend on two major parameters, the strike price K, which gives a trigger price at which an action is performed and the maturity T, which gives the time at which the option expires. For this work only European call options are important but for the sake of completeness, we add a collection of different common derivative types

European Call Option $C_{call} = (S_T^i - K)^+$. The owner of a European call option has the right but not the obligation to buy the asset i at maturity T for a fixed strike price K.

European Put Option $C_{put} = (K - S_T^i)^+$. The owner of a European put option has the right but not the obligation to sell the asset i at maturity T for a fixed strike price K.

European Binary Call Option $C_{bc} = \mathbb{1}_{S_T > K}$. This asset is usually cash settled. The owner of a European binary call option gains a fixed amount if the asset i at maturity T is higher than a boundary price K.

European Binary Put Option $C_{bp} = \mathbb{1}_{S_T<K}$. This asset is usually cash settled. The owner of a European binary put option gains a fixed amount if the asset i at maturity T is lower than a boundary price K.

Barrier Options $C_{barrier} = \mathbb{1}_{\max_t S_t<B_u}\mathbb{1}_{\min_t S_t>B_l}C^*$. Let C^* be any arbitrary derivative. A barrier option bounds the underlying in between or outside of two boundary values $B_l, B_u \in \mathbb{R}\cup\{-\infty,\infty\}$. It holds $B_l = -\infty$ or $B_u = \infty$ if only one bound applies. If the underlying crosses these bounds before maturity the derivative expires immediately. Even if the underlying option is European-style, pricing is more complicated as the payoff structure $C_{barrier}$ is strongly path-dependent.

Obviously there are many more derivative styles and as those assets are usually traded over the counter, theoretically everything imaginable is possible to trade as long as a fitting counterparty is found. But as this thesis does not focus solely on option pricing we refer the reader to Hull (2001).

Definition 2.12 (Attainable)**.** A contingent claim C is called attainable (replicable) if there exists a self-financing strategy ξ such that $\xi_T S_T = C$ $\mathbb{P}$-as. In this case ξ is called a replicating strategy.
Remark: For a contingent claim C we consider the corresponding discounted claim $H := \frac{C}{S_T^0}$. Then, C is attainable if and only if

$$H = \xi_T X_T = V_T = V_0 + \sum_{t=1}^{T} \xi_t (X_t - X_{t-1})$$

for a self-financing strategy $\overline{\xi} = (\xi_0, \xi)$.

Theorem 2.13. Let H be an attainable discounted claim. Then $\mathbb{E}_{\mathbb{Q}}[H] < \infty$ for every $\mathbb{Q} \in \mathcal{P}(\mathbb{P})$. The value process of any replicating strategy ξ satisfies $V_t = \mathbb{E}_{\mathbb{Q}}[H|\,\mathcal{F}_t]$ $\mathbb{P}$-as for all $t \in \{0,\dots,T\}$ and every $\mathbb{Q} \in \mathcal{P}(\mathbb{P})$. In particular, the value process V is a $\mathbb{P}$-martingale.

Proof. See Kupper (2018). □

Remark:

i. The value process V does not depend on the replicating strategy ξ.

ii. We have $\mathbb{E}_{\mathbb{Q}}[V_T|\,\mathcal{F}_t] = \mathbb{E}_{\mathbb{Q}^*}[V_T|\,\mathcal{F}_t]$ $\mathbb{P}$-as for all $\mathbb{Q}$, $\mathbb{Q}^* \in \mathcal{P}(\mathbb{P})$

Definition 2.14 (Arbitrage Free Price)**.** A real number $\pi_H \geq 0$ is called an arbitrage-free price of H, if there exists an adapted, non-negative process X^{d+1} such that $X_0^{d+1} = \pi_H$, $X_T^{d+1} = H$ and the extended discounted market model $(X^0, \ldots, X^d, X^{d+1})$ is arbitrage-free. The set of arbitrage-free prices for H is denoted by $\Pi(H)$. Further, let $\pi_{\inf(H)} := \inf \Pi(H)$ and $\pi_{\sup(H)} := \sup \Pi(H)$

Theorem 2.15. Let H be a discounted claim.

- If H is attainable, then the set $\Pi(H)$ consists of one single element.
- If H is not attainable, then $\pi_{\inf(H)} < \pi_{\sup(H)}$ and $\Pi(H) = (\pi_{\inf(H)}, \pi_{\sup(H)})$ is an open interval.

Proof. See Kupper (2018). □

Definition 2.16 (Completeness)**.** A market model is called complete if every contingent claim is attainable. This implies that every contingent claim has one unique fair price and there is a replicating strategy, which allows for synthetically reconstructing and perfectly hedging the asset.

Analogously to Theorem 2.10 and the no-arbitrage property, it is important to characterise completeness with a proper stochastic claim:

Theorem 2.17 (Second Fundamental Theorem of Asset Pricing)**.** An arbitrage-free market model is complete with respect to $\mathbb{P}$ if and only if $|\mathcal{P}(\mathbb{P})| = 1$

Proof. If a market model is complete with respect to $\mathbb{P}$, every contingent claim is attainable. We know that for every $A \in \mathcal{F}_T$ the trivial contingent claim $H = \mathbb{1}_A$ is attainable. It follows from Theorem 2.13, that for every $\mathbb{Q}, \mathbb{Q}^* \in \mathcal{P}(\mathbb{P})$ holds $\mathbb{E}_{\mathbb{Q}}[H] = \mathbb{E}_{\mathbb{Q}^*}[H]$. This leads to $\mathbb{Q}^*(A) = \mathbb{Q}(A)$. Therefore the mapping $\mathcal{P}(\mathbb{P}) \to [0,1]$ with $\mathbb{Q} \mapsto \mathbb{E}_{\mathbb{Q}}[H]$ is constant. As $\mathbb{E}_{\mathbb{Q}}[H] = \mathbb{Q}(A)$ and A is arbitrary, it follows $\mathbb{Q} = \mathbb{Q}^*$ and we conclude $|\mathcal{P}(\mathbb{P})| = 1$.
On the other hand if $|\mathcal{P}| = 1$, the set of arbitrage free prices consists of a single element for any arbitrary contingent claim. By negation of Theorem 2.15, it follows immediately that this contingent claim is attainable. □

2.3 Black-Scholes Model and Implied Volatility

In this section we review the popular Black-Scholes pricing formula and introduce the closely linked concept of implied volatilities.

2.3.1 Model Description and Pricing

The first arbitrage free market model which let to a closed-form solution for European-style call and put options was introduced by Black and Scholes (1973) and Merton (1973). On a filtered probability space $(\Omega, \mathcal{F}, \mathbb{P}, (\mathcal{F}_t)_t)$, a stock with price process S and a risk free transfer account B used as Numéraire are considered. Several assumptions are made to ensure absence of arbitrage. First, the short term interest rate r is known and constant through time and interests are calculated continuously. The variance σ of the stock is constant over time and follows no dynamic. The return process is log-normal distributed with parameters $((\mu - \frac{1}{2}\sigma^2)t\ , \sigma^2)$, where $\sigma > 0$. Here, $\mu \in \mathbb{R}$ can be interpreted as long term growth of the stock and $\frac{1}{2}\sigma^2$ as market price of the underlying risk. The stock pays no dividends. We assume a frictionless market in which no transaction costs occur and lending and borrowing is possible at unlimited amounts. Neither are any assumptions made on trading strategies, e.g. no limitation in short selling nor to the amount of stocks held or traded. Hence we get

$$S_t = S_0 \exp\left((\mu - \frac{1}{2}\sigma^2)t + \sigma W_t\right),$$

where W is a Wiener process on $(\Omega, \mathcal{F}, \mathbb{P}, (\mathcal{F}_t)_t)$ and $B_t = B_0 \exp(rT)$. The dynamics of S and B under $\mathbb{P}$ follow the system of SDEs (2.1).

$$\begin{aligned} dS_t &= \mu S_t\, dt + \sigma S_t\, dW_t \\ dB_t &= rB_t\, dt \end{aligned} \tag{2.1}$$

Ultimately, we want to extend this market model consisting of two assets with an European-style call option C with maturity T and strike K. First, consider the discounted market model, where the only asset follows

$$\begin{aligned} X_t = \frac{S_t}{B_t} &= S_t \exp(-rt) = S_0 \exp\left((\mu - \frac{1}{2}\sigma^2 - r)t + \sigma W_t\right) \\ &:= S_0 \exp\left(\frac{1}{2}\sigma^2 t + \sigma W_t^*\right), \end{aligned}$$

where, $W_t^* = W_t + \frac{\mu - r}{\sigma} t$. By Girsanov's Theorem we know that W_t^* is a Wiener process under the measure $\mathbb{Q}$ with

$$\frac{\mathrm{d}\,\mathbb{Q}}{\mathrm{d}\mathbb{P}} := \exp\left(-\frac{1}{2} \left(\frac{\mu - r}{\sigma} \right)^2 T - \frac{\mu - r}{\sigma} W_T \right).$$

We show that $(X_t)_t$ is a martingale with respect to $\mathbb{Q}$ or more precisely $\mathbb{E}_{\mathbb{Q}}[X_t | \mathcal{F}_s] = X_s$ for all $0 \leq s \leq t$):
It's clear that the natural filtration $(\mathcal{F}_t)_t$ of $(X_t)_t$ coincides with natural filtration $(\mathcal{F}_t^W)_t$ of $(W_t^*)_t$. It holds $\mathbb{E}_{\mathbb{Q}}[Z | \mathcal{F}_t] = \mathbb{E}_{\mathbb{Q}}[Z | \mathcal{F}_t^W]$ for all $0 \leq t$ and integrable random variable Z. Hence,

$$\begin{aligned} \mathbb{E}_{\mathbb{Q}}[X_t | \mathcal{F}_s] &= X_s e^{-\frac{1}{2}\sigma^2(t-s)} \, \mathbb{E}_{\mathbb{Q}} \left(e^{\sigma(W_t^* - W_s^*)} \right) \\ &= X_s e^{-\frac{1}{2}\sigma^2(t-s)} e^{\frac{1}{2}\sigma^2(t-s)} = X_s. \end{aligned}$$

It holds $\mathbb{Q} \in \mathcal{P}$ and we conclude with Theorem 2.13, that the value process of H equals

$$V_t = \mathbb{E}_{\mathbb{Q}}[H | \mathcal{F}_t] = \mathbb{E}_{\mathbb{Q}} \left[(X_T - K)^+ \middle| \mathcal{F}_t \right]. \tag{2.2}$$

To finalise, the pricing of H only formula (2.2) must be calculated:

$$\begin{aligned} &\mathbb{E}_{\mathbb{Q}} \left[(X_T - K)^+ \,\middle|\, \mathcal{F}_t \right] = e^{-r(T-t)} \, \mathbb{E}_{\mathbb{Q}} \left[\left(S_t e^{r - \frac{1}{2}\sigma^2(T-t) + \sigma(W_T^* - W_t^*)} - K \right)^+ \,\middle|\, \mathcal{F}_t \right] \\ &= e^{-r(T-t)} \int_{-\infty}^{\infty} \left(S_t e^{r - \frac{1}{2}\sigma^2(T-t) + \sigma\sqrt{T-t}x} - K \right)^+ \phi(x)\, dx \\ &= e^{-r(T-t)} \int_{-d_-}^{\infty} \left(S_t e^{r - \frac{1}{2}\sigma^2(T-t) + \sigma\sqrt{T-t}x} - K \right) \phi(x)\, dx \\ &= e^{-r(T-t)} \int_{-d_-}^{\infty} S_t e^{r - \frac{1}{2}\sigma^2(T-t) + \sigma\sqrt{T-t}x} \phi(x)\, dx - e^{-r(T-t)} K \int_{-d_-}^{\infty} \phi(x)\, dx \\ &= e^{-r(T-t)} \int_{-d_-}^{\infty} S_t e^{r - \frac{1}{2}\sigma^2(T-t) + \sigma\sqrt{T-t}x} \phi(x)\, dx - e^{-r(T-t)} K \Phi(d_-) \\ &= S_t \int_{-d_-}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(x - \sigma\sqrt{T-t})^2}\, dx - e^{-r(T-t)} K \Phi(d_-) \\ &= S_t \int_{-d_+}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}\, dx - e^{-r(T-t)} K \Phi(d_-) = S_t \Phi(d_+) - e^{-r(T-t)} K \Phi(d_-), \end{aligned}$$

where ϕ/Φ denotes the standard normal probability/cumulative density function and $d_\pm = \frac{\ln\left(\frac{K}{S_t}\right) \pm (r - \frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{T-t}}$. The final expression is also known as Black-Scholes-Formula. The payoff of buying a call option and selling a put option with same maturity T and strike price K equals the payoff at maturity of buying one unit of the stock and $e^{-r(T-t)}K$ units of the risk free bond. This property can be used to derive the price of a put option. Since we assume no-arbitrage, it follows

$$C_{call}(S_t, K, T, t) - C_{put}(S_t, K, T, t) = S_t - e^{-r(T-t)}K \tag{2.3}$$

This property is called Put-Call-Parity and leads to a put price of

$$C_{put}(S_t, K, T, t) = Ke^{-r(T-t)}\Phi(-d_-) - S_t\Phi(-d_+)$$

2.3.2 Implied Volatility

An option pricing model, such as the previously introduced model, uses a variety of inputs to derive a theoretical value for an option. These inputs vary depending on the type of option being priced and the pricing model used. One parameter, which most models depend on is an estimate of the volatility σ of the underlying price process. Economically, it is easy to argue that the option price, which can only have a positive payoff, increases in the volatility as the underlying is more likely to show extreme outcomes.
Let C be the value of an option such that $C = f(\sigma, \cdot)$ for a function f and the volatility σ of the underlying asset. As f is increasing in σ, the option price C is invertible with respect to the volatility σ by the inverse function theorem. Hence, it is possible to backtrack characteristics of the underlying from option prices. This concept is called implied volatility, $\sigma_{imp} = f^{-1}(C, \cdot)$. In general, pricing models do not have a closed-form solution for σ_{imp}. Usually, root-finding methods are used to solve the problem. As prices change quickly in modern financial markets, it is important to compute implied volatilities in an extremely low amount of time. The most efficient methods are usually gradient based and require an estimate of $\frac{\partial C}{\partial \sigma}$. While the BS model has an analytical form for the partial derivatives, most models do not. Liu, Oosterlee, and Bohte (2019) show that neural networks can be used as efficient tools to calculate implied volatilities, which gives another motivation to focus on deep learning methods in financial mathematics.
The implied volatility of an option can be used as a substitute for the option price as there is a one-to-one mapping between both. It can be seen as a

more useful measure for the relative value of an option as it is mostly scale-independent while the option price highly depends on the magnitude of the underlying.
Even though the Black-Scholes-Merton model implies constant implied volatilities, this does not hold true in empirical studies. When applying the Black-Scholes implied volatility on realised option prices, a quadratic relation between strike price and volatility is observed. This phenomenon is usually explained by the following observation. Empirical asset returns show a strong

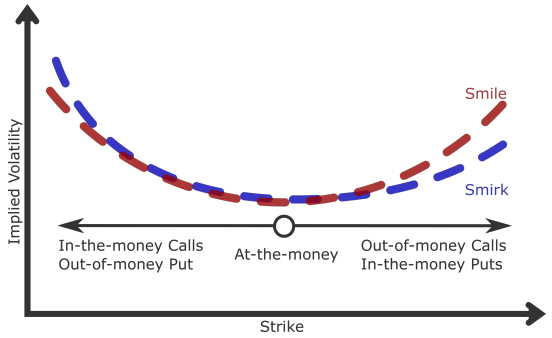


FIGURE 2.1: Exemplary Illustration of a Volatility Smile and Smirk

overkurtosis. As extreme returns are more likely to occur than expected by normally distributed returns, deep in-of-the-money and deep-out-of-money options are priced with a premium. While short-term equity and currency (forex) options tend to align with a characteristic volatility smile, a different phenomenon is observed for index options and long-term equity options: The implied volatilities for in-the-money puts and out-of-the-money calls are higher than the implied volatilities of out-of-the-money puts and in-the-money calls. This pattern, known as volatility smirk, is usually explained by two factors. On the on hand, the risk-aversion of long-term investors. Out-of-money puts are bought as insurance against bearish markets, leading to a increased demand of such options. Interestingly, the volatility smirk occurred first after the crash of 1987 (Black Monday), which led to a greater appreciation of large jumps in asset prices and an increased fear of substantial downward movements. On the other hand, in-the-money calls are a popular tool among investors that believe in rising prices. Such calls reassemble its underlying payoff structure but offer leverage and lead to higher returns-on-equity. Both lines of argument explain the increased demand for low strike options and the skewed pattern of a volatility smirk. On modern exchange markets, options are often quoted in terms of implied volatility instead of the price, which again stresses the importance of the concept.

Chapter 3

Heston-Nandi-GARCH Model

The development of a pricing model for European style options by Black and Scholes (1973) and Merton (1973) was crucial for the analysis and hedging of such assets. However, the simple structure does not reflect the behaviour of modern financial markets. Neither normal distributed returns nor a time invariant volatility can be observed in financial time-series. While the BS model implies a flat volatility surface, real volatility surfaces tend to depend on the time to maturity as well as the strike of an option. To account for the complex structure of the underlying volatility process, researchers developed a variety of different solutions. Engle's generalised auto-regressive conditional heteroscedasticity (GARCH) models proved helpful and reasonable in an economic setting (cf. Engle, 1982).
Heston and Nandi (2000) present a discrete-time model preserving the beneficial structure of Heston's continuous-time market model, while tackling previously mentioned problems by modelling the volatility via GARCH-like processes. They introduced the first discrete-time stochastic volatility which allows to price options with a closed-form solution. Heston and Nandi (2000) point out that in comparison to continuous-time stochastic volatility models, a GARCH-like structure allows to filter the volatility process from observed returns. Therefore the model provides the first framework to estimate option prices and calibrate parameters solely on the basis of historic observations.

3.1 Model Construction

The following assumptions are made in the HNG model. A discrete-time market with equally spaced trading dates is considered. For simplicity we assume daily observations. Secondly, the yearly risk-free rate r is constant and discount rates are continuously compounded. The market is frictionless market which no transaction costs and lending and borrowing is possible at unlimited amounts. No assumptions made on trading strategies, e.g. no

limitation in short selling nor to the amount of stocks hold or traded. Hence, we are in the setting of Chapter 2, where the Numéraire $B_t = S_t^0 := \exp(rt)$ and $t \in \{0, \dots, T\}$. The market consists of one additional tradeble asset with price process $(S_t)_t$ on the filtered probability space $(\Omega, \mathcal{F}, (\mathcal{F}_t)_t, \mathbb{P})$, where $(F_t)_t$ is generated by $(S_t)_t$. The log-return process $(Y_t)_t$ follows the dynamic (3.1)

$$\begin{cases} Y_t &= \ln(S_t) - \ln(S_{t-1}) = r + \lambda h_t + \sqrt{h_t} z_t \\ h_t &= \omega + \sum_{i=1}^{q} \alpha_i \left(z_{t-1} - \gamma_i \sqrt{h_{t-1}} \right)^2 + \sum_{i=1}^{p} \beta_i h_{t-1} \\ S_0 &= s \in \mathbb{R}^+ \\ h_0 &= \eta \in \mathbb{R}^+ \end{cases} \tag{3.1}$$

where $z_t \overset{iid}{\sim} \mathcal{N}(0,1), t \in \{1, \dots, T\}, \omega, \lambda \in \mathbb{R}, \alpha, \gamma \in R^q, \beta \in \mathbb{R}^p$ and $p, q \in \mathbb{N}$. In this model h_t represents the conditional variance of the returns which has GARCH(p,q) structure and is $\mathcal{F}_t$-predictable. Furthermore the conditional variance h_t appears in the mean as premium and the expected returns depend on the level of risk. The work of Hansen and Lunde (2005) indicates that the case of $p = q = 1$ is sufficient to provide good out-of-sample forecasts. Hence, we consider HNG(1,1) model from now on, which is given by (3.2)

$$\begin{cases} Y_t &= \ln(S_t) - \ln(S_{t-1}) = r + \lambda h_t + \sqrt{h_t} z_t \\ h_t &= \omega + \alpha \left(z_{t-1} - \gamma \sqrt{h_{t-1}} \right)^2 + \beta h_{t-1} \\ S_0 &= s \in \mathbb{R}^+ \\ h_0 &= \eta \in \mathbb{R}^+ \end{cases} \tag{3.2}$$

for $t \in \{1, \dots, T\}$. Positivity of $(h_t)_t$ can be assured by assuming $\alpha \geq 0$, $\beta \geq 0$ and $\omega > 0$. In this model the conditional expectation of returns equals

$$\begin{aligned} \mathbb{E}\left[Y_t \middle| \mathcal{F}_{t-1} \right] &= r + \lambda \, \mathbb{E}\left[h_t \middle| \mathcal{F}_{t-1} \right] \\ &= r + \lambda h_t. \end{aligned}$$

and for the conditional variance of returns holds

$$\mathbb{V}\left[Y_t \middle| \mathcal{F}_{t-1} \right] = \mathbb{E}\left[h_t z_t^2 \middle| \mathcal{F}_{t-1} \right] = h_t.$$

for $t \in \{1, \dots, T\}$. Similar to Engle's GARCH(1,1) process, the conditional volatility $(h_t)_t$ is weakly stationary with finite mean and variance if and only

if $|\alpha\gamma^2 + \beta| < 1$.

Proof. Assuming weak stationarity of $(h_t)_t$ we find

$$\begin{aligned}\mathbb{E}[h_t] &= \omega + \beta\,\mathbb{E}[h_{t-1}] + \alpha\,\mathbb{E}[z_{t-1}^2 + \gamma^2 h_{t-1} - 2\gamma z_{t-1}\sqrt{h_{t-1}}] \\ &= w + \beta\,\mathbb{E}[h_t] + \alpha + \alpha\gamma^2\,\mathbb{E}[h_t]\end{aligned}$$

for all $t \in \{1, \dots, T\}$ and we obtain $\mathbb{E}[h_t] = \frac{\omega+\alpha}{1-\beta-\alpha\gamma^2}$.
Due to the non-linear dependency of h_t on h_{t-1} the proof necessity does not follow analogously to stationarity proofs of other GARCH-like models. Hence we refer to Brautmeier (2020). □

Furthermore this directly implies that $(Y_t)_t \in L^2(\Omega)$ if $(h_t)_t$ is weakly stationary, as for all $t \in \{1, \dots, T\}$ holds

$$\begin{aligned}\mathbb{V}[Y_t] &= \mathbb{E}\left[\mathbb{V}\left[Y_t \middle| \mathcal{F}_{t-1}\right]\right] + \mathbb{V}\left[\mathbb{E}\left[Y_t \middle| \mathcal{F}_{t-1}\right]\right] \\ &= \mathbb{E}[h_t] + \mathbb{V}[r + \lambda h_t] \\ &= \mathbb{E}[h_t] + \lambda^2(\mathbb{E}[h_t^2] - \mathbb{E}[h_t]^2)\end{aligned}$$

In addition to Engle's GARCH Heston and Nandi account for leverage effects via the skewness parameter γ. The leverage effect refers to the generally negative correlation between an asset return and its volatility. It is not only important because it is observed in nearly all modern financial assets but furthermore a recent study by Roch (2017) relates the existence of leverage to the occurrence of bubbles. Hence, allowing leverage effect is of high importance for modern market models. The conditional covariance between Y_t and h_{t+1} is

$$\operatorname{Cov}\left[h_{t+1}, Y_t \middle| \mathcal{F}_{t-1}\right] = \operatorname{Cov}\left[h_{t+1}, \ln(S_t) \middle| \mathcal{F}_{t-1}\right] = -2\alpha\gamma h_t.$$

A positive γ leads to a leverage effect. The calculation of $\mathbb{V}[Y_t]$, proof of previous claims as well as further properties of the HNG(1,1) model can be found in Appendix A.
Under GARCH-like models the market is incomplete. Hence, by Theorem 2.17 there does not exist an unique risk-neutral measure to price in the HNG model. Heston and Nandi (2000) provide one risk-neutral process, which has an identical GARCH form as (3.2) with replacing λ by $\lambda^* = -\frac{1}{2}$ and γ by $\gamma^* = \lambda + \gamma + \frac{1}{2}$. This specific measure is obtained, by additionally assuming that value process of a call option with one period to expiration obeys the BS

formula (2.2). This results in the following risk-neutral model:

$$\begin{cases} Y_t &= \ln(S_t) - \ln(S_{t-1}) = r - \frac{1}{2}h_t + \sqrt{h_t}z_t^* \\ h_t &= \omega + \alpha\left(z_{t-1}^* - \gamma^*\sqrt{h_{t-1}}\right)^2 + \beta h_{t-1} \\ S_0 &= s \in \mathbb{R}^+ \\ h_0 &= \eta \in \mathbb{R}^+ \end{cases} \tag{3.3}$$

where $z_t^* \overset{iid}{\sim} \mathcal{N}(0,1)$ under the risk neutral measure $\mathbb{Q}$. This change in measure has two important implications. The expected return of the underlying equals, similar to the BS model, the risk-free rate r. and, given a positive λ, it holds $\gamma < \gamma^*$. This implies that in the risk-neutral setting, a higher average volatility and a stronger degree of leverage and persistence is observed. A more detailed analysis can be found in Chapter 4.2 of Chorro, Guégan, and Ielpo (2015).
Under the introduced risk-neutral measure $\mathbb{Q}$ the price C of a European call option with Strike K at time t and expiration date T is given by

$$\begin{aligned} \mathbb{E}_{\mathbb{Q}}\left[(S_T - K)^+ | \mathcal{F}_t\right] &= \frac{1}{2}e^{r(T-t)}S_t + \frac{1}{\pi}\int_0^\infty \mathrm{Re}\left(\frac{K^{-i\phi}f_t(i\phi+1)}{i\phi}\right)d\phi \\ &\quad - K\left(\frac{1}{2} + \frac{1}{\pi}\int_0^\infty \mathrm{Re}\left(\frac{K^{-i\phi}f_t(i\phi)}{i\phi}\right)d\phi\right) \\ C_{call}(S_t, K, T, t) &= e^{-r(T-t)}\,\mathbb{E}_{\mathbb{Q}}\left[(S_T - K)^+ | \mathcal{F}_t\right] \end{aligned} \tag{3.4}$$

where f_t denotes the conditional generating function of Y_t under $\mathbb{Q}$. In the case of $p = q = 1$, Chorro, Guégan, and Ielpo (2015) provide a formula to calculate f_t recursively using the following expressions

$$\begin{aligned} f_t(\phi) = \mathbb{E}_{\mathbb{Q}}\left[S_T^\phi | \mathcal{F}_t\right] &= S_t^\phi \exp\left(A_t + B_t h_{t+1}\right) \\ A_t &= A_{t+1} + r\phi + \omega B_{t+1} - \frac{1}{2}\ln(1 - 2\alpha B_{t+1}) \\ B_t &= -\frac{1}{2}\phi + \beta B_{t+1} + \frac{\frac{1}{2}\phi^2 - 2\alpha(\gamma^*)^2 B_{t+1}\phi + \alpha(\gamma^*)^2 B_{t+1}}{1 - 2\alpha B_{t+1}} \end{aligned} \tag{3.5}$$

with terminal condition $A_T = B_T = 0$. Furthermore, a corresponding put price can be easily derived using put-call-parities as introduced in equation (2.3).
Since we will need to calculate millions of option prices in Chapter 4, it is important to think about efficient calculation of (3.4). The easiest way to

save computational resource is by simplifying the expression and thereby reducing numerical errors when estimating the integrals. Simplifying both integrals leads to one integral with an integrand of equal complexity, resulting in approximately half the number of total operations:

$$\frac{1}{2}e^{r(T-t)}S_t + \frac{1}{\pi}\int_0^\infty \mathrm{Re}\left(\frac{K^{-i\phi}f_t(i\phi+1)}{i\phi}\right)d\phi - K\left(\frac{1}{2} + \frac{1}{\pi}\int_0^\infty \mathrm{Re}\left(\frac{K^{-i\phi}f_t(i\phi)}{i\phi}\right)d\phi\right)$$
$$= \frac{1}{2}(e^{r(T-t)}S_t - K) - \frac{1}{\pi}\int_0^\infty \frac{1}{\phi}\mathrm{Re}\left(iK^{-i\phi}(f_t(i\phi+1) - Kf_t(i\phi))\right)d\phi$$

This also decreases the number of operations of division needed which leads to a higher accuracy while approximating the integral. There are several further approaches to optimise this calculation, which allow write pricing formula (3.4) as an Inverse Laplace or Fourier transform (see for example Hubalek, Kallsen, and Krawczyk (2006)).
One property of the HNG(1,1) that will be heavily utilised in Chapter 4 is the positive homogeneity of the model.

