

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 fulfillment of the requirements  
for the degree of Master of Science in Mathematical Finance*

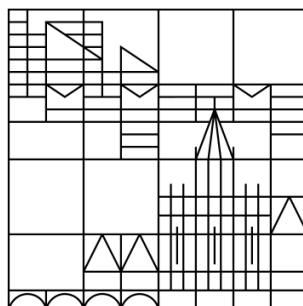
*in the*

Graduate School of Decision Sciences  
Department of Mathematics and Statistics

*at the*

University of Konstanz

May 27, 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: May 27, 2020

---



*“Building advanced AI is like launching a rocket. The first challenge is to maximize 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 CNN-based calibration and pricing method for discrete-time volatility models. We show that this approach can price and calibrate the HNG(1,1)-model in real time with extremely high accuracy. This opens a lot of possible applications for today's risk management as it allows to get rid of computation-intensive simulation as most modern market models strongly depend on Monte-Carlo simulations. The approach is loosely based the work of Horvath, Muguruza, and Tomas (2019). In contrast to their work, our model is based on discrete-time volatility models, able to incorporate fully non-linear parameter constraints and calibrates in real time with no further optimizations needed.





## *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

<b>Declaration of Authorship</b>	<b>iii</b>
<b>Abstract</b>	<b>vii</b>
<b>Acknowledgements</b>	<b>ix</b>
<b>List of Figures</b>	<b>xiii</b>
<b>List of Tables</b>	<b>xvii</b>
<b>List of Abbreviations</b>	<b>xix</b>
<b>List of Symbols</b>	<b>xxi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Context of Research . . . . .	1
1.2 Research Question and Motivation . . . . .	2
1.3 Summary of Results . . . . .	4
1.4 Structure . . . . .	4
<b>2 Options, Pricing and Finance Theory</b>	<b>7</b>
2.1 Financial Markets in discrete Time . . . . .	7
2.2 Contingent Claims, Options and Completeness . . . . .	10
2.3 Black Scholes & Implied Volatility . . . . .	12
2.3.1 Model Description and Pricing . . . . .	12
2.3.2 Implied Volatility . . . . .	14
<b>3 Heston Nandi GARCH Model</b>	<b>17</b>
3.1 Model Construction . . . . .	17
3.2 Empirical Performance . . . . .	22
3.2.1 Log-Likelihood Estimation . . . . .	22
3.2.2 Calibration . . . . .	24
3.2.3 Estimation vs. Calibration . . . . .	25

<b>4</b>	<b>Methodology</b>	<b>27</b>
4.1	Formalisation of the Deep Learning Approach . . . . .	28
4.2	Learning the Pricer . . . . .	30
4.2.1	Network Architecture and Training . . . . .	31
4.3	Learning the Calibration Mapping . . . . .	34
4.3.1	Network Architecture and Training . . . . .	34
4.4	Benchmark Model . . . . .	36
4.4.1	Feed-Forward Neural Network Approach . . . . .	36
4.4.2	Classic Calibration . . . . .	39
4.4.3	Monte Carlo simulation. . . . .	39
4.5	Generating a Trainingset . . . . .	40
4.5.1	Calibration . . . . .	40
4.5.2	How to draw a "good" random sample . . . . .	43
4.5.3	Further Specification of the Trainingset . . . . .	48
<b>5</b>	<b>Summary and Results</b>	<b>51</b>
5.1	Numerical Experiments and Results . . . . .	51
5.1.1	Pricing Volatility . . . . .	52
5.1.2	Calibration . . . . .	55
5.1.3	Autoencoder . . . . .	58
5.1.4	Different Datasets . . . . .	59
5.2	Conclusions and Outlook . . . . .	63
<b>A</b>	<b>Properties of HNG(1,1)</b>	<b>65</b>
<b>B</b>	<b>MATLAB and Python 3.7 Code</b>	<b>69</b>
<b>C</b>	<b>Additional Results</b>	<b>73</b>
C.1	Additional Results for Section 4.3 . . . . .	73
C.2	Additional Result of Section 5.1 . . . . .	74
	<b>Bibliography</b>	<b>79</b>

# List of Figures

2.1	Exemplary Illustration of a Volatility Smile . . . . .	15
4.1	<b>PRELIMINARY GRAPHIC</b> Volatility Pricing CNN . . . . .	33
4.2	<b>PRELIMINARY GRAPHIC</b> Calibration CNN . . . . .	35
4.3	FFNN Visualisation from Horvath, Muguruza, and Tomas (2019), page 20. Adapted . . . . .	37
4.4	Histogram of empirically calibrated parameters . . . . .	41
4.5	Histogram of filtered empirically calibrated parameters . . . . .	42
4.6	Histogram: Empirical Parameter Distribution based on S&P500 calibrated Data. Underlying Distribution: Normal Distribution. . . . .	44
4.7	Histogram: Empirical Parameter Distribution based on S&P500 calibrated Data. Extreme Scenarios are filtered out. Underlying Distribution: Normal Distribution. . . . .	44
4.8	Histogram: Empirical Parameter Distribution based on S&P500 calibrated Data. Underlying Distribution: Uniform Distribution. . . . .	45
4.9	Histogram: Empirical Parameter Distribution based on S&P500 calibrated Data. Extreme Scenarios are filtered out. Underlying Distribution: Uniform Distribution. . . . .	45
4.10	Histogram: Empirical Parameter Distribution based on S&P500 calibrated Data. Underlying Distribution: Mean-Symmetric Uniform Distribution. . . . .	46
4.11	Histogram: Empirical Parameter Distribution based on S&P500 calibrated Data. Extreme Scenarios are filtered out. Underlying Distribution: Mean-Symmetric Uniform Distribution. . . . .	46
4.12	Histogram: Empirical Parameter Distribution based on S&P500 calibrated Data. Underlying Distribution: Log-Normal Distri- bution. . . . .	47
4.13	Histogram: Empirical Parameter Distribution based on S&P500 calibrated Data. Extreme Scenarios are filtered out. Underlying Distribution: Log-Normal Distribution. . . . .	47





4.14	Histogram: Empirical Parameter Distribution based on S&P500 calibrated Data. Underlying Distribution: Preliminary MinMax-scaling . . . . .	48
4.15	Histogram: Empirical Parameter Distribution based on S&P500 calibrated Data. Extreme Scenarios are filtered out. Underlying Distribution: Preliminary MinMax-scaling . . . . .	49
5.1	Test Sample Performance of the Volatility Pricing Network $F$ . The errors are always measured with respect to the closed form solution. . . . .	52
5.2	Test Sample Performance of the Feed-Forward Volatility Pricing Network. The errors are always measured with respect to the closed form solution. . . . .	53
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. . . . .	53
5.4	Test Sample Performance of the Volatility Pricing Network $F$ . Scatter-Plot: True Volatility vs. Average Error . . . . .	54
5.5	Test Sample Performance of the Feed-Forward Volatility Pricing Network. Scatter-Plot: True Volatility vs. Average Error . . . . .	54
5.6	Test Sample Performance of the Volatility Pricing with Monte Carlo (100.000 paths). Scatter-Plot: True Volatility vs. Average Error . . . . .	54
5.7	Test Sample Performance of the Volatility Pricing. Empirical CDF of the relative errors for each approach. . . . .	55
5.8	Calibration relative error per parameter in the test set of the CNN	55
5.9	Relative error per parameter in the test set after Levenberg-Marquardt-FFNN-Calibration. . . . .	56
5.10	Relative error per parameter in the test set after Interior-Point Calibration (first 1000 scenarios) . . . . .	56
5.11	Calibration relative error per parameter in the test set of the CNN. Only parameter combination that fulfill the stationarity constraint . . . . .	56
5.12	Calibration relative error per parameter in the test set of the CNN. Only parameter combination that do not fulfill the stationarity constraint . . . . .	56
5.13	Test Sample Performance of the Autoencoder Network $F \circ G$ . The errors are always measured with respect to the closed form solution. . . . .	59

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 fulfill the stationarity constraint . . . . .	59
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 fulfill the stationarity constraint . . . . .	60
5.16	Performance of the Volatility Pricing Network $F$ based on the uniform symmetric distributed testing set. The errors are al- ways measured with respect to the closed form solution. . . .	61
5.17	Calibration relative errors per parameter in the uniform sym- metric testing set of trained CNN. . . . .	61
5.18	Performance of the Volatility Pricing Network $F$ based on the log-normal distributed testing set. The errors are always mea- sured with respect to the closed form solution. . . . .	61
5.19	Calibration relative errors per parameter in the log-normal testing set of trained CNN. . . . .	62
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. . . . .	62
5.21	Calibration relative errors per parameter in the uniform sym- metric testing set of pre-trained CNN. . . . .	62
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. . .	63
5.23	Calibration relative errors per parameter in the log-normal testing set of pre-trained CNN. . . . .	63
C.1	Test Sample Performance of the Volatility Pricing Network $F$ . Empirical Density of the relative error. . . . .	74
C.2	Test Sample Performance of the Feed Forward Volatility Pricing Network. Empirical Density of the relative error. . . . .	74
C.3	Test Sample Performance of the Volatility Pricing with Monte Carlo (100.000 paths). Empirical Density of the relative error. .	74
C.4	Test Sample Performance of the Autoencoder Network $G \circ F$ . Empirical Density of the relative error. . . . .	75

C.5	Test Sample Performance of the Autoencoder Network $G \circ F$ . Relative error per parameter in the testing set. . . . .	75
C.6	Test Sample Performance of the Autoencoder Network $G \circ F$ . Relative error per parameter in the testing set. Only parameter combinations which fulfill the stationarity constraint . . . . .	75
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 . . . . .	75
C.8	Calibration relative error per parameter in the uniform sym- metric distributed testing set of trained CNN. Only parameter combination that fulfill the stationarity constraint . . . . .	76
C.9	Calibration relative error per parameter in the uniform sym- metric testing set of trained CNN. Only parameter combination that do not fulfill the stationarity constraint . . . . .	76
C.10	Calibration relative error per parameter in the log-normal dis- tributed testing set of trained CNN. Only parameter combina- tion that fulfill the stationarity constraint . . . . .	76
C.11	Calibration relative error per parameter in the log-normal test- ing set of trained CNN. Only parameter combination that do not fulfill the stationarity constraint . . . . .	76
C.12	Calibration relative error per parameter in the uniform symmet- ric distributed testing set of pre-trained CNN. Only parameter combination that fulfill the stationarity constraint . . . . .	77
C.13	Calibration relative error per parameter in the uniform symmet- ric distributed testingset of pre-trained CNN. Only parameter combination that do not fulfill the stationarity constraint . . . . .	77
C.14	Calibration relative error per parameter in the log-normal dis- tributed testing set of pre-trained CNN. Only parameter com- bination that fulfill the stationarity constraint . . . . .	77
C.15	Calibration relative error per parameter in the log-normal dis- tributedf testing set of pre-trained CNN. Only parameter com- bination that do not fulfill the stationarity constraint . . . . .	77



# List of Tables

3.1	MLE Estimates (Wednesdays, 10 year window) . . . . .	23
3.2	Basic Features of the option pricing (calls) dataset (Wednesdays,2010-2018) . . . . .	24
3.3	Calibrated Parameters on Wednesdays with respect to Call-price MSE . . . . .	25
3.4	Performance of MLE-Parameters on Wednesdays . . . . .	26
3.5	1-week forecasting performance of MLE-estimated parameters and $h_0^Q = h_t^P$ . . . . .	26
3.6	1-week forecasting performance of calibrated parameters . . . . .	26
4.1	Structure of Pricing CNN: The network consists of 7 convolutional layers and a total of 26,473 trainable weights. . . . .	32
4.2	Calibration CNN: The CNN consists of 8 convolution layers, 1 max pooling layer and 123,781 trainable parameters . . . . .	35
4.3	Descriptive statistics of the underlying data . . . . .	41
4.4	Correlation matrix of underlying data . . . . .	42
4.5	Correlation matrix of cleaned underlying data . . . . .	42
5.1	Descriptive Statistics of the Volatility Pricing Approaches with respect to MAPE in % . . . . .	52
5.2	Average Computation Time for pricing one full surface . . . . .	55
5.3	Spearman-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 . . . . .	57
5.4	Average computation time for one surface calibration . . . . .	57
5.5	Descriptive Statistics of the Autoencoder Network $F \circ G$ with respect to MAPE in % . . . . .	58
C.1	Calibrated Parameters on Wednesdays with respect to Options-Likelihood . . . . .	73



# List of Abbreviations

## Model

---

<b>GARCH</b>	<b>Generalised AutoRegressive Conditional Heteroskedasticity</b>
<b>HNG</b>	<b>Heston Nandi Garch</b>
<b>BS</b>	<b>Black, Scholes &amp; Merton</b>

## Neural Networks

---

<b>CNN</b>	<b>Convolution Neural Network</b>
<b>FFNN</b>	<b>Feed-Forward Neural Network</b>
<b>LSTM</b>	<b>Long Short-Term Memory neural network</b>
<b>GRU</b>	<b>Gated Recurrent Unit</b>

## Error Measures

---

<b>MAPE</b>	<b>Mean Absolute Percentage Error</b>
<b>MSE</b>	<b>Mean Squared Error</b>
<b>RMSE</b>	<b>RootMean Squared Error</b>
<b>RMSRE</b>	<b>RootMean Squared Relative Error</b>
<b>CMSE</b>	<b>Constrained Mean Squared Error</b>

## Miscellaneous

---

<b>SDE</b>	<b>Stochastic Differential Equation</b>
------------	---



# List of Symbols

## Sets and Spaces

---

$\mathbb{N}, \mathbb{Z}, \mathbb{R}$	set of natural numbers, whole numbers and real numbers
$\mathbb{N}_{\leq k}$	$\{1, \dots, k\}$ for $k \in \mathbb{N}$
$\mathbb{R}^+$	set of positive reals
$\Omega$	usually a set of events
$\Sigma, \mathcal{F}$	$\sigma$ -algebra on a set $\Omega$
$\sigma(M)$	$\sigma$ -algebra generated by a set $M$
$\mathcal{L}^p(\Omega)$	stochastic $\mathcal{L}^p$ -space on a probability space $(\Omega, \Sigma, \mathbb{P})$
$\theta, \bar{\theta}$	parameter or statistic with corresponding estimator

## Stochastic Processes

---

i.i.d	corresponds to a family of identically distributed, pairwise independent random variables.
$\mathcal{N}(\mu, \Sigma)$	multivariate normal distribution, with expected value $\mu$ and covariance matrix $\Sigma$
$\mathbb{P}$ -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 $X$
$\mathbb{E}(X   \mathcal{F})$	conditional expectation of $X$ with respect to $\mathcal{F}$
$\mathbb{V}(X   \mathcal{F})$	conditional variance of $X$ with respect to $\mathcal{F}$
$\ \cdot\ _2$	euclidean norm on $\mathbb{R}^d$
$\ \cdot\ _\infty$	maximum norm
$\mathbb{1}_C(\cdot)$	indicator function. 1 if proposition $C$ is true, 0 else.
$\ln(\cdot)$	natural logarithm
$\text{id}_X(\cdot)$	identity function of a space $X$
$\text{Re}(\cdot)$	real part of a complex vector
$e$	vector or matrix of constant 1 in each entry
$(\cdot)^+$	positive part of a function



# Chapter 1

## Introduction


### 1.1 Context of Research

Over the past two decades society and economy were disrupted by digitalisation. It fundamentally transformed major business sectors as the automobile, communication or commerce. Nowadays, 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. Especially the increase of computational power and the linked rise of big data and artificial intelligence algorithm are the main driver of those developments.

Hence it's no surprise that this development doesn't not stop at the financial markets. The availability of intra-day 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. According to the European Financial Management Association, roughly 35% of European financial organizations have deployed at least one machine learning solution at the end of 2018. Further, they state that for those organizations that have yet to deploy a solution, 23% believed they would have an AI solution in place within in a year (EFMA, 2018). Besides huge improvement in automating customer Experience, e.g. detecting customer behavior patterns or automatised chatbot programs, huge improvements can be developed with respect to modern risk management.


On the other hand, the subprime crisis of 2007 and the following global financial crisis outlined the weaknesses of the used rating and pricing models.

The appearance of financial markets shaped the modern economy and contributed to the 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** end of last century, their importance increased significantly. In 2019, the total volume of futures and options contracts traded on exchanges worldwide reached 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 (BIS, 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. In light of such numbers, practitioners as well as researchers can not afford to not thoroughly understand modelling and managing them.

Options are used in the economy for various reasons. Besides the obvious purely monetary driven reasons as profits through speculation, they are widely used to increase a portfolio performance and, more generally, to transfer and manage risk. Consequently, pricing and analysing them must be dealt with great care and with sufficiently realistic models. Hence, researches have spend years investigating pricing and hedging of such financial products. Starting with the work of Black and Scholes (1973) and Merton (1973), tons of market models were introduced and analysed, each with it's own upsides and flaws. However, the over-reliance on this same formula and more generally the deviance in pricing of complex derivatives products have been highly criticized during the recent financial crisis. Nowadays, regulatory (e.g. **Basel IV, MiFID, IFRS9**) and environmental pressures as institutional investors are pushing banks to upgrade their risk management.

## 1.2 Research Question and Motivation

 Today's stochastic pricing models are able to capture most empirically observed properties as leptokurtosis and skewness of returns or even advanced characteristics as market shocks (e.g jump-diffusion processes (Runggaldier, 2003)). But, such sufficiency comes at a significant expense in computational load and memory usage. Lately, many practitioners and researchers focus on non-parametric approaches to forecast financial time-series as LSTM or GRU NN-structures (cf. Kim and Won, 2018 or Cao, Z. Li, and J. Li, 2019). One major advantage of such 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. Nonetheless statistical



inference and analysis of the given estimators and forecasts is difficult (cf. Geman, Bienenstock, and Doursat, 1992). Also recent work implies that those approaches do not outperform standard-theory in out of sample comparison, but it indicates that deep learning techniques can be used to significantly decrease computation time (cf. Liu, Oosterlee, and Bohte, 2019).