Theorem 3.1. Let $\lambda > 0$ be a positive real number. For a European call option with strike K, maturity T and an underlying asset value S_t at time t holds

$$\lambda C(S_t, K, T, t) = C(\lambda S_t, \lambda K, T, t)$$

Proof. Kreps and Wallis (1997) prove that an option pricing function is homogeneous of degree one with respect to (S_t, K) if and only if the risk-free pricing measure $\mathbb{Q}_t$ at time t does not depend on S_t. Heston and Nandi (2000) show that for the risk adjusted probability density function q_t of $\ln(S_t)$ under $\mathbb{Q}$ holds

$$q_t(x) = \frac{\exp(x)p_t(x)}{f_t(1)} = \frac{p_t(x)}{\exp(A_t + B_t h_{t+1})},$$

where p_t denotes the real world probability density function $\ln(S_t)$ under $\mathbb{P}$ and A_t, B_t as in (3.5). Hence, we conclude the call pricing function is homogeneous for the HNG(1,1) model. An alternative proof by explicitly calculating the homogeneity property is provided in Appendix A. □

We want to highlight the usefulness of the introduced homogeneity property. When analysing derivative prices, it allows to normalise the underlying price $S_t = 1$. Therefore it effectively reduces the number of derivative parameters by one as only the moneyness $\frac{S_t}{K}$ and maturity of an option contract matters. In particular comparing derivative prices is significantly simplified as the

prices can be expressed with respect to the moneyness.
In this section, we introduced of the HNG model, derived a closed-form pricing formula formula for option prices and provided some useful properties of the HNG(1,1) model. In the following sections, we will apply this theoretical fundamentals.

3.2 Empirical Performance

So far the theoretical foundations of option pricing and the HNG model were tackled, but ultimately one is interested in applying these on real option prices. Hence, we compare two approaches to calibrate the HNG(1,1) model from observed market quotes. First, a Maximum Likelihood Estimation (MLE) approach based historical returns and secondly, a calibration approach to fit parameters to observed option prices. At this point, several beneficial properties of the HNG model take effect: The GARCH-like structure allows for easy estimation of parameters based on historical returns, while the exponentially affine characteristic function still leads to fast option pricing. The empirical study in this section is based on S&P 500 returns as well as vanilla European-style call options on the S&P 500. These contracts usually show a high liquidity in comparison to call option contracts traded on other assets. This leads to reduced trading noise and less problems generated by the market microstructure, which ensures a better framework for the model calibration.

3.2.1 Maximum Likelihood Estimation

From Section 3.1 we know that the log returns under $\mathbb{P}$ satisfy $Y_t = r + \lambda h_t + \sqrt{h_t} z_t$ for all $t \in \{1, \dots, T\}$, where $z_t \overset{iid}{\sim} \mathcal{N}(0,1)$. This implies a distribution of Y_t conditional on $\mathcal{F}_{t-1}$ of $\mathcal{N}(r_t + \lambda h_t, h_t)$. Hence, the Log-Likelihood function $\mathcal{L}$ of the HNG(1,1) model based on observations $y_1, \dots, y_T$ is

$$
\begin{aligned}
&\mathcal{L}\left(y_1, \dots, y_T | (\omega, \alpha, \beta, \gamma, \lambda, h_0)\right) \\
:=& \ln(f(y_1, \dots, y_T | \omega, \alpha, \beta, \gamma, \lambda, h_0)) \\
=& \ln\left(\prod_{t=1}^{T} f_t(y_t | \omega, \alpha, \beta, \gamma, \lambda, h_0)\right) \\
=& \sum_{t=1}^{T} \ln(f_t(y_t | \omega, \alpha, \beta, \gamma, \lambda, h_0)) \\
=& -\frac{T}{2}\ln(2\pi) - \frac{1}{2}\sum_{t=1}^{T}\left(\ln(h_t) + \frac{(y_t - r - \lambda h_t)^2}{h_t}\right),
\end{aligned}
$$

where f/f_t correspond to the joint/single probability density functions of $Y_1, \ldots, Y_T$. The Log-Likelihood function $\mathcal{L}$ can be calculated recursively using

$$h_{t+1} = \omega + \beta h_t + \alpha \frac{(y_t - r - (\lambda + \gamma)h_t)^2}{h_t}, \tag{3.6}$$

which is obtained directly from (3.2).
A reliable set of estimated parameters can be estimated by numerically solving the maximisation problem (3.7) .

$$\begin{aligned} &\max_{\omega,\alpha,\beta,\gamma,\lambda,h_0} && \mathcal{L}\left(y_1, \ldots, y_T | (\omega, \alpha, \beta, \gamma, \lambda, h_0)\right) \\ &\text{subject to} && \begin{cases} \alpha\gamma^2 + \beta &< 1 \\ \alpha &\geq 0 \\ \beta &\geq 0 \\ \omega &> 0 \\ h_0 &> 0 \end{cases} \end{aligned} \tag{3.7}$$

On every Wednesday between 2010 and 2018, the daily historical returns of S&P 500 in the last 2520 trading days (10 years) are used to estimate parameters via optimisation problem (3.7) in a moving window approach. The interest rate r is taken as the average yearly T-Bill rates over the estimated period. By testing several optimisation algorithms, the most robust results are given by the interior-point algorithm. All optimisations run on MATLAB R2020a using the Globalised Optimisation Toolbox. We use the Globalsearch algorithm with the following specifications:

i. XTolerance: 10^{-9}

ii. Function Tolerance: 10^{-9}

iii. NumTrailPoints: 2000

iv. Default settings in all other configuration options.

The results are summarised in Table 3.1.

Several challenges occur in this optimisation problem. Until now, most researchers do not optimise h_0 but take it as an external parameter. The most common approach is to use the historical return variance as well as the usage of the unconditional expectation of $(h_t)_t$. We do not share this perception for a number of reasons. First, empirically these approaches underestimates the observed (via backtracking estimated) initial variance. Secondly, the model is extremely sensitive on the initial conditional variance h_0 if the size

θ	2010	2011	2012	2013	2014	2015	2016	2017	2018
ω	$4.4871e-12$ $(2.9653e-12)$	$5.2773e-12$ $(3.3908e-12)$	$4.5585e-12$ $(3.6039e-12)$	$2.2018e-12$ $(1.0564e-12)$	$4.0732e-12$ $(4.1726e-12)$	$4.2019e-12$ $(5.9165e-12)$	$5.4833e-08$ $(7.0281e-08)$	$1.0753e-08$ $(3.9681e-08)$	$2.4323e-08$ $(5.3003e-08)$
α	$2.8645e-06$ $(1.6263e-07)$	$3.0251e-06$ $(1.4964e-07)$	$3.3309e-06$ $(6.9975e-08)$	$3.4392e-06$ $(7.5825e-08)$	$3.2333e-06$ $(9.6807e-08)$	$3.8471e-06$ $(4.3881e-07)$	$5.0427e-06$ $(1.8607e-07)$	$4.7757e-06$ $(5.0985e-07)$	$4.2730e-06$ $(6.2838e-07)$
β	0.7557 (0.0087)	0.7817 (0.0088)	0.7782 (0.0036)	0.7762 (0.0032)	0.7518 (0.0074)	0.7363 (0.0065)	0.7175 (0.0056)	0.7197 (0.0043)	0.7333 (0.0125)
γ	281.1041 (14.0375)	255.9472 (9.1781)	243.9845 (3.5547)	239.2679 (3.2157)	262.6245 (5.3793)	247.9217 (12.7544)	220.8570 (3.6400)	227.3857 (15.1128)	232.9116 (19.5619)
λ	-0.6690 (0.1861)	0.1138 (0.1653)	0.8717 (0.4157)	1.6130 (0.1274)	1.6335 (0.1329)	1.5308 (0.1714)	1.1750 (0.1260)	1.1505 (0.1079)	1.8159 (0.5644)
h_0	$1.7769e-04$ $(1.0680e-04)$	$1.5068e-04$ $(9.0268e-05)$	$2.8818e-04$ $(2.0276e-04)$	$1.6060e-04$ $(1.2234e-04)$	$4.8486e-05$ $(2.5280e-05)$	$3.9643e-05$ $(3.4890e-05)$	$3.4331e-05$ $(2.8584e-05)$	$1.1573e-04$ $(7.9826e-05)$	$1.8279e-03$ $(2.0366e-03)$

TABLE 3.1: MLE average estimates and their standard errors reported in brackets of S&P 500 returns. Estimated on every Wednesday with a 10-year moving window. The standard errors are computed as the sample standard deviation over each year.

of available data points is low. In these cases estimating h_0 leads to more robust estimations. To support this claim, we repeat the estimation task (3.7) but fix $h_0 = \mathbb{E}[h_t] = \frac{\omega+\alpha}{1-\beta-\alpha\gamma^2}$ as unconditional expectation of $(h_t)_t$. Here, the values of $\omega, \alpha, \beta, \gamma$ are chosen as the corresponding initial parameters. The results are summarised in Table 3.2. As one can see by comparing the

θ	2010	2011	2012	2013	2014	2015	2016	2017	2018
ω	$1.2975e-08$ $(9.3546e-08)$	$3.1068e-12$ $(1.4067e-12)$	$2.1492e-12$ $(7.3306e-13)$	$1.8801e-12$ $(4.4392e-13)$	$1.8372e-12$ $(3.9309e-13)$	$9.1863e-10$ $(6.6076e-09)$	$1.1265e-07$ $(1.9704e-07)$	$2.8398e-08$ $(7.3100e-08)$	$1.5230e-08$ $(4.5894e-08)$
α	$3.2426e-06$ $(5.1624e-07)$	$3.2882e-06$ $(5.6062e-07)$	$3.6342e-06$ $(3.8884e-07)$	$3.6247e-06$ $(3.7563e-07)$	$3.2728e-06$ $(2.1511e-07)$	$3.9255e-06$ $(4.8846e-07)$	$5.0398e-06$ $(3.0156e-07)$	$4.9122e-06$ $(5.6934e-07)$	$5.1993e-06$ $(9.4273e-07)$
β	0.7641 (0.0160)	0.7873 (0.0143)	0.7820 (0.0068)	0.7807 (0.0089)	0.7541 (0.0097)	0.7391 (0.0078)	0.7146 (0.0146)	0.7218 (0.0072)	0.7320 (0.0112)
γ	259.9067 (27.2832)	243.5280 (27.3660)	231.3805 (15.9869)	230.6035 (17.5080)	259.8050 (11.5973)	243.9296 (15.6664)	222.1951 (10.6345)	222.9149 (16.8273)	209.6187 (22.9595)
λ	-0.6690 (0.1867)	0.1158 (0.1646)	0.8707 (0.4152)	1.6144 (0.1284)	1.6355 (0.1332)	1.5327 (0.1708)	1.1768 (0.1267)	1.1490 (0.1073)	1.9568 (0.5959)
h_0	$1.2971e-04$ $(8.7116e-05)$	$1.6015e-04$ $(1.0235e-04)$	$8.9183e-05$ $(4.3327e-05)$	$6.1203e-05$ $(3.1452e-05)$	$6.5388e-05$ $(3.8166e-05)$	0.0001 $(6.5918e-05)$	$9.8876e-05$ $(7.2937e-05)$	$4.1014e-05$ $(2.3070e-05)$	$1.1552e-04$ $(9.1559e-05)$

TABLE 3.2: MLE average estimates with fixed $h_0 = \mathbb{E}[h_t]$ and their standard errors reported in brackets of S&P 500 returns. Estimated on every Wednesday with a 10-year moving window. The standard errors are computed as the sample standard deviation over each year.

results in Table 3.1 and Table 3.2, the sample standard deviation is decreased for every parameter, but h_0 and λ. The standard deviation of estimated h_0 and λ is comparable between the approaches. Especially for ω and α, the estimates show a significantly lower standard deviation. For β and γ, the sample yearly standard deviation is at least 40% lower when estimating h_0.

The improvements are significantly stronger for the option price calibration introduced in the following section. For a detailed exposition of this claim, we refer to Badescu, Brautmeier, Grigoryeva, and Orthega (2020).
Another problem occurring when optimising (3.7) is, that the magnitude of estimated parameters differs between α, ω, h_0 and β, γ. While an average α is of magnitude 10^{-6}, γ^* tends to be around 250. The magnitude of multi-dimensional optimisation algorithms do not allow for different step sizes in each dimension and only consider the norm of each step. Hence, small parameters are systemically discriminated. This leads to inelastic results and only small movement in these parameters. This results in a strong (artificial) dependence on the initial values, even though the optimization is usually not relient on the choice of initial parameters. We overcome this challenge by rescaling the parameters to the same magnitude before each optimisation step.

3.2.2 Calibration with respect to Option Prices

As Chorro, Guégan, and Ielpo (2015) point out, calibration of a model under $\mathbb{Q}$ using option prices, leads to better results in terms of option pricing than using a model with MLE based on historically returns estimated parameters. Hence, we proceed as follows: We consider all call option contracts traded on Wednesdays between Monday, 04.01.2010, and Friday 28.12.2018, whose

i. Maturity is between 8 and 250 days,

ii. Moneyness is between 0.9 and 1.1, and

iii. Open interest as well as trading volume is over 100.

The data is obtained from the Wharton Research Data Services. We divide the data into different groups of moneyness and maturities and illustrate the basic features in Table 3.3. On every Wednesday between 2010 to 2019, with N observed call prices $\{P_i^{obs}(S_i, K_i, M_i, r_i)\}_{i\in\{1,\dots,N\}}$ the optimisation problem (3.8) is solved.

$$\begin{aligned} \min_{\omega,\alpha,\beta,\gamma^*,h_0} \quad & \tfrac{1}{N}\sum_{i=1}^{N}\left(P(\theta, S_i, K_i, M_i, r_i) - P_i^{obs}(S_i, K_i, M_i, r_i)\right)^2 \\ \text{subject to} \quad & \begin{cases} \alpha(\gamma^*)^2 + \beta < 1 \\ \alpha \geq 0 \\ \beta \geq 0 \\ \omega > 0 \\ h_0 > 0 \end{cases} \end{aligned} \tag{3.8}$$

		Moneyness S_0/K						Across
	Maturities	[0.900, 0.950]	(0.950, 0.975]	(0.975, 1.000]	(1.000, 1.025]	(1.025, 1.050]	(1.050, 1.100]	Moneyness
Number of Contracts	$\mathbf{8 \leq T \leq 30}$	5847	9137	14852	5362	860	312	36370
	$\mathbf{30 < T \leq 80}$	4077	4418	4432	2683	461	227	16298
	$\mathbf{80 < T \leq 180}$	1674	1019	1081	816	211	146	4947
	$\mathbf{180 < T \leq 250}$	593	338	474	415	106	67	1993
Across Maturities		12191	14912	20839	9276	1638	752	59608
Average Prices	$\mathbf{8 \leq T \leq 30}$	1.051	3.083	10.696	32.322	73.111	119.694	12.832
	$\mathbf{30 < T \leq 80}$	4.730	11.213	30.083	54.253	89.829	132.246	25.718
	$\mathbf{80 < T \leq 180}$	17.971	38.017	65.160	86.598	118.064	163.207	52.287
	$\mathbf{180 < T \leq 250}$	40.296	69.839	99.497	113.667	146.267	191.009	85.367
Across Maturities		6.514	9.392	19.664	47.079	88.341	138.285	22.055
Average Implied Volatilities	$\mathbf{8 \leq T \leq 30}$	0.151	0.118	0.107	0.130	0.171	0.220	0.123
	$\mathbf{30 < T \leq 80}$	0.126	0.114	0.127	0.148	0.172	0.198	0.129
	$\mathbf{80 < T \leq 180}$	0.130	0.139	0.153	0.162	0.178	0.195	0.146
	$\mathbf{180 < T \leq 250}$	0.147	0.157	0.165	0.177	0.184	0.193	0.163
Across Maturities		0.140	0.119	0.115	0.140	0.173	0.206	0.128

TABLE 3.3: Descriptive statistics of the option pricing dataset obtained from Wharton Data (Wednesdays, 2010-2018). The option are divided into different groups of moneyness and maturities.

where $\theta = (\alpha, \beta, \gamma^*, \omega, h_0)$ and $P(\theta, \cdot)$ corresponds to the model price for call options according to formula (3.4). The used interest rates $(r_i)_i$ are interpolated from daily treasury yield curve rates. Again, several different optimisation algorithms are tested and the interior-point algorithm leads to the best results. All optimisations run on MATLAB R2020a using the Optimisation Toolbox. We use fmincon with the following specifications:

i. TolFun: 10^{-6}

ii. TolX: 10^{-9}

iii. MaxIterations: 4000

iv. MaxFunctionEvaluations: 2500

v. Default settings in all other configuration options.

Analogously to problem (3.7), one should account for the magnitude of parameters. Table 3.4 and Table 3.5 show the results of (3.8) with calibration of h_0 and with fixed $h_0 = \mathbb{E}_Q[h_t]$. As highlighted in Section 3.2.1, optimising h_0 increases the stability of the calibrated parameters. To measure the goodness of fit of the calibration, we use the mean squared error (MSE) of option prices, which is used as goal function in the optimisation (3.8). Furthermore, we report the mean average percentage error (MAPE) of option prices

$$\text{MAPE}(\hat{\theta}) == \frac{1}{N} \sum_{i=1}^{N} \left| \frac{P(\hat{\theta}, S_i, K_i, M_i, r_i) - P_i^{obs}(S_i, K_i, M_i, r_i)}{P_i^{obs}(S_i, K_i, M_i, r_i)} \right|$$

and the log-likelihood of the option prices (OptLL) introduced by Trolle and Schwartz (2009)

$$\begin{aligned}\text{OptLL}(\hat{\theta}) = &-\frac{1}{2}N - \frac{1}{2}N\ln(2\pi) + \sum_{i=1}^{N}\ln(\nu_i) \\ &+ \sum_{i=1}^{N}\log\left(\frac{P(\hat{\theta}, S_i, K_i, M_i, r_i) - P_i^{obs}(S_i, K_i, M_i, r_i)}{\nu_i}\right)^2,\end{aligned}$$

where $\nu_i = \frac{\partial P_i^{obs}}{\partial \sigma_{BS}^i}$ denotes the Option Vega of the observed price P_i^{obs}, which measures the sensitivity of the option price to the (implied) volatility. Calibrating h_0 leads to an overall performance increase. This conclusion holds for all error measures and through the whole observation windows. This finding indicates, that the calibration of h_0 does not only increase the stability of the estimates but also the in-sample fit. Nonetheless, we remark that this could also be a result of the increased number of optimised parameters and we did not check for significance of the performance increase.

θ	2010	2011	2012	2013	2014	2015	2016	2017	2018
ω	$1.0488e-07$ ($4.3237e-07$)	$5.8246e-07$ ($9.9623e-07$)	$2.5115e-07$ ($5.7761e-07$)	$1.6648e-07$ ($4.5215e-07$)	$2.3430e-07$ ($4.5167e-07$)	$7.7768e-08$ ($2.6235e-07$)	$1.1626e-07$ ($2.7833e-07$)	$8.2065e-08$ ($3.2339e-07$)	$7.6453e-08$ ($3.3182e-07$)
α	$8.4165e-06$ ($6.7016e-06$)	$4.4508e-06$ ($2.4687e-06$)	$2.8014e-06$ ($1.4378e-06$)	$2.5121e-06$ ($1.4269e-06$)	$2.5227e-06$ ($2.2280e-06$)	$2.9788e-06$ ($1.3795e-06$)	$2.2257e-06$ ($9.4056e-07$)	$1.3120e-06$ ($7.8262e-07$)	$1.4577e-06$ ($7.2948e-07$)
β	0.6871 (0.1397)	0.5490 (0.2245)	0.7000 (0.1376)	0.7605 (0.1253)	0.6585 (0.1859)	0.5583 (0.1226)	0.5809 (0.1377)	0.6908 (0.1482)	0.6496 (0.1324)
γ^*	197.5895 (79.0995)	347.0532 (210.7790)	349.9407 (182.3969)	311.1355 (155.5853)	419.7989 (230.8533)	397.9111 (128.9083)	439.0339 (115.1693)	454.7184 (207.7471)	502.6705 (132.3138)
h_0^Q	$1.2420e-04$ ($7.7985e-05$)	$1.7303e-04$ ($1.3864e-04$)	$7.7115e-05$ ($3.0317e-05$)	$4.6121e-05$ ($2.5813e-05$)	$4.3171e-05$ ($3.8513e-05$)	0.0001 ($4.8647e-05$)	$6.1981e-05$ ($4.8685e-05$)	$1.7690e-05$ ($1.1101e-05$)	$6.7046e-05$ ($5.9643e-05$)
MSE	0.3344	0.4992	0.3164	0.1865	0.2756	0.4952	0.5942	0.8425	1.4562
MAPE	0.1024	0.1053	0.1555	0.1366	0.1616	0.1886	0.1722	0.2196	0.1849
OptLL	207.0992	216.2553	244.4436	345.9152	369.4851	433.9732	544.1547	617.0931	679.5187

TABLE 3.4: Average option price calibrated parameters and their standard errors reported in brackets of S&P 500 European-style call options with respect to MSE. Estimated at every Wednesday. The standard errors are computed as the sample standard deviation over each year.

3.2.3 Estimation vs. Calibration

To analyse the differences of both introduced approaches (historical estimation and option calibration), we proceed as follows:
First, we compute the option prices using the parameters obtained in the MLE procedure based on historical returns and compare them to the observed

θ	2010	2011	2012	2013	2014	2015	2016	2017	2018
ω	$1.5697e-07$	$5.7491e-06$	$2.1523e-07$	$2.7628e-07$	$2.3732e-08$	$1.6110e-08$	$2.0066e-07$	$1.2489e-08$	$2.2192e-07$
	$(5.6788e-07)$	$(1.6332e-05)$	$(6.2554e-07)$	$(7.5402e-07)$	$(1.1658e-07)$	$(1.1044e-07)$	$(1.2615e-06)$	$(6.3331e-08)$	$(7.7929e-07)$
α	$1.4628e-05$	$1.0156e-05$	$9.7869e-06$	$7.8367e-06$	$7.9205e-06$	$9.5198e-06$	$8.2525e-06$	$5.0611e-06$	$9.1095e-06$
	$(1.0156e-05)$	$(9.5801e-06)$	$(8.8243e-06)$	$(7.6776e-06)$	$(5.5364e-06)$	$(3.9620e-06)$	$(5.2317e-06)$	$(3.5988e-06)$	$(6.2168e-06)$
β	0.5256	0.2930	0.4448	0.4003	0.2776	0.1191	0.2523	0.1647	0.1895
	(0.2334)	(0.3055)	(0.2970)	(0.3662)	(0.3251)	(0.1824)	(0.2875)	(0.2930)	(0.2762)
γ^*	206.9885	393.3237	321.9917	420.5704	386.1910	300.0024	317.9679	464.4275	343.7525
	(168.6065)	(361.2128)	(294.8291)	(320.3050)	(252.4873)	(61.9570)	(103.3799)	(201.6139)	(242.9145)
h_0^Q	$1.2504e-04$	$1.6094e-04$	$8.8020e-05$	$6.3516e-05$	$6.4968e-05$	$1.0677e-04$	$9.4593e-05$	$4.2065e-05$	$1.2042e-04$
	$(8.4350e-05)$	$(1.0127e-04)$	$(3.9993e-05)$	$(3.0169e-05)$	$(3.7802e-05)$	$(5.3934e-05)$	$(6.6163e-05)$	$(2.5624e-05)$	$(9.2499e-05)$
MSE	0.5696	2.7958	0.9089	1.1220	2.1939	2.3500	4.2735	8.5872	5.6765
MAPE	0.1199	0.1306	0.1811	0.1706	0.2157	0.2114	0.2289	0.3632	0.2216
OptLL	199.1370	198.2037	232.4967	308.1413	325.3681	406.4867	478.1218	511.1234	629.4424

TABLE 3.5: Average option price calibrated parameters with fixed $h_0 = \mathbb{E}_{\mathbb{Q}}[h_t]$ and their standard errors reported in brackets of S&P 500 European-style call options with respect to MSE. Estimated at every Wednesday. The standard errors are computed as the sample standard deviation over each year.

prices of them same day. Hereby, the estimated parameters are transformed to their risk neutral equivalent as proposed in Section 3.1. For the initial variance h_0 under $\mathbb{Q}$, the last update h_t obtained from formula (3.6) under $\mathbb{P}$ is used. Comparing the in-sample fit of MLE, illustrated in Table 3.6, and the option price calibration in Table 3.4, we find, that the latter approach leads to an improvement in every applied performance measure. At the same time the standard deviation of parameters, which are option price calibrated, is substantially higher. The MLE shows a particular poor fit in terms of MSE. This indicates that the MLE parameters do not lead to an accurate price of deep-in-the-money calls.

	2010	2011	2012	2013	2014	2015	2016	2017	2018
MSE	21.6631	37.0986	14.30646	72.3288	135.3291	128.5007	174.5775	377.5813	283.7601
MAPE	0.3008	0.3442	0.3359	0.5719	1.0299	1.49736	1.54036	2.2963	1.0283
OptLL	-136.4921	-136.4657	-186.6858	-201.1627	-182.8358	-191.3204	-247.6674	-239.7448	-341.8202

TABLE 3.6: Average performance for the in-sample option pricing of the MLE parameters on S&P 500 European-style call options. Estimated at every Wednesday.

Next, we focus on the forecasting performance of the two approaches. Ultimately, the out-of-sample performance is the true test for a calibrated model. To check for this, we use the historically estimated and option prices calibrated parameters to price options on the following Wednesday. Hereby, we consider the same option contracts as in previous sections. No further adjustments are made and the parameter are not updated in any kind. Besides the already

applied error measures, we check for several wide-know information criteria, namely Akaike (AIC), small sample adjusted Akaike (AICc) and Bayes (BIC), to measure the performance of each approach. The results are summarised in Table 3.7 and 3.8. The option calibration approach provides to a better out-of-sample performance than the historical estimation. The performance of the both approaches worsens with the time horizon, the effect is lower for the option calibration. Especially in the years following 2015, the option calibrated parameters lead to better forecasting results in comparison to the historically estimated parameters. Interestingly, the calibration forecasts are most stable in respect to the relative deviation.

	2010	2011	2012	2013	2014	2015	2016	2017	2018
MSE	25.1600	45.5099	17.7076	81.0171	148.2900	145.0361	197.4627	388.4116	302.6200
MAPE	0.3566	0.4147	0.3856	0.6144	1.1283	1.6699	1.6469	2.2636	1.0177
OptLL	−137.0339	−161.6806	−151.2927	−256.5975	−310.6469	−403.4363	−499.6559	−620.0705	−648.6887
AIC	147.7208	169.6806	162.3185	264.5975	318.6469	411.4363	507.6559	628.0705	683.7173
AICc	148.5249	170.4493	162.9532	265.0716	319.0970	411.7996	507.9429	628.3312	683.9361
BIC	295.5488	339.6501	325.5824	531.2467	639.5872	825.9617	1019.2740	1260.4996	1372.4488

TABLE 3.7: 1-week option price forecasting performance of MLE Parameters and $h_0^Q = h_t^P$

	2010	2011	2012	2013	2014	2015	2016	2017	2018
MSE	12.7466	27.2788	9.6810	7.4448	13.5402	16.4519	17.5702	7.9150	39.0847
MAPE	0.2236	0.2808	0.2448	0.2426	0.3006	0.3798	0.3724	0.2900	0.3300
OptLL	−104.5406	−129.6401	−125.8304	−169.0302	−193.4868	−245.2058	−337.7765	−312.4968	−489.4748
AIC	116.5904	139.6401	138.3470	179.0302	203.4868	255.2058	347.7765	322.4968	519.8696
AICc	117.8227	140.8171	139.3148	179.7501	204.1702	255.7560	348.2103	322.8906	520.1995
BIC	233.3149	279.6412	277.8755	360.6249	409.8403	514.2730	700.5059	650.4419	1046.0068

TABLE 3.8: 1-week option price forecasting performance of the option calibrated parameters

Finally, we focus on the difference between in-sample and out-of-sample performance and analyse the ability of the approaches to generalise. Comparing the in-sample fit with the 1-week forecasting performance of the option calibrated parameters, the MAPE is roughly twice as high each year in the 1-week forecasting task. Simultaneously, the MSE increases by factor 100. The varying degree of increase in MAPE and MSE indicates, that the calibrated parameters performs out-of-sample worse for in-the-money call, which have significantly higher price than at-the-money and out-of-money calls. Interestingly, the 1-week forecasting performance is in line with the in-sample performance for historically estimated parameters. We conclude that the estimated parameters tend to be more robust than their calibrated counterparts due to the large observation window, while the option price calibrated parameters lead to a

significantly higher accuracy in terms of option pricing.
Summarising, we find that the option price calibration performs better in terms of insample-fit as well as 1-week forecasting performance than the MLE based on historic returns. Furthermore, the additional optimisation of h_0 leads to more stable parameter estimates than the usage of a fixed h_0. Hence, the option price calibration with additional h_0 optimisation will be used as basis to generate the training set for our neural network approaches in Section 4.6 and as benchmark for the them in the conducted numerical experiments.

Chapter 4

Methodology

The previous chapters provide the mathematical and economical background for derivative pricing in discrete-time volatility models. This chapter finally links these models to deep learning methods.
First, we briefly introduce the concept of neural networks and lay the theoretical foundation to understand the design and function of these. Secondly, we present an approach on option pricing in the HNG(1,1) model using convolutional neural networks and explain the methods used for measuring its performance.

4.1 A brief Introduction to Neural Networks and Statistical Learning Theory

Artificial neural networks are computing systems vaguely inspired by the biological neural network. Similar to neurological structures, they are collection of (artificial) neurons, which process a given input. Such systems are trained to perform tasks by considering large sets of examples, generally without being given any task-specific rules. In the last decade a plethora of different neural network architectures has been developed and applied to a wide variety of problems. Some prominent examples are face and pattern recognition tasks or more recent and sophisticated projects as AlphaStar, which plays the video game StarCraft II on a superhuman level.
One of the most applied neural networks are feed-forward neural networks (FFNN). The goal is to find an efficient method to approximate a function $F^* : \mathbb{R}^m \to \mathbb{R}^k$, which is only available through sample points $\{(y_i, x_i)\}_{i\in\{1,\dots,n\}}$ with $n \in \mathbb{N}$, $x_i \in \mathbb{R}^m$, $y_i \in \mathbb{R}^k$ and $F^*(x_i) = y_i$. A feed-forward neural network is a mapping $F : \mathbb{R}^m \times \Omega \to \mathbb{R}^k$ such that $F(\cdot, \nu) \approx F^*(\cdot)$ for some $\nu \in \Omega$. Hereby, Ω denotes the set of additional variables the networks depends on besides the data input x. The optimal values for ν are "trained" (calibrated) on

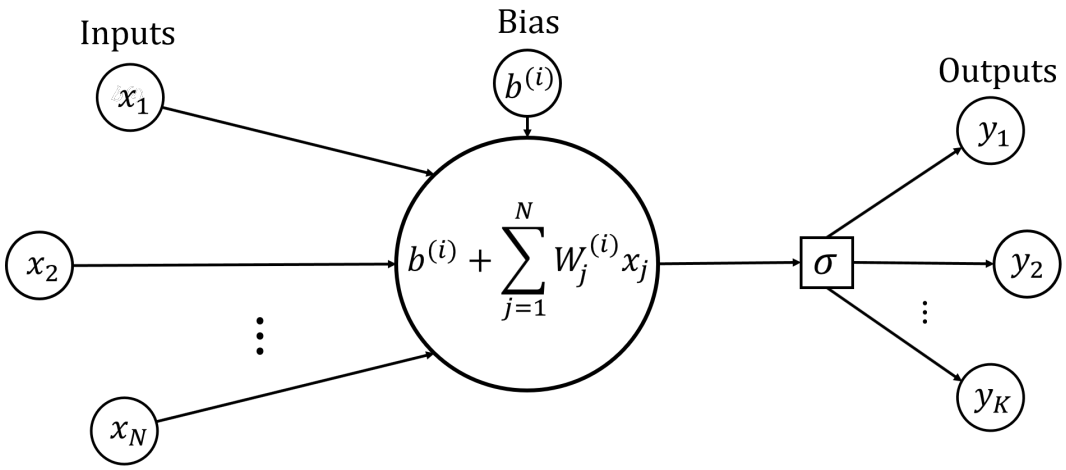


FIGURE 4.1: Visualisation of Neuron in a FFNN

the sample set $\{(y_i, x_i)\}_{i\in\{1,\dots,n\}}$. This outlines the importance of the sample set $\{(y_i, x_i)\}_{i\in\{1,\dots,n\}}$ as the accuracy of the approximation highly depends on the training. In the following, we formalise this idea, introduce two major neural network types, and state some important properties and principles of function approximation via neural networks.

Definition 4.1 (Feed-Forward Neural Network)**.** Let $L \in \mathbb{N}$ and the tuple $(N_1, \dots, N_L) \in \mathbb{N}^L$ denote the number of layers (depth) and the number of nodes (neurons) on each layer respectively. Let $W^{(i)} \in \mathbb{R}^{N_i \times N_{i-1}}$ be the weight matrix connecting layer $i-1$ and i. Let the vector $b^{(i)} \in \mathbb{R}^{N_i}$ denote the bias term in the i-th layer. Let the function $\sigma^{(i)} \in \mathcal{F}(\mathbb{R}^{N_i}, \mathbb{R}^{N_i})$ denote the activation function in the i-th layer. Usually, the activation function is applied componentwise, i.e. $\sigma^{(i)}(x)_k = \sigma(x_k)$ for some function $\sigma : \mathbb{R} \to \mathbb{R}$ and $k \in \mathbb{N}_{N_i}$. Define the i-th layer $A^{(i)} : \mathbb{R}^{N_i} \to \mathbb{R}^{N_{i+1}}$ as $A^{(i)}(x) := \sigma^{(i)}\left(W^{(i)}x + b^{(i)}\right)$. The first and last layers, F_1 and F_L, are called the input and output layers. The other layers are called hidden layers. These layers are also called fully-connected layers as each output component is dependent on every input through the weights. A feed-forward neural network F is defined as the composition

$$F := A_L \circ \cdots \circ A_1.$$

If the dependence of a neural network F on the weights and biases needs to be emphasised, it will be denoted as $F(\cdot, \nu) : \mathbb{R}^{N_0} \to \mathbb{R}^{N_L}$.
An exemplary neuron of a FFNN is illustrated in Figure 4.1. A fully-connected FFNN with three hidden layers is used in Section 4.5 and visualised in Figure 4.7.

Definition 4.2 (Convolutional Neural Network)**.** A convolutional neural network (CNN) is a special type of feed-forward neural networks that is tailored for image processing. More generally, it is suitable for analysing data with spatial structures. Analogously to the FFNNs, a convolutional neural network is defined as the composition of a finite set of compatible layers. We consider a type of CNNs, where the input (image) and features of each hidden layer are represented by a tensor $X \in \mathbb{R}^{d_1 \times d_2 \times d_3}$. Here, the first two dimensions d_1, d_2 of X indicate the coordinates of an image (pixel dimensions) while the third d_3 indicates the number of channels (e.g. RBG colour codes). We introduce four types of layers used to build CNNs, namely the convolutional layer, the pooling layer, the padding layer and the fully connected layer.
The most important, eponymous layer, the convolutional layer, provides the functionality of a fully-connected FFNN layer. It transforms the data input using a affine transformation and a non-linear activation function. It utilises the convolution operation. We consider an input $X \in \mathbb{R}^{d_1 \times d_2 \times d_3}$ with $(d_1, d_2, d_3) \in \mathbb{N}^3$. Let $W_k \in \mathbb{R}^{f_1 \times f_2 \times n}$ for $k \in \{1, \dots, n\}$ be the filters and n the number of filters. Each filter convolves with the input feature, resulting in a feature map

$$O_{ij}^k = \langle [X]_{ij}, W_k \rangle = \sum_{i'=1}^{f_1} \sum_{j'=1}^{f_2} \sum_{l=1}^{d_3} [X]_{i+i'-1, j+j'-1, l} [W_k]_{i', j', l},$$

where $[X]_{ij} \in \mathbb{R}^{f_1 \times f_2 \times d_3}$ denotes the sub-tensor of X starting at location $(i, j, 1)$ and ending at location $(i + f_1 - 1, j + f_2 - 1, d_3)$. The mappings O_{ij}^k are concatenated into a tensor $\mathcal{O} \in \mathbb{R}^{(d_1 - f_1 + 1) \times (d_2 - f_2 + 1) \times n}$. Analogously to FFNNs, an activation function $\sigma : \mathbb{R} \to \mathbb{R}$ is applied componentwise to the output, resulting in the layer $A : \mathbb{R}^{d_1 \times d_2 \times d_3} \to \mathbb{R}^{(d_1 - f_1 + 1) \times (d_2 - f_2 + 1) \times n}$ with $A(X) := \sigma(\mathcal{O}(X))$.
It is common to reduce the dimension even further by computing O_{ij}^k for only few pairs (i, j) and not every $(i, j) \in \{1, \dots, d_1\} \times \{1, \dots, d_2\}$. In these cases, the difference between two used pairs (i, j) and (i', j') is called stride. This is useful in plenty of applications. On the one hand it reduces the dimension if the desired output is of lower dimension. On the other hand, the computational load is drastically reduced.
However, one might be interested in unchanging or increasing dimensions. This can be achieved by Zero Padding, where zeros are appended to the margins of the spatial dimensions.
The Max Pooling layer reduces the dimension and aggregates the information of features. It has two important properties, the stride s and the filter size

$(f_1, f_2) \in \mathbb{N}^2$. For every channel $k \in \{1, \dots, d_3\}$

$$A_{i,j,k}(X) = \max\left([X]_{ij,k}\right)$$

is computed, where $(i, j) \in \{1, \dots, d_1\} \times \{1, \dots, d_2\}$ according to the wanted stride. For a stride $s = (1, 1)$, it results in a layer $A : \mathbb{R}^{d_1 \times d_2 \times d_3} \to \mathbb{R}^{(d_1-1)\times(d_2-1)\times d_3}$. A stride $s = (2, 2)$ results in a layer $A : \mathbb{R}^{d_1 \times d_2 \times d_3} \to \mathbb{R}^{\lceil d_1/f_1 \rceil \times \lceil d_2/f_2 \rceil \times d_3}$. The Max Pooling layer is illustrated in Figure 4.2. Furthermore, it is common to use fully-connected layers, as introduced for the FFNNs. Hereby, a tensor X is vectorised and the layer is defined according to Definition 4.1 with input vector $\mathrm{Vec}(X)$. These layers are also known as dense layers. As fully-connected layers allow to either increase or decrease the the spatial dimension, they provide flexibility to a CNN. Two different CNN architectures utilising the introduced layer types are visualised in the Figures 4.5 and 4.6.

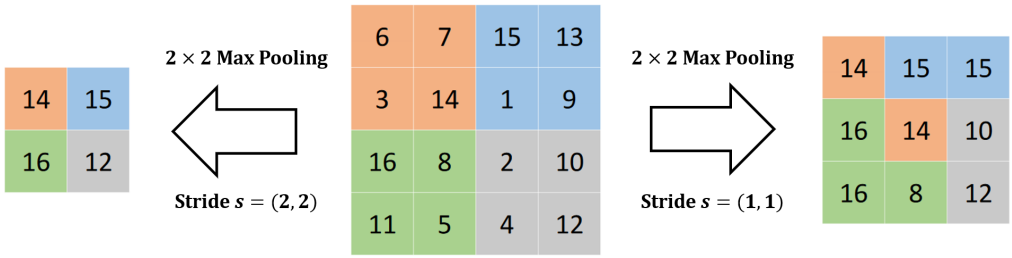


FIGURE 4.2: Illustration of the Functionality of a Max Pooling Layer from Fan, Ma, and Zhong (2019), p.10. Adapted.

The usability of FFNNs as approximation tool for a wide class of functions is justified by a result of Hornik, Stinchcombe, and White (1989):

Theorem 4.3 (Universal Approximation Theorem)**.** Let $\mathcal{NF}(\sigma, d_0, d_1)$ be the set of feed-forward neural networks with a single non-constant activation function $\sigma \in C^0(\mathbb{R}, \mathbb{R})$, input dimension $d_0 \in \mathbb{N}$ and output dimension $d_1 \in \mathbb{N}$. Then, $\mathcal{NF}(\sigma, d_0, d_1)$ is dense in $L^p(\mu)$ for all finite measures μ and $1 < p < \infty$.

The literature on the approximation properties of neural networks is growing. There are plenty of approximation theorems which focus on different architectures, the ability of neural networks to approximate derivatives or the approximation in specific function spaces. See for example Hornik, Stinchcombe, and White (1990), Zhou (2018), Grohs, Perekrestenko, Elbrächter, and Bölcskei (2019), or T. Chen and H. Chen (1993).
So far, we introduced the architecture of FFNNs and CNNs and showed the

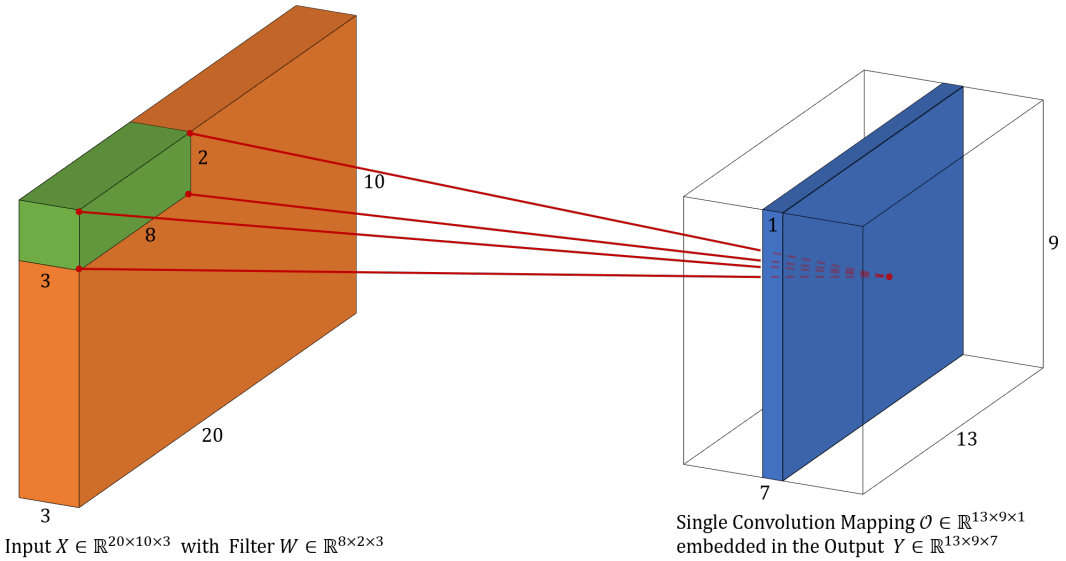


FIGURE 4.3: Illustration of a Convolutional Layer with Input $X \in \mathbb{R}^{20\times10\times3}$, Output $Y \in \mathbb{R}^{13\times9\times7}$, Filter $W \in \mathbb{R}^{8\times2\times3}$ and feature mapping $\mathcal{O} \in \mathbb{R}^{13\times9\times1}$

existence of a good function approximator in the class of neural networks. Nonetheless, this does not imply that it is possible to find a fitting neural network in practice application. The reason for the widespread use of neural networks is the development of algorithms which allow to solve this problem with relative ease. In the following we provide the foundation of neural network training and introduce a basic algorithm to efficiently solve a function approximation problem with a specific neural network architecture.
The training of neural networks follows the concept of empirical risk minimisation (ERM). A trained neural network F solves the optimisation problem (4.1)

$$\min_{\nu\in\Omega} \hat{\mathcal{R}}_n(\nu) := \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}\left(F(x_i,\nu), y_i\right) \tag{4.1}$$

for a dataset $\left\{(x_i, y_i) \,\middle|\, i \in \{1,\dots,n\}\right\}$ of input parameters x_i and true values y_i and sample size $n \in \mathbb{N}$. $\nu \in \Omega$ denotes the values of trainable weights in the neural network. $\mathcal{L}$ denotes an error measure as MSE or MAPE as introduced in Chapter 3.1. Here, $\hat{\mathcal{R}}_n(\nu)$ measures the training's error of the neural network, called empirical risk. If one considers the class $\mathcal{H}$ of neural networks with the same architectures as F, the ERM (4.1) can be denoted as

$$\min_{F\in\mathcal{H}} \hat{\mathcal{R}}_n(F) := \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}\left(F(x_i), y_i\right)$$

Furthermore, we denote with

$$\mathcal{R}(F) := \mathbb{E}_{\mathbb{Q}}[\mathcal{L}(F(x,v),y)]$$

the out-of-sample error of the neural network F, where $\mathbb{Q}$ denotes the joint distribution over (x,y). Also known as statistical risk or generalisation error. After all, one always aims to minimise the out-of-sample error. As $\mathbb{Q}$ is usually not known, $\mathcal{R}$ can only be approximated. To motivate the empirical risk minimisation as substitute for the minimisation of the statistical risk, consider the following risk decomposition:
For a class of neural networks $\mathcal{H}$ define the ERM-optimal network as $F_n := \underset{F\in\mathcal{H}}{\operatorname{argmin}}\, \hat{R}_n(F)$. Furthermore, we denote with $\mathcal{R}^* := \min_{G\in L^0} \mathcal{R}(G)$ the minimal risk value possible in the class of $\mathbb{Q}$-measurable functions, called Bayes Risk. Then,

$$\underbrace{\mathcal{R}(F_n) - \mathcal{R}^*}_{\text{Distance to Bayes Risk}} = \underbrace{\mathcal{R}(F_n) - \min_{G\in\mathcal{H}} \mathcal{R}(G)}_{\text{Excess Risk or Estimation Error}} + \underbrace{\min_{G\in\mathcal{H}} \mathcal{R}(G) - \mathcal{R}^*}_{\text{Approximation Error}}$$

The excess risk measures the quality of the ERM. The approximation error determines how good the architecture of a network fits the problem. If the class of neural networks $\mathcal{H}$ is chosen correctly, ERM leads to a minimisation of generalisation error. But, with increasing complexity of the class $\mathcal{H}$, solving the ERM becomes more difficult and the excess risk increases. On the other hand, the approximation error will decrease. This effect results in a "U"-shaped risk curve, also known as Bias-Variance trade-off, which is illustrated in Figure 4.4. Usually, one assumes that $\mathcal{R}(F_n) \to \min_{G\in\mathcal{H}} \mathcal{R}(G)$ if the training sample is large enough. The proof of this claim for specific losses or functions classes is beyond the scope of this thesis and we refer to Vapnik (2013) for an introduction to the theory of statistical learning.
Fan, Ma, and Zhong (2019) divide the challenges of empirical risk minimisation in three categories:

i. **Scalability and Nonconvexity:** For modern deep learning applications the number of trainable weights as well as the sample size can be huge. The approach we present in Section 4.3 consists of 85,970 trainable weights and a sample size of (179,605). Many optimisation algorithms are not practical due to the computational costs and memory constraints. Furthermore, it is not possible to ensure convexity of the empirical loss function $\hat{\mathcal{R}}_n$. Therefore, it is unclear whether an optimisation algorithm

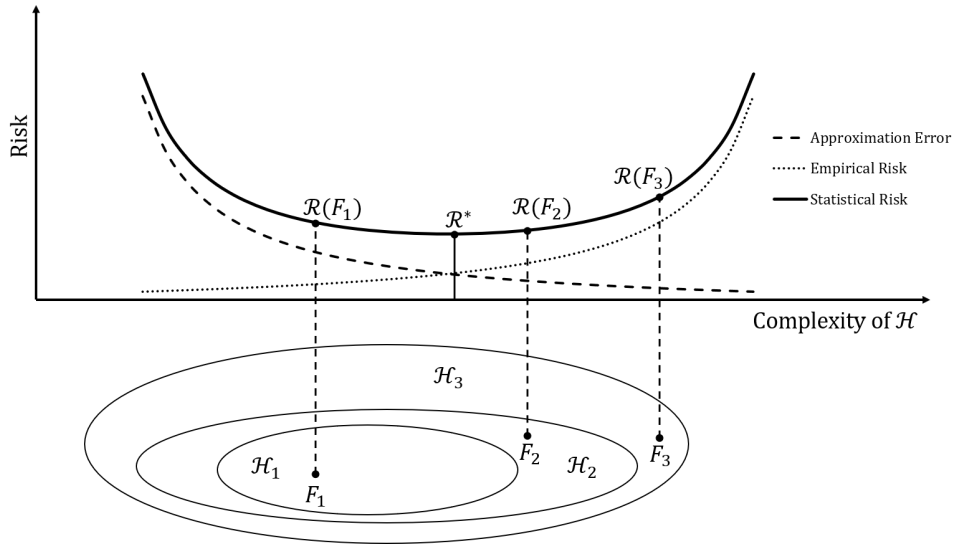


FIGURE 4.4: Visualisation of the Statistical Risk Trade-Off

converges and if it does, how fast the algorithm leads to small errors.

ii. **Numerical Stability:** The recursive structure of a neural network leads to a high dependency between the weights in different layers. This may lead to exponentially increasing or decreasing gradients, when optimisation the empirical loss. These effects are known as "exploding gradients" or "vanishing gradients".

iii. **Generalisation:** Ultimately, the goal is to find weights ν, which lead to a small out-of-sample error $\mathcal{R}(\nu)$. However, if an neural network architecture consists of more trainable weights than samples, the neural network might be able to fit the training data perfectly while performing poorly out-of-sample.

The Scalability problem is usually tackled by the right choice of optimisation algorithm. The stochastic gradient descend algorithm (SGD) is the most popular algorithm to solve the empirical risk minimisation (4.1) for large scale problems. It is based on the gradient descend algorithm but uses a single randomly chosen datapoint to update instead of the entire dataset. This leads to a considerably lower need for computational power and memory. The update rule is

$$\nu^{t+1} = \nu^t - \alpha_t \nabla \mathcal{L}(F(x_{i_t}, \nu^t, y_{i_t})),$$

for $t \in \mathbb{N}$ and a family of step sizes $(\alpha_t)_t$. ν^0 are initial weights, which are usually initialised randomly or as zeros. The datapoint (x_{i_t}, y_{i_t}) is chosen at every step randomly from the trainingset. The recursive structure of the introduced neural networks allows for an efficient calculation of the pointwise

gradients $\nabla \mathcal{L}(F(x_{i_t}, v^t, y_{i_t}))$. This method is called backpropagation. It is worth mentioning that in practice many adjustments are made to the SGD in order to obtain more accurate results or improve the speed of the algorithm. Some prominent variants are mini-batch SGD, momentum-based SGD or SGD with adaptive learning rates. Regularisation of the weights is widely applied tool to prevent "exploding gradients" or "vanishing gradients" as well as to reduce the risk of over-fitting. We refer to Fan, Ma, and Zhong (2019) and Aggarwal (2018) for more detailed view on training algorithms or regularisation techniques.

4.2 Formalisation of the Deep Learning Approach

Using deep learning methods to price options has the goal to overcome high computational effort caused by the implementation of volatility models, especially for those which have no closed-form solution available. Monte-Carlo based approaches approximate pricing functions pointwise with no way to store the pricer. Even if one stores the set of generated paths to overcome this problem, it leads to correlated errors and is therefore not a desirable procedure. Contrary, a neural network needs to learn the pricing function only once. After this, it is capable to evaluate new parameter scenarios at an incredibly high pace. Horvath, Muguruza, and Tomas (2019) present a highly efficient and accurate deep learning approach to implement option pricing in continuous-time stochastic volatility models. Their approach consists of two steps:

i. Learning a pricing function via a neural network by using model parameters as input and option price surfaces as an output.

ii. Calibration for the parameters that fit observed prices the best using the training network.

Our approach slightly differs from theirs. First, they only consider stochastic volatility models in continuous time with no closed-form solution available. In our analysis we deal with the discrete-time Heston-Nandi-GARCH model which has a closed-form pricing formula derived by Heston and Nandi (2000). The reasons to do so are multifaceted. First, discrete-time models provide a framework that allows to incorporate observed market data with no additional effort (see section 3.2). Secondly, there is no application of neural networks to option pricing in discrete-time models in the literature up to this point. We aim to close this gap. The choice on the HNG(1,1) model is

solely based on analytic reasons. The approximation of a pricing function with a closed-form solution is available allows to focus on the approximation quality. Contrary, if no closed-form solution is available, the neural network approach is highly dependent on the choice and accuracy on the method used to evaluate this pricing function (e.g. Monte Carlo simulation) in the first step. Clearly, our approach can be generalised to discrete-time models without a closed-form solution available. In this case one has to focus even more on the choice of the training set (see Section 4.6.

Furthermore, we focus on using deep CNN instead of feed-forward networks. The usage of a deep architecture is reasoned by Poggio et al. (2016) and Schindler, Lidy, and Rauber (2016). They find that with increasing dataset size and problem complexity, deep neural networks architectures perform better than shallow ones. This has two major impacts on our methodology in comparison to the approach proposed by Horvath, Muguruza, and Tomas (2019). The image-based approach proposed by them is tailor-made for a CNN architecture as we outlined in the previous section. Nonetheless, the usage of deep CNNs results in a higher amount of trainable weights and deeper network structure. This increases training times significantly, but does not effect evaluation times. But more importantly, the calibration of the model can no longer be done efficiently via standard optimisation algorithms, as the architecture of the proposed CNNs does not allow for fast computation of the gradients. Hence, we train a separate CNN for this task. These alleged disadvantages are compensated by the significantly increased performance as well as the increased calibration speed, we obtained in our numerical experiments in Section 5.1.