This being said, this thesis focuses on the latter. With increasing complexity of models, it's not possible to ensure the existence of closed form pricing formulas. Therefore calibrating models and estimating optimal parameters lead to the necessity of very computation-intensive simulations, which themselves add approximation errors to each calculation. Additionally, the computation time of Monte Carlo based approaches scales at least linear with number of path's while accuracy often scales sublinear. Hence, such methods become unattractive with higher needs for accuracy. This explains the attractiveness of Black's, Scholes' and Merton's 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 calibrating continuous volatility models (namely Rough Bergomi Model) using feed-forward neural networks. 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 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 a sufficient approach on training a neural nets for the calibration task. Also, the usage of continuous-time stochastic volatility models often results in problems when one is asked not only to calibrate a model to observable market data, but to forecast given data.

This thesis is loosely based on the approach of Horvath, Muguruza, and Tomas (2019), but will focus on discrete-time models, namely the model introduced by Heston and Nandi (2000). Moving from continuous time to a discrete-time model allows to incorporate real data with no additional effort and furthermore estimate parameters under real conditions and not only in risk-neutral markets. Another advantage of using Heston and Nandi's model is the existence of a closed form solution. Hence, we do not deal with additional approximation errors resulting from Monte-Carlo simulations. Nonetheless, this approach is easily generalizable to any other discrete-time model, which will be shortly discussed in chapter 4. With increased model complexity and additional parameter constraints, FFNNs are not able to fully

capture the properties of the model, but using CNN resolves this issue. However, CNNs are not widely deployed in a financial setting yet and this thesis shows a first approach on incorporating such network structures on pricing models. It is crucial to point out that this thesis neither aims to find a well performing volatility model nor to compare existing models. The interested reader is referred to relevant publications as Hansen and Lunde (2005) or Ahmend et al. (2010). This thesis focuses on the ability of CNN's to approximate discrete-time volatility models and, therefore, their suitability as efficient approximation tool. It is obvious, that our approach can only fit data as good as the underlying model and questions with respect to model choice will not be tackled in the following.

### 1.3 Summary of Results

We are able to show that our proposed approach is able to significantly outperforms all given benchmarks. This holds true for the volatility pricing as well as the calibration task. Hereby, the performance is measured in terms of accuracy and speed. Our approach reaches an average 0.6298% deviation to the closed form solution in the implied volatility pricing task. For calibration we decreased the computation in comparison to the closest benchmark by 99.97% while increasing the accuracy by atleast 50% for each parameter. Combining pricing and calibration task lead to promising results in terms of the robustness of our approach. Furthermore we analysed the performance of the proposed neural networks when used on different datasets, indicating that our propose is indeed able to generalize and is not bounded to the introduced HNG(1,1) mode.

### 1.4 Structure

Having introduced the context of our work, the research questions and our main results, the rest of this master's thesis is structured along the following chapters.

**Chapter 2** We begin by reviewing the concepts that are necessary to grasp the remaining theory of the thesis. We introduce the mathematical foundation of market models and focus on risk-neutral valuation, the important concepts of no-arbitrage and market completeness. We cover elements on options pricing, the Black-Scholes model and implied volatility.

**Chapter 3** This chapter covers the Heston-Nandi-Garch model, which will be used as model in the majority of this thesis. Based on the model assumptions, we derive the closed form solution and cover the most important properties of the model. Finally, we review the model limitations and its empirical performance. For the latter historic estimation as well as option price based calibration on S&P500 data is compared.

**Chapter 4** Here, we detail the machine learning methodology that we followed to answer our research question. We describe the neural net structure and the error measures used to assess the performance of the models. Furthermore we describe the used benchmark model as well as an auto-encoder approach which can be used for real-world application. Finally, we describe the underlying data used and how to generate a fitting trainingset.

**Chapter 5** This chapter concludes the thesis. We detail the results of the conducted numerical experiments and discuss the outcomes. We point out implications and underline the limitations of our work to finish with some 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 script to the lecture Mathematical Finance of Kupper (2018) at the University of Konstanz.

We want to provide a mathematical structure for financial market models. Since we are dealing with discrete-time models later on, this section focuses on discrete-time models. Especially when dealing with high frequency trading and the corresponding analysis of connected event e.g. Flash Crashes (cf. Kirilenko et al., 2017), such models are mandatory. We want to point out that ultimately every real financial market is of discrete nature independent of its size and liquidity, once the observation frequency is increased on a tick basis.



## 2.1 Financial Markets in discrete Time

**Definition 2.1.1** (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  $t \in \{0, \dots, T\}$ . We model that by 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  $\bar{S} = (S^0, S) = (S_t^0, S_t^1, \dots, S_t^d)_{t \in \{0, \dots, T\}}$ , 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  $\mathcal{F}_0 = \{\emptyset, \mathcal{F}\}$  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 a

money stored in a transaction account and therefore we will assume  $S_t^0 > 0$  for all  $t \in \{0, \dots, T\}$ . Later on, more assumption on the distribution of  $S^0$  will be made (cf. BS model in 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 2.1.2** (Trading Strategy). A trading strategy  $\xi$  is an  $\mathbb{R}^{d+1}$ -valued process on  $(\Omega, \mathcal{F}, (\mathcal{F}_t)_{t \in \{0, \dots, T\}}, \mathbb{P})$ , where  $\xi_{t+1}$  is measurable with respect to  $\mathcal{F}_t$  for all  $t \in \{0, \dots, T-1\}$ .  $\xi_t^i$  can be interpreted as the quantity of shares of the  $i$ -th asset at time  $t$ . Therefore, a trading strategy can be interpreted as portfolio of asset in a given market over the lifetime of this market.

For later purposes it's useful to assume that no external capital is added to portfolio while trading.

**Definition 2.1.3** (self-financing). A trading strategy  $\xi$  is called self-financing if  $\xi_t S_t = \xi_{t+1} S_t$  holds for every  $t \in \{0, \dots, T-1\}$

Especially when replicating contingent claims with a trading strategy, it is necessary to assume self-financing. Elsewise, it would be 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  $\xi$  can be changed to a self-financing strategy by adjusting the number of assets hold in the Numeraire.

**Definition 2.1.4** (Discounted Price). Since all values are usually expressed in terms of the Numéraire  $S^0$ , it is easier to deal with the discounted prices price process

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

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 computation power.

Finally, all tools are given to define the value of a portfolio.

**Definition 2.1.5.** (Value Process) The value  $V$  of a given trading strategy  $\xi$  is given by

$$V_0 = \xi_0 \bar{X}_0, W_t = \xi_t \bar{X}_t,$$

where  $V_0$  represents the initial endowment of an investor. The accumulated trading gains  $G$  of such strategy  $\xi$  are given by

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

**Definition 2.1.6** (Arbitrage). To ensure a market model in which it is not possible to generate higher gains than the Numairère at anytime without taking additional risk, we need additional assumptions on the set of trading strategies:

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. It is interesting to notice that it doesn't matter whether the value at the time  $T$  is a.s. positive or at any other point in time  $t^*$ . As one can transport the gains from time  $t^*$  always  $T$  by closing the portfolio and investing everything into  $S^0$ . Furthermore it is possible to show that if a arbitrage opportunity is found it is always possible to find a an arbitrage opportunity which is bounded and with  $V_0$ . The proof can be found in Kupper (2018). This claim is quiet remarkably 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 only unbounded arbitrage opportunities would undermine the economic sense of such a definition.

As the assumption of lack of arbitrage builds the basis for every further pricing, it is of utmost importance to find characterisation of this property. Especially connecting the structure of  $\mathbb{P}$  with it allows for a deeper stochastic analysis of such markets:

**Theorem 2.1.1** (First Fundamental Theorem of Asset Pricing). The market model  $S$  is arbitrage-free if and only if the set  $\mathcal{P}$  of equivalent martingale measure of  $\mathbb{P}$  is nonempty. In this case there is even a  $\mathbb{Q} \in \mathcal{P}$  which has a bounded Radon–Nikodym density  $\frac{d\mathbb{Q}}{d\mathbb{P}}$ .

A proof of the above claim can be found in Kupper (2018) Theorem 1.20.

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).

## 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 lack of arbitrage.

**Definition 2.2.1** (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  $S_0, \dots, S_d$  if  $C$  is measurable with respect to  $\sigma(S_0, \dots, S_T)$ .

Remark: Obviously the relation between a derivative and a specific underlying can be trivial and does not have to depend on every single asset.

European-style derivatives usually depends 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 an 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 an European call 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 an European binary call option gains a fixed amount if the asset  $i$  at maturity  $T$  is higher then a boundary price  $K$

**European Binary Put Option**  $C_{bp} = \mathbb{1}_{S_T < K}$ . This asset is usually cash settled. The owner of an European binary call option gains a fixed amount if the asset  $i$  at maturity  $T$  is lower then 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 be  $C^*$  any arbitrary derivative. A barrier option bounds the underlying in between or outside of two boundary values  $B_l, B_u \in \mathbb{R} \cup \{-\infty, \infty\}$ . If the underlying crosses these boundaries before maturity the derivative expires immediately. Even if the underlying option is European-style, pricing such a derivative 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 2.2.2** (Attainable). A contingent claim  $C$  is called attainable (replicable) if there exists a self-financing strategy  $\xi$  such that  $\xi_T S_T = C$   $\mathbb{P}$ -as. In this case  $\xi$  is called a replicating strategy.

Remark: For a contingent claim  $C$  we consider the corresponding discounted claim  $H := \frac{C}{S_0^0}$ . 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  $\bar{\xi} = (\xi_0, \xi)$ .

**Theorem 2.2.1.** Let  $H$  be an attainable discounted claim. Then  $\mathbb{E}_{\mathbb{Q}}[H] < \infty$  for every  $\mathbb{Q} \in \mathcal{P}$ . The value process of any replicating strategy  $\xi$  satisfies  $V_t = \mathbb{E}^*[H | \mathcal{F}_t]$   $\mathbb{P}$ -as for all  $t$  and  $\mathbb{P}^*$ . In particular, the value process  $V$  is a  $\mathbb{P}$ -martingale

Remark:

- The value process  $V$  does not depend on the replicating strategy  $\xi$ .

- We have  $\mathbb{E}_{\mathbb{Q}}(V_T | \mathcal{F}_t) = \tilde{\mathbb{E}}(V_T | \mathcal{F}_t)$   $\mathbb{P}$ -as for all  $\mathbb{Q}, \tilde{\mathbb{P}}$

**Definition 2.2.3** (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  $(X_0, \dots, X^d, X^{d+1})$  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 2.2.2.** 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.

**Definition 2.2.4** (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 the First Fundamental Theorem of Asset Pricing and the lack of arbitrage, it is important to characterise this economic property with a proper stochastic claim:

**Theorem 2.2.3** (Second Fundamental Theorem of Asset Pricing). An arbitrage-free market model is complete if and only if  $|\mathcal{P}| = 1$

*Proof.* If a market model is complete, by definition 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. But it follows from Theorem 2.2.1, that for every  $\mathbb{Q}, \mathbb{P}^* \in \mathcal{P}$  holds  $\mathbb{E}_{\mathbb{Q}}(H) = \mathbb{E}^*(H)$ . This leads to  $\mathbb{P}^*(A) = \mathbb{Q}(A)$ . Therefore the mapping  $\mathcal{P} \rightarrow [0, 1]$ , where  $\mathbb{P}^* \mapsto \mathbb{E}^*(H)$ , is constant. Hence  $\mathcal{P} = \mathbb{Q}$  and we conclude  $|\mathcal{P}| = 1$ .

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 2.2.2, it follows immediately that this contingent claim is attainable.  $\square$

## 2.3 Black Scholes & Implied Volatility

### 2.3.1 Model Description and Pricing

In this section, we want to show the first arbitrage free market model which let to a closed form solution for European call and put options, introduced by Black and Scholes (1973). A stock  $S$  and a risk free transfer account  $B$  used as Numéraire are considered and several assumptions are made to ensure lack of arbitrage.

First, short term interest rate  $r$  is known and constant through time and interests are calculated continuously. The variance  $\sigma$  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  $\sigma > 0$  and 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 hold or traded. Hence we get

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

where the process  $W$  is a Wiener Process and  $B_t = B_0 \exp(rt)$ . The dynamics of  $S$  and  $B$  follow the SDEs (2.1).

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

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

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

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

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

We show that  $(X_t)_t$  is a martingale with respect to  $\mathbb{Q}$  and  $\mathbb{E}_{\mathbb{Q}}(X_t | \mathcal{F}_s) = X_s$  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 2.2.1, that the value process of  $H$  equals

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

To finalise the pricing of  $H$  only formula (2.2) needs to be calculated:

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

$$\begin{aligned}
&= e^{-r(T-t)} \int_{-d_-}^{\infty} S_t e^{r-\frac{1}{2}\sigma^2(T-t)+\sigma\sqrt{T-t}x} \phi(x) dx - e^{-r(T-t)} K \Phi(d_-) \\
&= S_t \int_{-d_-}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(x-\sigma\sqrt{T-t})^2} dx - e^{-r(T-t)} K \Phi(d_-) \\
&= S_t \int_{-d_+}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2} dx - e^{-r(T-t)} K \Phi(d_-) = S_t \Phi(d_+) - e^{-r(T-t)} K \Phi(d_-),
\end{aligned}$$

where  $\phi/\Phi$  denotes the standard normal probability density/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. It is nice to see that the payoff function of buying a call option and selling a put option with same characteristics is affine linear and equals the pay off at maturity of buying one unit of the stock and  $\exp(-r(T-t))K$  units of the risk free bond. This property can be used to derive the price of a put option. Since we assume absence of arbitrage, it follows

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

This property is called Put-Call-Parity.

### 2.3.2 Implied Volatility

An option pricing model, such as 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 the most model depend on is an estimate of the volatility  $\sigma$  the underlying price shows till maturity. Economically it is easy to argue that the option price, which can only have an positive pay off, increases in the volatility as the underlying is more likely to show extreme outcomes. Let  $H$  be the value of an option such that  $H = f(\sigma, \cdot)$ . As  $f$  is increasing in  $\sigma$ , by the inverse function theorem, the option price  $H$  is, atleast locally, invertible with respect to the volatility. Hence, it is possible to backtrack characteristics of the underlying from option prices. This concept is called implied volatility,  $\sigma_{imp} = f^{-1}(H, \cdot)$ . In general, a pricing model does not have a closed-form solution for its inverse. Usually, root-finding methods are used to solve the problem. As prices change quickly in modern financial market, it is important to calculate 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 nets can be used as efficient tool 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 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 model implies a constant implied volatilities, this does not hold true in empirical studies. When applying the BS implied volatility on realized option prices, an quadratic relation between strike price and volatility is observed. Empirical asset returns show a strong overkurto-sis. As extreme returns are more likely to occur than expected by normally

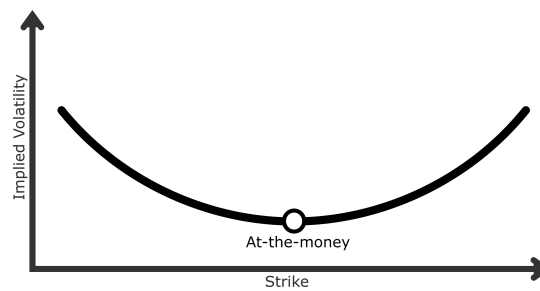


FIGURE 2.1: Exemplary Illustration of a Volatility Smile

distributed returns, deep in-of-the-money and deep-out-of-the-money options are priced with an premium. Resulting the characteristics volatility smile. 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 modern financial markets. 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 generalized autoregressive 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 Hestons continuous time market model, while tackling previously mentioned problems by modeling 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 point out that in comparison to continuous-time stochastic volatility models, which are not observable, a GARCH-like structure provides the possibility to use historic prices of the underlying for estimation and calibration.

### 3.1 Model Construction

The following assumptions are made in the HNG model. First, a discrete-time market on equidistant grid, for simplicity we assume with  $\Delta t = 1$ , is considered. Secondly, the yearly risk-free rate  $r$  is constant and discount rates are continuously compounded. Hence, we are in the setting of chapter 2, where the Numéraire  $B_t = S_t^0 := \exp(rt)$  and  $t \in \{0, \dots, T\}$ . Third, the market consists of one additional asset with price process  $(S_t)_t$  that follows

the dynamic of (3.1).

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

where  $z_t \stackrel{iid}{\sim} N(0, 1)$ ,  $\omega, \alpha_i, \beta_i, \gamma_i, \lambda \in \mathbb{R}$  and  $p, q \in \mathbb{N}$ .

Previous research (cf. Hansen and Lunde, 2005) indicate that the case of  $p = q = 1$  is sufficient. Hence, we consider the model (3.2) from now on.

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

In this model the conditional expectation of returns equals

$$\begin{aligned} \mathbb{E}(Y_t | \mathcal{F}_{t-1}) &= r + \lambda \mathbb{E}(h_t | \mathcal{F}_{t-1}) \\ &= r + \lambda h_t. \end{aligned}$$

Hence, we get for the conditional variance of returns

$$\begin{aligned} \mathbb{V}(Y_t | \mathcal{F}_{t-1}) &= \mathbb{E} \left( (Y_t - \mathbb{E}(Y_t | \mathcal{F}_{t-1}))^2 | \mathcal{F}_{t-1} \right) \\ &= \mathbb{E} \left( h_t z_t^2 | \mathcal{F}_{t-1} \right) = h_t. \end{aligned}$$

Similar to Engle's GARCH(1,1) process the volatility  $(h_t)_t$  is weakly stationary if and only if  $|\alpha\gamma^2 + \beta| < 1$ . Due to the non-linear dependency of  $h_t$  to  $h_{t-1}$  the proof of this claim does not work analogously to stationarity proofs of other GARCH-like models. Therefore we only provide the proof of sufficiency. The proof of necessity will be added in future publications.

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

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



and conclude  $\mathbb{E}(h_t) = \frac{\omega + \alpha}{1 - \beta - \alpha\gamma^2}$ . Resulting directly in a representation of the long term expectation of  $h_t$ .  $\square$

Positivity of  $h_t$  can be assured by assuming  $\alpha \geq 0$ ,  $\beta \geq 0$  and  $\omega > 0$ . Furthermore this directly implies that  $(Y_t)_t \in \mathcal{L}^2$  if  $(h_t)_t$  is weakly stationary, as

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

In addition to Engle's GARCH Heston and Nandi accounts for leverage effects via the skewness parameter  $\gamma$ . The conditional covariance between  $Y_t$  and  $h_{t+1}$  is

$$\text{Cov}(h_{t+1}, Y_t | \mathcal{F}_{t-1}) = \text{Cov}(h_{t+1}, \ln(S_t) | \mathcal{F}_{t-1}) = -2\alpha\gamma h_t.$$

A positive  $\gamma$  leads to a leverage effect. Leverage is not only important as it is observed in nearly all modern financial assets but furthermore a recent studies by Roch (2017) correlates the existence of leverage to bubbles. Hence allowing leverage effect is of high importance for modern market models. The calculation of  $\mathbb{V}(Y_t)$ , proofs of previous claims as well as further properties of the HNG(1,1) model can be found in Appendix A.