Let $\mathcal{M}^{HNG}(\theta)_{\theta\in\Theta^{HNG}}$ be a the model, we wish to approximate, with parameters

$$\theta = (\alpha, \beta, \gamma^*, \omega, h_0) \in \Theta^{HNG} \subset \mathbb{R}^5 . \tag{4.2}$$

The parameter combination $\theta \in \Theta^{HNG}$ completely describes the dynamics of the risk-neutral HNG(1,1) model $\mathcal{M}^{HNG}(\theta)$ as introduced in equation (3.3). The pricing map is stated by $P : \mathcal{M}^{HNG}(\theta, \zeta) \to \mathbb{R}^+$, which is available in closed-form for HNG(1,1) (see equation (3.4)). Thereby ζ includes the additional attributes of the HNG market as well as the characteristics needed of a call option, namely strike price K,maturity T, as well as yield curves $(r_t)_t$ or stock specific parameters as a dividend rate d. There are two major goals to achieve in this setting. First, finding an efficient way to approximate P. Hence,

we aim to find a way learn option prices by a neural network F. In section 2.1 we introduced the concept of implied volatilities and argued that it can be used as scale-independent substitute for option prices. We will utilise this concept and consider implied volatilities instead of option prices: The input of the network F are parameters of a volatility model (see equation (4.2)), the output is a strike-maturity determined set of implied volatilities. Formalised, we want to find

$$F(\Theta^{HNG}, \nu, \zeta) \approx \sigma_{BS}\left(P(\mathcal{M}^{HNG}(\Theta^{HNG}, \zeta))\right),$$

where σ_{BS} denotes the implied volatility function of the Black-Scholes-Formula (2.2) Our approach to this task will be elaborated in detail in Section 4.3. We will denote in short form $\sigma_{BS}^{HNG}(\theta, \zeta) \equiv \sigma_{BS}\left(P(\mathcal{M}^{HNG}(\Theta^{HNG}, \zeta))\right)$. Our approach to this task will be elaborated in detail in Section 4.3.
Secondly, we focus on calibrating of the HNG(1,1) model. The goal is to find a network, which approximates the inverse of the pricing mapping, $P^{-1} : R^{+} \to \Theta^{HNG}$. There are two main approaches to this - find an optimal parameters $\hat{\theta}$ by solving either

$$\hat{\theta} = \underset{\theta \in \Theta^{HNG}}{\operatorname{argmin}}\, \delta\left(\sigma_{BS}^{HNG}(\theta, \zeta), F(\theta, \nu, \zeta)\right) \tag{4.3}$$

or

$$\hat{\theta} = \underset{\theta \in \Theta^{HNG}}{\operatorname{argmin}}\, \delta\left(G\left(\sigma_{BS}^{HNG}(\theta, \zeta)\right), \theta\right), \tag{4.4}$$

where $\delta(\cdot, \cdot)$ is a suitable metric defined in the corresponding topological space. While both approaches (4.3) and (4.4) aim to find fitting parameters to given prices, they focus on different problems. Minimising the distance between predicted and observed surfaces ensures that the parameters induce accurate implied volatilities. On the other hand, minimising the difference between predicted and real parameters directly, leads to a more direct approximation of the inverse mapping $P^{-1} : \mathcal{X} \to \Theta^{HNG}$. This approach was originally proposed by Hernandez (2016). Theoretically, it should lead to a neural network G which is able to capture the dependency between the parameters better. Horvath, Muguruza, and Tomas (2019) focus on the former. Formalised, their approach consists of the two steps:

i. Train a neural network F to approximate the pricing map P.

ii. Calibrate the deterministic learned pricing map in (4.3).

While this approach works very well for their specific model, its usage is limited on models, which require only simple parameter constraints or to be more precise models with a low dependence between the model parameters. Furthermore, it requires a minimalist neural network architecture, which allows for quick backpropagation. Therefore, we follow Hernandez (2016), which will be elaborated in Section 4.4.

4.3 Learning the Pricer

When learning the option pricing function P with a NN, two different approaches come to mind. On the one hand, option and market parameters ζ are included in the input. This results in a network that is able to price every type of option and is flexible in general, but comes with a huge downside. The output of the network is the price of exactly one specific option. This leads to problems if one is asked to approximate a whole set or surface of options. The number of function evaluations increase linear in the number of options. Furthermore, the input dimension of the network increases drastically and therefore the training effort increases. To tackle this problem we focus on an image-based learning approach. As a first step, a fixed grid of strikes and maturities, $\Delta := \{K_i, T_j\}_{i=1,\,j=1}^{n\quad m}$ is defined. This grid specifies the set of call options we are interested in, namely call options with strike K_i, $i = 1, \ldots, n$, and maturity T_j, $j = 1, \ldots, m$. The "new" training task is defined as follows: The neural network $F(\theta, \hat{v})$ learns the set of implied volatilities

$$F^*(\theta) = \{\sigma_{BS}^{HNG}(\theta, T_i, K_j, \zeta)\}_{i=1,\,j=1}^{n\quad m},$$

where $\hat{v}$ are the trained network weights and

$$\begin{aligned} F^* : \Theta^{HNG} &\to \mathbb{R}^{m \times n} \\ \theta &\mapsto F^*(\theta) \end{aligned}$$

correspond to the implied volatilities of the closed-form solution. The optimal weights are calculated via

$$\hat{v} = \underset{v \in \Omega}{\operatorname{argmin}} \frac{1}{N_{train}} \sum_{u=1}^{N_{train}} \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{m} \left(\frac{F(\theta_u, v)_{ij} - F^*(\theta_u)_{ij}}{F^*(\theta_u)_{ij}} \right)^2}$$

The network does not train a single option in a market, but fixes a whole market structure and prices all options in it at once. The output surface of

prices can be interpreted as a grey-scale picture and our training evolves from a forecasting task to an image-recognition task. Our training problem corresponds to a decoding problem resulting from file compression. Compressed data needs to be decompressed to get a usable data type. In our case the parameters symbolise the compressed/encrypted volatility surfaces. The optimal weights $\hat{v}$ implicitly depend on the structure of the grid Δ and influence the neural network function $F(\theta, \hat{v})$. Hence the approximation quality of the neural network can differ across different strike-maturity grid structures. In theory, the strike-maturity grid can be chosen arbitrarily, however if the chosen scenarios are too extreme there might be numerical issues, especially when implied volatilities are calculated (e.g. for option contracts which are deep out of the money but simultaneously have a large time to maturity, prices tend to converge to zero, resulting in a badly conditioned problem for the implied volatility calculation.).
However Horvath, Muguruza, and Tomas (2019) list even more advantages of the image-based implicit learning approach. Examples are

i. exploiting the structure of having a full price grid which improves the learning process since neighbouring outputs are incorporated.

ii. properties (as injectivity or convexity) of the pricing map can be more easily guaranteed than in the pointwise training.

iii. dimension reduction to a finite optimisation problem. If we sample enough strike and maturity points $(k, t) \in [k_{min}, k_{max}] \times [0, T]$ the error of the neural network approximation gets smaller and smaller.

iv. fast training speed since the gridpoints stay fixed.

v. freedom of choosing the grid. As modern financial markets are often standardised with respect to option contracts, it is easy to adjust the neural network to such market requirements.

4.3.1 Network Architecture and Training

The detailed structure of the convolutional neural network is summarised in Table 4.1. The major specifications are as follows:

i. The input is of size $(5, 1, 1)$, where each input has the structure $(\alpha, \beta, \gamma^*, \omega, h_0)$.

ii. The output dimension is $(9, 9, 1)$ which corresponds to a discrete-time volatility surface as generated in Section 4.6.

iii. Zero Padding is used to increase the dimension.

iv. Every hidden convolutional layer uses elu-activation and bias terms

v. The total number of trainable weights is 85,970.

vi. To ensure positivity the final layer uses a linear activation and the weights are forced to be non-negative.

vii. the network is split into two network paths which separately price the low and high maturity options.

Volatility Pricing Network				
Layer	**Shape**			
Initialisation	(5,1,1)			
Zero Padding	(9,5,1)			
	Network Part (a)		**Network Part (b)**	
Layer	**Shape**	**Param #**	**Shape**	**Param #**
2D-Convolutional (elu)	(7, 5,32)	128	(7, 5,32)	128
Zero Padding	(13,7,32)		(13,7,32)	
2D-Convolutional (elu)	(12, 6, 32)	4128	(12, 6, 32)	4128
2D-Convolutional (elu)	(6, 5, 32)	4128	(6, 5, 32)	4128
Zero Padding	(10, 9, 32)		(10, 7, 32)	
2D-Convolutional (elu)	(9, 8, 32)	4148	(9, 6, 32)	4148
Zero Padding	(11, 10, 32)		(11, 6, 32)	
2D-Convolutional (elu)	(5, 9, 32)	4128	(5, 5, 32)	4128
Zero Padding	(7, 11, 32)		(7, 7, 32)	
2D-Convolutional (elu)	(3, 10, 32)	4128	(3, 6 32)	4128
Zero Padding	(5, 12, 32)		(5, 8, 32)	
2D-Convolutional (elu)	(2, 5, 32)	9248	(2, 3 32)	9248
Zero Padding	(6, 7, 32)		(6, 5, 32)	
2D-Convolutional (elu)	(3, 6, 32)	4128	(3, 4, 32)	4128
2D-Convolutional (elu)	(1, 5, 32)	4128	(1, 3, 32)	4128
Zero Padding	(5, 7, 32)		(5, 5, 32)	
2D-Convolutional (elu)	(2, 6, 32)	4128	(2, 4, 32)	4128
2D-Convolutional (linear)	(1, 5, 9)	585	(1, 4, 9)	585
Total params		42,985		42,985
Concatenation	(9,9,1)			
Total params		85,970		

TABLE 4.1: Architecture of the Pricing CNN: The network consists of 11x2 convolutional layers and a total of 85,970 trainable weights.

We realised that the models usually showed low accuracy for options with extreme specification e.g. low maturities in combination with high strike

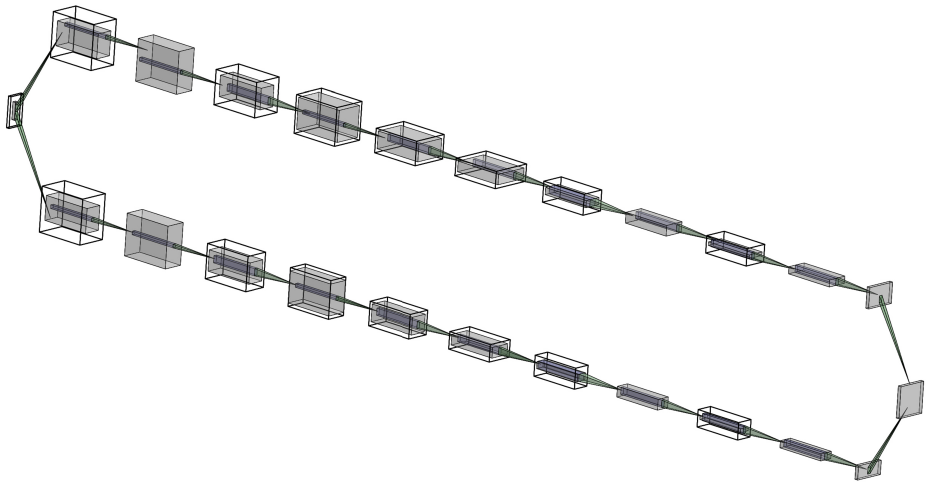


FIGURE 4.5: Visualisation of the Volatility Pricing CNN Architecture

prices. This led to main the characteristic of the introduced CNN, the 2-path structure. The neural network separately trains low and high maturity assets. Using only one half of the network still leads to reasonable results if one wants to decrease the training times. The activation function for the hidden layers is elu $\sigma_{elu} = \alpha(e^x - 1)$ with $\alpha = 1$ and for the output layer it is the linear function, i.e. no activation. Using Zero Padding Layers is of high importance in this specific neural network architecture as it increases the dimension. We tested using fully-connected layers to increase the dimension in contrast to Zero Padding but such approaches led to decreased performance. We also optimised the number of layers as well as filter size, leading to the non natural structure of the proposed network. Furthermore, the output is forced to be positive to ensure no violation of trivial economic properties (positive volatility and price). We did not account for other properties of volatility surfaces that one can think of as convexity. Often, it is desirable to transform the data so that such conditions are simplified to boundary conditions and as such, can be simply applied to every neural net. But, this is not possible in most advanced cases. However, by adding a penalisation to the error term, those adjustments are possible (see Section 4.4 for more details). We find that adding Max-Pooling as well as Dropout layers do not increase the overall network performance. The input parameters are normalised by min-max-scaling,

$$\theta_{norm} = \frac{2\theta - (\theta_{max} + \theta_{min})}{\theta_{max} - \theta_{min}} \in [-1, 1],$$

where θ_{max} and θ_{min} are the componentwise maxima, and minima respectively, of the set of parameters Θ^{HNG}. Normalisation of input parameters is extremely important for the network performance. One reason is the massive discrepancies between the average magnitude of parameters. A "normal" α is of order between 10^{-9} and 10^{-6}, on the contrary a "normal" γ^* is between 100 and 1000. We find that scaling the output parameters does not lead to a significant performance improvement for implied volatilities. Hence, the output is not scaled. We use 1000 epochs and a batch size of 64 to train the network. The error measure used for training the network is Root-Mean-Squared-Relative-Error (RMSRE):

$$RMSRE(\hat{\theta}) = \sqrt{\frac{1}{nm}\sum_{i=1}^{n}\sum_{j=1}^{m}\left(\frac{F(\hat{\theta},\hat{v})_{ij} - \sigma_{BS}^{HNG}(\theta, T_i, K_j, \zeta)}{\sigma_{BS}^{HNG}(\theta, T_i, K_j, \zeta)}\right)^2}$$

RMSRE behaves similar to Mean-Absolute-Percentrage-Error (MAPE) while staying differentiable. RMSRE is the $\mathcal{L}^2$-norm of the relative deviation, while MAPE corresponds to the $\mathcal{L}^1$-norm of the relative deviation.

4.4 Learning the Calibration Mapping

Following the reasoning from the previous section, it is more efficient to map a whole surface of implied volatilities or prices to its parameters instead of using the option specifications as additional variables. Therefore, the calibration task can be formalised as finding a neural network G that learn the set of parameters θ from the set of true implied volatilities surfaces $F^*(\theta)$. In contrast to the training task in Section 4.3, the output of the calibration Network should account for our parameter constraints. While positivity constraints are easy to implement, the incorporation of the stationarity constraint is a more sophisticated task. Two approaches are suggested by today's research to deal with constrained approximation tasks, even though this is not a widely tackled problem in a deep learning context. First, transforming the data and transforming constraints into boundaries which can be ensured by a specific activation function (e.g. restriction as $\theta \in (-1, 1)$ can be ensured by *tanh*-activation). The complexity of the HNG constraint does not allow for such an elegant solution. Hence, we propose a simple but efficient approach. We penalise the network for every violated constraint. We use a penalised mean

squared error (CMSE) error,

$$CMSE(\hat{\theta}) := \frac{1}{n}\sum_{u=1}^{n}\left(\frac{1}{5}\sum_{i=1}^{5}\left(G\left(F^*(\theta_u),\hat{v}_u\right)_i - \theta_{ui}\right)^2 + \lambda \mathbb{1}_{\text{VioSet}}\left(G\left(F^*(\theta_u),\hat{v}_u\right)\right)\right),$$

where $\lambda > 0$ is a hyper parameter trained via cross validation and the constraint set is defined as

$$\text{VioSet} = \{\theta \in \Theta^{HNG} : \alpha(\gamma^*)^2 + \beta \geq 1\}.$$

4.4.1 Network Architecture and Training

The detailed structure of the convolutional neural network is summarised in Table 4.2. The major specifications are as follows:

i. The input dimension is $(9, 9, 1)$ which corresponds to a discrete-time volatility surface as generated in Section 4.6.

ii. The output is of size $(5, 1, 1)$, where each output has the structure $(\alpha, \beta, \gamma^*, \omega, h_0)$.

iii. Every hidden convolutional layer uses *tanh*-activation and bias terms.

iv. The total number of trainable weights is 123,781.

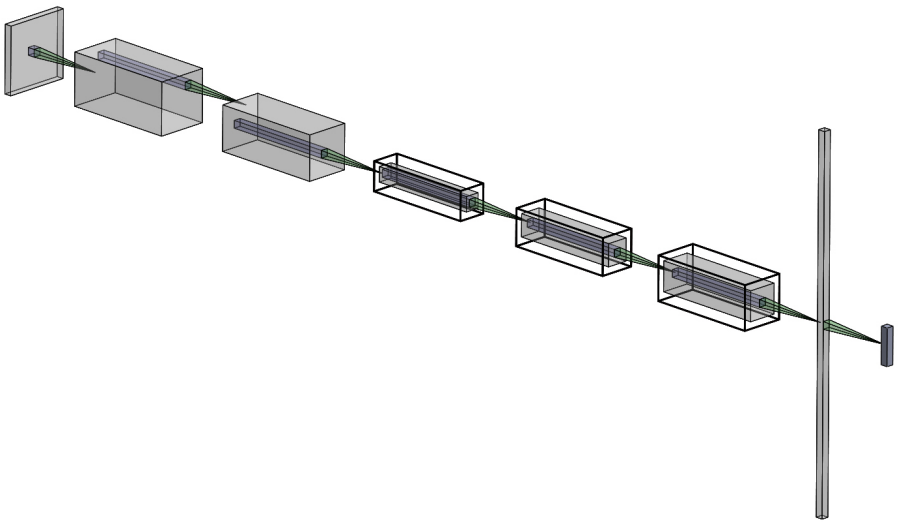


FIGURE 4.6: Visualisation of the Calibration CNN Architecture

To ensure that the forecasted parameters $G\left(\sigma_{BS}^{MKT}(T,K),\hat{v}\right)$ are in the same domain as the transformed parameters ($[-1,1]^5 \subset \mathbb{R}^5$), we use *tanh* activation functions. The same training and testing set as in the pricing task are used.

Calibration Network		
Layer	**Shape**	**Param #**
Initialisation	(9,9,1)	
2D-Convolutional (tanh)	(7,7,64)	640
2D-Convolutional (tanh)	(6, 6,64)	16448
2D-Max-Pooling	(3, 3, 64)	
2D-Convolutional (tanh)	(2, 2, 64)	16448
Zero Padding	(4, 4, 64)	
2D-Convolutional (tanh)	(3, 3, 64)	16448
Zero Padding	(5, 5, 64)	
2D-Convolutional (tanh)	(4, 4, 64)	16448
2D-Convolutional (tanh)	(3, 3, 64)	16448
Zero Padding	(5, 5, 64)	
2D-Convolutional (tanh)	(5, 4, 64)	16448
Zero Padding	(6, 6, 64)	
2D-Convolutional (tanh)	(5, 5, 64)	16448
Flatten	(1,1600)	
Dense (linear)	(1,5)	8005
Total params		123,781

TABLE 4.2: Architecture of the Calibration CNN: The CNN consists of 8 convolution layers, 1 Max-Pooling layer and 123,781 trainable parameters.

4.5 Benchmark Models

Horvath, Muguruza, and Tomas (2019) state that "In spite of the promising results by Hernandez the main drawback of [a direct calibration by a neural network], as Hernandez observes, is the lack of control on the function P^{-1}". They also state that for risk management purposes, one has no guarantee of how well the out-of-sample performance of the learned mapping of P^{-1} will be when directly approximated by a neural network. Hernandez (2016) points out that for such inverse approaches, the out-of-sample performance tends to differ from the in sample one, suggesting a not fully satisfactory generalisation of the learned map. Hence, we use their approach as benchmark for our models. Furthermore, to get more insight of in the performance of the calibration network, we compare it to standard theory using the calibration approach introduced in Section 4.4.

4.5.1 Feed-Forward Neural Network Approach

In contrast to our approach of using two separate neural networks, they use one network for pricing and calibration. We briefly introduced their

approach in Section 4.2. As their proposed network structure allows for fast backpropagation, the calibration conducted efficiently. Their approach adjusted to our data can be summarises as follows:

i. The neural network $H(\theta, \hat{v})$ learns the set of implied volatilities $F^*(\theta) = \{\sigma_{BS}^{HNG}(\theta, T_i, K_j, \zeta)\}_{i=1,\, j=1}^{n\quad m}$, where $\hat{v}$ denotes the trained network weights and

$$\begin{aligned} F^*(\theta) : \Theta^{HNG} &\to \mathbb{R}^{m \times n} \\ \theta &\mapsto F^*(\theta). \end{aligned}$$

The optimal weights are calculated via

$$\hat{v} = \underset{v \in \mathbb{R}^l}{\operatorname{argmin}} \sum_{u=1}^{N_{train}} \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{m} \left(H(\theta_u, v)_{ij} - F^*(\theta_u)_{ij} \right)^2}$$

ii. Find the optimal parameters $\hat{\theta}$ which solve

$$\begin{aligned} \hat{\theta} \quad = \quad & \underset{\theta \in \Theta^{HNG}}{\operatorname{argmin}} \quad \sum_{i=1}^{n} \sum_{j=1}^{m} (H(\theta, \hat{v})_{ij} - F^*(\theta)_{ij})^2 \\ & \text{subject to} \quad \begin{cases} \alpha(\gamma^*)^2 + \beta & < \quad 1 \\ \alpha & \geq \quad 0 \\ \beta & \geq \quad 0 \\ \omega & > \quad 0 \\ h_0 & > \quad 0 \end{cases} \end{aligned} \tag{4.5}$$

Neural Network architecture

They use a fully connected feed-forward neural network with 3 hidden layers and 30 nodes on each layer. The input vector contains the five variable parameters $\alpha, \beta, \gamma^*, \omega$ and h_0. The output dimension is defined by the product of the number of strikes and the number of the maturities in the grid. Theoutput vector for parameters θ corresponds to vectorised volatility surface $\text{Vec}(F^*(\theta))$. The activation function for the hidden layers is *elu* $\sigma_{elu} = \alpha(e^x - 1)$ with $\alpha = 1$ and for the output layer it is the linear function, i.e. no activation. We use 1000 epochs and a batch size of 64 to train the network. The inputs of the network are normalised in the same way as in Section 4.3.1. The structure of the network H is visualised in Figure 4.7.

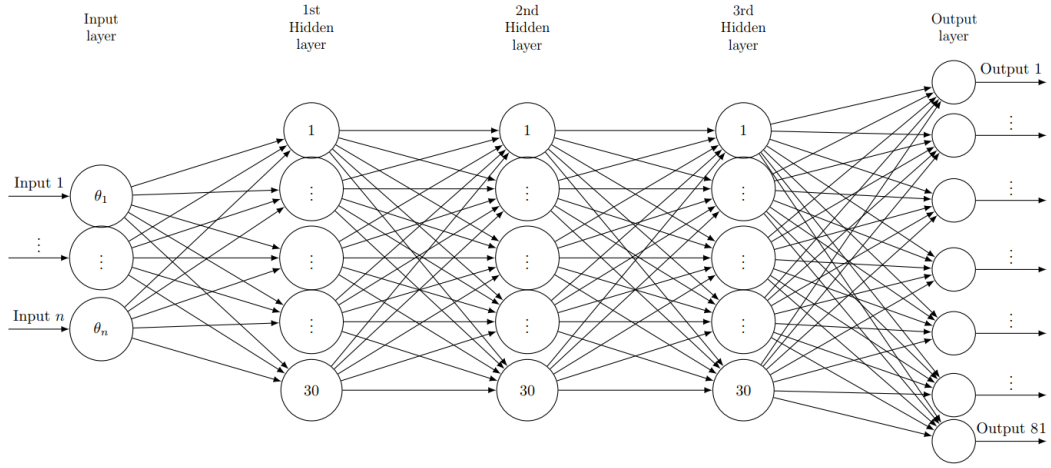


FIGURE 4.7: Visualisation of the FFNN Architecture from Horvath, Muguruza, and Tomas (2019), page 20. Adapted

Calibration

Due to the simple structure of the network H, it is easy to calculate the gradient. For $i \in \mathbb{N}_{\leq 5}$ let $A_j^{(i)}$ to be the output of the network H in the i-th layer for the j-th datapoint in dataset of size n. Let $W^{(i)}$ and $b^{(i)}$ the weights and the bias of the i-th layer, and therefore the parameters we wish to optimise. Let $Z_j^{(i)} = W^{(i)} A_j^{(i)} + b^{(i)}$ be the linear combination within the i-th layer for the j-th data point. We get the following identities for $A^{(i)}$:

$$A^{(i)} = \sigma_{elu}\left(Z^{(i)}\right) \quad \text{for } i \in \{2,3,4\}$$
$$A^{(5)} = Z^{(5)}$$

By C_j we denote the true implied volatilities for the i-th parameter combination. The loss of function L of H is defined by

$$L(W,b) = \frac{1}{n}\sum_{j=1}^{n}\left\|A_j^{(5)} - C_j\right\|_2^2$$

Hence we get

$$\frac{\partial L}{\partial W^{(4)}} = \frac{2}{n}\sum_{j=1}^{n} A_j^{(4)}(A_j^{(5)} - C_j)^T \imath$$
$$\frac{\partial L}{\partial b^{(4)}} = \frac{2}{n}\sum_{j=1}^{n} \imath^T(A_j^{(5)} - C_j)^T \imath$$

and for $i \in \{1,2,3\}$ the partial derivative can be calculated via

$$\frac{\partial L}{\partial W^{(i)}} = \frac{\partial L}{\partial A^{(i+1)}} \frac{\partial A^{(i+1)}}{\partial W^{(i)}}$$
$$\frac{\partial L}{\partial b^{(i)}} = \frac{\partial L}{\partial A^{(i+1)}} \frac{\partial A^{(i+1)}}{\partial b^{(i)}}$$

where

$$\frac{\partial L}{\partial A^{(4)}} = \frac{2}{n} \sum_{j=1}^{n} W_j^{(4)} (A_j^{(5)} - C_j)^T \iota$$
$$\frac{\partial A^{(i+1)}}{\partial W^{(i)}} = \begin{cases} A^{(i)} \exp\left(A^{(i+1)}\right) & \text{if } A^{(i+1)} \leq 0 \\ A^{(i)} & \text{if } A^{(i+1)} > 0 \end{cases}$$
$$\frac{\partial A^{(i+1)}}{\partial b^{(i)}} = \begin{cases} \iota \exp\left(A^{(i+1)}\right) & \text{if } A^{(i+1)} \leq 0 \\ \iota & \text{if } A^{(i+1)} > 0 \end{cases}$$

Horvath, Muguruza, and Tomas (2019) test several gradient based optimisers and find that the Levenberg-Marquardt algorithm is the most balanced optimiser regarding speed and convergence. Hence, it is used to solve the optimisation problem (4.5).

4.5.2 Classic Calibration

Besides different neural network approaches we introduced for calibration, we compare the performance of our approach to standard optimisation theory. We minimise the MSE of the training set implied volatility and the implied volatility surface of the HNG(1,1) model. For each surface, the following optimisation problem (4.6) is solved:

$$\hat{\theta} = \underset{\theta \in \mathbb{R}^5}{\operatorname{argmin}} \sum_{i=1}^{n} \sum_{j=1}^{m} \left(\sigma_{BS}^{HNG}(\theta, \zeta)_{ij} - F^*(\theta)_{ij}\right)^2$$
$$\text{subject to} \begin{cases} \alpha(\gamma^*)^2 + \beta < 1 \\ \alpha \geq 0 \\ \beta \geq 0 \\ \omega > 0 \\ h_0 > 0 \end{cases} \tag{4.6}$$

This approach is explained in more detail in Section 3.2.2 and we proceed as proposed there: A interior-point algorithm is used. All optimisations run on

MATLAB R2020a using the Optimisation Toolbox. We use fmincon with the following specifications:

i. TolFun: 10^{-6}

ii. TolX: 10^{-9}

iii. MaxIterations: 4000

iv. MaxFunctionEvaluations: 2500

v. Default settings in all other configuration options.