Heston and Nandi show that the corresponding risk-neutral process has an identical GARCH form as (3.1) with replacing  $\lambda$  by  $\lambda^* = -\frac{1}{2}$  and  $\gamma$  by  $\gamma^* = \lambda + \gamma + \frac{1}{2}$ . This results in the following risk-neutral model:

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

where  $z_t^* \stackrel{iid}{\sim} N(0, 1)$  but only under the risk neutral measure  $\mathbb{Q}$ . We want to remark that even if the assumption of  $z_t \stackrel{iid}{\sim} N(0, 1)$  is released, the model still works. But the value of a call option cannot be calculated via Black-Scholes formula. In this case formula (3.4) does not hold. Furthermore only pseudo Maximum Likelihood estimation is possible in section 4.5.

This change in measure has two important implications. The expected return

of the underlying equals, similar to BS, the risk-free rate  $r$  and, given a positive  $\lambda$ , it holds  $\gamma < \gamma^*$ . This implies that in risk-neutral setting a higher average volatility and a stronger degree of leverage is observed. A more detailed analysis can be found in chapter 4.2 of Chorro, Guégan, and Ielpo (2015).

Under the risk-neutral measure  $\mathbb{Q}$  the price  $C$  of an European call option with Strike  $K$  at time  $t$  and expiration date  $T$  is given by

$$\begin{aligned} \mathbb{E}_{\mathbb{Q}} [(S_T - K)^+ | \mathcal{F}_t] &= \frac{1}{2} e^{r(T-t)} S_t + \frac{1}{\pi} \int_0^{\infty} \operatorname{Re} \left( \frac{K^{-i\phi} f_t(i\phi + 1)}{i\phi} \right) d\phi \\ &\quad - K \left( \frac{1}{2} + \frac{1}{\pi} \int_0^{\infty} \operatorname{Re} \left( \frac{K^{-i\phi} f_t(i\phi)}{i\phi} \right) d\phi \right) \end{aligned} \quad (3.3)$$

and

$$C = e^{-r(T-t)} \mathbb{E}_{\mathbb{Q}} [(S_T - K)^+ | \mathcal{F}_t] \quad (3.4)$$

where  $f_t$  denotes the conditional generating function of  $Y_t$  under  $\mathbb{Q}$ . In the case of  $p = q = 1$  the generating function Heston and Nandi (2000) give a formula to calculate  $f_t$  recursively with the following expressions

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

with terminal condition  $A_T = B_T = 0$ . A proof of this expression can be found in Heston and Nandi (2000). We want to highlight that a corresponding put price can be easily derived using put-call-parities (cf. section (2.3)).

Since we will need to calculate millions of option price in chapter 4, it is important to think about efficient calculation of (3.4). The easiest way of saving computational power is by simplifying the expression and thereby reducing numerical errors when estimating the integrals. Summarizing both integrals leads to one integral with an integrand of equal complexity, resulting in approximately half the amount of operations:

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

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

We also decrease the number of division needed which leads to a higher accuracy while approximating the integral. Unluckily, there is no elegant way of simplifying the term  $f_t(i\phi + 1) - Kf_t(i\phi)$ . One property that will majorly exploited in chapter 4 is the positive homogeneity of the model:

For all  $\lambda > 0$  at time  $t$  holds for a European call option with strike  $K$ , maturity  $T$  and a underlying stock value  $S_t$

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

*Proof.* Kreps and Wallis (1997) proof that a 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 density  $q_t$  holds

$$q_t(x) = \frac{\exp(x)p_t(x)}{\mathbb{E}(\exp(S_T) | \mathcal{F}_t)},$$

where  $p_t$  denotes the real world probability density function. Hence, we conclude the call pricing function is homogeneous for the HNG(1,1) model.  $\square$

This property allows us to normalize call prices and set  $S_t = 1$  and therefore reduce the number of parameter by one as only the moneyness  $\frac{S}{K}$  and maturity of an option contract matters.

## 3.2 Empirical Performance

So far only the theoretical foundations of option pricing and the HNG model were tackled, but ultimately one is interested in applying these on real option prices. In this section we will compare the performance of the HNG(1,1) model with likelihood-estimated parameters to the calibration of parameters based on option prices. At this point several beneficial properties of Heston's and Nandi's model take effect: The GARCH-like structure allows for easy estimation of parameters based on historic returns while the exponentially affine characteristic function still leads to fast option pricing. The analysis used in this section is based on S&P500 returns as well as vanilla European Style Call Options on the S&P500. These contracts show usually a high liquidity in comparison to other option contracts. This leads to reduced trading noise, problems generated by the micro market structure or a lag of observations due to little trading activity.

### 3.2.1 Log-Likelihood Estimation

From section 3.1 we know that for the log returns under  $\mathbb{P}$  it holds  $Y_t = r + \lambda h_t + \sqrt{h_t} z_t$ , where  $z_t$  i.i.d.  $\mathcal{N}(0, 1)$ . This implies a conditional distribution of  $Y_t$  under the filtration  $\mathcal{F}_{t-1}$  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

$$\mathcal{L}(y_1, \dots, y_T | (\omega, \alpha, \beta, \gamma, \lambda, h_0)) = -\frac{T}{2} \ln(2\pi) - \frac{1}{2} \sum_{t=1}^T \left( \ln(h_t) + \frac{(y_t - r - \lambda h_t)^2}{h_t} \right) \quad (3.5)$$

The form (3.5) can be calculated recursively using identity (3.6), which is obtained directly from (3.1):

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

A reliable set of estimated parameters can be estimated by numerically solving the maximization problem (3.7) .

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

The estimation is done by calibrating problem (3.7) by applying a moving window approach for S&P500 returns on every Wednesday between 2010 and 2018. The used interest rate  $r$  is interpolated from the corresponding daily treasury yield curve rates. The observation length  $T$  is set to 2520 trading days which corresponds to roughly 10 years. By testing several globalised optimization algorithms the most robust results are given by the interior-point algorithm. All optimizations run on MATLAB R2020a. The results are summarized in Table 3.1.

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

TABLE 3.1: MLE Estimates (Wednesdays, 10 year window)

Several challenges occur in this optimization problem. Till now most researchers do not optimize  $h_0$  but take it as an external parameter, often set to the unconditional variance of  $(h_t)_t$  (The calculations to derive  $\mathbb{E}(h_t)$  under  $\mathbb{P}$  can be found in Appendix A). We do not share this perception for a variety of reasons. First, empirically the unconditional variance underestimates the observed (via back propagation estimated) initial variance. Secondly, the model is extremely sensitive on initial conditional variance  $h_0$ . By estimating  $h_0$  forecasting errors can be drastically reduced. For a detailed exposition of this claim, we refer to Badescu et al. (2020). Furthermore the magnitude

of good parameters differs extremely between  $\alpha, \omega, h_0$  and  $\beta, \gamma^*$ . While a average  $\alpha$  is of magnitude  $10^{-6}$ ,  $\gamma^*$  tends to be around 200. The most multi-dimensional optimization algorithms do not allow for different steps 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 the problem to be (artificially) highly dependent on its initial values, even though it is usually not. A good example of this problem can be found in chapter 4.2 of Chorro, Guégan, and Ielpo (2015). One solution to overcome this problem is to rescale the parameter beforehand to the interval  $[0,1]$ .

### 3.2.2 Calibration

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 then using a model with historically estimated parameters. Hence, we proceed as follows: We consider all call option contracts traded on Wednesdays between Monday, 03.01.2011, and Friday 28.12.2018, whose

- i. Maturity is in between 8 and 250 days,
- ii. **Moneyess** is in between 0.9 and 1.1, and
- iii. open interest as well as trading volume is over 100.



		Moneyess $S_0/K$						Across
		[0.900, 0.950]	(0.950, 0.975]	(0.975, 1.000]	(1.000, 1.025]	(1.025, 1.050]	(1.050, 1.100]	Moneyess
Number of Contracts	$8 \leq T \leq 30$	5847	9137	14852	5362	860	312	36370
	$30 < T \leq 80$	4077	4418	4432	2683	461	227	16298
	$80 < T \leq 180$	1674	1019	1081	816	211	146	4947
	$180 < T \leq 250$	593	338	474	415	106	67	1993
Across Maturities		12191	14912	20839	9276	1638	752	59608
Average Prices	$8 \leq T \leq 30$	1.051	3.083	10.696	32.322	73.111	119.694	12.832
	$30 < T \leq 80$	4.730	11.213	30.083	54.253	89.829	132.246	25.718
	$80 < T \leq 180$	17.971	38.017	65.160	86.598	118.064	163.207	52.287
	$180 < T \leq 250$	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	$8 \leq T \leq 30$	0.151	0.118	0.107	0.130	0.171	0.220	0.123
	$30 < T \leq 80$	0.126	0.114	0.127	0.148	0.172	0.198	0.129
	$80 < T \leq 180$	0.130	0.139	0.153	0.162	0.178	0.195	0.146
	$180 < T \leq 250$	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 3.2: Basic Features of the option pricing (calls) dataset (Wednesdays, 2010-2018)

At every Wednesday with observed call prices  $(P_i^{obs}(S_i, K_i, M_i, r_i))_{i \in \{1, \dots, N\}}$  the optimization problem (3.8) is solved.

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

The used interest rates are interpolated from daily treasury yield curve rates. Again, several different optimization algorithms were tested and a globalised interior point algorithm led to the best results. Analogously to problem (3.7), one should account for the magnitude of parameters. Besides using the MSE in problem (3.8), we also optimized over MAPE and the option-likelihood function. Table 3.3 show the results using MSE as criterion. As the other optimizations lead to similar results, those are attached to Appendix C.1.

$\theta$	2010	2011	2012	2013	2014	2015	2016	2017	2018
$\omega$	1.0488e-07	5.8246e-07	2.5115e-07	1.6648e-07	2.3430e-07	7.7768e-08	1.1626e-07	8.2065e-08	7.6453e-08
std	(4.3237e-07)	(9.9623e-07)	(5.7761e-07)	(4.5215e-07)	(4.5167e-07)	(2.6235e-07)	(2.7833e-07)	(3.2339e-07)	(3.3182e-07)
$\alpha$	8.4165e-06	4.4508e-06	2.8014e-06	2.5121e-06	2.5227e-06	2.9788e-06	2.2257e-06	1.3120e-06	1.4577e-06
std	(6.7016e-06)	(2.4687e-06)	(1.4378e-06)	(1.4269e-06)	(2.2280e-06)	(1.3795e-06)	(9.4056e-07)	(7.8262e-07)	(7.2948e-07)
$\beta$	0.6871	0.5490	0.7000	0.7605	0.6585	0.5583	0.5809	0.6908	0.6496
std	(0.1397)	(0.2245)	(0.1376)	(0.1253)	(0.1859)	(0.1226)	(0.1377)	(0.1482)	(0.1324)
$\gamma^*$	197.5895	347.0532	349.9407	311.1355	419.7989	397.9111	439.0339	454.7184	502.6705
std	(79.0995)	(210.7790)	(182.3969)	(155.5853)	(230.8533)	(128.9083)	(115.1693)	(207.7471)	(132.3138)
$h_0^Q$	1.2420e-04	1.7303e-04	7.7115e-05	4.6121e-05	4.3171e-05	0.0001	6.1981e-05	1.7690e-05	6.7046e-05
std	(7.7985e-05)	(1.3864e-04)	(3.0317e-05)	(2.5813e-05)	(3.8513e-05)	(4.8647e-05)	(4.8685e-05)	(1.1101e-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 3.3: Calibrated Parameters on Wednesdays with respect to Call-price MSE

### 3.2.3 Estimation vs. Calibration

Even though it is obvious that a model which is optimized based on the option prices leads to a better performance in terms of in-sample option price fitting than a model based on parameters which have only seen historic returns, we want to verify that claim. The parameter estimated in section 3.2.1 are transformed to their risk neutral equivalent. Afterwards those parameters are used to price the call options of the corresponding Wednesday. For the initial

variance  $h_0$ , the last update  $h_t$  as obtained from formula (3.6) is used. As we can see by comparing Table 3.4 to Table 3.3 calibrating parameters leads to an improve in every single year and for every error measure.

After all, we are more interested in the forecasting performance of the two

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

TABLE 3.4: Performance of MLE-Parameters on Wednesdays

approaches. To check for those, we use the estimated/calibrated parameters to price options one week later. No further adjustments are made to the parameters. The results are summarized in Table 3.5 and 3.6. Besides the error measured, we used for the insample-task, we checked for several wide-know Information criteria, namely Akaike and Bayes to measure the performance of each approach.

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

TABLE 3.5: 1-week forecasting performance of MLE-estimated parameters and  $h_0^Q = h_t^P$

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

TABLE 3.6: 1-week forecasting performance of calibrated parameters

Again, the calibration approach outperforms the estimation for every chosen criteria. Hence, the calibration approach will be used as basis to generate the neural net dataset in section 4.5 and as benchmark for our model.



## Chapter 4

# Methodology

The previous chapters provide the mathematical and economical foundation to finally link stochastic volatility models and deep learning methods. This chapter summarizes our approach and introduces the methods used for measuring the performance of it.

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. The neural network needs to learn the pricing function only once. Afterwards it is capable to evaluate new parameter scenarios at an incredibly high pace. On the contrary 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 desirable at all.

In their paper *Deep Learning Volatility* Horvath, Muguruza, and Tomas (2019) present a highly efficient and accurate deep learning approach to implement continuous stochastic volatility models. Their approach consists of two steps: First learning a pricing function via a neural network by using model parameters as input and option price surfaces as an output. The second step contains the optimization of the parameters that fit observed prices best.

Our approach slightly differs from theirs. Firstly, they only considers 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. Discrete-time models capture real markets and dynamics better. Hence want to know if Neural Nets approximation of discrete-time models yield comparable results to continuous models. Furthermore, a deep learning implementation of models that provide closed form solutions can be realized in more precise manner since the numerical pricing errors are negligible compared to other numerical pricing approximations

like Monte Carlo based approaches. This is done solely for analytic reason. It allows us isolate the approximation error of the neural net from the errors occurring from Monte Carlo simulations. Obviously, our approach can be generalised to discrete-time models without a closed form solution available. Furthermore we focus on using CNNs instead of feed-forward networks. This has two major impacts on our methodology in comparison to Horvath, Muguruza, and Tomas (2019) approach. The usage of CNN results in a higher amount of trainable parameters and deeper network structure. This increases training's time significantly. But more importantly, the calibration of the model cannot no longer be done efficiently via standard optimization algorithms, since backpropagation through a CNN is not as easy as for feed-forward networks. Hence, we train a separate CNN for this task. Nonetheless, these disadvantages are compensated by the increased performance we obtained in our numerical experiments (cf. section 5.1).

## 4.1 Formalisation of the Deep Learning Approach

In the following sections we stick to the formalisation which is used in Chapter 1 of Horvath, Muguruza, and Tomas (2019). Let  $\mathcal{M}^{HNG}(\theta)_{\theta \in \Theta^{HNG}}$  be a our model, we wish to approximate, with parameters

$$\theta = (\alpha, \beta, \gamma^*, \omega, h_0) \in \Theta^{HNG} \subset \mathbb{R}^5. \quad (4.1)$$

The parameter combination  $\theta \in \Theta^{HNG}$  completely describes the dynamics of the risk-neutral HNG(1,1) model  $\mathcal{M}^{HNG}(\theta)$ .

The pricing map is stated by  $P : \mathcal{M}^{HNG}(\theta, \zeta) \rightarrow \mathcal{X}$ , which is available in closed form for Heston-Nandi-GARCH. Thereby  $\zeta$  includes the additional attributes of the market as well as the characteristics needed to price 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  $d$ .  $\mathcal{P}^{MKT}(\zeta) \in \mathcal{X}$  denote the observed market prices depending on  $\zeta$ , while  $\mathcal{X}$  denotes the set of call options prices. 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  $F$ . The input of the network are parameters of a volatility model (e.g. equation (4.1)), the output is a strike-maturity determined set of implied volatilities (or option prices if necessary). Formalised, we want to

find

$$F(\Theta^{HNG}, \zeta) \approx P(\mathcal{M}^{HNG}(\Theta^{HNG}, \zeta)).$$

Our approach to this task will be elaborated in section 4.2.

Secondly, calibrating a model is of utmost importance. The goal is to find a network  $G$ , which approximates the inverse of the pricing mapping,  $P^{-1} : \mathcal{X} \rightarrow \Theta^{HNG}$ . There are two main approaches to this - find the optimal parameters  $\hat{\theta}$  which solves either

$$\hat{\theta} = \underset{\theta \in \Theta^{HNG}}{\operatorname{argmin}} \delta(P(\mathcal{M}^{HNG}(\theta, \zeta)), \mathcal{P}^{MKT}(\zeta)) \quad (4.2)$$

or

$$\hat{\theta} = \min_{x \in \mathcal{X}} \delta(P^{-1}(x, \zeta), \Theta^{HNG}) \quad (4.3)$$

where  $\delta(\cdot, \cdot)$  is a suitable metric defined in the corresponding topological space. While both approaches (4.2) and (4.3) aim to find good fitting parameters to given prices, they focus on different problems. Minimizing the distance between forecasted and observed surfaces ensures that the calibrated parameters lead to parameters which induce high quality in-sample-surfaces. On the other hand minimizing the difference between predicted and real parameters directly, leads to a more direct approximation of the inverse mapping  $P^{-1} : \mathcal{X} \rightarrow \Theta^{HNG}$ . This approach was originally proposed by Hernandez (2016). Theoretically it should lead to a neural net  $G$  which is able to capture the dependency between the parameters better. Horvath, Muguruza, and Tomas (2019) focuses on the former. Their approach consists of two steps:

- i. Learning a neural net  $F$  to approximate the pricing map.
- ii. Calibrate the deterministic learned pricing map.

$$\hat{\theta} = \underset{\theta \in \Theta^{HNG}}{\operatorname{argmin}} \delta(F(\theta, \zeta), \mathcal{P}^{MKT}(\zeta))$$

While this approach works really good for their specific model, it's 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. Therefore we follow Hernandez (2016) approach, which will be elaborated in section 4.3.