4.5.3 Monte-Carlo Simulation

One standard approach for approximating derivative prices in stochastic models are Monte-Carlo simulations. Instead of calculating expected values analytically, a large number of independent stock price paths is used to approximate the expected value of an option. It is based on the law of large numbers:

Theorem 4.4 (Law of Large Numbers)**.** Let be $(\Omega, \mathcal{F}, \mathbb{P})$ a probability space. For every integrable random variable X and measurable function f on $(\Omega, \mathcal{F}, \mathbb{P})$, such that $f(X)$ is integrable holds $\frac{1}{N}\sum_{i=1}^{N} f(X_n)) \xrightarrow{\mathbb{P}-a.s.} \mathbb{E}[f(X)]$, where $(X_n)_n$ is a family of i.i.d. copies of X. Additionally, if $f(X) \in L^2(\Omega)$, it follows

$$\mathbb{V}\left[\frac{1}{N}\sum_{i=1}^{N} f(X_n)\right] = \frac{1}{N^2}\sum_{i=1}^{N} \mathbb{V}[f(X_n)] = \frac{1}{N}\mathbb{V}[g(X)] \xrightarrow{\mathbb{P}-a.s.} 0$$

The law of large numbers implies for a set of "good" pseudo random variables $Y_1, \ldots, Y_n$ of X and large $N \in \mathbb{N}$, that

$$\frac{1}{N}\sum_{i=1}^{N} f(Y_i) \approx \mathbb{E}(f(X)).$$

We proceed as follows: For a given parameter set of the HNG(1,1) model under $\mathbb{Q}$, i.e. $(\omega, \alpha, \beta, \gamma^*, h_0)$, we generate 100.000 independent, (pseudo-)random paths following the stock prices dynamic (3.3). For every path, we compute the payoff of an option at maturity. The estimate of the fair price is given by the average of the discounted payoffs. This estimate is used to calculate the implied volatility of the option. As this approach only exploits the law of large numbers, it is one of the easiest, yet most commonly utilised

approaches. Especially for derivative contracts with no closed-form solution available, it is often the only way of estimating the fair value. Therefore, we use this approach to get a non deep learning based benchmark for our pricing network.

4.6 Generating a Training Set

As for every deep learning task, it is utmost importance to use a sufficiently large dataset of high quality. In contrast to the majority of neural network applications, there are no datasets we can use for our analysis. Hence, the generation procedure should be handled extremely careful. For this reason, we need to clarify what properties are necessary to deal with the given problem. The Heston-Nandi-GARCH(1,1) model under $\mathbb{Q}$ consists of 4 visible parameters, $(\alpha, \beta, \gamma^*, \omega)$, as well as one "hidden" parameter, namely the initial conditional variance h_0. As we want to learn the pricing as well as the calibration mapping, we need a huge variety of different parameter combinations and the corresponding prices or implied volatilities. Hereby, it is important that the parameters should represent real world scenarios as we ultimately want to apply the approach on real option prices. Furthermore, a parameter combination must fulfil mathematical conditions as stationarity and positivity constraints of HNG(1,1) model. This is to ensure proper functionality of the model and, with a view to financial regulations, to ensure stable results. This being said, the question of how to generate such a dataset is still unanswered. Our approach on generating a training set is summarised in the following:
First, we calibrate the HNG(1,1) model on observed call option prices on a weekly basis over several years. Hereby, we follow the approaches established in Section 3.2. Afterwards, a random sample of parameters is drawn which captures the statistical properties of the priorly calibrated prices and parameters. This random sample is then filtered such that only parameter sets which fulfil all model constraints remain. Prices and implied volatilities are generated for those remaining parameters. Last but not least, parameter which lead to extreme prices and volatilities are filtered out (see Section 4.6.2). This is necessary to ensure a realistic training set as volatilities of e.g. 10^{-8} % are not reasonable. Nonetheless, this procedure might decreases the significance of the results as scenarios which are supposedly difficult to price are excluded from the training set.

4.6.1 Calibration

To obtain a set of parameters with good concordance to observed prices, the calibration task (3.8) from Section 3.2 is repeated. On all Wednesdays between January 1st 2010 and December 31st 2018 European call option contracts on the S&P 500 with the following characteristics are considered:

i. The maturity is between 8 and 250 days.

ii. The Moneyness is between 0.9 and 1.1.

iii. The open interest and trading volume is over 100 at the observed Wednesday.

The parameters are scaled the same magnitude to ensure proper convergence. Again, we want to remark that this step is of utmost importance as on one hand, the model is extremely sensitive to the parameters and on the other hand, the magnitude between typical α, γ, h_0 and β, γ^* differs extremely. The initial conditional variance h_0 is calibrated. The used interest rates are interpolated from T-Bill rates of the corresponding Wednesday. This optimisation leads to 464 parameter combinations (9 years with 51 or 52 trading weeks). Table 4.3, 4.4 and Figure 4.8 show the descriptive statistics of the calibrated parameters.

	α	β	γ^*	ω	h_0
mean	3.1962e-06	0.6479	379.6678	1.8837e-07	7.5785e-05
median	2.2875e-06	0.6813	350.5501	1.3915e-09	5.0526e-05
max	3.8 669e-05	0.9909	1373.9331	4.6840e-06	6.2074e-04
min	7.2297e-12	0.0019	-529.5509	2.3293e-11	3.3337e-06
2.5% qu.	5.5113e-07	0.1892	148.3569	7.6257e-11	9.8376e-06
97.5% qu.	1.1426e-05	0.8632	900.1006	1.9948e-06	3.0903e-04

TABLE 4.3: Descriptive statistics of the calibrated parameters. The parameters are calibrated as in Section 3.2.

θ	ω	α	β	γ^*	h_0
ω	1	-0.0782	-0.4865	0.4088	0.3970
α	-0.0782	1	-0.2009	-0.4679	0.2863
β	-0.4865	-0.2009	1	-0.4892	-0.5434
γ^*	0.4088	-0.4679	-0.4890	1	-0.0099
h_0	0.3970	0.2863	-0.5434	-0.0099	1

TABLE 4.4: Correlation matrix of the calibrated parameters. The parameters are calibrated as in Section 3.2.

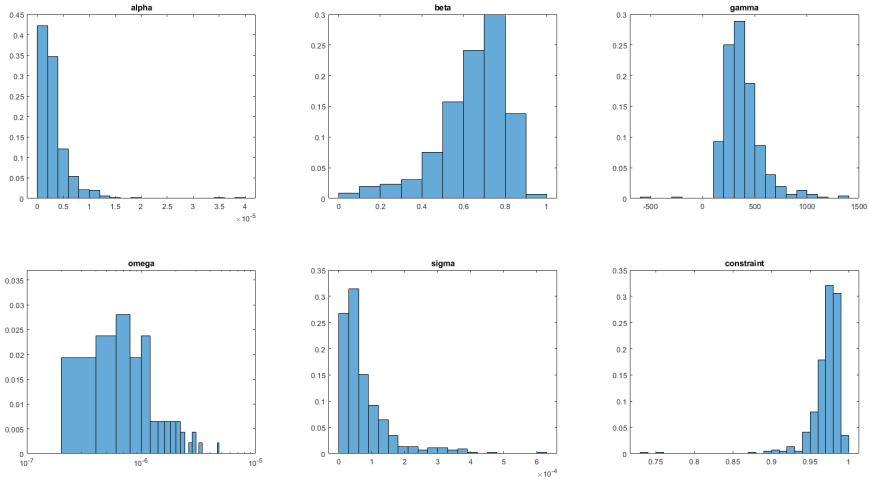


FIGURE 4.8: Histogram of calibrated parameters. The parameters are calibrated as in Section 3.2.

Furthermore, a second parameter set is created by restricting the previously calibrated parameters. For this dataset, every scenario is filtered out for which the beta and gamma values are outside of the symmetric 95% confidence interval. This corresponds to the 9.05% of data. The descriptive statistics of this restricted dataset is shown in Table 4.5 and Figure 4.9. Those to parameter

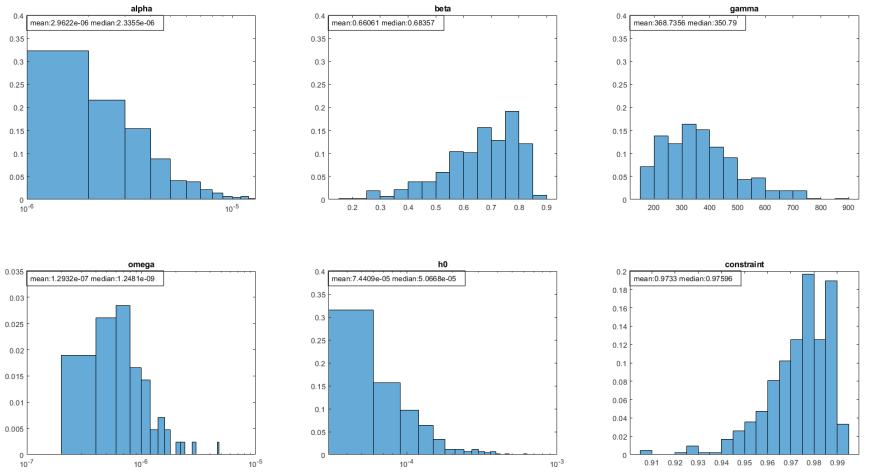


FIGURE 4.9: Histogram of filtered calibrated parameters. The parameters are calibrated as in Section 3.2.

sets are used as foundation to generate the training set for the neural network approaches.

θ	ω	α	β	γ^*	h_0
ω	1	0.0319	-0.3641	0.2473	0.4315
α	0.0319	1	-0.2188	-0.6867	0.3814
β	-0.3641	-0.2188	1	-0.3576	-0.5719
γ^*	0.2473	-0.6867	-0.3576	1	-0.0353
h_0	0.4315	0.3814	-0.5719	-0.0353	1

TABLE 4.5: Correlation matrix of cleaned underlying option prices

4.6.2 How to draw a "good" Random Sample

As it is desirable that the training set captures the properties of those previously calibrated parameters, several points need to be considered: On the one hand, the distribution of calibrated parameters and the parameters in the training set should be similar. Thereby, it is beneficial for training that some measures of central tendency, as mean or median, coincide. Furthermore, as we are dealing with highly dependent parameters, especially with stationarity constraints in mind, the correlation between the parameters should be similar. But also, the support and tail distribution must be carefully attended. In theory, creating such a training set should provide a good in-sample fit. On the other hand, it is not clear if a neural network is able to capture the properties of extreme scenarios as tails of empirical distributions are usually thin. Therefore, the question arises: How strong do the similarities between reality and artificial sample have to be? Weighting out these aspects against each other, we come up with several different approaches; each with a focus on different aspect of training.

Normal Distribution

The bell shaped empirical distribution is a remarkable feature of the calibrated parameters. Hence, a natural approach is to generate a normal distributed training sample. A multivariate normally distributed random sample with mean μ and covariance matrix Σ is generated, where μ and σ correspond to the sample mean and variance of the calibrated parameter set. Parameter combinations which do not satisfy positivity or stationarity constraints are filtered out. The mean and covariance of the underlying calibrated is approximately conserved. But, due to the ad hoc filtering, the mean of the random sample has a positive bias. This issue is solved by shifting the random sample. Figure 4.10 and 4.11 show the empirical parameter distributions of the random sample based on the raw calibrated parameters and the cleaned parameters.

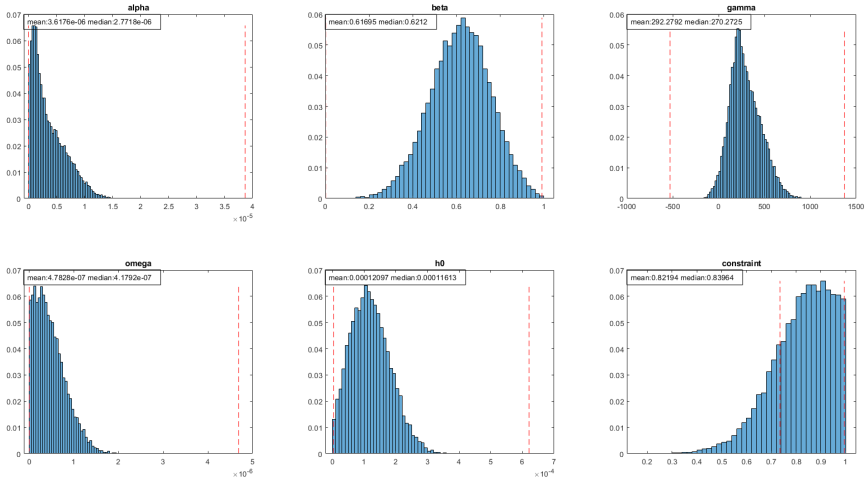


FIGURE 4.10: Histogram: Empirical Parameter Distribution based on S&P 500 option price calibrated parameters. Underlying Distribution: Normal Distribution.

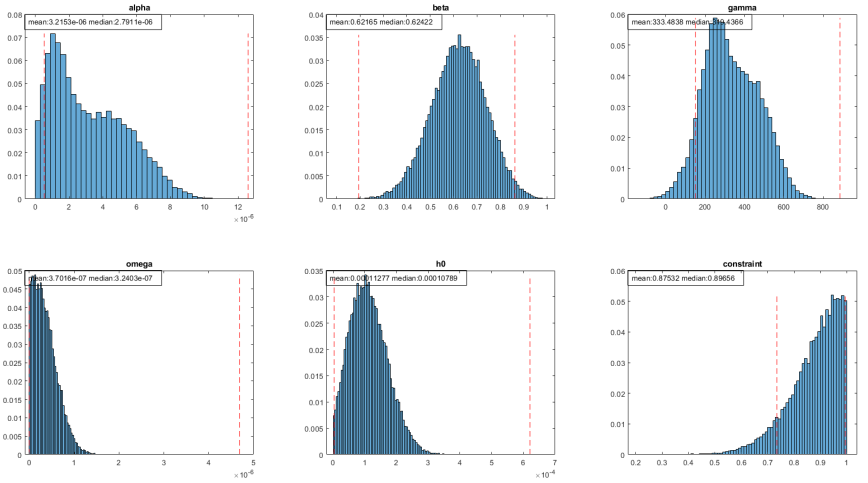


FIGURE 4.11: Histogram: Empirical Parameter Distribution based on S&P 500 option price calibrated parameters. Extreme Scenarios are filtered out. Underlying Distribution: Normal Distribution.

MinMax-Uniform Distribution

Another simple approach utilises the multivariate uniform distribution. The procedure looks as follows:
Calculate the minimum and maximum value for each parameter. Draw a multivariate uniformly distributed random sample on the interval between the minimum and maximum value of the underlying dataset. This approach does not capture the moments of the underlying dataset. In contrast to the previous

method, the support of the underlying data is preserved. For this method, independence of the parameters is assumed which is clearly not the case. The main argument in favour of this approach is, that extreme scenarios are highly over represented which could lead to a better out-of-sample performance.

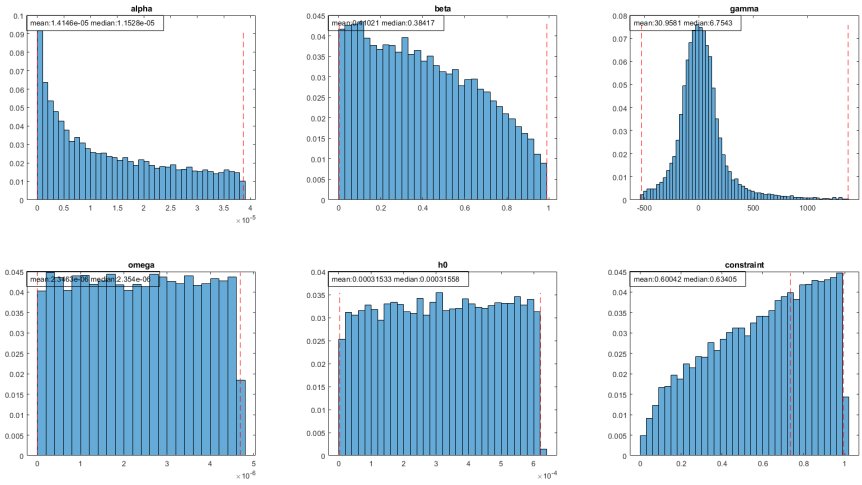


FIGURE 4.12: Histogram: Empirical Parameter Distribution based on S&P 500 option price calibrated parameters. Underlying Distribution: Uniform Distribution.

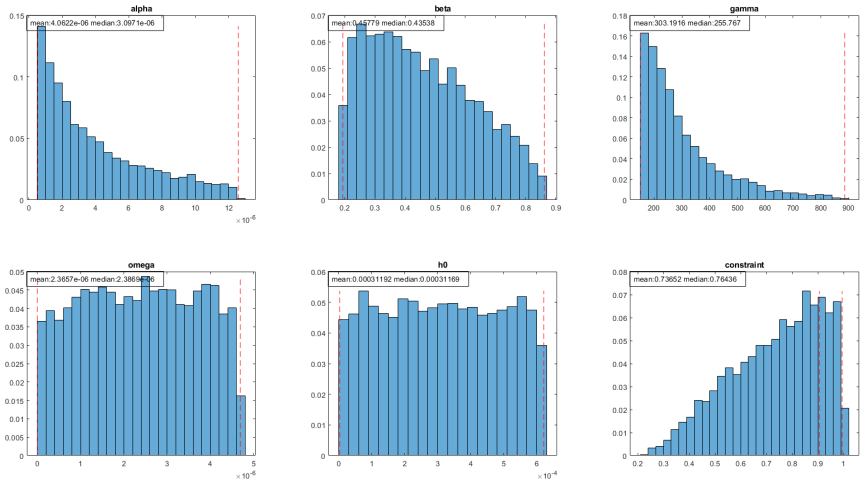


FIGURE 4.13: Histogram: Empirical Parameter Distribution based on S&P 500 option price calibrated parameters. Extreme Scenarios are filtered out. Underlying Distribution: Uniform Distribution.

Mean-Symmetric-Uniform Distribution

As stated previously, drawing uniformly from the interval between minimum and maximum value, neither captures the standard deviation nor the mean of the underlying dataset. The following adjustments are made to tackle this problem, but keep extreme scenarios over represented in the training set: A uniform random sample is generated. Thereby, the boundaries of the uniform distribution are chosen such that the first two sample moments coincide with the set of calibrated parameters. The sample is drawn symmetric around the mean of the underlying data $[\bar{x} - a, \bar{x} + a]$. Here a fulfils the following equation:

$$\sigma_{sample} = \frac{2a + \bar{x}}{\sqrt{12}} \stackrel{!}{=} \sigma_x$$

Hence the random sample is drawn from the interval $[\bar{x} - \frac{\sqrt{12}\sigma_x - \bar{x}}{2}, \bar{x} + \frac{\sqrt{12}\sigma_x - \bar{x}}{2}]$. 4.12 and 4.13 show the empirical parameter distributions of the random sample based on the raw calibrated parameters and the cleaned parameters.

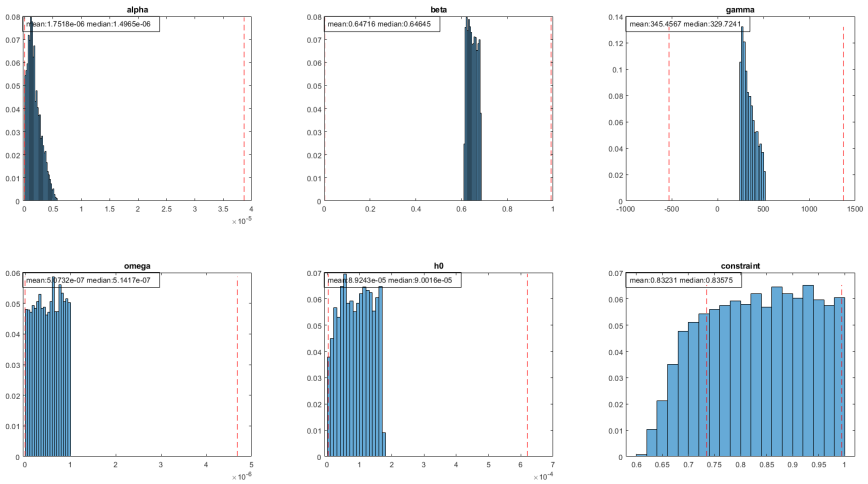


FIGURE 4.14: Histogram: Empirical Parameter Distribution based on S&P 500 option price calibrated parameters. Underlying Distribution: Mean-Symmetric Uniform Distribution.

Partial-Log-Normal Distribution

Filtering out the parameter combinations which do not satisfy the positivity constraints ad hoc, might look inelegant for some readers. This drawback can be compensated by taking the logarithm of the affected parameters, namely ω, α, β and h_0. Following the approach based on the normal distribution, leads to a dataset which is log-normal distributed in the transformed parameters

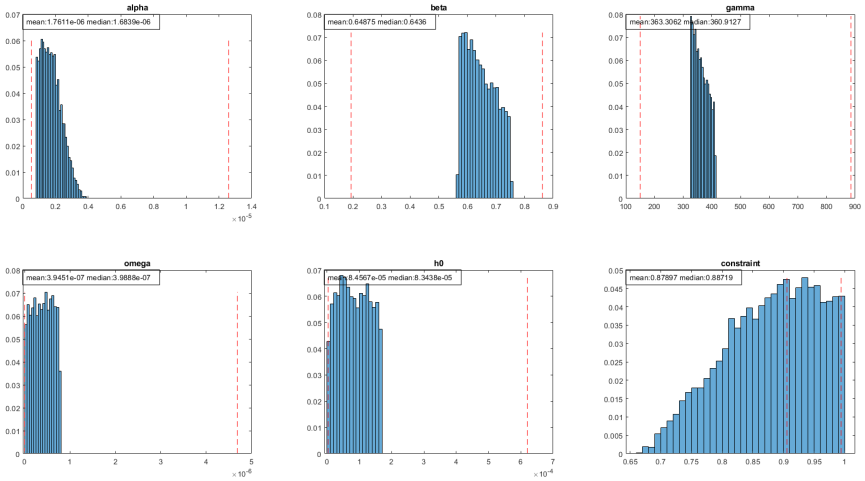


FIGURE 4.15: Histogram: Empirical Parameter Distribution based on S&P 500 option price calibrated parameters. Extreme Scenarios are filtered out. Underlying Distribution: Mean-Symmetric Uniform Distribution.

and normal distributed in γ^*. As the reader can see in Figure 4.16 and 4.17, such an approach lead to extremely slim tails which, again, might cause performance problems when pricing out-of-sample.

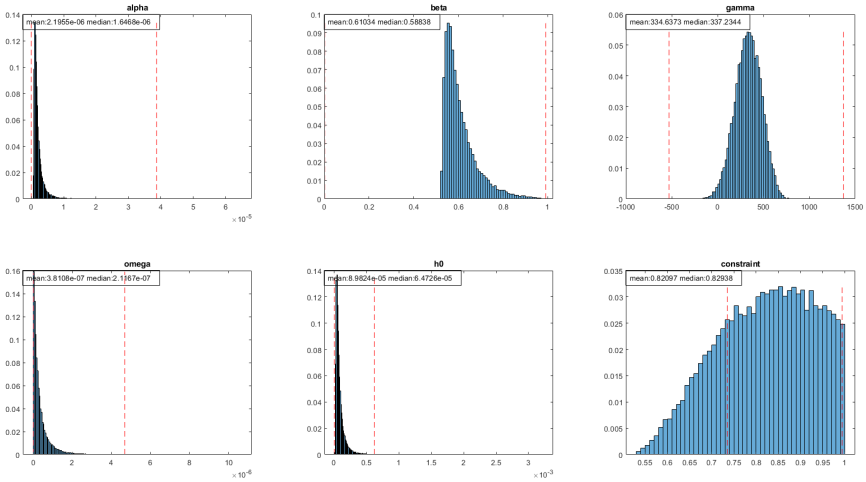


FIGURE 4.16: Histogram: Empirical Parameter Distribution based on S&P 500 option price calibrated parameters. Underlying Distribution: Log-Normal Distribution.

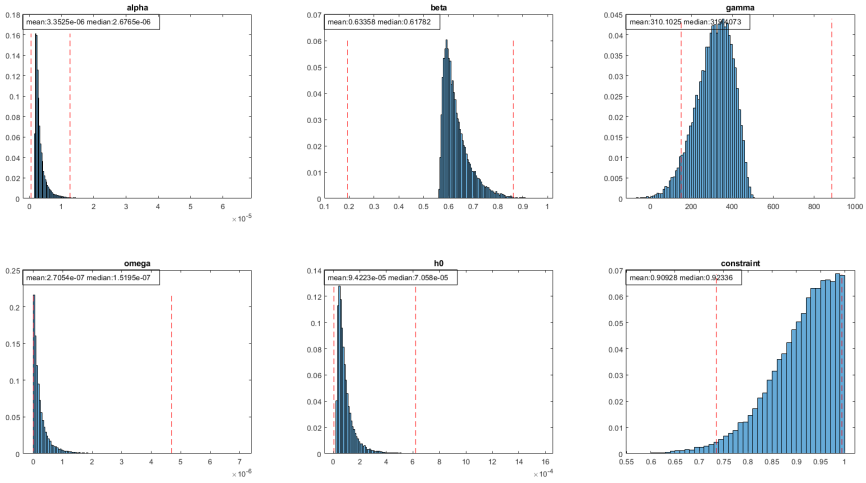


FIGURE 4.17: Histogram: Empirical Parameter Distribution based on S&P 500 option price calibrated parameters. Extreme Scenarios are filtered out. Underlying Distribution: Log-Normal Distribution.

4.6.3 Further Specification of the Training Set

In all further analysis and numerical experiments, we consider a grid of maturities $M \in \{10, 40, \ldots, 250\}$ days and (inverse) Moneyness of $\frac{K}{S} \in \{0.9, 0.925, \ldots, 1.1\}$ to match the option contract specification used for calibration. As the HNG model is homogeneous in (S_t, K), the underlying price is set to 1. For each parameter set a surface of implied volatilities is computed using the closed-form solution of HNG (3.4) on the grid $M \times \frac{K}{S}$. The used yield-curves are chosen randomly from the set of historic T-Bill curves observed on Wednesdays between 2010-2018. The algorithm introduced by Jäckel (2015) is used to efficiently and accurately compute implied volatilities from each price surface. This results in a 9×9 surface of implied volatilities. Each dataset is split randomly into three part, with a splitting ratio of (72,25%, 12.75%, 15%) to generate training-, validation- and testing set. Finally, the dataset contains the used yield-curves, as such data would be available in application. Depending on the numerical experiment, the parameter set to compute implied volatilities is generated by one of the previously introduced approaches.

Chapter 5

Summary and Results

This chapter focuses on the performance of the proposed neural network approaches to price and calibrate implied volatilities in HNG(1,1) model. We summarise the thesis and provide an outlook on future research.