## 4.2 Learning the Pricer

When learning the pricing function  $P$  with a NN, two different approaches come to mind. On the one hand option and market parameters  $\zeta$  are included in the input. This results in a network that is able to price every type of option and is really 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 evaluation increases linear with the number of options. Furthermore the input dimension of the network increases drastically and therefore training effort increases. To tackle these problem we focus on a image-based learning approach. As a first step, a fixed grid of strikes and maturities,  $\Delta := \{K_i, T_j\}_{i=1, j=1}^{n, m}$  is defined. This grid specifies the set of call options we are interested in, namely call options with strike  $K_i, i = 1, \dots, n$ , and maturity  $T_j, j = 1, \dots, m$ . The "new" trainings task is defined as follows: The neural network  $F(\theta, \hat{v})$  learns the set of implied volatilities

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

(resp. prices  $F^*(\theta) = \{P(\mathcal{M}^{HNG}(\theta, (T_i, K_j)))\}_{i=1, j=1}^{n, m}$ ), where  $\hat{v}$  are the trained network weights and

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

correspond to the closed form solution. The optimal weights are calculated via

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

The network does not train a single option in a market, but fixes a whole market-structure and prices all option in it at once. The output surface of prices can be interpreted as grey-scale picture and our training evolves from a forecasting task to a image-recognition task. Our training problem corresponds to a decoding problem known from file compression. Compressed data need to be decompressed to get a usable data type. In our case the parameters symbolize the compressed/encrypted volatility surface.

The optimal weights  $\hat{v}$  implicitly depend on the structure of the grid  $\Delta$  and so 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 arbitrary, 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 net to such Market requirements.

### 4.2.1 Network Architecture and Training

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

- i. The input is of size  $5 \times 1 \times 1$ , where each input has the structure  $(\alpha, \beta, \gamma^*, \omega, h_0)$ .
- ii. The output dimension is  $9 \times 9 \times 1$  which corresponds to a discrete-time volatility surface as generated in section 4.5.
- iii. ZeroPadding 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)			
ZeroPadding	(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
ZeroPadding	(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
ZeroPadding	(10, 9, 32)		(10, 7, 32)	
2D-Convolutional (elu)	(9, 8, 32)	4148	(9, 6, 32)	4148
ZeroPadding	(11, 10, 32)		(11, 6, 32)	
2D-Convolutional (elu)	(5, 9, 32)	4128	(5, 5, 32)	4128
ZeroPadding	(7, 11, 32)		(7, 7, 32)	
2D-Convolutional (elu)	(3, 10, 32)	4128	(3, 6, 32)	4128
ZeroPadding	(5, 12, 32)		(5, 8, 32)	
2D-Convolutional (elu)	(2, 5, 32)	9248	(2, 3, 32)	9248
ZeroPadding	(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
ZeroPadding	(5, 7, 32)		(5, 5, 32)	
2D-Convolutional (elu)	(2, 6, 32)	4128	(2, 4, 32)	4128
2D-Convolutional (elu)	(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: Structure of Pricing CNN: The network consists of 7 convolutional layers and a total of 26,473 trainable weights.

We realise that the models usually showed high error for options with extreme specification e.g. low maturities in combination with high strike prices. This lead to main characteristic of the introduced CNN, it's 2-path structure. The neural net nearly separately trains low and high maturity assets. The activation function for the hidden layers is  $\text{elu } \sigma_{\text{elu}} = \alpha(e^x - 1)$  with  $\alpha = 1$  and for the output layer it is the linear function, i.e. no activation. Using ZeroPadding is of high importance in this neural net as it is the only way to increase the dimension of the whole network. We tested running a FFNN to increase the dimension in contrast to ZeroPadding but such approaches led to decreased performance. Furthermore the output is forced to be positive

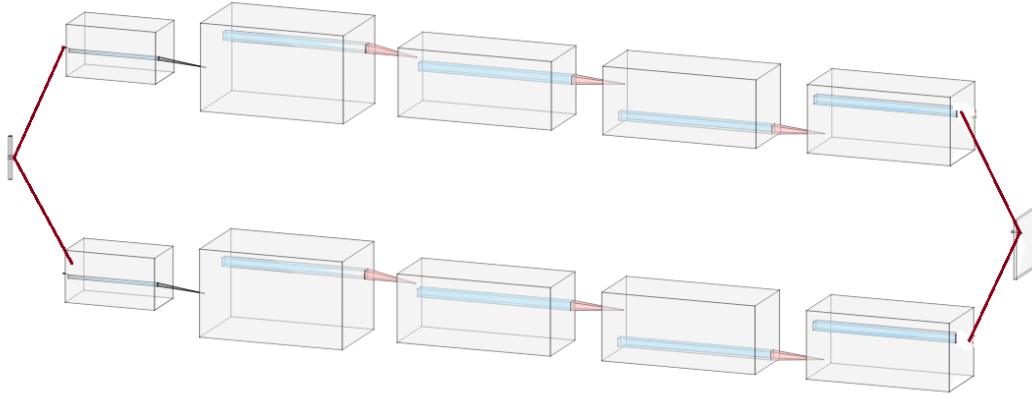


FIGURE 4.1: PRELIMINARY GRAPHIC Volatility Pricing CNN

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. Usually it is desirable to transform the data that such conditions are simplified to boundary conditions as such can be simply apply to every neural net. But, this is not possible in most advanced cases. By adding penalisation to the error term those adjustments are possible, though (cf. section 4.3). We find that adding MaxPooling or Dropout layers do not increase the overall network performance. 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  $\theta_{max}$  and  $\theta_{min}$  are the componentwise maxima, and minima respectively, of the set of parameters  $\Theta$ . Normalization of input parameters is extremely important for the network performance. One reason is the massive discrepancies between the average magnitude of parameters. A "normal"  $\alpha$  is of order between  $10^{-9}$  and  $10^{-6}$ , on the contrary a "normal"  $\gamma^*$  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 net is Root-Mean-Squared-Relative-Error (RMSRE):

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

RMSRE behaves similar to Mean-Absolute-Percentage-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

### 4.3 Learning the Calibration Mapping

Following the reasoning from 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 net  $G$  that learn the set of parameters  $\theta$  from the set of true implied volatilities surfaces  $F^*(\theta)$ . In contrast to the training task in section 4.2, 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 elegant solution. Hence, we propose a simple but efficient approach. We penalize the network for each constraint is violated. We use a penalized mean squared error (CMSE) error,

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

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

$$\text{VioSet} = \{\theta \in \mathbb{R}^5 : \alpha(\gamma^*)^2 + \beta \geq 1\}.$$

#### 4.3.1 Network Architecture and Training

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

- i. The input dimension is  $9 \times 9 \times 1$  which corresponds to a discrete-time volatility surface as generated in section 4.5.



- ii. The output is of size  $5 \times 1 \times 1$ , 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.

Calibration Network		
Layer	Shape	Param #
Initialisation	(9,9,1)	
2D-Convolutional (tanh)	(7,7,64)	640
2D-Convolutional (tanh)	(6, 6,64)	16448
2D-MaxPooling	(3, 3, 64)	
2D-Convolutional (tanh)	(2, 2, 64)	16448
ZeroPadding	(4, 4, 64)	
2D-Convolutional (tanh)	(3, 3, 64)	16448
ZeroPadding	(5, 5, 64)	
2D-Convolutional (tanh)	(4, 4, 64)	16448
2D-Convolutional (tanh)	(3, 3, 64)	16448
ZeroPadding	(5, 5, 64)	
2D-Convolutional (tanh)	(5, 4, 64)	16448
ZeroPadding	(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: Calibration CNN: The CNN consists of 8 convolution layers, 1 max pooling layer and 123,781 trainable parameters

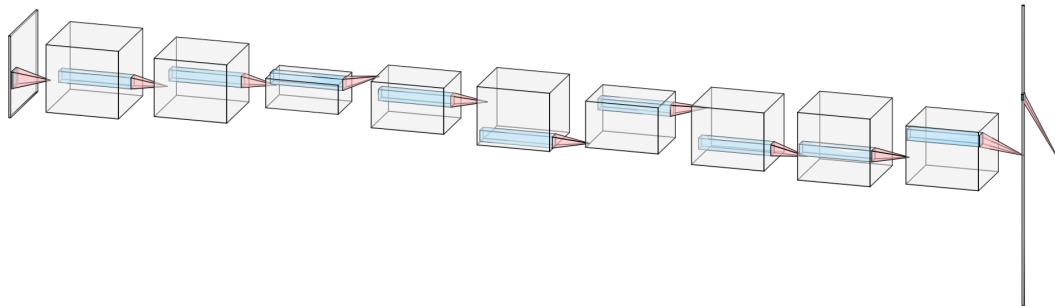


FIGURE 4.2: **PRELIMINARY GRAPHIC** Calibration CNN

To ensure that the forecasted parameters  $G(\sigma_{BS}^{MKT}(T, K), \hat{v})$  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.

## 4.4 Benchmark Model

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 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 calibration theory using globalised interior point algorithm.

### 4.4.1 Feed-Forward Neural Network Approach

In contrast to our approach of using two separate neural networks, they use one network for pricing and calibration. This can be done since their simple network structure allows for fast backpropagation. Their approach adjusted to our data can be summarized as follows:

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

$$F^*(\theta) : \Theta^{HNG} \rightarrow \mathbb{R}^{m \times n}$$

$$\theta \mapsto F^*(\theta).$$

The optimal weights are calculated via

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

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

$$\begin{aligned} \hat{\theta} = \underset{\theta \in \Theta^{HNG}}{\operatorname{argmin}} \quad & \sum_{i=1}^n \sum_{j=1}^m (H(\theta, \hat{v})_{ij} - \sigma_{BS}^{MKT}(T_i, K_j))^2 \\ \text{subject to} \quad & \begin{cases} \alpha(\gamma^*)^2 + \beta < 1 \\ \alpha \geq 0 \\ \beta \geq 0 \\ \omega > 0 \\ h_0 > 0 \end{cases} \end{aligned} \quad (4.4)$$

### 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.

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 normalized the same way as in section 4.3.1. The structure of the network  $G$  is visualized in Figure 4.3.

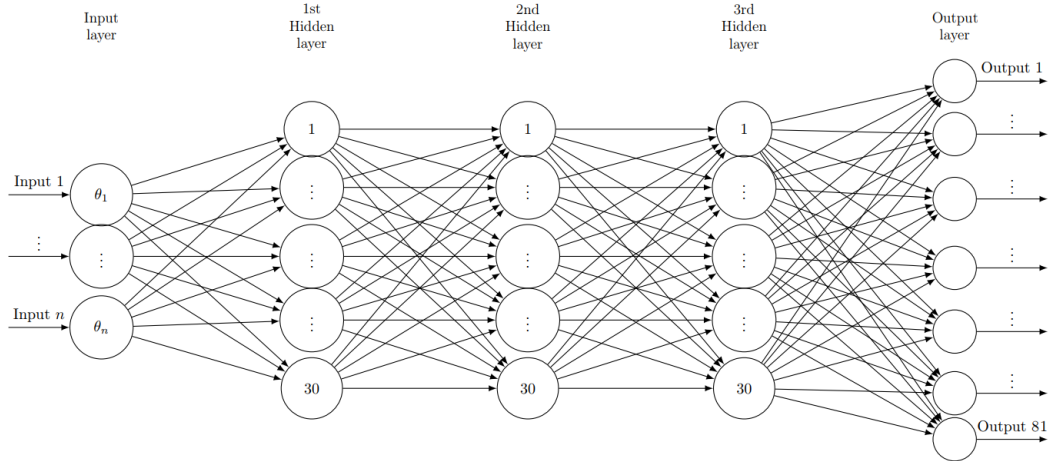


FIGURE 4.3: FFNN Visualisation from Horvath, Muguruza, and Tomas (2019), page 20. Adapted

### Calibration

Due to the simple structure of the network  $G$ , it is easy to calculate the gradient. For  $i \in \mathbb{N}_{\leq 5}$  let  $A_j^{(i)}$  be the output of the network  $G$  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 optimize. Let  $Z_j^{(i)} = W^{(i)} A_j^{(i)} + b^{(i)}$  be the linear combination within the  $i$ -th layer for the  $j$ -th datapoint. We get the following identities for  $A^{(i)}$  :

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

By  $C_j$  we denote the true implied volatilities for the  $i$ -th parameter combination. The loss of function  $L$  of  $G$  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

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

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

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

where

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

Horvath, Muguruza, and Tomas (2019) test several gradient based optimizers and find that the Levenberg-Marquardt algorithm is the most balanced

optimizer regarding speed and convergence. Hence it is used to solve the optimization problem (4.4).

#### 4.4.2 Classic Calibration

Besides different neural network approaches we introduced for calibration, we compare the performance of our method to standard optimization theory. We minimize the MSE over all surfaces and the implied volatility or prices, respectively. The optimization problem (4.5) is solved.

$$\begin{aligned} \hat{\theta} = \underset{\theta \in \mathbb{R}^5}{\operatorname{argmin}} \quad & \sum_{i=1}^n \sum_{j=1}^m (\sigma_{BS}(P(\theta, \hat{v}))_{ij} - \sigma_{BS}^{MKT}(T_i, K_j))^2 \\ \text{subject to} \quad & \begin{cases} \alpha(\gamma^*)^2 + \beta < 1 \\ \alpha \geq 0 \\ \beta \geq 0 \\ \omega > 0 \\ h_0 > 0 \end{cases} \end{aligned} \quad (4.5)$$

This approach and its choice of goal function is elaborated in more detail in section 3.2 as it was already used to evaluate the empirical performance of the model. It is similar to the previously introduced calibration approach, but instead of using a neural net, the closed form solution is used for evaluation.

#### 4.4.3 Monte Carlo simulation.

One standard approach for approximating stochastic models are Monte Carlo Simulations. Instead of calculating stochastic integrals/expected values analytically, a large number of independent stock price paths is used to approximate the expected value of an option. As it only exploits the law of large numbers, it is one of the easiest yet widest applied approaches. For every probability measure  $\mathbb{P}$ , integrable random variable  $X$  and measurable function  $f$  holds  $\frac{1}{N} \sum_{i=1}^N f(X(\omega_n)) \xrightarrow{\mathbb{P}-a.s.} \mathbb{E}(f(X))$ , where  $(\omega_n)_n$  is a family of i.i.d. drawn paths. We use this approach to get a non-deeplearning based benchmark for our pricer. As the trainigset consists of risk neutral parameters and pricing option is only possible under the risk neutral measure  $\mathbb{Q}$ , the stock prices paths have to be generated under  $\mathbb{Q}$  as well. We choose to draw the path directly under  $\mathbb{Q}$  while it is also possible to generate paths under  $\mathbb{P}$  and weight them based on the difference between  $\mathbb{P}$  and  $\mathbb{Q}$ .

## 4.5 Generating a Trainingset

As for every deep learning task, it is utmost important 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 done extremely carefully. Therefore we need to clarify what properties are necessary to deal with the given problem.

The Heston Nandi GARCH(1,1) model under 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. Hereby it is important that the parameters should represent real world scenarios as we want to apply the approach ultimately on real option prices. Furthermore a parameter combination must fulfill mathematical conditions as stationarity and positivity constraints of HNG (cf. section 3.1). 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. The procedure used for all neural net approaches can be summarized as follows:

First, on a weakly basis over several years the model is calibrated on observed call option prices. Afterwards a random sample of parameters is drawn which captures the statistical properties of the priorly calibrated prices. Such a random sample is then filtered such that only parameter sets which fulfill all model constraints remain. Prices and implied volatility are generated for those remaining parameters. Last but not least parameter sets with lead to extreme prices and volatilities are filtered out (cf. section 4.5.2). One could argue that this may decrease the significance of the results as supposedly scenarios which are difficult to map are excluded, but it is necessary to ensure a more realistic trainingset as prices or volatilities of e.g.  $10^{-8}$  % of the underlying asset price are not reasonable.

### 4.5.1 Calibration

To obtain a set of fitting parameters with good concordance to observed prices, the calibration task from section 3.2 is repeated. On all Wednesdays between January 1<sup>st</sup> 2010 and December 31<sup>th</sup> 2018 European call option contracts on the S&P500 with characteristics that lie within the following bounds are considered:

- i. The maturity is in between 8 and 250 days.

- ii. The **Money**ness is in between 0.9 and 1.1.
- iii. The open interest and trading volume is over 100 at the observed Wednesday.

We use the calibration method which led to the best results in our previous analysis (cf. section 3.2): The MSE of call prices is minimized on every Wednesday with respect to the parameter constraints. A globalised interior-point algorithms is used. The parameters are scaled to be in the same magnitude. Again, we want to remark that this step is of utmost importance as the model is extremely sensitive to the parameters on one hand and on the other hand the magnitude between typical  $\alpha, \gamma, h_0$  and  $\beta, \gamma^*$  differs extremely. The initial conditional variance  $h_0$  is calibrated. The interest rate used to estimate prices is calibrated from T-Bill rates on the corresponding day. This optimization leads to 464 parameter combinations. Table 4.3, 4.4 and Figure 4.4 summarize the results.

	$\alpha$	$\beta$	$\gamma^*$	$\omega$	$h_0$
<b>mean</b>	3.1962e-06	0.6479	379.6678	1.8837e-07	7.5785e-05
<b>median</b>	2.2875e-06	0.6813	350.5501	1.3915e-09	5.0526e-05
<b>max</b>	3.8669e-05	0.9909	1373.9331	4.6840e-06	6.2074e-04
<b>min</b>	7.2297e-12	0.0019	-529.5509	2.3293e-11	3.3337e-06
<b>2.5% qu.</b>	5.5113e-07	0.1892	148.3569	7.6257e-11	9.8376e-06
<b>97.5% qu.</b>	1.1426e-05	0.8632	900.1006	1.9948e-06	3.0903e-04

TABLE 4.3: Descriptive statistics of the underlying data

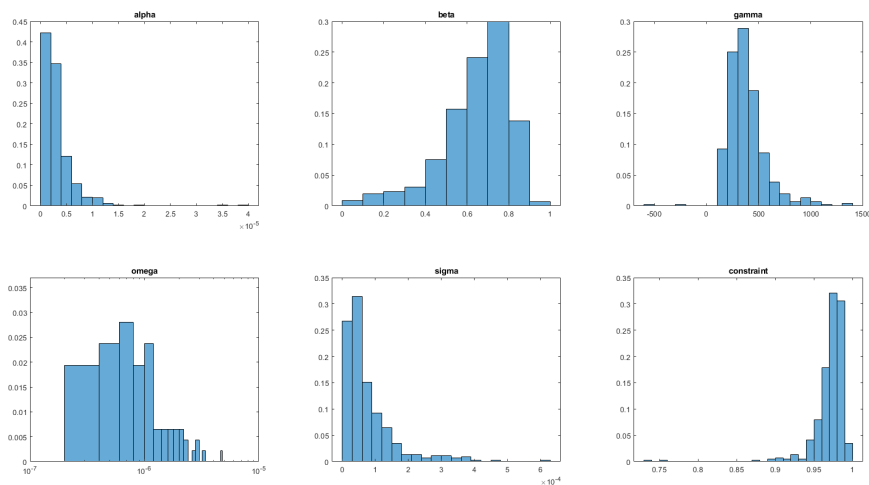


FIGURE 4.4: Histogram of empirically calibrated parameters

$\theta$	$\omega$	$\alpha$	$\beta$	$\gamma^*$	$h_0$
$\omega$	1	-0.0782	-0.4865	0.4088	0.3970
$\alpha$	-0.0782	1	-0.2009	-0.4679	0.2863
$\beta$	-0.4865	-0.2009	1	-0.4892	-0.5434
$\gamma^*$	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 underlying data

Furthermore a second underlying dataset was created by restricting this data. In that restricted underlying dataset a historic calibrated parameter set is filtered out if beta and gamma values are outside of the symmetric 95% CI. This corresponds to the 9.05% of data. The descriptive statistics of this restricted dataset is shown in Table 4.5 and Figure 4.5.

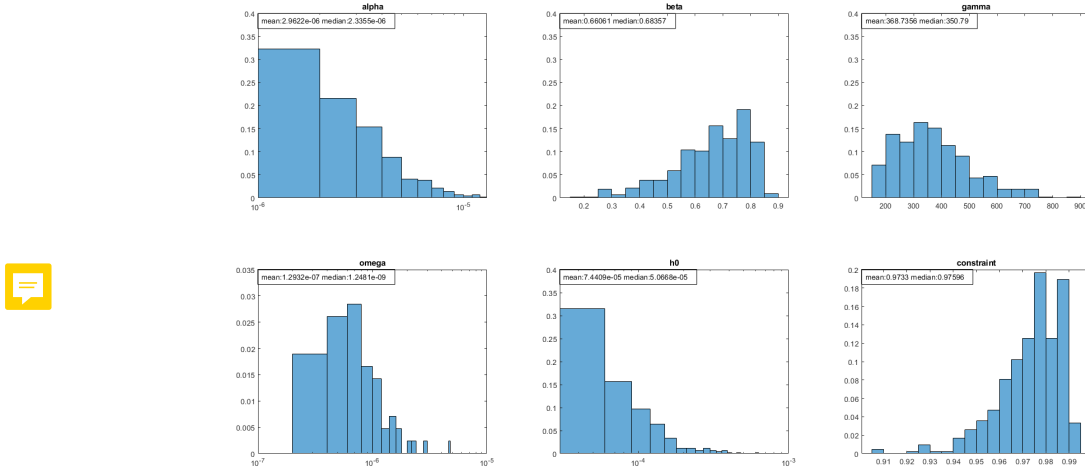


FIGURE 4.5: Histogram of filtered empirically calibrated parameters

$\theta$	$\omega$	$\alpha$	$\beta$	$\gamma^*$	$h_0$
$\omega$	1	0.0319	-0.3641	0.2473	0.4315
$\alpha$	0.0319	1	-0.2188	-0.6867	0.3814
$\beta$	-0.3641	-0.2188	1	-0.3576	-0.5719
$\gamma^*$	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 data



### 4.5.2 How to draw a "good" random sample

As it is desirable that the trainingset captures the properties of those previously calibrated parameters, several points need to be considered: On the one hand, the distribution of parameters in the underlying dataset and in the trainingset should be similar. 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, support and tail distribution have to be carefully attended. In theory, creating such a trainingset should provide a nice in-sample fit. On the other hand, it is not clear if a neural net is able to capture the properties of extreme scenarios as tails of empirical distributions are usually thin. Therefore the question arises how strong the similarities between reality and artificial sample have to be. Weighting out these aspects against each other, we came up with several different approaches; each with a focus on different aspect of training:

#### Normal Distribution

When analysing the underlying calibrated data, the bell shaped distribution are remarkable features. Hence one of the easiest and most obvious ways to approach is to generate a normal distributed sample. The sample mean  $\mu$  and covariance matrix  $\Sigma$  of the underlying dataset are estimated. After a multivariate normally distributed random sample with mean  $\mu$  and covariance matrix  $\Sigma$  are drawn. Finally scenarios which do not satisfy positivity or stationarity constraints are filtered out. As one can see in Figure .. and ... the dataset are really similar. Furthermore the mean and covariance of the data is approximately conserved - obviously due to the adhoc filtering the mean of the random sample has a positive bias. This issue can be tackled by shifting the random sample.

#### MinMax-Uniform Distribution

Another simple approach focuses on 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 nicely, but its support. It assumes independence of the parameters which is clearly not the case. The

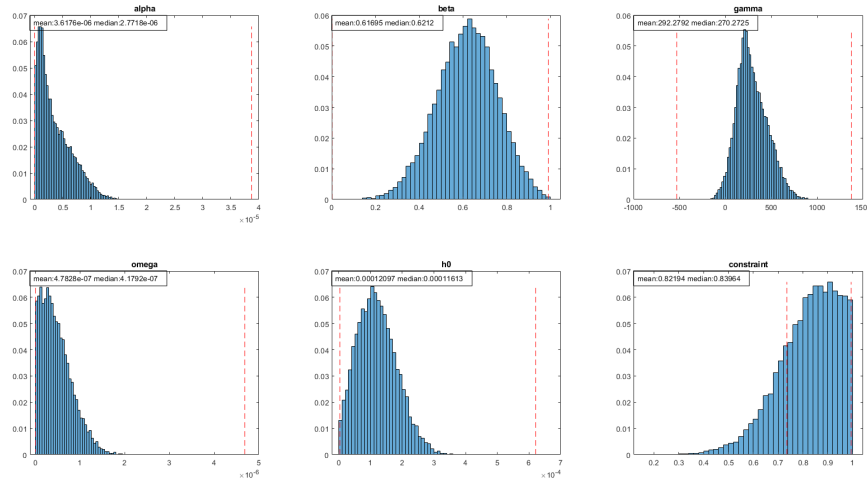


FIGURE 4.6: Histogram: Empirical Parameter Distribution based on S&P500 calibrated Data. Underlying Distribution: Normal Distribution.

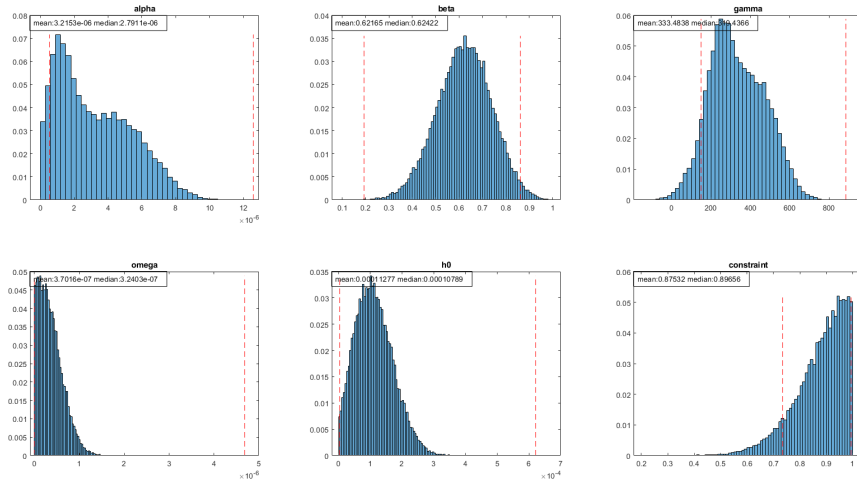


FIGURE 4.7: Histogram: Empirical Parameter Distribution based on S&P500 calibrated Data. Extreme Scenarios are filtered out. Underlying Distribution: Normal Distribution.

main argument in favor of this approach is that extreme scenarios are highly over represented which could lead to a better out-of-sample performance.

### Mean-Symmetric-Uniform Distribution

As stated previously Drawing uniformly from the interval in between minimum and maximum value, does neither capture the standard deviation or the mean of the underlying dataset. To tackle that problem, the following adjustments are done: Again a uniform random sample is drawn .but the

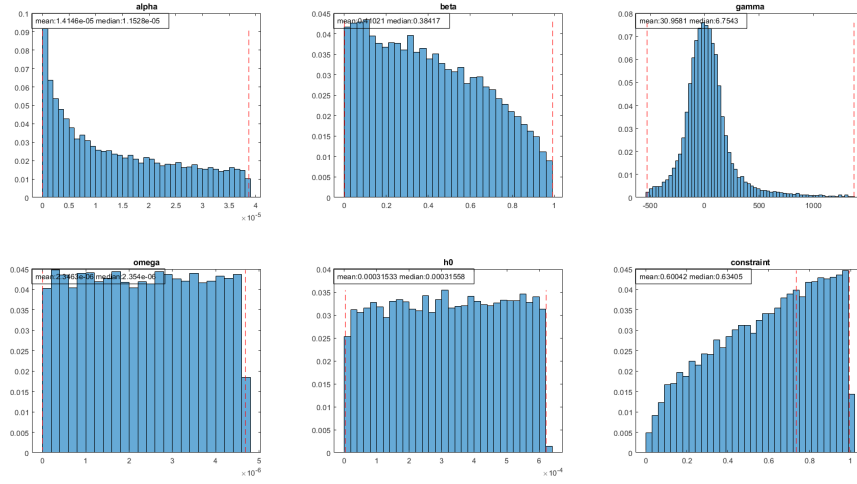


FIGURE 4.8: Histogram: Empirical Parameter Distribution based on S&P500 calibrated Data. Underlying Distribution: Uniform Distribution.

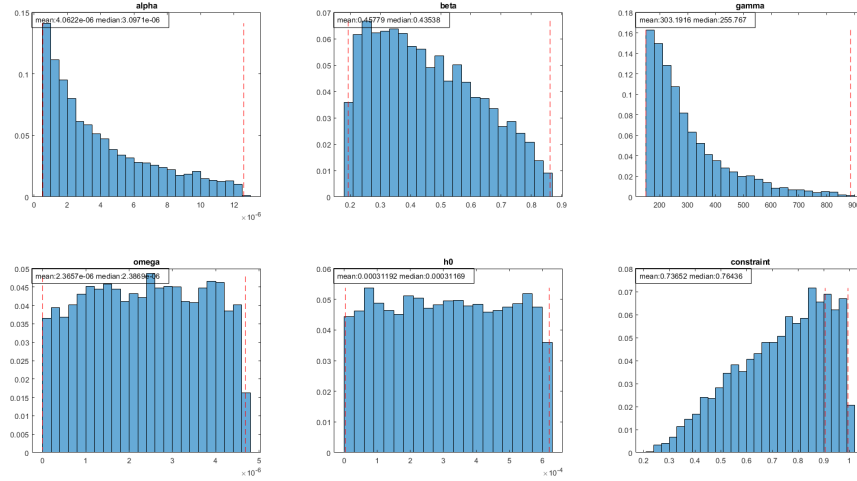


FIGURE 4.9: Histogram: Empirical Parameter Distribution based on S&P500 calibrated Data. Extreme Scenarios are filtered out. Underlying Distribution: Uniform Distribution.

boundaries of the uniform distribution are chosen to fit the standard deviations. The sample is drawn symmetric around the mean of the underlying data  $[\bar{x} - a, \bar{x} + a]$ . Here  $a$  fulfills the following equation:

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

Hence the random sample is drawn from the interval  $[\bar{x} - \frac{\sqrt{12}\sigma_{\text{data}} - \bar{x}}{2}, \bar{x} + \frac{\sqrt{12}\sigma_{\text{data}} - \bar{x}}{2}]$

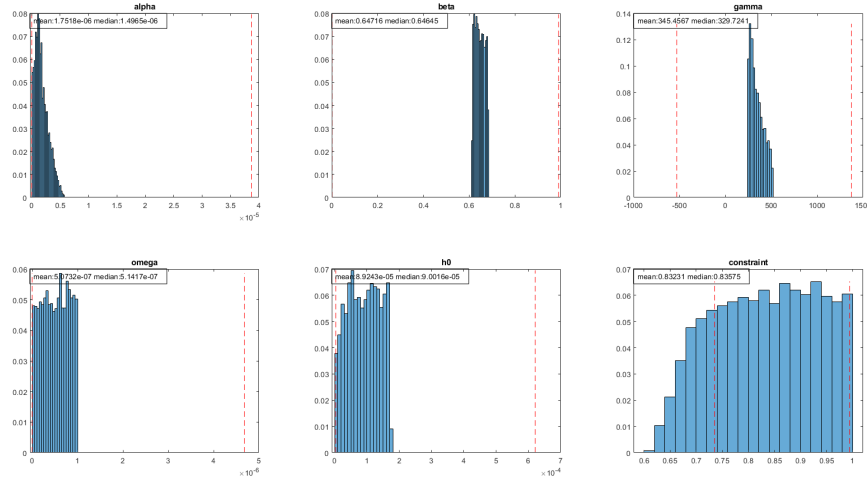


FIGURE 4.10: Histogram: Empirical Parameter Distribution based on S&P500 calibrated Data. Underlying Distribution: Mean-Symmetric Uniform Distribution.

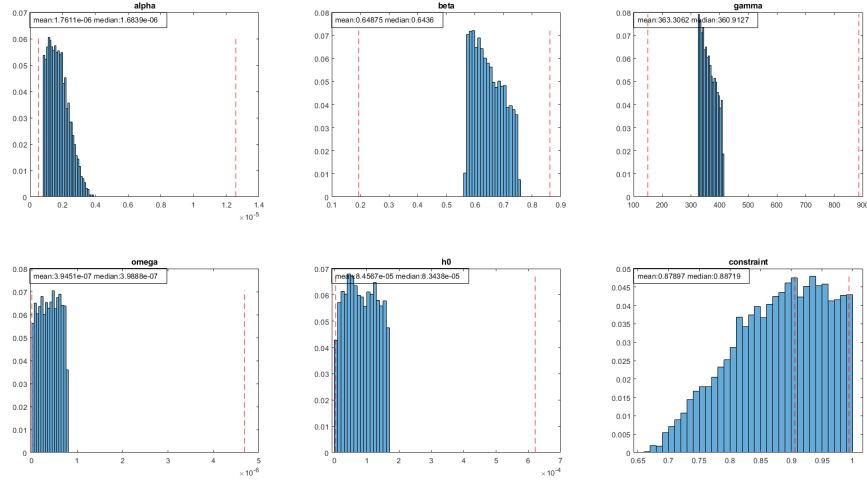


FIGURE 4.11: Histogram: Empirical Parameter Distribution based on S&P500 calibrated Data. Extreme Scenarios are filtered out. Underlying Distribution: Mean-Symmetric Uniform Distribution.

### Partial-Log-Normal Distribution

Filtering out parameters which do not satisfy the positivity constraints adhoc, might look inelegant for some readers. This drawback can be compensated by taking the logarithm of the affected parameters, namely  $\omega, \alpha, \beta$  and  $h_0$ . Following the approach based on the normal distribution leads to a dataset which is log-normal distributed in the transformed parameters and normal distributed in  $\gamma^*$ . As the reader can see in figure – and ..., such an approach

lead to really slim tails which, again, might cause performance problem when pricing out-of-sample.

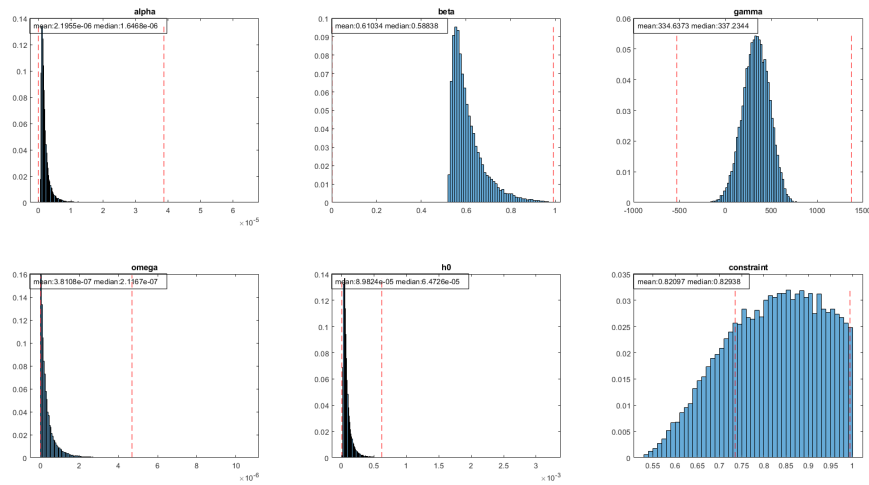


FIGURE 4.12: Histogram: Empirical Parameter Distribution based on S&P500 calibrated Data. Underlying Distribution: Log-Normal Distribution.