5.1 Numerical Experiments and Results

If not stated differently, the results shown in this chapter are based on a normal-distributed dataset of size 179,605, and 30,878 testing set size respectively, as specified in Section 4.6.2 and 4.6.3. The neural networks are implemented and training in Pyhton 3.7 using Tensorflow 2.1 with the Keras Backend. We use MATLAB 2020a for the computation of closed-form solutions and Monte-Carlo simulations. All computations are run on Microsoft Windows Server 2016 Datacenter with 88 Intel Xeon Gold 6152 (2.1 GHz) CPUs. All code used is parallelised to optimise the performance. The numerical results demonstrate that our two CNN approaches are indeed able to outperform all defined benchmarks. This holds for the pricing as well as the calibration task. We only state the test sample results.
First, we focus on the performance of the volatility pricing network F. Hereby, we measure the accuracy of the approach and the introduced benchmarks by using the MAPE as well as the MSE between forecasted surfaces and the closed-form solution. Additionally, we test for significance by applying the two-sample one-sided Kolmogorov-Smirnov test and the Anderson test to the empirical error distribution. Furthermore, we analyse the pricing speed. Second, we repeat this exercise for calibration task. We compute the average deviation in each parameter and analyse the empirical error distributions. Besides the accuracy, we also compare the approximation speed of the different calibration approaches. Third, we check the performance when calibration and forecasting are used in combination. This task is of utmost importance as a good performance with respect to this autoencoding task

shows that the calibrated parameters lead to good surface forecasts even if they differ from the real parameters. Therefore these experiments can be seen as a measure of robustness. Finally, we analyse the ability of the proposed neural network architecture to generalise. Hereby, the neural network performance on several different datasets is tested.
The code to reproduce the numerical experiments is publicly available at Github.com/henrikbrautmeier/HNGDeepVola

5.1.1 Pricing Volatility

Figures 5.1, 5.2 and 5.3 show the test sample performance of the volatility pricing task of each introduced approach splitted by maturity and strike price. In table 5.1 we state the descriptive statistics of the results. The proposed CNN approach achieves an average deviation to the closed-form solution of 0.62%, while the approach of Horvath, Muguruza, and Tomas (2019) and a Monte-Carlo simulation lead to average deviations of 0.91% and 30.04%, respectively. It is striking, that the benchmark approaches tent to perform worse for low

Approach	Mean	Median	Max
CNN	0.6298	0.4603	37.1354
FFNN	0.9133	0.6275	53.6112
MC	30.0421	56.0395	1919.3118

TABLE 5.1: Descriptive Statistics of the Volatility Pricing Approaches with respect to MAPE in %

maturities while our approach leads to better results with decreasing maturity. This finding can be attributed to the 2-path architecture of the CNN. By artificially splitting the training task by maturity, the network is able to capture the different characteristics of option prices better. Analysing the dependency between the magnitude of the volatilities and the obtain error in Figures 5.4, 5.5 and 5.6, we find a clear dependency between implied volatlility and the approximation error in the Monte-Carlo simulation. This holds not true for both neural network approaches. This observation implies, that neural network approaches may lead to less correlated errors and more stable results. In view of modern financial regulatory frameworks, this re-emphasises the potential of neural networks for option pricing in practical application. Figure 5.7 shows the empirical error distribution of the three approaches. For the empirical cummulative distribution functions holds $CDF^{emp}_{CNN} > CDF^{emp}_{FFNN} > CDF^{emp}_{MC}$. We use the two-sample one-sided Kolmogorov-Smirnov test as well as test procedure introduced by Anderson (1996) to test for significant first order

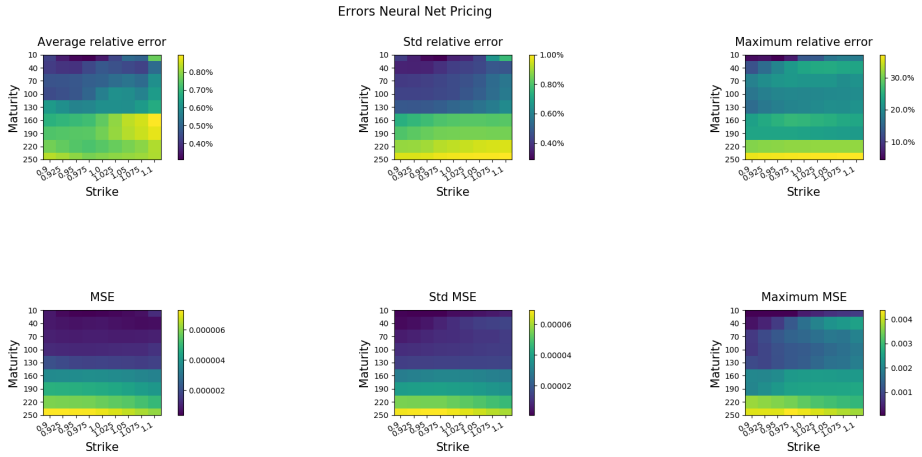


FIGURE 5.1: Test Sample Performance of the Volatility Pricing Network F. The errors are always measured with respect to the closed-form solution.

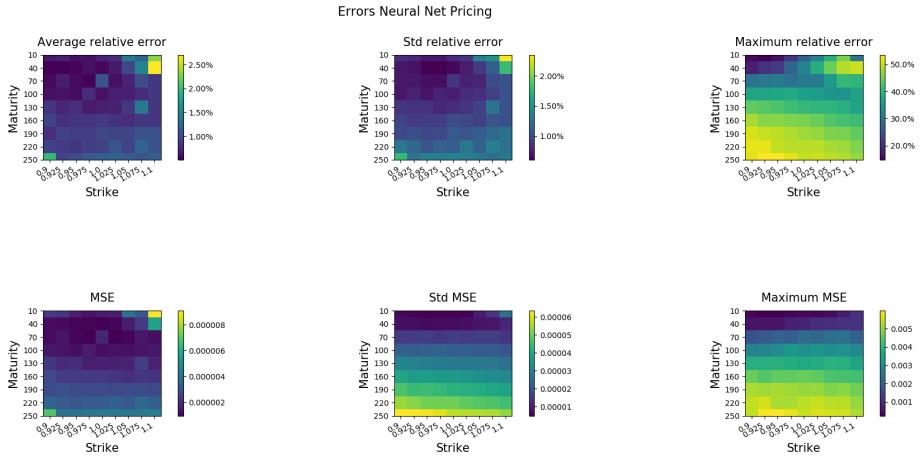


FIGURE 5.2: Test Sample Performance of the Feed-Forward Volatility Pricing Network. The errors are always measured with respect to the closed-form solution.

stochastic dominance in the relative errors. We find no evidence against the claim that our proposed CNN approach is stochastically dominated by the FFNN approach as well as the Monte-Carlo approach. The estimated p-values are always smaller 10^{-50} for both test. We conclude that there is strong evidence that the proposed volatility pricing approach is dominated by the benchmark models with respect to the relative deviation. Summarising, the proposed approach fits volatilities with significant higher accuracy then the benchmark models. Additional Figures showing the empirical density functions of the MAPE for each approach can be found in Appendix C.2.

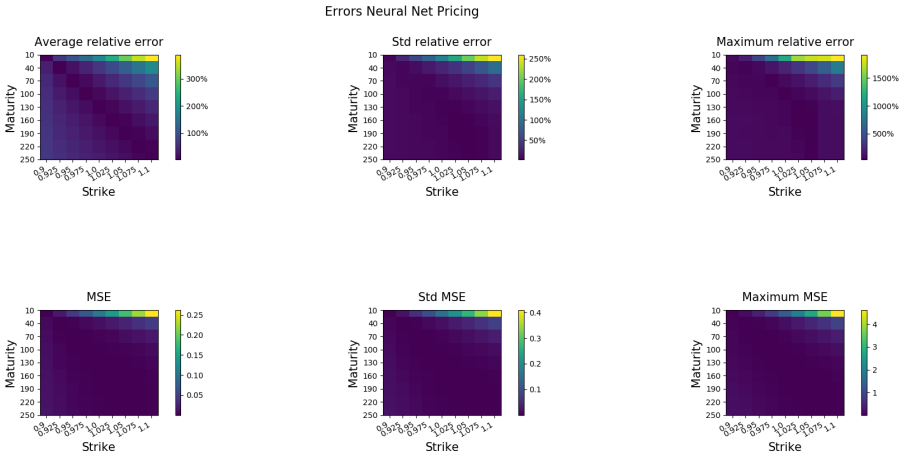


FIGURE 5.3: Test Sample Performance of the Volatility Pricing with Monte-Carlo (100.000 paths). The errors are always measured with respect to the closed-form solution.

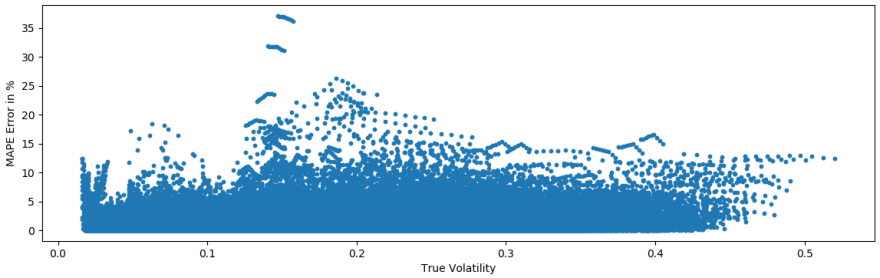


FIGURE 5.4: Test Sample Performance of the Volatility Pricing Network F. Scatter-Plot: True Volatility vs. Average Error

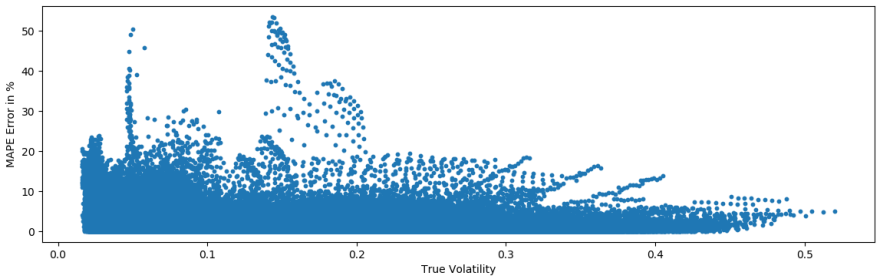


FIGURE 5.5: Test Sample Performance of the Feed-Forward Volatility Pricing Network. Scatter-Plot: True Volatility vs. Average Error

Table 5.2 states the average computation time for pricing one full surface. The FFNN approach leads to lowest times, while the CNN approach still prices in same magnitude of time. Both deep learning approach need less time to price one surface then the usage of the closed-form solution. Only the Monte-Carlo

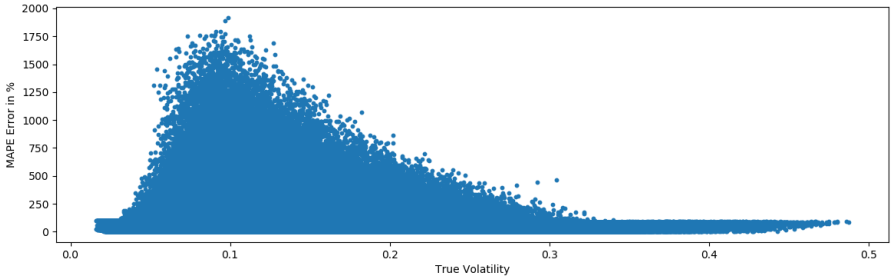


FIGURE 5.6: Test Sample Performance of the Volatility Pricing with Monte-Carlo (100.000 paths). Scatter-Plot: True Volatility vs. Average Error

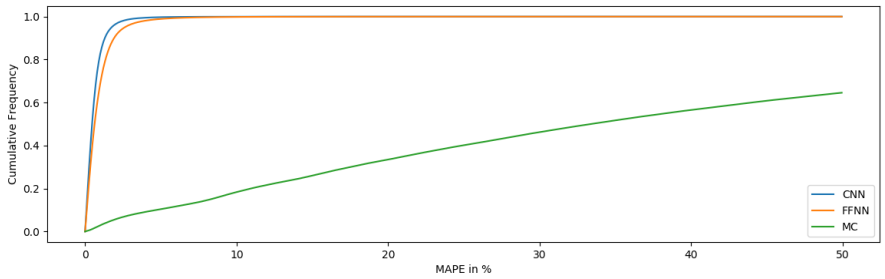


FIGURE 5.7: Test Sample Performance of the Volatility Pricing. Empirical CDF of the relative errors for each approach.

simulations does not price faster then the close form solution.

Approach	Avg. Comp. Time	Difference to the CNN
CNN	585.9µs	
FFNN	18.2µs	-96.9%
MC	593.6ms	101310.1%
closed-form	39.1ms	6673.5%

TABLE 5.2: Average Computation Time for Pricing a single Surface

5.1.2 Calibration

Analogously to the volatility pricing task, we compute the average percentage deviation for each parameter and calibration approach in Figures 5.8, 5.9 and 5.10. The CNN approach calibrates every parameter with the highest average and median accuracy. Thereby, the mean errors are at least two times lower than for the closest benchmark.

As the proposed CNN approach only soft-forces the parameters to fulfil the stationarity constraint, we analyse the performance for scenarios which

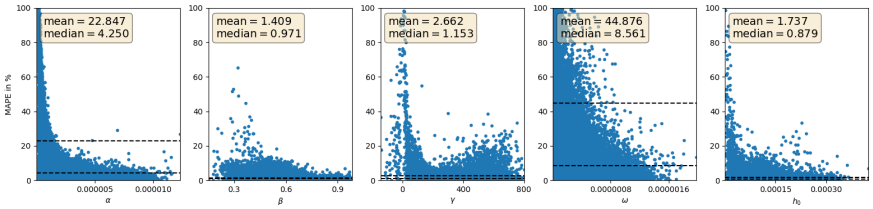


FIGURE 5.8: Calibration relative error per parameter in the testing set of the CNN

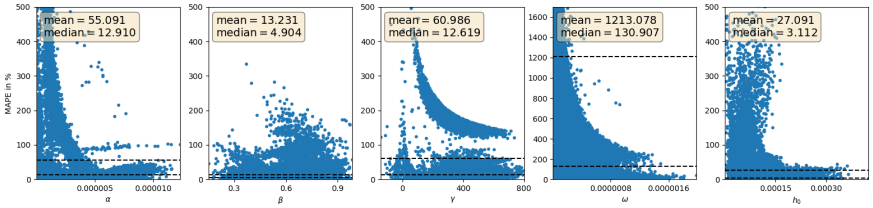


FIGURE 5.9: Relative error per parameter in the testing set after Levenberg-Marquardt-FFNN-Calibration.

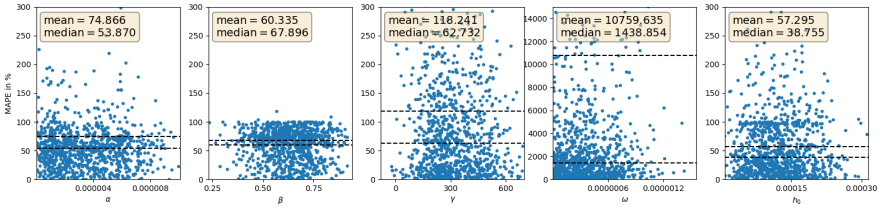


FIGURE 5.10: Relative error per parameter in the testing set after Interior-Point Calibration (first 1000 scenarios)

satisfy and those which do not separately. Figure 5.11 and 5.12 show the performance split by parameter. Out of 30878 parameter combinations only 65 do not fulfil the stationarity constraint. Even though the accuracy on these scenarios is worse, such forecasts are on average more accurate than the provided benchmarks. Nonetheless, this stresses the importance of carefully handling parameter constraints. A strong correlation between the true

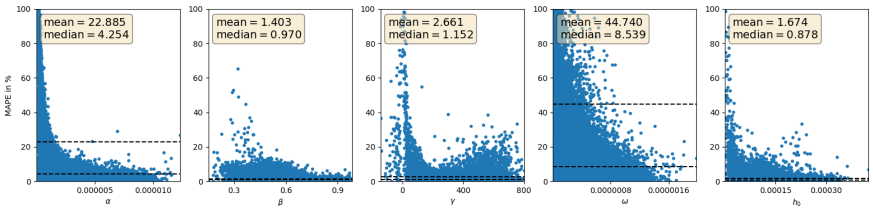


FIGURE 5.11: Calibration relative error per parameter in the test set of the CNN. Only parameter combination that fulfil the stationarity constraint

parameters size and the corresponding error is noticeable for both neural

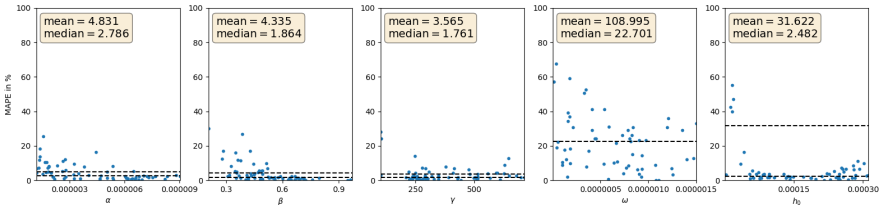


FIGURE 5.12: Calibration relative error per parameter in the test set of the CNN. Only parameter combination that do not fulfil the stationarity constraint

network approaches. Especially for α and ω, the relative errors magnify for values close to zeros. For the classic calibration approach using an interior-point algorithm such an effect can not be observed. Table 5.3 shows the Spearman's rank correlation between the absolute parameter size and the corresponding relative deviation from the true parameter. For the CNN approach, a significance rank correlations between the parameters and the error is observed. Except for β, this holds also true for the FFNN-Levenberg approach. Only for the interior-point algorithm no significant correlations are observed. This indicates that the neural network based approaches perform worse for numerically difficult values than classic algorithms. In terms of

θ	α	β	γ^*	ω	h_0
CNN	-0.8459 (0)	-0.4362 (0)	-0.1609 (0)	-0.5335 (0)	-0.3242 (0)
FFNN-Levenberg	-0.3722 (0)	-0.0036 (0.5300)	0.0817 (0)	-0.6687 (0)	-0.5725 (0)
Interior-Point	-0.02156 (0.4959)	-0.0166 (0.5992)	-0.0091 (0.7750)	0.0163 (0.6063)	-0.0037 (0.9077)

TABLE 5.3: Spearman's Rank Correlation (and corresponding two-sided p-value of H_0 : both samples are uncorrelated. 0 is stated if a p-value is smaller than 10^{-30}) between relative error and absolute parameters size of the testing set

efficiency the CNN approach outperforms both benchmarks significantly. Our proposed approach needs 5000 times less computation time than the FFNN-Levenberg approach to calibrate one surface and 26000 times less computation time than the interior-point algorithm. Another advantage of the CNN approach is that it can handle collection of surfaces at once while both benchmark can only calibrate one surface at the time.

Approach	Avg. Comp. Time	Difference to the CNN
CNN	286.1µs	
FFNN-Levenberg	1423.2ms	4974.5%
Interior-Point	74.9s	26223.7%

TABLE 5.4: Average Computation Time for one Surface Calibration

5.1.3 Autoencoder

In application it is of utmost importance that the calibrated parameter do not only fit the true parameters but lead to good volatility surfaces. To ensure that our approach leads to parameters which induce good surfaces, we consider the mapping $Auto : \mathbb{R}^{m\times n} \to \mathbb{R}^{m\times n}$ with $Auto := F \circ G \approx P \circ P^{-1} \overset{!}{=} \mathrm{id}_{\mathbb{R}^{m\times n}}$. We claim the trained network $F \circ G$ provides a good approximation of $\mathrm{id}_{\mathbb{R}^{m\times n}}$. As the HNG model is really sensitive in some parameters, a small deviation can lead to strong fluctuations in option prices. One has no guarantee, that a presumed good calibration mapping leads to fitting surfaces at a out-of-sample comparison. To support our claim, we proceed as follows: First the trained network G is used to to calibrate parameters from the surfaces of the testing set. Those forecast are priced by the network F. Table 5.5 summarises the results. The performance of the auto-encoding network $F \circ G$ does not differ significantly the pricing performance of the network F. Similar to the results of the previous sections, we observe that parameters which do not fulfil the stationarity constraint perform slightly worse than those who do. Interestingly, such scenarios show a lower maximum deviation overall surfaces. But, this finding might be a result of the small sample size and is probably not statistical significant.

Figure 5.13, 5.14 and 5.15 show the average errors split by maturity and

	Mean	Median	Max	Count
no constraint violation	0.6260	0.4595	37.1354	30813
constraint violation	2.4141	1.5671	17.2712	65
total	0.6298	0.4603	37.1354	30878

TABLE 5.5: Descriptive Statistics of the Autoencoder Network $F \circ G$ with respect to MAPE in %

strike. Analogously to the results from Section 5.1.1, the smaller the maturity the smaller the average deviation. We conclude that the proposed CNN calibration approach does not only results in parameter which differ less from

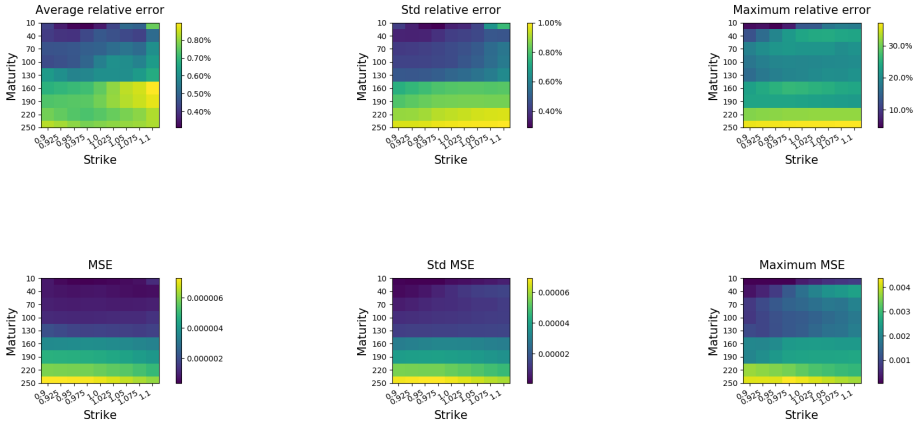


FIGURE 5.13: Test Sample Performance of the Autoencoder Network $F \circ G$. The errors are always measured with respect to the closed-form solution.

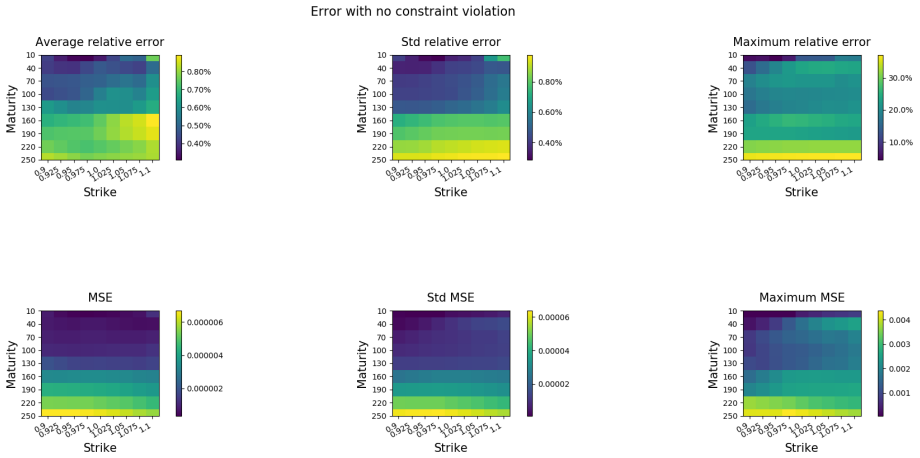


FIGURE 5.14: Test Sample Performance of the Autoencoder Network $F \circ G$. The errors are always measured with respect to the closed-form solution. Only parameter combinations which fulfil the stationarity constraint

the real parameters then the given benchmarks, but also leads to accurate surfaces if used for pricing. We also analysed the inverse autoencoder network $F \circ G \approx \mathrm{id}_{\Theta^{HNG}}$. As this network has no straight-forward practice application or easy interpretation, the results can be found in Appendix C.2.

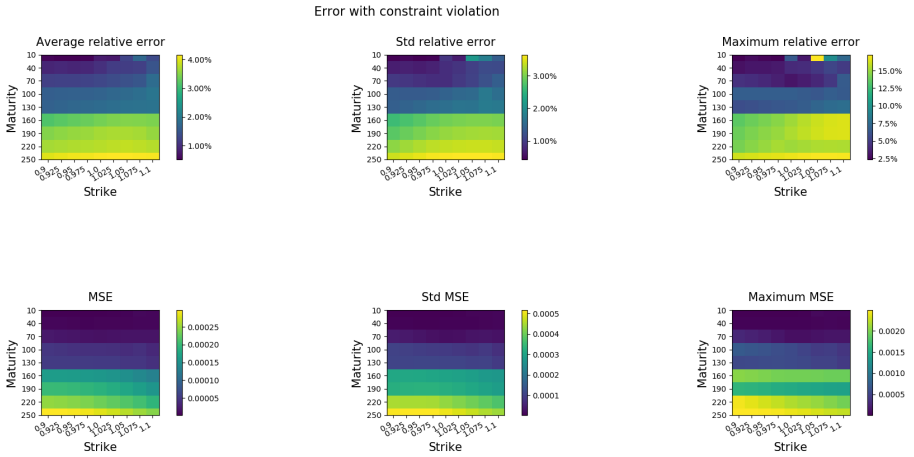


FIGURE 5.15: Test Sample Performance of the Autoencoder Network $F \circ G$. The errors are always measured with respect to the closed-form solution. Only parameter combinations which that do not fulfil the stationarity constraint

5.1.4 Different Datasets

Finally, we want to test the performance of our network when applied to other datasets. We proceed in two steps. First, we train the neural networks introduced in Section 4.3.1 and 4.4.1 on several different datasets. Secondly, we use already trained neural networks and test their performance on differently generated datasets. While the first task aims on the ability of our proposed neural network architecture to approximate discrete-time volatility models, the second task focuses on the generability of the pre-trained neural network. Hereby, we compare the performance on the dataset generated from a normal distribution (as seen previous experiments) to datasets generated from the uniform symmetric distribution and the log-normal distribution as introduced in Section 4.6.

The results of the volatility pricing task where the network is trained on the new datasets is shown in Figure 5.16 and 5.18. The approximation is as accurate as approximation of the original dataset. This also applies to the calibration task as seen in Figures 5.17 and 5.19. We conclude that the chosen network structure can be used for different dataset types and indicates a usability for different discrete-time models. Furthermore, we analyse the performance of the networks trained on the normal distribution dataset when applied to other differently generated datasets. Interestingly, both pre-trained networks perform roughly as accurate as the networks which are trained on the generated datasets. Nonetheless, for the log-normal distributed dataset,

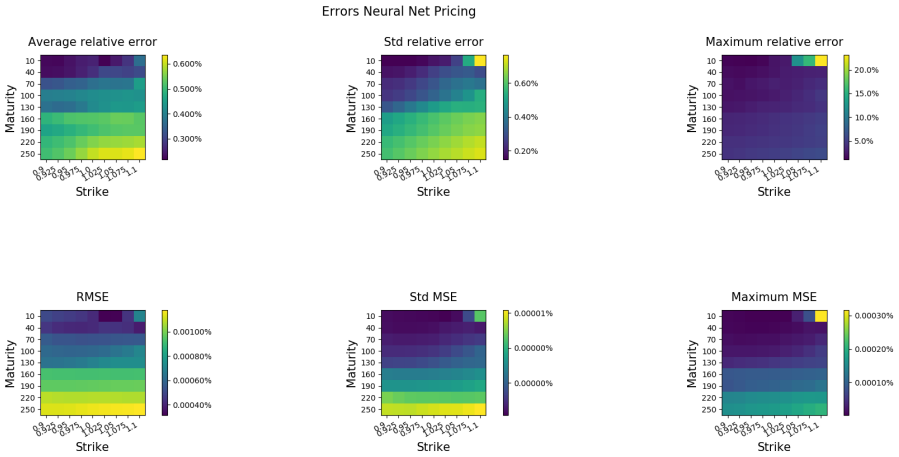


FIGURE 5.16: Performance of the Volatility Pricing Network F based on the uniform symmetric distributed testing set. The errors are always measured with respect to the closed-form solution.