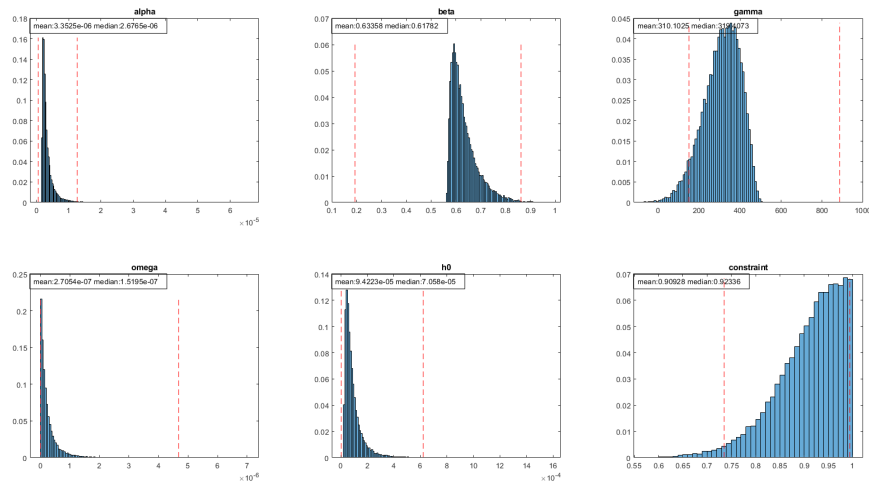


FIGURE 4.13: Histogram: Empirical Parameter Distribution based on S&P500 calibrated Data. Extreme Scenarios are filtered out. Underlying Distribution: Log-Normal Distribution.

### Preliminary MinMax-Scaling

Combining all previous mentioned positive effects and minimizing the pitfalls, lead to the following approaches: The idea is to generate only parameters within the minimum and maximum of the underlying dataset but still account for correlations between the parameters. First, use min-max scaling to bound the underlying dataset on the interval  $[-1, 1]$  (to avoid numerical issues in the next step for minimum and maximum, subtract/add a small values, e.g.  $10e-8$ , from every value). Take the Areatangens hyperbolicus of the dataset. Now the data is again located on  $\mathbb{R}$ . Normalise the transformed data and calculate the mean  $\mu^*$  and covariance matrix  $\Sigma^*$ . Draw a multivariate normally distributed random sample with mean  $\mu^*$  and covariance matrix  $\Sigma^*$ . Re-scale the sample to the statistics of the transformed data, apply Tangens Hyperbolicus, normalise the data and re-scale them to the statistics of the min-max scaled underlying dataset. Finally, invert the min-max scaling.

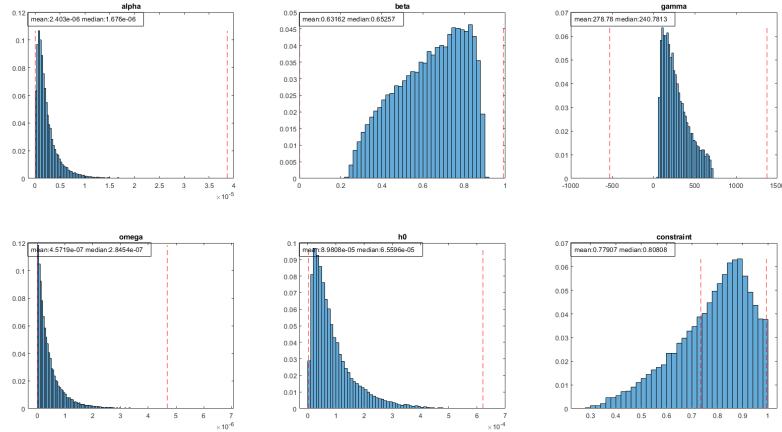


FIGURE 4.14: Histogram: Empirical Parameter Distribution based on S&P500 calibrated Data. Underlying Distribution: Preliminary MinMax-scaling

### 4.5.3 Further Specification of the Trainingset

In all further analysis, we consider a grid of maturities  $M \in \{10, 40, \dots, 250\}$  days and (inverse) Moneyness of  $\frac{K}{S} \in \{0.9, 0.925, \dots, 1.1\}$  to match the option properties of the underlying dataset. As the HNG model is homogeneous in  $(S_t, K)$ , the underlying price is set to 1. For each parameter set a surface of prices is calculated using the closed form solution of HNG (3.4) on each point on the grid. The used yield-curves are chosen uniformly from the set of historic T-Bill curves on the underlying Wednesdays between 2010-2018.

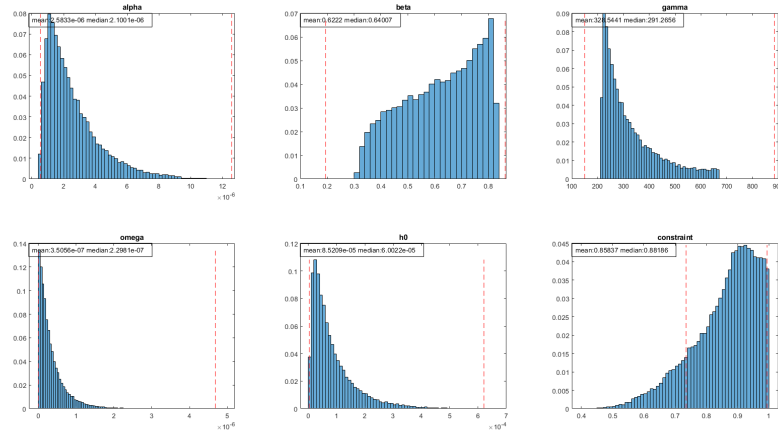


FIGURE 4.15: Histogram: Empirical Parameter Distribution based on S&P500 calibrated Data. Extreme Scenarios are filtered out. Underlying Distribution: Preliminary MinMax-scaling

The algorithm introduced by Jäkel (2015) is used to efficiently and accurately calculate implied volatilities from each price surface. Again, this results in a  $9 \times 9$  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 testingset. Finally the dataset contains the used yield-curves as such data would be available in application.





## Chapter 5

# Summary and Results

### 5.1 Numerical Experiments and Results

If not stated else wise 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 introduced in section 4.5.2. 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 compute the MAPE between the forecast and the closed form solution. To check for significant difference in the pricing result, we use the two-sample one-sided Kolmogorov test and the Anderson test. To compare the results of the calibration task, we compute the average deviation in each parameter. Additionally we use the Kolmogorov-Smirnov test to check whether the empirical distribution of forecasted parameters differs significantly from the true parameter distribution. Besides the accuracy we also compare the approximation speed of the different calibration approaches. Furthermore we check the performance when calibration and forecasting are used in combination. For doing so we use the pricing network on the calibration network. This task is of utmost importance as a good performance with respect to this autoencoding task shows that the calibrated parameters lead to good forecasts even if they might differ from the real parameters. Therefore these experiments can be seen as a measure of robustness. Finally, We check the performance of the already trained networks on different datasets. The code needed to reproduce the experiments is publicly available at [Github.com/henrikbrautmeier/HNGDeepVola](https://github.com/henrikbrautmeier/HNGDeepVola)

### 5.1.1 Pricing Volatility

Figures 5.1, 5.2 and 5.3 show the test sample performance of volatility pricing task of each introduced approach. In table 5.1 we state the descriptive statistics of the results. Our 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 interesting to note that the benchmark

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 %

approach tend to obtain worse results for low maturities while our approach leads to better results with decreasing maturity. Analysing the dependency

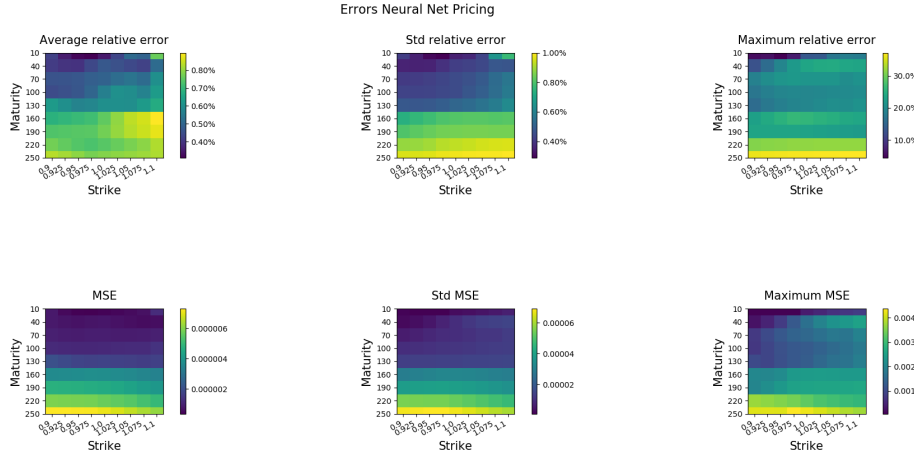


FIGURE 5.1: Test Sample Performance of the Volatility Pricing Network  $F$ . The errors are always measured with respect to the closed form solution.

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 size and error for the Monte Carlo simulation. This holds not true for both deep learning approaches. In Figure 5.7 we compute the empirical error distribution of the three approaches. It holds  $CDF_{CNN}^{emp} > CDF_{FFNN}^{emp} > CDF_{MC}^{emp}$ . We use the two-sample one-sided Kolmogorov-Smirnov test as well as test procedure introduced by Anderson

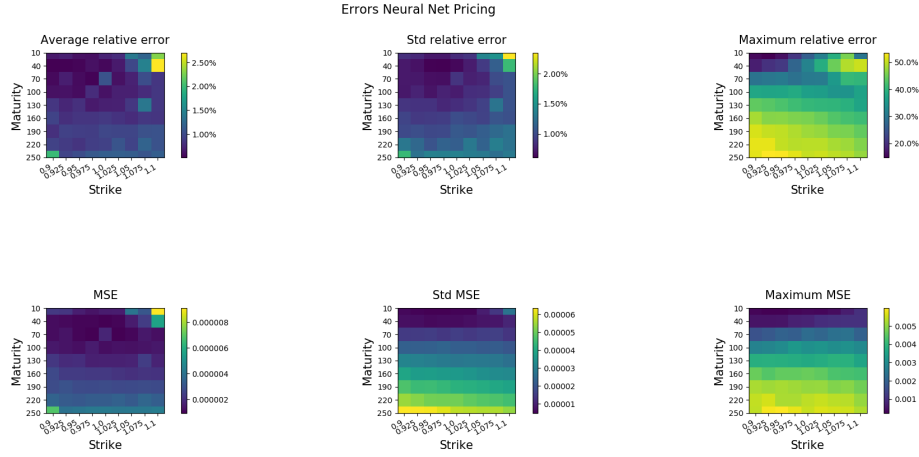


FIGURE 5.2: Test Sample Performance of the Feed-Forward Volatility Pricing Network. The errors are always measured with respect to the closed form solution.

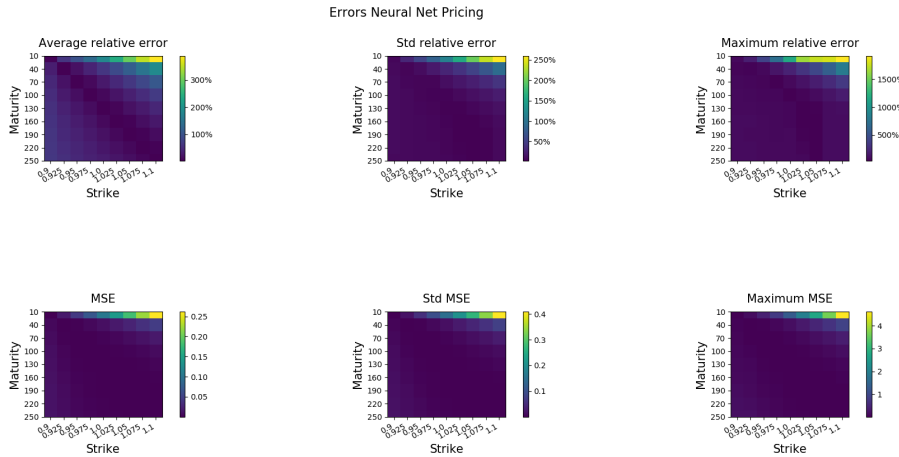


FIGURE 5.3: Test Sample Performance of the Volatility Pricing with Monte Carlo (100.000 paths). The errors are always measured with respect to the closed form solution.

(1996) to test for significant first order 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 your Volatility-Pricing approach is dominated by the benchmark models with respect to the relative deviation. Our approach fits volatilities with significant higher accuracy then the benchmark models. Additional Figures showing the empirical density of the MAPE for each approach can be found in Appendix C.2. In Table 5.2 we compute

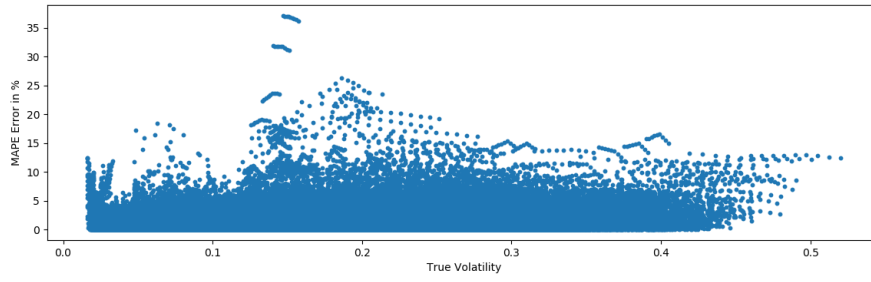


FIGURE 5.4: Test Sample Performance of the Volatility Pricing Network  $F$ . Scatter-Plot: True Volatility vs. Average Error

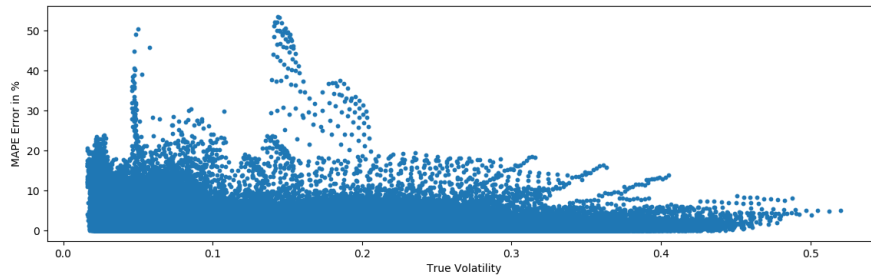


FIGURE 5.5: Test Sample Performance of the Feed-Forward Volatility Pricing Network. Scatter-Plot: True Volatility vs. Average Error

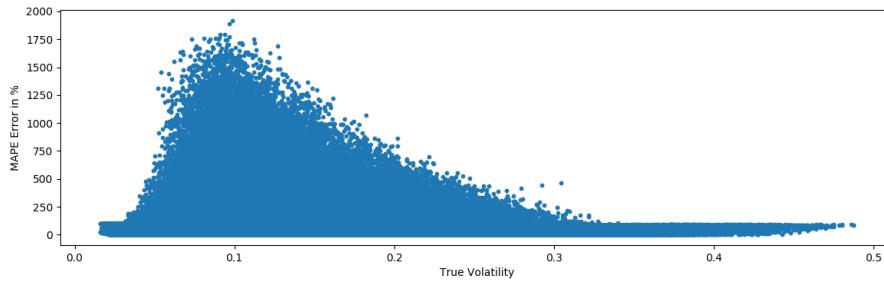


FIGURE 5.6: Test Sample Performance of the Volatility Pricing with Monte Carlo (100,000 paths). Scatter-Plot: True Volatility vs. Average Error

the average Computation Time for pricing one full surface. The FFNN does lead to lowest times, while our approach still performs in same magnitude of time. Both deep learning approach need less time to price one surface then the using the closed form solution. Only the Monte Carlo simulations does not price faster then the close form solution. The deep learning approaches are implemented in Pyhton 3.7 using Tensorflow 2.1 with Keras Backend. we use MATLAB 2020a for the closed form solution and monte carlo simulation. All computations are run on Microsoft Windows Server 2016 Datacenter with 88 Intel Xeon Gold 6152 (2.1 GHz) CPUs. All code used is parallelized to

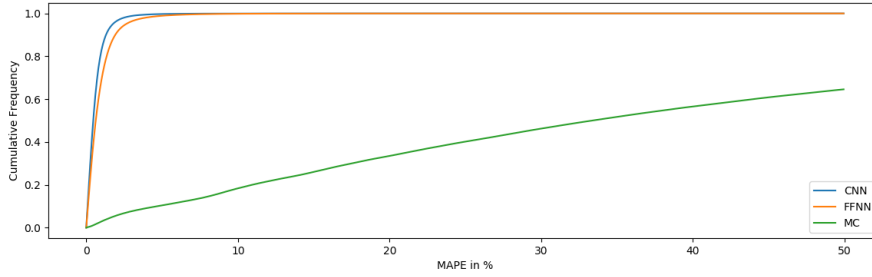


FIGURE 5.7: Test Sample Performance of the Volatility Pricing. Empirical CDF of the relative errors for each approach.

optimize the performance.

Approach	Avg. Comp. Time	Difference to the CNN
CNN	585.9 $\mu$ s	
FFNN	18.2 $\mu$ s	-96.9%
MC	593.6ms	101310.1%
Closed Form	39.1ms	6673.5%

TABLE 5.2: Average Computation Time for pricing one full 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 leads for every parameters to the lowest average and median errors. The mean errors are always at least 2 times smaller then for every other benchmark.

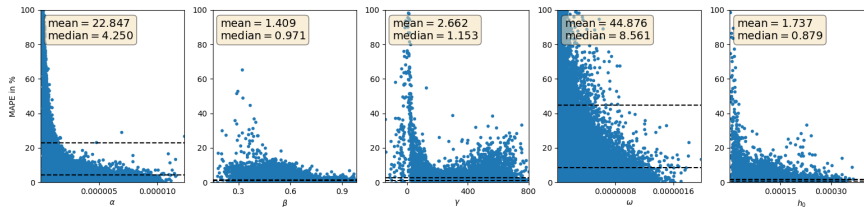


FIGURE 5.8: Calibration relative error per parameter in the test set of the CNN

As our CNN approach only soft-forces the parameters to fulfill the stationarity constraint, we analyse the performance for scenarios which satisfied and those which did not separately in Figure 5.11 and 5.12. Out of 30878 parameter

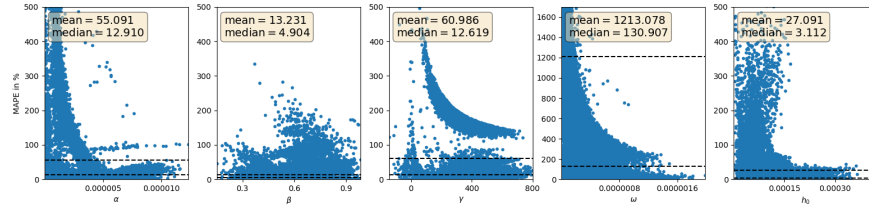


FIGURE 5.9: Relative error per parameter in the test set after Levengerg-Marquardt-FFNN-Calibration.

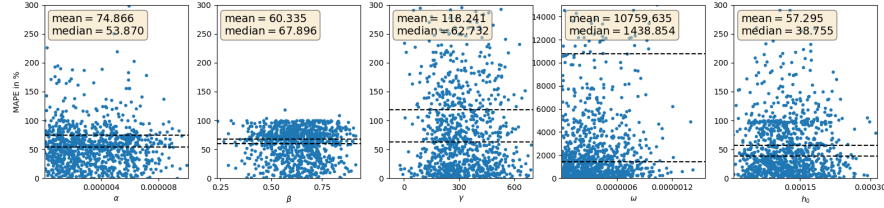


FIGURE 5.10: Relative error per parameter in the test set after Interio-Point Calibration (first 1000 scenarios)

combinations only 65 do not fulfill the stationarity constraint. Even though the accuracy on these scenarios is worse, they are more accurate on average than the benchmarks. This shows that it is of utmost importance to carefully handle parameter constraints. when analysing the scatter plots of the deep

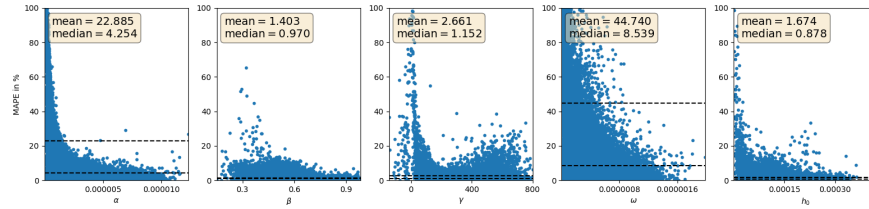


FIGURE 5.11: Calibration relative error per parameter in the test set of the CNN. Only parameter combination that fulfill the stationarity constraint

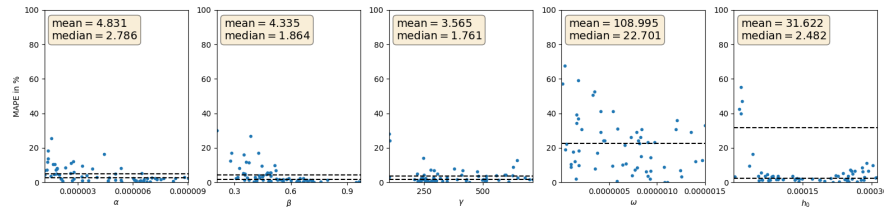


FIGURE 5.12: Calibration relative error per parameter in the test set of the CNN. Only parameter combination that do not fulfill the stationarity constraint

learning approaches a strong correlation between the true parameters size

and the corresponding error is noticeable. Especially for  $\alpha$  and  $\omega$ , the relative errors magnify for values close to zeros. For the classic calibration approach using an interior-point algorithm such an effect is not observed. Computing the spearman-rank-correlation between the absolute parameter size and the corresponding relative deviation in Table 5.3 supports this claim. For our CNN approach, we observe significance rank-correlations between all parameters and the error. Nearly the same holds true for the FFNN-Levenberg approach. Only for the interior-point algorithm no significant correlations are observed. It seems like deep learning approaches deal worse with numerically difficult values than classic algorithms do. In terms of efficiency the CNN approach

$\theta$	$\alpha$	$\beta$	$\gamma^*$	$\omega$	$h_0$
<b>CNN</b>	-0.8459 (0)	-0.4362 (0)	-0.1609 (0)	-0.5335 (0)	-0.3242 (0)
<b>FFNN-Levenberg</b>	-0.3722 (0)	-0.0036 (0.5300)	0.0817 (0)	-0.6687 (0)	-0.5725 (0)
<b>Interior-Point</b>	-0.02156 (0.4959)	-0.0166 (0.5992)	-0.0091 (0.7750)	0.0163 (0.6063)	-0.0037 (0.9077)

TABLE 5.3: Spearman-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

outperforms both benchmarks significantly. Our proposed approach needs 5000 times less computation time than the FFNN 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 top the CNN
<b>CNN</b>	286.1 $\mu$ s	
<b>FFNN-Levenberg</b>	1423.2ms	4974.5%
<b>Interior-Point</b>	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 price surfaces, too. To ensure that our approach leads to parameters which induce good surface, we consider the mapping  $Auto : \mathcal{X} \rightarrow \mathcal{X}$  with  $Auto := F \circ G \approx P \circ P^{-1} \stackrel{!}{=} \text{id}$ . We claim the provided networks  $F, G$  result in a good performance of the  $Auto$  mapping. As the HNG model is really sensitive in some parameters, a small deviation can lead to strong fluctuations, one has no guarantee that presumed good calibration mapping leads to fitting surfaces at a out-of-sample comparison. To proof 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 then given to the pricing network  $F$ . Table 5.5 summarizes the results. The results do not differ significantly from the results of the pure pricing network  $F$ . Similar to the previous section, we see that parameter combination that do not fulfil the stationarity constraint perform slightly worse than those who do. Still, it is remarkable that such scenarios show a lower maximum deviation overall surfaces. This finding might be a result of the small sample size. The allocation of deviations on a surface does not

	Mean	Median	Max	Count
<b>no constraint violation</b>	0.6260	0.4595	31.1354	30813
<b>constraint violation</b>	2.4141	1.5671	17.2712	65
<b>total</b>	0.6298	0.4603	37.1354	30878

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

differ from the allocation observed by the pure pricing network as we see in Figure 5.13, 5.14 and 5.15. Again, the smaller the maturity the smaller the average deviation. We conclude that our calibration approach does not only results in parameter which differ less from the real parameters then the given benchmarks, but also provides fitting surfaces if applied again. We also analysed the inverse autoencoder network  $F \circ G$ . As this network has no clear application or interpretation the results can be found in Appendix C.2.



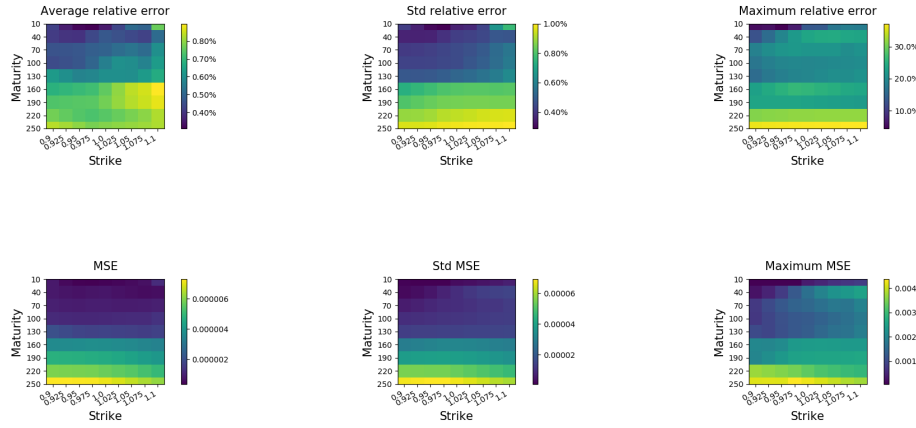


FIGURE 5.13: Test Sample Performance of the Autoencoder Network  $F \circ G$ . The errors are always measured with respect to the closed form solution.

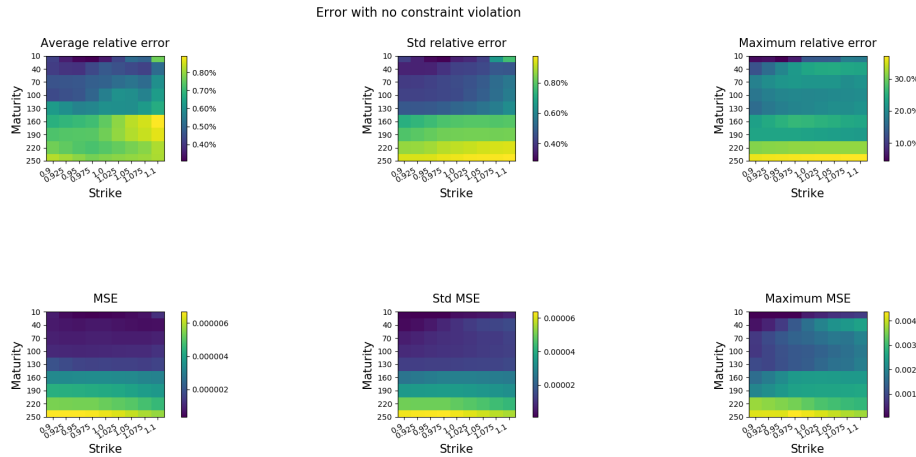


FIGURE 5.14: Test Sample Performance of the Autoencoder Network  $F \circ G$ . The errors are always measured with respect to the closed form solution. Only parameter combinations which fulfill the stationarity constraint

### 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 network structure introduced in section 4.3.1 and 4.2.1 on other datasets. Secondly, we use already trained neural networks from previous sections and test their performance on differently generated datasets. While the first task aims on the ability of our proposed neural network structure to approximate discrete-time volatility

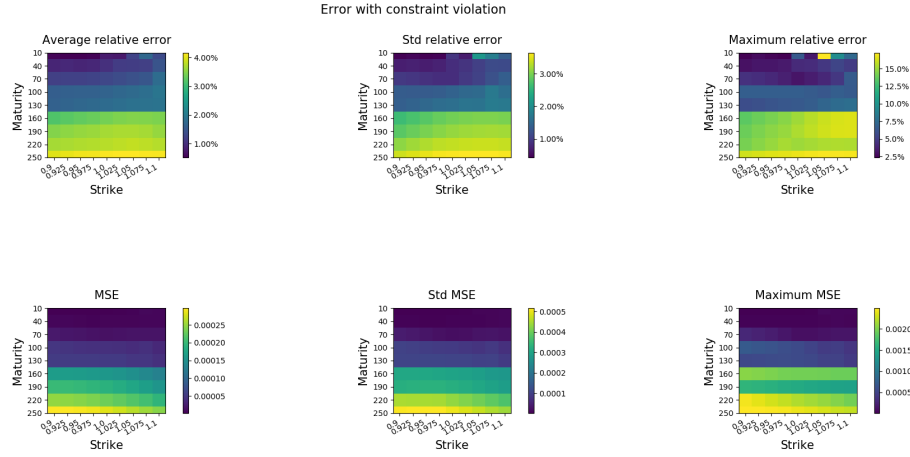


FIGURE 5.15: Test Sample Performance of the Autoencoder Network  $F \circ G$ . The errors are always measured with respect to the closed form solution. Only parameter combinations which that do not fulfill the stationarity constraint

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 4.5.

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 an usability for different discrete-time models. Furthermore we analyse the performance of the networks trained on the normal distribution dataset when they are used to forecast different datasets. It is interesting to see that the calibration network still works reasonable accurate while volatility approximation is roughly as accurate as the Monte Carlo approach. We notice an extreme dependency of the error size to the size of the parameters. This might results from the fact that small values are difficult to predict and as the network is not trained on this kind of small values cannot fit them precisly.

The results 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 not only of the chosen dataset. When using a pre-trained network on different dataset the approximation and calibration

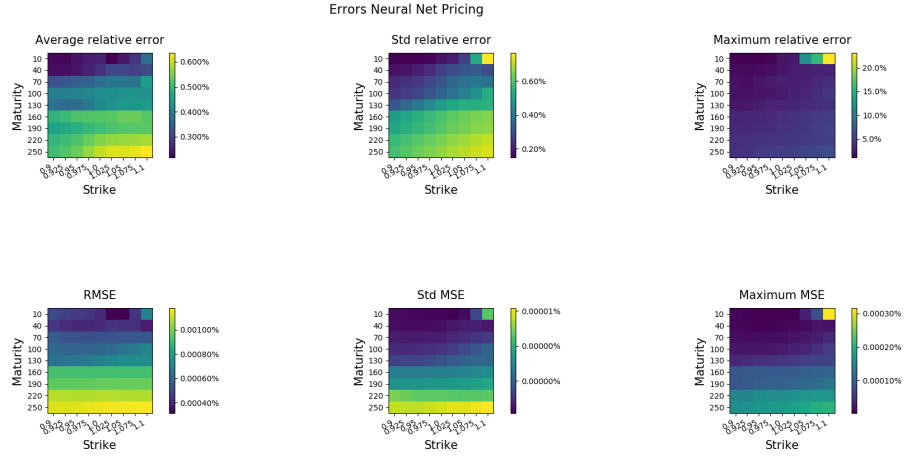


FIGURE 5.16: Performance of the Volatility Pricing Network  $F$  based on the uniform symmetric distributed testing set. The errors are always measured with respect to the closed form solution.

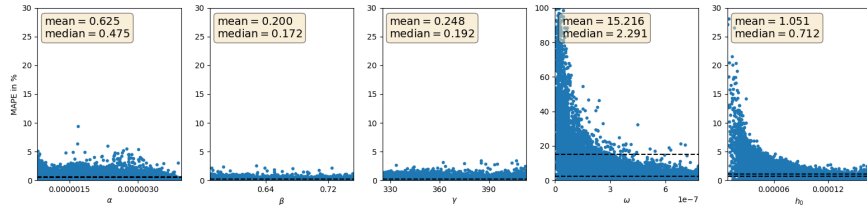


FIGURE 5.17: Calibration relative errors per parameter in the uniform symmetric testing set of trained CNN.

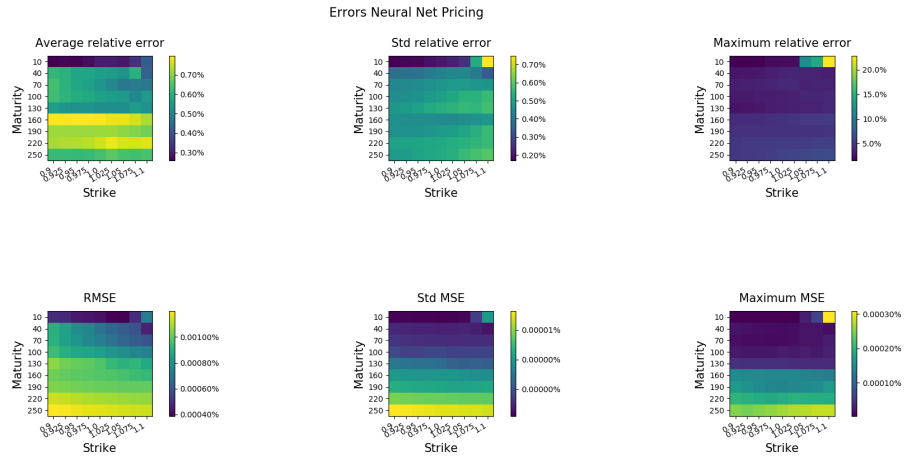


FIGURE 5.18: Performance of the Volatility Pricing Network  $F$  based on the log-normal distributed testing set. The errors are always measured with respect to the closed form solution.

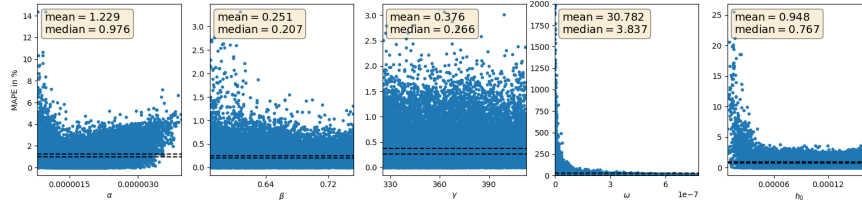


FIGURE 5.19: Calibration relative errors per parameter in the log-normal testing set of trained CNN.

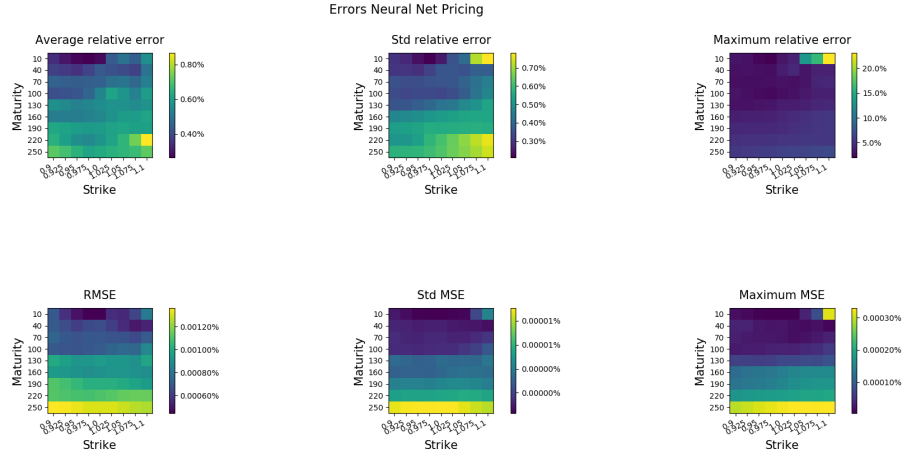


FIGURE 5.20: Performance of the pre-trained Volatility Pricing Network  $F$  based on the uniform symmetric distributed testing set. The errors are always measured with respect to the closed form solution.

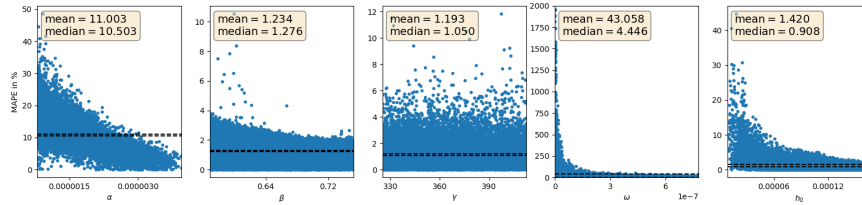


FIGURE 5.21: Calibration relative errors per parameter in the uniform symmetric testing set of pre-trained CNN.

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.