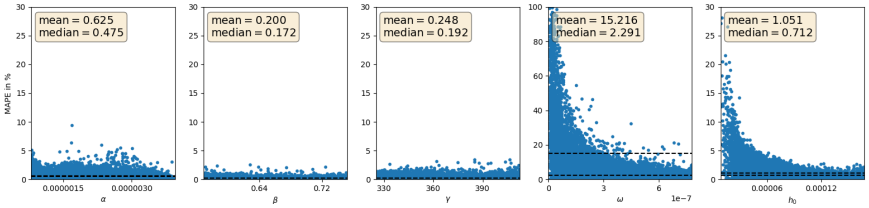


FIGURE 5.17: Calibration relative errors per parameter in the uniform symmetric testing set of trained CNN.

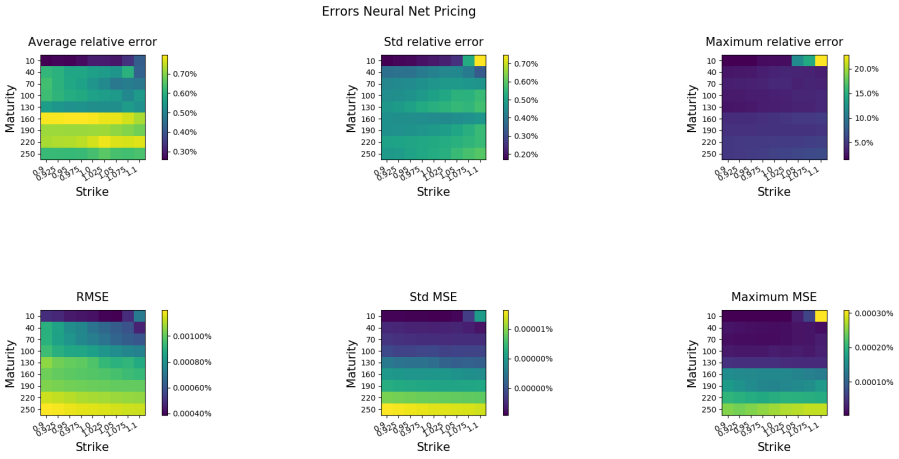


FIGURE 5.18: Performance of the Volatility Pricing Network F based on the log-normal distributed testing set. The errors are always measured with respect to the closed-form solution.

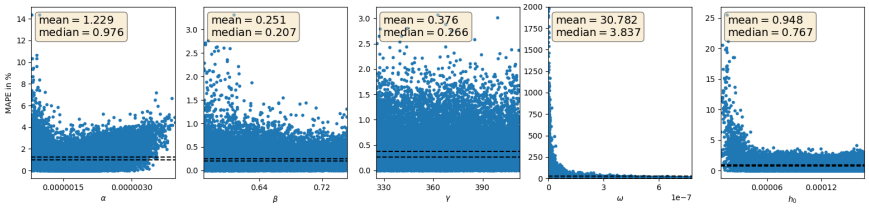


FIGURE 5.19: Calibration relative errors per parameter in the log-normal testing set of trained CNN.

an extreme dependency of the error size to the parameters is observed. This might results from logarithmic data transformation. The pre-trained networks are not able to fully capture the dynamic of the transformed data. The results

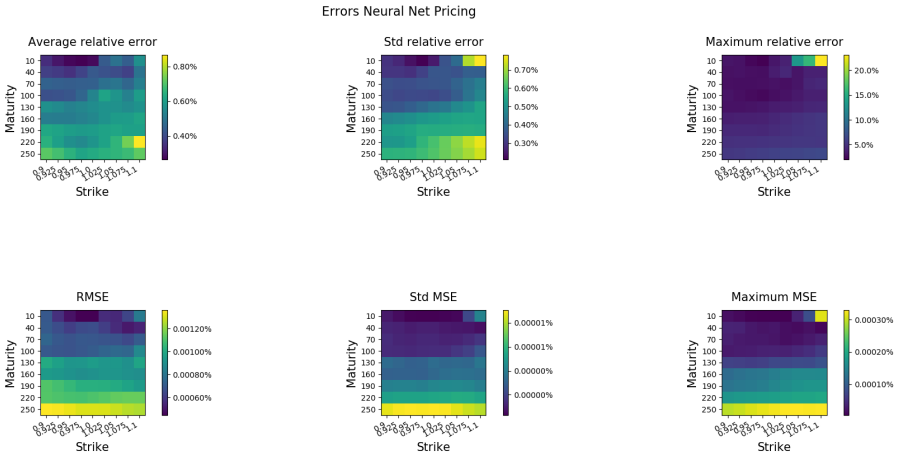


FIGURE 5.20: Performance of the pre-trained Volatility Pricing Network F based on the uniform symmetric distributed testing set. The errors are always measured with respect to the closed-form solution.

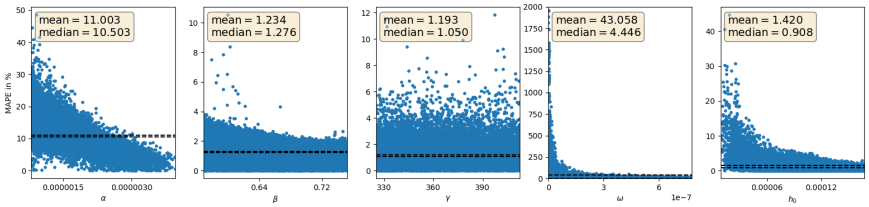


FIGURE 5.21: Calibration relative errors per parameter in the uniform symmetric testing set of pre-trained CNN.

show that the network structure performs well on different dataset implying that the proposed structure is able to capture the underlying structure of the HNG model and the good performance is not solely based on the right

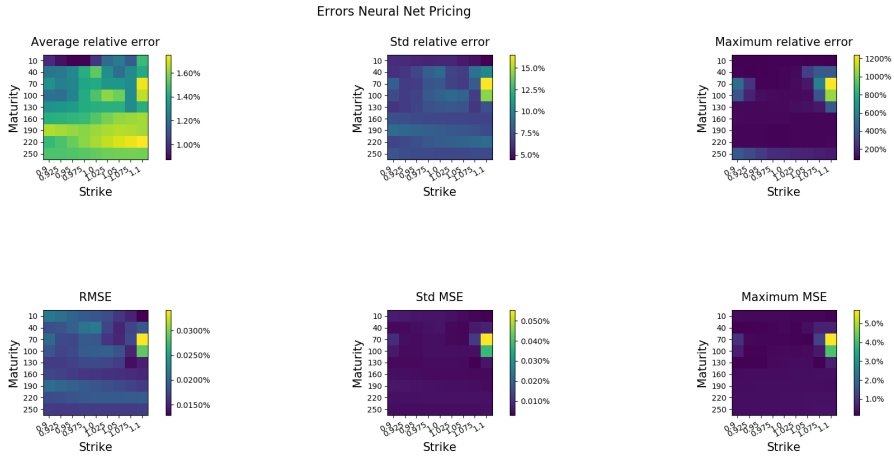


FIGURE 5.22: Performance of the pre-trained Volatility Pricing Network F based on the log-normal distributed testing set. The errors are always measured with respect to the closed-form solution.

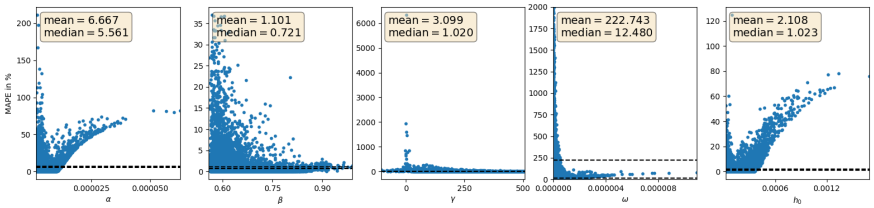


FIGURE 5.23: Calibration relative errors per parameter in the log-normal testing set of pre-trained CNN.

choice of data. When using a pre-trained network on different dataset the approximation and calibration quality decreases, especially for small values. However, the results are still comparable. Especially for the calibration task it is a surprising results as it is considered the more sophisticated problem.

5.2 Conclusions and Outlook

We show that our approach can be used as highly efficient and accurate tool for calibration and pricing of the HNG(1,1) model. It significantly outperforms all tested benchmark models in terms of accuracy as well as calibration speed. Horvath, Muguruza, and Tomas (2019) state that "neural networks have the potential to efficiently approximate complex functions, which are difficult to represent and time-consuming to evaluate by other means. Using deep neural networks [...] to approximate the pricing map (or equivalently the implied volatility mapping) from parameters of traditional models to (approximate) shapes of the implied volatility surfaces opens up a whole new landscape for financial modelling, while maintaining control over reliability and interpretability of network outputs". The results of the conducted numerical experiments imply that this hold true even for discrete-time models.
The numerical tests show that the proposed network architectures are an extremely powerful tool to approximate implied volatilities. Nonetheless, they are highly dependent on the choice of the grid. Going forward, a more elastic framework which allows for variability in the grid while still being computationally efficient, would certainly be of use. Such an improvement would simplify the incorporation of observed option prices as such contract might not correspond to a point on the predefined grid. Also, the calibration approach only soft-forces the parameters to fulfil stationarity constraints and we show that the performance decrease if such constraints are violated. Hence, the penalisation of constraints should be refined in future work.
Some research opposes the assumption of first order homogeneity as it can be link to dynamics in delta hedging which might not be desirable (see for example Kreps and Wallis (1997)). Even though this work does not answer questions related to model choice or quality, we exploited the homogeneity of HNG(1,1). The proposed approach is not able to simultaneously capture different values of the underlying asset. Therefore, further adjustments to this approach with respect to the underlying should be made to capture non-homogeneity.
Again, in view on real applications, it is utmost important to apply this procedure to more sophisticated models. Especially models with no closed-form solution available should be tackled thoroughly. Clearly, much more empirical research is needed to verify the usefulness of deep learning frameworks in the field of option pricing.

Appendix A

Properties of HNG(1,1)

This is a collection of properties of the HNG(1,1) model under $\mathbb{P}$. It provides additional proofs and calculus for Chapter 3.

Leverage Effect

$$\begin{aligned}
\operatorname{Cov}\left[h_{t+1}, Y_t \middle| \mathcal{F}_{t-1}\right] &= \operatorname{Cov}\left[\omega + \alpha(z_t - \gamma\sqrt{h_t})^2 + \beta h_t, r + \lambda h_t + \sqrt{h_t} z_t \middle| \mathcal{F}_{t-1}\right] \\
&= \alpha\lambda \operatorname{Cov}\left[(z_t - \gamma\sqrt{h_t})^2, h_t \middle| \mathcal{F}_{t-1}\right] && \text{(a)} \\
&\quad + \alpha \operatorname{Cov}\left[(z_t - \gamma\sqrt{h_t})^2, \sqrt{h_t} z_t \middle| \mathcal{F}_{t-1}\right] && \text{(b)} \\
&\quad + \beta \operatorname{Cov}\left[h_t, \sqrt{h_t} z_t \middle| \mathcal{F}_{t-1}\right] && \text{(c)} \\
&\quad + \lambda\beta \mathbb{V}\left[h_t \middle| \mathcal{F}_{t-1}\right] && \text{(d)}
\end{aligned}$$

Since $(z_t - \gamma\sqrt{h_t})^2) = z_t^2 - 2\gamma z_t\sqrt{h_t} + \gamma^2 h_t$, we get

$$\begin{aligned}
\frac{(a)}{\alpha\lambda} &= \operatorname{Cov}\left[z_t^2 - 2\gamma z_t\sqrt{h_t} + \gamma^2 h_t, h_t \middle| \mathcal{F}_{t-1}\right] \\
&= \operatorname{Cov}\left[z_t^2, h_t \middle| \mathcal{F}_{t-1}\right] - 2\gamma \operatorname{Cov}\left[z_t\sqrt{h_t}, h_t \middle| \mathcal{F}_{t-1}\right] + \gamma^2 \mathbb{V}\left[h_t \middle| \mathcal{F}_{t-1}\right] \\
&= 0
\end{aligned}$$

and

$$\begin{aligned}
\frac{(b)}{\alpha} &= \operatorname{Cov}\left[z_t^2 - 2\gamma z_t\sqrt{h_t} + \gamma^2 h_t, z_t\sqrt{h_t} \middle| \mathcal{F}_{t-1}\right] \\
&= -2\gamma \mathbb{V}\left[z_t\sqrt{h_t} \middle| \mathcal{F}_{t-1}\right] = -2\gamma \mathbb{E}\left[h_t \middle| \mathcal{F}_{t-1}\right] = -2\gamma h_t
\end{aligned}$$

Furthermore it holds $(c) = (d) = 0$ and we conclude

$$\operatorname{Cov}\left[h_{t+1}, Y_t \middle| \mathcal{F}_{t-1}\right] = -2\alpha\gamma h_t.$$

Squared Process

$$\begin{aligned}
h_t^2 =&(\omega + \beta h_{t-1} + \alpha(z_{t-1} - \gamma\sqrt{h_{t-1}})^2))^2 \\
=&\omega^2 + \beta^2 h_{t-1}^2 + 2\omega\beta h_{t-1} + 2\alpha(\omega + \beta h_{t-1})(z_{t-1} - \gamma\sqrt{h_{t-1}})^2 \\
&+ \alpha^2(z_{t-1}^4 - 4\gamma z_{t-1}^3\sqrt{h_{t-1}} + 6\gamma^2 z_{t-1}^2 h_{t-1} - 4\gamma^3 z_{t-1} h_{t-1}^{\frac{3}{2}} + \gamma^4 h_{t-1}^2)
\end{aligned}$$

Under Stationarity of $(h_t)_t$ and $(h_t^2)_t$, it holds

$$\begin{aligned}
\mathbb{E}[h_t^2] =&\omega^2 + \beta^2\,\mathbb{E}[h_t^2] + 2\omega\beta\,\mathbb{E}[h_t] + \alpha^2(3 + 6\gamma^2\,\mathbb{E}[h_t] + \gamma^4\,\mathbb{E}[h_t^2)] \\
&+ 2\alpha\omega + 2\alpha(\omega\gamma^2 + \beta)\,\mathbb{E}[h_t] + 2\alpha\beta\gamma^2\,\mathbb{E}[h_t^2] \\
=&\omega^2 + 3\alpha^2 + 2\alpha\omega + (2\omega\beta + 6\alpha^2\gamma^2 + 2\alpha(\omega\gamma^2 + \beta))\,\mathbb{E}(h_t) \\
&+ (\beta^2 + \alpha^2\gamma^4 + 2\alpha\beta\gamma^2)\,\mathbb{E}(h_t^2)
\end{aligned}$$

which leads to

$$\begin{aligned}
\mathbb{E}[h_t^2] &= \frac{\omega^2 + 3\alpha^2 + 2\alpha\omega + (2\omega\beta + 6\alpha^2\gamma^2 + 2\alpha(\omega\gamma^2 + \beta))\,\mathbb{E}[h_t]}{1 - (\beta^2 + \alpha^2\gamma^4 + 2\alpha\beta\gamma^2)} \\
&= \frac{(\omega^2 + 3\alpha^2 + 2\alpha\omega)(1 - \beta - \alpha\gamma^2) + (2\omega\beta + 6\alpha^2\gamma^2 + 2\alpha(\omega\gamma^2 + \beta))(\omega + \alpha)}{(1 - \beta^2 - \alpha^2\gamma^4 - 2\alpha\beta\gamma^2)(1 - \beta - \alpha\gamma^2)}
\end{aligned}$$

and gives directly a sufficient condition for stationarity of $(h_t^2)_t$, namely

$$1 \neq \beta^2 + \alpha^2\gamma^4 + 2\alpha\beta\gamma^2 = (\beta + \alpha\gamma^2)^2,$$

which is directly satisfied by the stationary constraint of $(h_t)_t$. We conclude $(\beta + \alpha\gamma^2)^2 < 1$.

Unconditional Variance of Returns

$$\begin{aligned}
\mathbb{V}[Y_t] &= \mathbb{E}\left[\mathbb{V}\left[Y_t \middle| \mathcal{F}_{t-1}\right]\right] + \mathbb{V}\left[\mathbb{E}\left[Y_t \middle| \mathcal{F}_{t-1}\right]\right] \\
&= \mathbb{E}[h_t] + \mathbb{V}[r + \lambda h_t] \\
&= \mathbb{E}[h_t] + \lambda^2\,\mathbb{V}[h_t] \\
&= \mathbb{E}[h_t] + \lambda^2(\mathbb{E}[h_t^2] - \mathbb{E}[h_t]^2)
\end{aligned}$$

Calculation of $\mathbb{E}[h_t^2] - \mathbb{E}[h_t]^2$:

$$\begin{aligned}
&(\mathbb{E}[h_t^2] - \mathbb{E}[h_t]^2)(1 - \alpha\gamma^2 - \beta) \\
=&\frac{(\omega^2 + 3\alpha^2 + 2\alpha\omega)(1 - \beta - \alpha\gamma^2) + (2\omega\beta + 6\alpha^2\gamma^2 + 2\alpha(\omega\gamma^2 + \beta))(\omega + \alpha)}{1 - \beta^2 - \alpha^2\gamma^4 - 2\alpha\beta\gamma^2} - \omega - \alpha \\
=&\frac{(\omega^2 + 3\alpha^2 + 2\alpha\omega)(1 - \beta - \alpha\gamma^2) + (2\omega\beta + 6\alpha^2\gamma^2 + 2\alpha(\omega\gamma^2 + \beta))}{1 - \beta^2 - \alpha^2\gamma^4 - 2\alpha\beta\gamma^2} - 1
\end{aligned}$$

Furthermore, we get:

$$\begin{aligned}
&(\mathbb{E}[h_t^2] - \mathbb{E}[h_t]^2)(1 - \alpha\gamma^2 - \beta)(1 - (\beta + \alpha\gamma^2)^2) \\
=&(\omega^2 + 3\alpha^2 + 2\alpha\omega)(1 - \beta - \alpha\gamma^2) + (2\omega\beta + 6\alpha^2\gamma^2 + 2\alpha(\omega\gamma^2 + \beta) \\
&- (1 - (\beta + \alpha\gamma^2)^2) \\
=&\omega^2 + 3\alpha^2 + 2\alpha\omega - \beta\omega^2 - 3\beta\alpha^2 - 2\beta\alpha\omega - \alpha\gamma^2\omega^2 - 3\gamma^2\alpha^3 - 2\gamma^2\alpha^2\omega \\
&+ 2\omega\beta + 6\alpha^2\gamma^2 + 2\alpha\omega\gamma^2 + 2\alpha\beta - 1 + (\beta + \alpha\gamma^2)^2 \\
=&(\omega + \alpha)^2(1 - \beta - \alpha\gamma^2) + (\beta + \alpha\gamma^2)^2 + 2\alpha^2(1 - \beta - \alpha\gamma^2) \\
&+ 2\omega\beta + 6\alpha^2\gamma^2 + 2\alpha\omega\gamma^2 + 2\alpha\beta - 1 \\
=&(\omega + \alpha)^2(1 - \beta - \alpha\gamma^2) + (\beta + \alpha\gamma^2)^2 + 2\alpha^2((1 - \beta - \alpha\gamma^2) + 3\gamma^2) \\
&+ (\beta + \alpha\gamma^2)(6\alpha + 2\omega) - 4\alpha\beta - 1
\end{aligned}$$

Forecasts

For a second order stationary process the optimal linear forecast is given by the conditional expectation. A proof of this claim is given in Beran (2018). The 1-step optimal forecast of the process $(h_t)_t$ at time t is

$$\begin{aligned}
\mathbb{E}\left[h_t \middle| \mathcal{F}_{t-1}\right] &= r + \lambda\, \mathbb{E}\left[\omega + \alpha(z_t - \gamma\sqrt{h_t})^2) + \beta h_t \middle| \mathcal{F}_t\right] \\
&= r + \lambda\, \mathbb{E}\left[\omega + \alpha(z_t^2 + \gamma^2 h_t - 2\gamma z_t\sqrt{h_t}) \middle| \mathcal{F}_t\right] \\
&= r + \lambda(\omega + \alpha(1 + \gamma h_t)).
\end{aligned}$$

Hence the optimal 1-step forecast and long-term expectation for log returns equals

$$\begin{aligned}
\mathbb{E}\left[Y_t \middle| \mathcal{F}_{t-1}\right] &= (1 + \lambda)r + \lambda^2(\omega + \alpha(1 + \gamma h_t)) \\
\mathbb{E}[Y_t] &= r + \lambda\, \mathbb{E}[h_t] = r + \lambda\frac{\omega + \alpha}{1 - \beta - \alpha\gamma^2}.
\end{aligned}$$

Homogeneity in S_t and K

We provide an additional proof for the homogeneity property of the HNG(1,1) model:

$$\begin{aligned}
&\mathbb{E}_{\mathbb{Q}}\left[(S_T-K)^+|\mathcal{F}_t\right]\\
=&\frac{1}{2}(e^{r(T-t)}S_t-K)-\frac{1}{\pi}\int_0^\infty\frac{1}{\phi}\,\mathrm{Re}\left(iK^{-i\phi}(f_t(i\phi+1)-Kf_t(i\phi))\right)d\phi
\end{aligned}$$

Clearly, the first term is linear in S_t and K. Hence, we focus on the integral part of the above expression. For $\lambda\in\mathbb{R}^+$ consider

$$\begin{aligned}
&\lambda\int_0^\infty\frac{1}{\phi}\,\mathrm{Re}\left(iK^{-i\phi}(f_t(i\phi+1)-Kf_t(i\phi))\right)d\phi\\
=&\int_0^\infty\frac{1}{\phi}\,\mathrm{Re}\left(\lambda iK^{-i\phi}(f_t(i\phi+1)-Kf_t(i\phi))\right)d\phi\\
=&\int_0^\infty\frac{1}{\phi}\,\mathrm{Re}\left(i(\lambda K)^{-i\phi}(\lambda^{i\phi+1}f_t(i\phi+1)-\lambda K\lambda^{i\phi}f_t(i\phi))\right)d\phi\\
=&\int_0^\infty\frac{1}{\phi}\,\mathrm{Re}\Big(i(\lambda K)^{-i\phi}\Big((\lambda S_t)^{i\phi+1}exp\left(A_t(i\phi+1)+B_t(i\phi+1)h_{t+1}\right)\\
&\quad-\lambda K(\lambda S_t)^{i\phi}\exp\left(A_t(i\phi)+B_t(i\phi)h_{t+1}\right)\Big)\Big)d\phi,
\end{aligned}$$

which proves the homogeneity of the HNG(1,1) model in S_t and K.

Appendix B

MATLAB and Python 3.7 Code

This section includes specific code examples. All code files used for this thesis are publicly available at Github.com/henrikbrautmeier/HNGDeepVola
The following code provides a minimalistic example to build the proposed neural networks with Python 3.7 using Tensorflow 2.1 and Keras Backend.

```
import numpy as np
from tensorflow.compat.v1.keras.models import Sequential,Model
from tensorflow.compat.v1.keras.layers import Reshape,InputLayer
    ,Dense,Flatten, Conv2D,Conv1D, Dropout, Input,ZeroPadding2D,
    ZeroPadding1D,MaxPooling2D
from tensorflow.compat.v1.keras import backend as K
from tensorflow.compat.v1.keras.callbacks import EarlyStopping
import tensorflow as tf
from tensorflow.compat.v1.keras.backend import set_session
from tensorflow.python.client import device_lib
config = tf.compat.v1.ConfigProto( device_count = {'GPU': 1 , '
    CPU': 56} )
sess = tf.compat.v1.Session(config=config)
tf.compat.v1.keras.backend.set_session(sess)
print(device_lib.list_local_devices())
tf.compat.v1.keras.backend.set_floatx('float64')

def root_relative_mean_squared_error(y_true, y_pred):
        return K.sqrt(K.mean(K.square((y_pred-y_true)/y_true)))
def mse_constraint(param):
    def rel_mse_constraint(y_true, y_pred):
            traf_a = 0.5*(y_pred[:,0]*diff[0]+bound_sum[0])
            traf_g = 0.5*(y_pred[:,2]*diff[2]+bound_sum[2])
            traf_b = 0.5*(y_pred[:,1]*diff[1]+bound_sum[1])
            constraint = traf_a*K.square(traf_g)+traf_b
            return K.mean(K.square(y_pred - y_true)) +param*K.
    mean(K.greater(constraint,1))
    return rel_mse_constraint

```

```
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1,
    patience = 50 ,restore_best_weights=True)

# Volatility Pricing Network
NN1a = Sequential()
NN1a.add(InputLayer(input_shape=(Nparameters,1,1,)))
NN1a.add(ZeroPadding2D(padding=(2, 2)))
NN1a.add(Conv2D(32, (3, 1), padding='valid',use_bias =True,
    strides =(1,1),activation='elu'))
NN1a.add(ZeroPadding2D(padding=(3,1)))
NN1a.add(Conv2D(32, (2, 2),padding='valid',use_bias =True,
    strides =(1,1),activation ='elu'))
NN1a.add(Conv2D(32, (2, 2),padding='valid',use_bias =True,
    strides =(2,1),activation ='elu'))
NN1a.add(ZeroPadding2D(padding=(2,2)))
NN1a.add(Conv2D(32, (2, 2),padding='valid',use_bias =True,
    strides =(1,1),activation ='elu'))
NN1a.add(ZeroPadding2D(padding=(1,1)))
NN1a.add(Conv2D(32, (2, 2),padding='valid',use_bias =True,
    strides =(2,1),activation ='elu'))
NN1a.add(ZeroPadding2D(padding=(1,1)))
NN1a.add(Conv2D(32, (2, 2),padding='valid',use_bias =True,
    strides =(2,1),activation ='elu'))
NN1a.add(ZeroPadding2D(padding=(1,1)))
NN1a.add(Conv2D(32, (3,3),padding='valid',use_bias =True,strides
     =(2,2),activation ='elu'))
NN1a.add(ZeroPadding2D(padding=(2,1)))
NN1a.add(Conv2D(32, (2, 2),padding='valid',use_bias =True,
    strides =(2,1),activation ='elu'))
NN1a.add(Conv2D(32, (2, 2),padding='valid',use_bias =True,
    strides =(2,1),activation ='elu'))
NN1a.add(ZeroPadding2D(padding=(2,1)))
NN1a.add(Conv2D(32, (2, 2),padding='valid',use_bias =True,
    strides =(2,1),activation ='elu'))
NN1a.add(Conv2D(Nstrikes, (2, 1),padding='valid',use_bias =True,
    strides =(2,1),activation ='linear', kernel_constraint = tf.
    keras.constraints.NonNeg()))
NN1a.summary()
NN1a.compile(loss = root_relative_mean_squared_error, optimizer
    = "adam",metrics=["MAPE","MSE"])
NN1b = Sequential()
NN1b.add(InputLayer(input_shape=(Nparameters,1,1,)))
NN1b.add(ZeroPadding2D(padding=(2, 2)))
NN1b.add(Conv2D(32, (3, 1), padding='valid',use_bias =True,
    strides =(1,1),activation='elu'))
NN1b.add(ZeroPadding2D(padding=(3,1)))
```

```
57 NN1b.add(Conv2D(32, (2, 2),padding='valid',use_bias =True,
       strides =(1,1),activation ='elu'))
58 NN1b.add(Conv2D(32, (2, 2),padding='valid',use_bias =True,
       strides =(2,1),activation ='elu'))
59 NN1b.add(ZeroPadding2D(padding=(2,1)))
60 NN1b.add(Conv2D(32, (2, 2),padding='valid',use_bias =True,
       strides =(1,1),activation ='elu'))
61 NN1b.add(ZeroPadding2D(padding=(1,0)))
62 NN1b.add(Conv2D(32, (2, 2),padding='valid',use_bias =True,
       strides =(2,1),activation ='elu'))
63 NN1b.add(ZeroPadding2D(padding=(1,1)))
64 NN1b.add(Conv2D(32, (2, 2),padding='valid',use_bias =True,
       strides =(2,1),activation ='elu'))
65 NN1b.add(ZeroPadding2D(padding=(1,1)))
66 NN1b.add(Conv2D(32, (3,3),padding='valid',use_bias =True,strides
       =(2,2),activation ='elu'))
67 NN1b.add(ZeroPadding2D(padding=(2,1)))
68 NN1b.add(Conv2D(32, (2, 2),padding='valid',use_bias =True,
       strides =(2,1),activation ='elu'))
69 NN1b.add(Conv2D(32, (2, 2),padding='valid',use_bias =True,
       strides =(2,1),activation ='elu'))
70 NN1b.add(ZeroPadding2D(padding=(2,1)))
71 NN1b.add(Conv2D(32, (2, 2),padding='valid',use_bias =True,
       strides =(2,1),activation ='elu'))
72 NN1b.add(Conv2D(Nstrikes, (2, 1),padding='valid',use_bias =True,
       strides =(2,1),activation ='linear', kernel_constraint = tf.
       keras.constraints.NonNeg()))
73 NN1b.summary()
74 NN1b.compile(loss = root_relative_mean_squared_error, optimizer
       = "adam",metrics=["MAPE","MSE"])
75
76 y_tmp_train = y_train[:,:,[0,1,2,3,4,5],:]
77 y_tmp_val = y_val[:,:,[0,1,2,3,4,5],:]
78 y_tmp_test= y_test[:,:,[0,1,2,3,4,5],:]
79 NN1a.fit(X_train, y_tmp_train, batch_size=64, validation_data =
       (X_val, y_tmp_val), epochs = 1000, verbose = True, shuffle
       =1,callbacks=[es])
80 y_tmp_train = y_train_[:,:,[5,6,7,8],:]
81 y_tmp_val = y_val[:,:,[5,6,7,8],:]
82 y_tmp_test= y_test[:,:,[5,6,7,8],:]
83 NN1b.fit(X_train, y_tmp_train, batch_size=64, validation_data =
       (X_val, y_tmp_val), epochs = 1000, verbose = True, shuffle
       =1,callbacks=[es])
84 prediction_a    = NN1a.predict(X_test_trafo).reshape((Ntest,6,
       Nstrikes))
```

```
prediction_b    = NN1b.predict(X_test_trafo).reshape((Ntest,4,
    Nstrikes))
prediction = np.concatenate((prediction_a[:,[0,1,2,3,4],:],
    prediction_b[:,[0,1,2,3],:]),axis =1)

# Calibration Network
NN2 = Sequential()
NN2.add(InputLayer(input_shape=(Nmaturities,Nstrikes,1)))
NN2.add(Conv2D(64,(3, 3),use_bias= True, padding='valid',strides
    =(1,1),activation ='tanh'))
NN2.add(Conv2D(64,(2, 2),use_bias= True,padding='valid',strides
    =(1,1),activation ='tanh'))
NN2.add(MaxPooling2D(pool_size=(2, 2)))
NN2.add(Conv2D(64,(2, 2),use_bias= True,padding='valid',strides
    =(1,1),activation ='tanh'))
NN2.add(ZeroPadding2D(padding=(1,1)))
NN2.add(Conv2D(64,(2, 2),use_bias= True,padding='valid',strides
    =(1,1),activation ='tanh'))
NN2.add(ZeroPadding2D(padding=(1,1)))
NN2.add(Conv2D(64,(2, 2),use_bias= True,padding='valid',strides
    =(1,1),activation ='tanh'))
NN2.add(Conv2D(64,(2, 2),use_bias= True,padding='valid',strides
    =(1,1),activation ='tanh'))
NN2.add(ZeroPadding2D(padding=(1,1)))
NN2.add(Conv2D(64,(2, 2),use_bias= True,padding='valid',strides
    =(1,1),activation ='tanh'))
NN2.add(ZeroPadding2D(padding=(1,1)))
NN2.add(Conv2D(64,(2, 2),use_bias= True,padding='valid',strides
    =(1,1),activation ='tanh'))
NN2.add(Flatten())
NN2.add(Dense(Nparameters,activation = 'linear',use_bias=True))
NN2.summary()
NN2.compile(loss =mse_constraint(0.75), optimizer = "adam",
    metrics=["MAPE", "MSE"])
NN2.fit(y_train,X_train, batch_size=50, validation_data = (y_val
    ,X_val), epochs=1000, verbose = True, shuffle=1,callbacks =
    [es])
```

Appendix C

Additional Results

C.1 Additional Results for Section 4.4

θ	2010	2011	2012	2013	2014	2015	2016	2017	2018
ω	$4.2779e-09$	$3.2992e-07$	$3.3648e-08$	$3.8491e-07$	$1.2743e-07$	$4.4951e-08$	$2.5272e-08$	$3.9321e-08$	$3.7781e-08$
std	$(1.6791e-08)$	$(1.5604e-06)$	$(1.6574e-07)$	$(1.3052e-06)$	$(4.5655e-07)$	$(2.0855e-07)$	$(1.4770e-07)$	$(1.7009e-07)$	$(2.2443e-07)$
α	$1.8528e-05$	$1.6271e-05$	$9.0589e-06$	$6.1070e-06$	$7.6946e-06$	$7.2374e-06$	$5.1346e-06$	$2.3951e-06$	$1.3159e-05$
std	$(1.9304e-05)$	$(2.1985e-05)$	$(1.2012e-05)$	$(7.9519e-06)$	$(9.6389e-06)$	$(7.2754e-06)$	$(5.8307e-06)$	$(3.0938e-06)$	$(1.6443e-05)$
β	0.6378	0.5560	0.7245	0.7258	0.6358	0.5520	0.6269	0.7263	0.5387
std	(0.2696)	(0.2971)	(0.2146)	(0.2478)	(0.2979)	(0.2466)	(0.2257)	(0.2586)	(0.3753)
γ^*	134.9727	191.7168	186.9011	254.4028	276.4433	280.6426	298.3299	331.9039	243.3202
std	(47.8695)	(93.1766)	(76.3909)	(194.7410)	(232.3643)	(175.7277)	(157.3293)	(112.0556)	(122.2386)
h_0^Q	$1.3056e-04$	$2.2460e-04$	$8.4830e-05$	$4.8801e-05$	$4.8652e-05$	0.0001	$7.5242e-05$	$1.9048e-05$	$1.3485e-04$
std	$(1.3959e-04)$	$(2.3120e-04)$	$(5.7765e-05)$	$(4.5932e-05)$	$(5.7911e-05)$	$(1.1307e-04)$	$(1.0294e-04)$	$(1.9023e-05)$	$(1.7128e-04)$
MSE	0.6499	1.0486	1.0785	0.7407	1.1260	1.2960	1.6303	1.7009	4.5699
MAPE	0.0672	0.0724	0.1105	0.1056	0.1224	0.1361	0.1324	0.1677	0.1248
OptLL	226.1068	234.9978	265.1968	365.6016	393.4111	469.1520	576.9261	651.9071	731.5026

TABLE C.1: Calibrated Parameters on Wednesdays with respect to Options-Likelihood

C.2 Additional Result of Section 5.1

Volatility Pricing

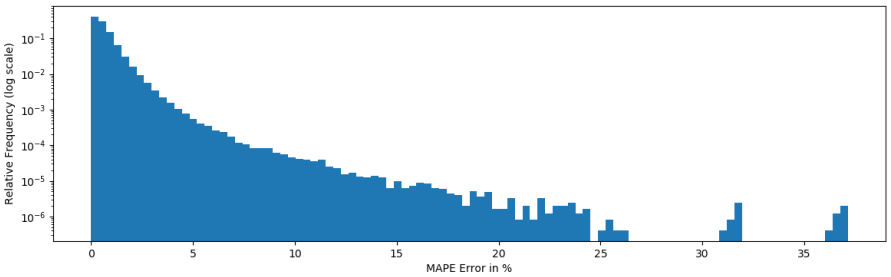


FIGURE C.1: Test Sample Performance of the Volatility Pricing Network F. Empirical Density of the relative error.

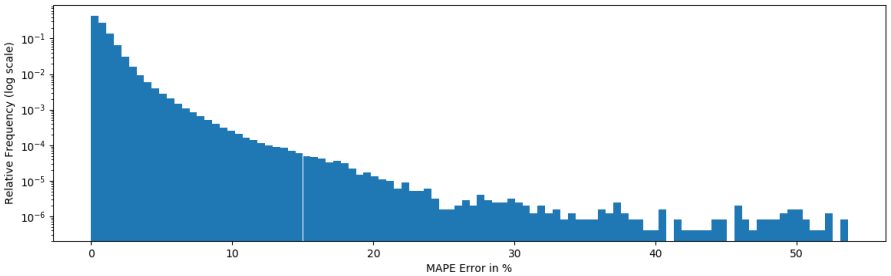


FIGURE C.2: Test Sample Performance of the Feed Forward Volatility Pricing Network. Empirical Density of the relative error.

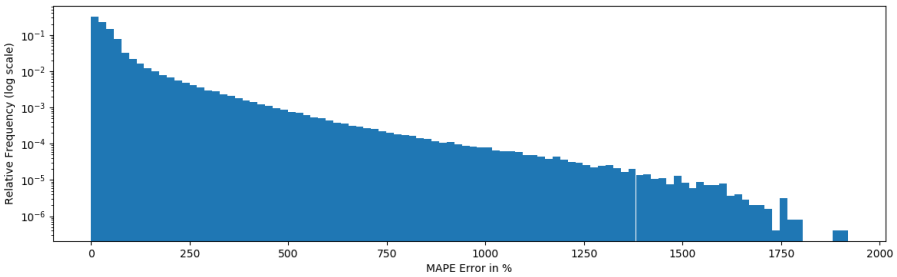


FIGURE C.3: Test Sample Performance of the Volatility Pricing with Monte-Carlo (100.000 paths). Empirical Density of the relative error.

Autoencoder

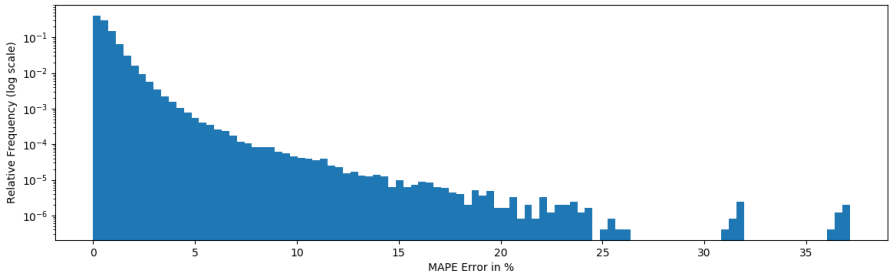


FIGURE C.4: Test Sample Performance of the Autoencoder Network $G \circ F$. Empirical Density of the relative error.

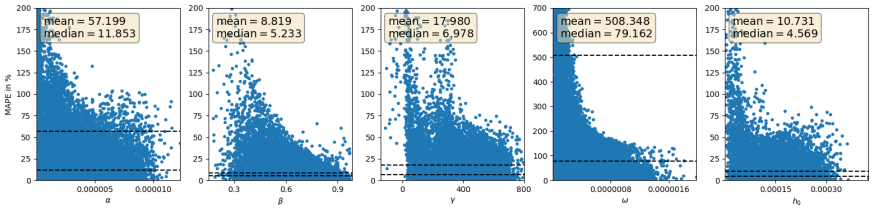


FIGURE C.5: Test Sample Performance of the Autoencoder Network $G \circ F$. Relative error per parameter in the testing set.

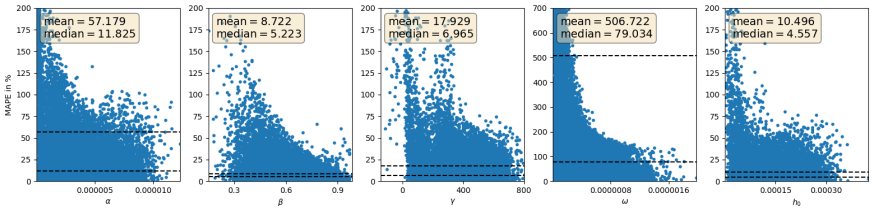


FIGURE C.6: Test Sample Performance of the Autoencoder Network $G \circ F$. Only parameter combinations which fulfill the stationarity constraint

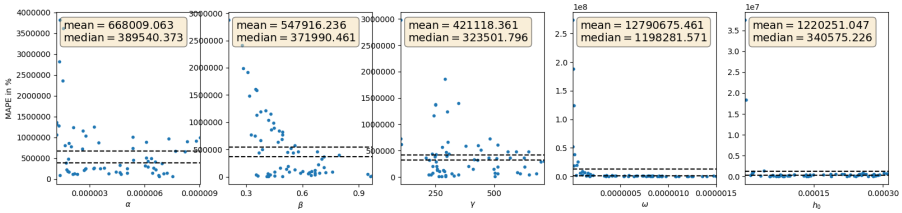


FIGURE C.7: Test Sample Performance of the Autoencoder Network $G \circ F$.Relative error per parameter in the testing set. Only parameter combinations which do not fulfill the stationarity constraint

Different Datasets

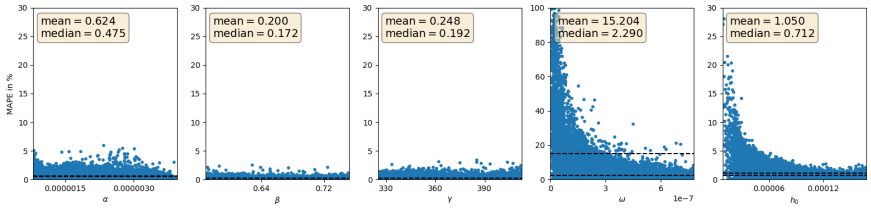


FIGURE C.8: Calibration relative error per parameter in the uniform symmetric distributed testing set of trained CNN. Only parameter combination that fulfill the stationarity constraint

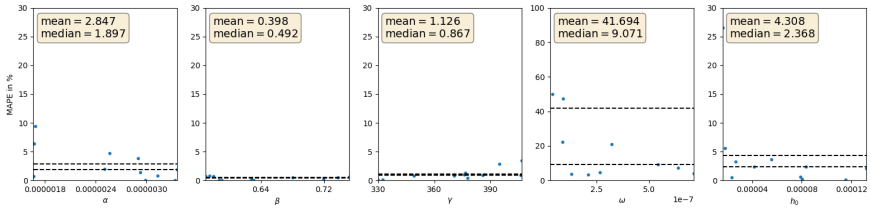


FIGURE C.9: Calibration relative error per parameter in the uniform symmetric testing set of trained CNN. Only parameter combination that do not fulfill the stationarity constraint

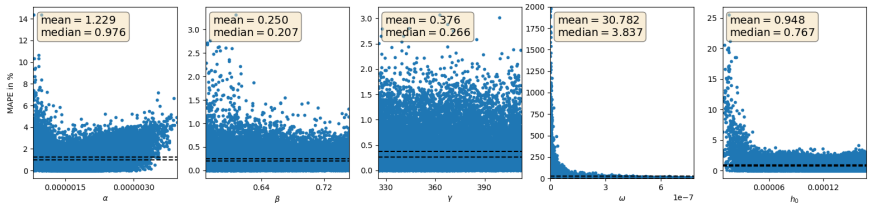


FIGURE C.10: Calibration relative error per parameter in the log-normal distributed testing set of trained CNN. Only parameter combination that fulfill the stationarity constraint

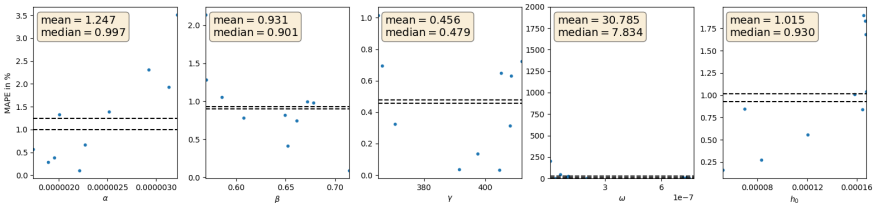


FIGURE C.11: Calibration relative error per parameter in the log-normal testing set of trained CNN. Only parameter combination that do not fulfill the stationarity constraint

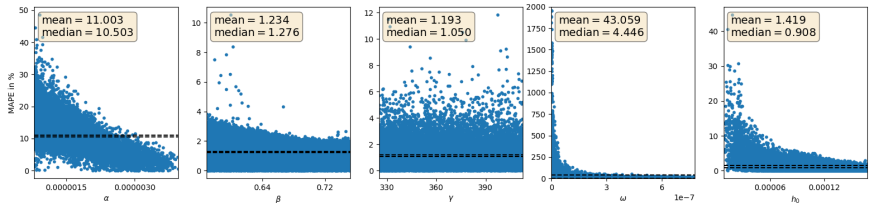


FIGURE C.12: Calibration relative error per parameter in the uniform symmetric distributed testing set of pre-trained CNN. Only parameter combination that fulfill the stationarity constraint

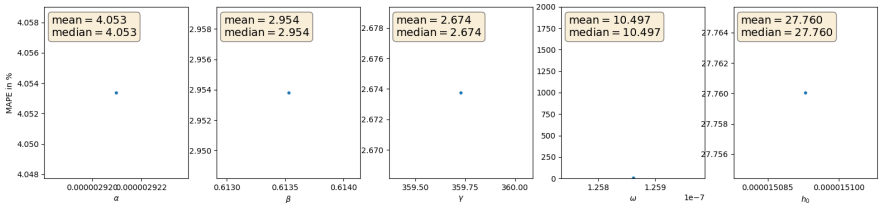


FIGURE C.13: Calibration relative error per parameter in the uniform symmetric distributed testing set of pre-trained CNN. Only parameter combination that do not fulfill the stationarity constraint

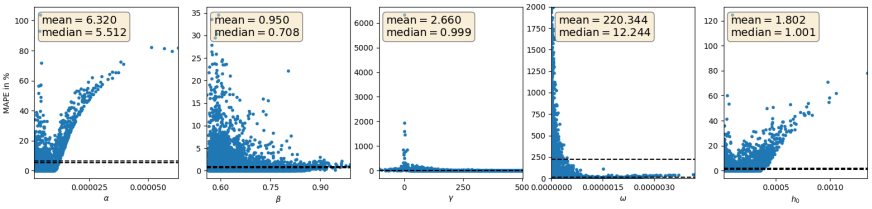


FIGURE C.14: Calibration relative error per parameter in the log-normal distributed testing set of pre-trained CNN. Only parameter combination that fulfill the stationarity constraint

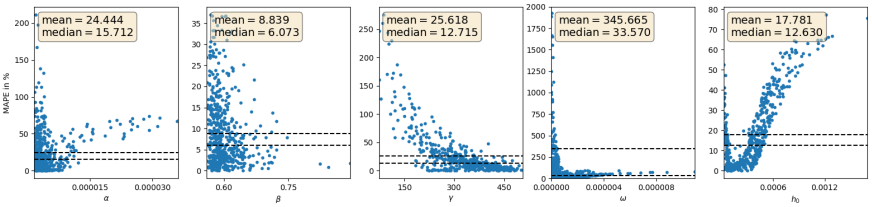


FIGURE C.15: Calibration relative error per parameter in the log-normal distributed testing set of pre-trained CNN. Only parameter combination that do not fulfill the stationarity constraint

Bibliography

Aggarwal, C.C. (2018). *Neural Networks and Deep Learning: A textbook*. Springer.

Ahmend, N.K., A.F. Atiya, N. El Gayar, and H. El-Shishiny (2010). "An Empirical Comparison of Machine Learning Models for Time Series Forecasting". In: *Econometric Reviews* 29.5-6, pp. 594–621. DOI: 10.1080/07474938.2010.481556.

Alaton, P., B. Djehiche, and D. Stillberger (2002). "On modelling and pricing weather derivatives". In: *Applied mathematical finance* 9.1, pp. 1–20. DOI: 10.1080/13504860210132897.

Anderson, G. (1996). "Nonparametric Tests of Stochastic Dominance in Income Distributions". In: *Econometrica* 64.5, pp. 1183–93. JSTOR: 2171961.

Association, European Financial Management (2018). *Machine learning in European financial institutions*. URL: efma.com/study/detail/28154 (visited on 04/20/2020).

Badescu, A., H. Brautmeier, L. Grigoryeva, and J.P. Orthega (2020). "How robust is GARCH option pricing calibration?" Working Paper.

Beran, J. (2018). *Mathematical foundations of time series analysis: a concise introduction*. Springer.

Black, F. and M. Scholes (1973). "The Pricing of Options and Corporate Liabilities." In: *Journal of Political Economy* 81.3, pp. 637–654. JSTOR: 1831029.

Brautmeier, H. (2020). "Stationarity of discrete-time Volatility Models". Bachelor Thesis.

Cao, J., Z. Li, and J. Li (2019). "Financial time series forecasting model based on CEEMDAN and LSTM". In: *Physica A: Statistical Mechanics and its Applications* 519, pp. 127–139. DOI: 10.1016/j.physa.2018.11.061.

Chen, T. and H. Chen (1993). "Approximations of continuous functionals by neural networks with application to dynamic systems". In: *IEEE Transactions on Neural Networks* 4.6, pp. 910–918.

Chorro, C., D. Guégan, and F. Ielpo (2015). *A Time Series Approach to Option Pricing: Models, Methods and Empirical Performances*. 2015th edn, Springer, Berlin, Heidelberg.

Deng, S. (2000). "Pricing Electricity Derivatives under Alternative Stochastic Spot Price Models". In: *2014 47th Hawaii International Conference on System*

Sciences. Vol. 5. IEEE Computer Society, p. 4025. DOI: 10.1109/HICSS.2000.926755.

Engle, R.F. (1982). "Autoregressive conditional heteroscedasticity with estimates of the variance of United Kingdom inflation." In: *Econometrica: Journal of the Econometric Society* 50.4, pp. 987–1007. JSTOR: 1912773.

Fan, Jianqing, Cong Ma, and Yiqiao Zhong (2019). *A Selective Overview of Deep Learning*. arXiv: 1904.05526.

FIA (2019). *Global Futures and Options Trading Reaches Record Level in 2019*. URL: fia.org/node/5542 (visited on 01/27/2020).

Föllmer, H. and A. Schied (2016). *Stochastic Finance: An Introduction in Discrete Time*. Walter de Gryuter, Berlin, Boston.

Frey, C. B. and M.A. Osborne (2017). "The future of employment: How susceptible are jobs to computerisation?" In: *Technological forecasting and social change* 114, pp. 254–280. DOI: 10.1016/j.techfore.2016.08.019.

Geman, S., E. Bienenstock, and R. Doursat (1992). "Neural Networks and the Bias/Variance Dilemma". In: *Neural Computation* 4.1, pp. 1–58. DOI: 10.1162/neco.1992.4.1.1.

Grohs, P., D. Perekrestenko, D. Elbrächter, and H. Bölcskei (2019). *Deep Neural Network Approximation Theory*. arXiv: 1901.02220.

Hansen, P.R. and A. Lunde (2005). "A forecast comparison of volatility models: does anything beat a GARCH (1, 1)?" In: *Journal of applied econometrics* 20.7, pp. 873–889. DOI: 10.1002/jae.800.

Hernandez, A. (2016). *Model Calibration with Neural Networks*. DOI: 10.2139/ssrn.2812140.

Heston, S.L. and S. Nandi (2000). "A closed-form GARCH option valuation model." In: *The review of financial studies* 13.3, pp. 585–625. JSTOR: 2645997.

Hornik, K., M. Stinchcombe, and H. White (1989). "Multilayer feedforward networks are universal approximators". In: *Neural Networks* 2.5, pp. 359–366. DOI: 10.1016/0893-6080(89)90020-8.

— (1990). "Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks". In: *Neural Networks* 3.5, pp. 551–560. DOI: 10.1016/0893-6080(90)90005-6.

Horvath, B., A. Muguruza, and A. Tomas (2019). *Deep Learning Volatility*. DOI: 10.2139/ssrn.3322085.

Hubalek, F., J. Kallsen, and L. Krawczyk (2006). "Variance-optimal hedging for processes with stationary independent increments". In: *Annals of Applied Probability* 16.2, pp. 853–885. DOI: 10.1214/105051606000000178.

Hull, J.C. (2001). *Fundamentals of Futures and Options Markets*. Prentice Hall, 2011.

IMF, International Monetary Fund (2019). *World Economic Outlook Database: October 2019*. URL: imf.org/en/publications/weo (visited on 05/24/2020).

International Settlements, Bank for (2019). *Statistical release: OTC derivatives statistics at end-June 2019*. URL: bis.org/publ/otc_hy1911.pdf (visited on 01/27/2020).

Jäckel, P. (2015). "Let's Be Rational". In: *Wilmott* 2015.75, pp. 40–53. DOI: 10.1002/wilm.10395.

Kim, H.Y. and C.H. Won (2018). "Forecasting the volatility of stock price index: A hybrid model integrating LSTM with multiple GARCH-type models". In: *Expert Systems with Applications* 103, pp. 25–37.

Kirilenko, A., A.S. Kyle, M. Samadi, and T. Tuzun (2017). "The flash crash: High-frequency trading in an electronic market". In: *The Journal of Finance* 72.3, pp. 967–998. DOI: 10.1111/jofi.12498.

Kreps, D.M. and K.F. Wallis (1997). *Advances in economics and econometrics: theory and applications: seventh world congress*. Vol. 3. Cambridge University Press.

Kupper, M. (2018). "Mathematical Finance". Lecture Notes at the University of Konstanz.

Liu, S., C. W. Oosterlee, and S.M. Bohte (2019). *Pricing options and computing implied volatilities using neural networks*. arXiv: 1901.08943.

Merton, R.C. (1973). "Theory of Rational Option Pricing." In: *The Bell Journal of Economics and Management Science* 4.1, pp. 141–183. JSTOR: 3003143.

Poggio, T.A. et al. (2016). *Why and When Can Deep - but Not Shallow - Networks Avoid the Curse of Dimensionality: a Review*. arXiv: 1611.00740.

Roch, A. (2017). "Asymptotic asset pricing and bubbles". In: *Mathematics and Financial Economics* 2018.2. DOI: 10.1007/s11579-017-0204-1.

Runggaldier, W.J. (2003). "Jump-diffusion models". In: *Handbook of heavy tailed distributions in finance*. Elsevier, pp. 169–209.

Schindler, A., T. Lidy, and A. Rauber (Nov. 2016). "Comparing Shallow versus Deep Neural Network Architectures for Automatic Music Genre Classification". In: *Conference: 9th Forum Media Technology (FMT2016)*.

Trolle, A. B. and E. S. Schwartz (2009). "Unspanned Stochastic Volatility and the Pricing of Commodity Derivatives". In: *Review of Financial Studies* 22.11, pp. 4423–4461. DOI: 10.1093/rfs/hhp036.

Vapnik, V. (2013). *The nature of statistical learning theory*. Springer science & business media.

Zhou, D. (2018). *Universality of Deep Convolutional Neural Networks*. arXiv: 1805.10769.