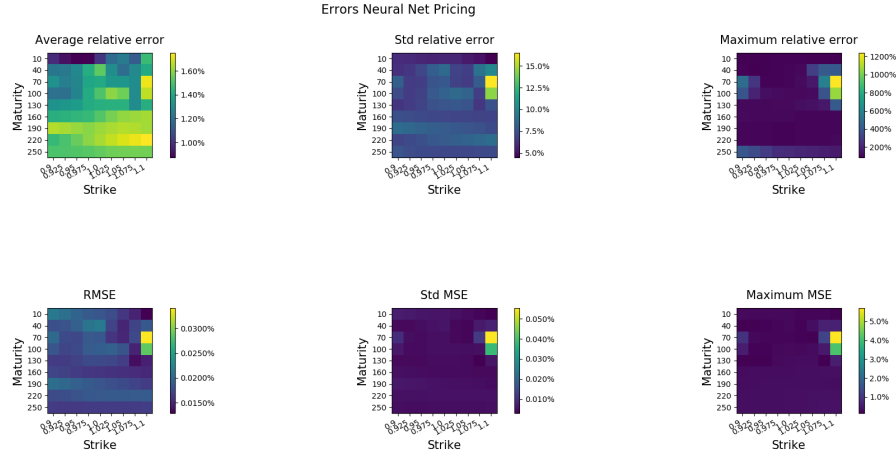


FIGURE 5.22: Performance of the pre-trained Volatility Pricing Network  $F$  based on the log-normal distributed testing set. The errors are always measured with respect to the closed form solution.

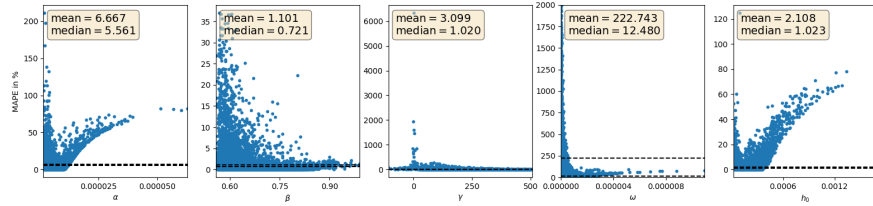


FIGURE 5.23: Calibration relative errors per parameter in the log-normal testing set of pre-trained CNN.

## 5.2 Conclusions and Outlook

We were able to show that our approach can be used as highly efficient and accurate tool for calibration and pricing of the HNG(1,1)-model. It outperforms significantly all given 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". We were able to show that this hold true even for discrete-time models. In the previous section numerical tests show that our networks are an extremely powerful tool to

approximate implied volatilities yet they are highly dependent on the choice of the grid. Now we are interested in a more elastic model which easily allow to change this grid while still being computationally efficient. Furthermore such an improvement would allow to incorporate observed option prices as such contract might not correspond to a point on our grid. Also, we did not focus on the fitting prices. Ultimately option prices are as important to practitioners as implied volatilities and further research should analyse the ability of neural nets to approximate option prices directly. As our calibration approach only soft-forces the parameters to fulfill stationarity constraints and we show that the performance decrease if such constraints are violated, the penalisation of constraints should be refined in future work. Again, in view on real applications, it is utmost important to apply this procedure to more sophisticated models. The last section of the numerical experiments indicate that the proposed approach is able to generalize but more research is needed to conclude this claim. Especially models with no closed form solution available should be tackled thoroughly. Some research opposes the assumption of first order homogeneity as it can be link to dynamics in delta hedging which might not realistic (cf. Kreps and Wallis, 1997). Even though this work does not answer question related to model choice or quality, at this point our approach is not able to capture different underlying prices as all values are normalised beforehand. Therefore further adjustments to this approach in terms of the underlying should be made to capture non-homogeneity. Nonetheless, we proof that deep neural networks are indeed able to approximate option pricing function of discrete-time volatility models.

## 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

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

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

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

and

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

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

$$\text{Cov}(h_{t+1}, Y_t | \mathcal{F}_{t-1}) = -2\alpha\gamma h_t.$$

### Squared Process

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

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

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

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

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



Calculation of  $\mathbb{E}(h_t^2) - \mathbb{E}(h_t)^2$ :

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

Furthermore we get:

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

### 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

$$\begin{aligned}
 h_{t+1:t1} &= r + \lambda \mathbb{E} \left( \omega + \alpha(z_t - \gamma\sqrt{h_t})^2 + \beta h_t \mid \mathcal{F}_t \right) \\
 &= r + \lambda \mathbb{E} \left( \omega + \alpha(z_t^2 + \gamma^2 h_t - 2\gamma z_t \sqrt{h_t}) \mid \mathcal{F}_t \right) \\
 &= r + \lambda(\omega + \alpha(1 + \gamma h_t)).
 \end{aligned}$$

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

$$\begin{aligned}
 Y_{t+1:t} &= (1 + \lambda)r + \lambda^2(\omega + \alpha(1 + \gamma h_t)) \\
 \mathbb{E}(Y_t) &= r + \lambda \mathbb{E}(h_t) = r + \lambda \frac{\omega + \alpha}{1 - \beta - \alpha\gamma^2}.
 \end{aligned}$$



## 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](https://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.

```

1 import numpy as np
2 from tensorflow.compat.v1.keras.models import Sequential, Model
3 from tensorflow.compat.v1.keras.layers import Reshape, InputLayer
   ,Dense, Flatten, Conv2D, Conv1D, Dropout, Input, ZeroPadding2D,
   ZeroPadding1D, MaxPooling2D
4 from tensorflow.compat.v1.keras import backend as K
5 from tensorflow.compat.v1.keras.callbacks import EarlyStopping
6 import tensorflow as tf
7 from tensorflow.compat.v1.keras.backend import set_session
8 from tensorflow.python.client import device_lib
9 config = tf.compat.v1.ConfigProto( device_count = {'GPU': 1 , '
   CPU': 56} )
10 sess = tf.compat.v1.Session(config=config)
11 tf.compat.v1.keras.backend.set_session(sess)
12 print(device_lib.list_local_devices())
13 tf.compat.v1.keras.backend.set_floatx('float64')
14
15 def root_relative_mean_squared_error(y_true, y_pred):
16     return K.sqrt(K.mean(K.square((y_pred-y_true)/y_true)))
17 def mse_constraint(param):
18     def rel_mse_constraint(y_true, y_pred):
19         traf_a = 0.5*(y_pred[:,0]*diff[0]+bound_sum[0])
20         traf_g = 0.5*(y_pred[:,2]*diff[2]+bound_sum[2])
21         traf_b = 0.5*(y_pred[:,1]*diff[1]+bound_sum[1])
22         constraint = traf_a*K.square(traf_g)+traf_b
23         return K.mean(K.square(y_pred - y_true)) +param*K.
mean(K.greater(constraint,1))
24     return rel_mse_constraint
25

```

```

26 es = EarlyStopping(monitor='val_loss', mode='min', verbose=1,
    patience = 50 ,restore_best_weights=True)
27
28 # Volatility Pricing Network
29 NN1a = Sequential()
30 NN1a.add(InputLayer(input_shape=(Nparameters,1,1)))
31 NN1a.add(ZeroPadding2D(padding=(2, 2)))
32 NN1a.add(Conv2D(32, (3, 1), padding='valid',use_bias =True,
    strides =(1,1),activation='elu'))
33 NN1a.add(ZeroPadding2D(padding=(3,1)))
34 NN1a.add(Conv2D(32, (2, 2),padding='valid',use_bias =True,
    strides =(1,1),activation = 'elu'))
35 NN1a.add(Conv2D(32, (2, 2),padding='valid',use_bias =True,
    strides =(2,1),activation = 'elu'))
36 NN1a.add(ZeroPadding2D(padding=(2,2)))
37 NN1a.add(Conv2D(32, (2, 2),padding='valid',use_bias =True,
    strides =(1,1),activation = 'elu'))
38 NN1a.add(ZeroPadding2D(padding=(1,1)))
39 NN1a.add(Conv2D(32, (2, 2),padding='valid',use_bias =True,
    strides =(2,1),activation = 'elu'))
40 NN1a.add(ZeroPadding2D(padding=(1,1)))
41 NN1a.add(Conv2D(32, (2, 2),padding='valid',use_bias =True,
    strides =(2,1),activation = 'elu'))
42 NN1a.add(ZeroPadding2D(padding=(1,1)))
43 NN1a.add(Conv2D(32, (3,3),padding='valid',use_bias =True,strides
    =(2,2),activation = 'elu'))
44 NN1a.add(ZeroPadding2D(padding=(2,1)))
45 NN1a.add(Conv2D(32, (2, 2),padding='valid',use_bias =True,
    strides =(2,1),activation = 'elu'))
46 NN1a.add(Conv2D(32, (2, 2),padding='valid',use_bias =True,
    strides =(2,1),activation = 'elu'))
47 NN1a.add(ZeroPadding2D(padding=(2,1)))
48 NN1a.add(Conv2D(32, (2, 2),padding='valid',use_bias =True,
    strides =(2,1),activation = 'elu'))
49 NN1a.add(Conv2D(Nstrikes, (2, 1),padding='valid',use_bias =True,
    strides =(2,1),activation = 'linear', kernel_constraint = tf.
    keras.constraints.NonNeg()))
50 NN1a.summary()
51 NN1a.compile(loss = root_relative_mean_squared_error, optimizer
    = "adam",metrics=["MAPE","MSE"])
52 NN1b = Sequential()
53 NN1b.add(InputLayer(input_shape=(Nparameters,1,1)))
54 NN1b.add(ZeroPadding2D(padding=(2, 2)))
55 NN1b.add(Conv2D(32, (3, 1), padding='valid',use_bias =True,
    strides =(1,1),activation='elu'))
56 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))

```

```

85 prediction_b = NN1b.predict(X_test_trafo).reshape((Ntest,4,
    Nstrikes))
86 prediction = np.concatenate((prediction_a[:,[0,1,2,3,4],:],
    prediction_b[:,[0,1,2,3],:]),axis =1)
87
88 # Calibration Network
89 NN2 = Sequential()
90 NN2.add(InputLayer(input_shape=(Nmaturities,Nstrikes,1)))
91 NN2.add(Conv2D(64,(3, 3),use_bias= True, padding='valid',strides
    =(1,1),activation = 'tanh'))
92 NN2.add(Conv2D(64,(2, 2),use_bias= True,padding='valid',strides
    =(1,1),activation = 'tanh'))
93 NN2.add(MaxPooling2D(pool_size=(2, 2)))
94 NN2.add(Conv2D(64,(2, 2),use_bias= True,padding='valid',strides
    =(1,1),activation = 'tanh'))
95 NN2.add(ZeroPadding2D(padding=(1,1)))
96 NN2.add(Conv2D(64,(2, 2),use_bias= True,padding='valid',strides
    =(1,1),activation = 'tanh'))
97 NN2.add(ZeroPadding2D(padding=(1,1)))
98 NN2.add(Conv2D(64,(2, 2),use_bias= True,padding='valid',strides
    =(1,1),activation = 'tanh'))
99 NN2.add(Conv2D(64,(2, 2),use_bias= True,padding='valid',strides
    =(1,1),activation = 'tanh'))
100 NN2.add(ZeroPadding2D(padding=(1,1)))
101 NN2.add(Conv2D(64,(2, 2),use_bias= True,padding='valid',strides
    =(1,1),activation = 'tanh'))
102 NN2.add(ZeroPadding2D(padding=(1,1)))
103 NN2.add(Conv2D(64,(2, 2),use_bias= True,padding='valid',strides
    =(1,1),activation = 'tanh'))
104 NN2.add(Flatten())
105 NN2.add(Dense(Nparameters,activation = 'linear',use_bias=True))
106 NN2.summary()
107 NN2.compile(loss =mse_constraint(0.75), optimizer = "adam",
    metrics=["MAPE", "MSE"])
108 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 4.3

$\theta$	2010	2011	2012	2013	2014	2015	2016	2017	2018
$\omega$ std	$4.2779e-09$ ( $1.6791e-08$ )	$3.2992e-07$ ( $1.5604e-06$ )	$3.3648e-08$ ( $1.6574e-07$ )	$3.8491e-07$ ( $1.3052e-06$ )	$1.2743e-07$ ( $4.5655e-07$ )	$4.4951e-08$ ( $2.0855e-07$ )	$2.5272e-08$ ( $1.4770e-07$ )	$3.9321e-08$ ( $1.7009e-07$ )	$3.7781e-08$ ( $2.2443e-07$ )
$\alpha$ std	$1.8528e-05$ ( $1.9304e-05$ )	$1.6271e-05$ ( $2.1985e-05$ )	$9.0589e-06$ ( $1.2012e-05$ )	$6.1070e-06$ ( $7.9519e-06$ )	$7.6946e-06$ ( $9.6389e-06$ )	$7.2374e-06$ ( $7.2754e-06$ )	$5.1346e-06$ ( $5.8307e-06$ )	$2.3951e-06$ ( $3.0938e-06$ )	$1.3159e-05$ ( $1.6443e-05$ )
$\beta$ std	0.6378 (0.2696)	0.5560 (0.2971)	0.7245 (0.2146)	0.7258 (0.2478)	0.6358 (0.2979)	0.5520 (0.2466)	0.6269 (0.2257)	0.7263 (0.2586)	0.5387 (0.3753)
$\gamma$ std	134.9727 (47.8695)	191.7168 (93.1766)	186.9011 (76.3909)	254.4028 (194.7410)	276.4433 (232.3643)	280.6426 (175.7277)	298.3299 (157.3293)	331.9039 (112.0556)	243.3202 (122.2386)
$\hat{h}_0^Q$ std	$1.3056e-04$ ( $1.3959e-04$ )	$2.2460e-04$ ( $2.3120e-04$ )	$8.4830e-05$ ( $5.7765e-05$ )	$4.8801e-05$ ( $4.5932e-05$ )	$4.8652e-05$ ( $5.7911e-05$ )	0.0001 ( $1.1307e-04$ )	$7.5242e-05$ ( $1.0294e-04$ )	$1.9048e-05$ ( $1.9023e-05$ )	$1.3485e-04$ ( $1.7128e-04$ )
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

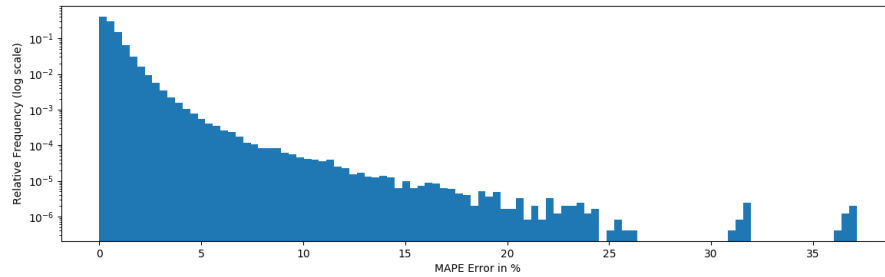


FIGURE C.1: Test Sample Performance of the Volatility Pricing Network  $F$ . Empirical Density of the relative error.

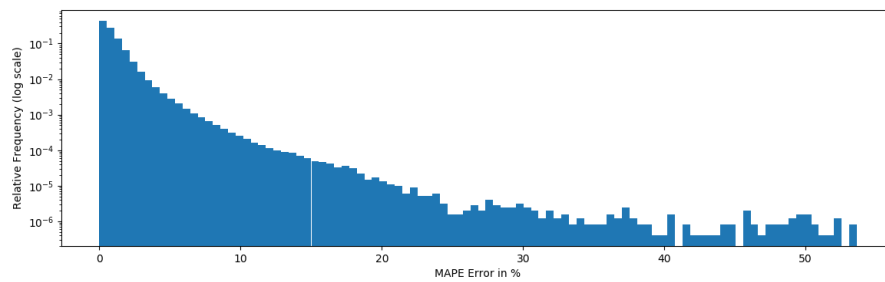


FIGURE C.2: Test Sample Performance of the Feed Forward Volatility Pricing Network. Empirical Density of the relative error.

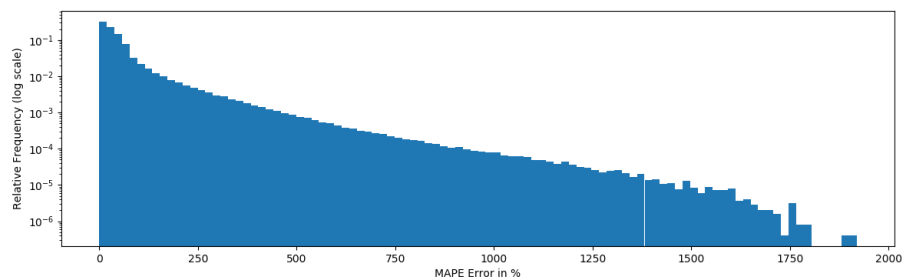


FIGURE C.3: Test Sample Performance of the Volatility Pricing with Monte Carlo (100.000 paths). Empirical Density of the relative error.



## Autoencoder

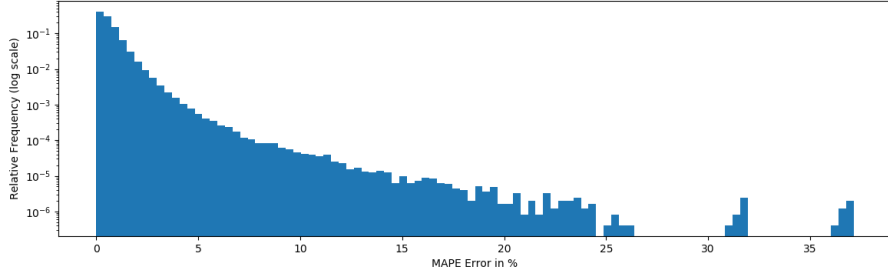


FIGURE C.4: Test Sample Performance of the Autoencoder Network  $G \circ F$ . Empirical Density of the relative error.

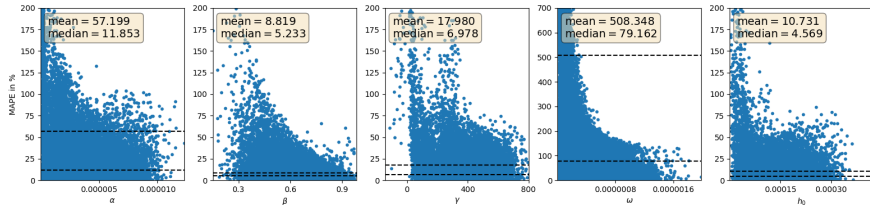


FIGURE C.5: Test Sample Performance of the Autoencoder Network  $G \circ F$ . Relative error per parameter in the testing set.

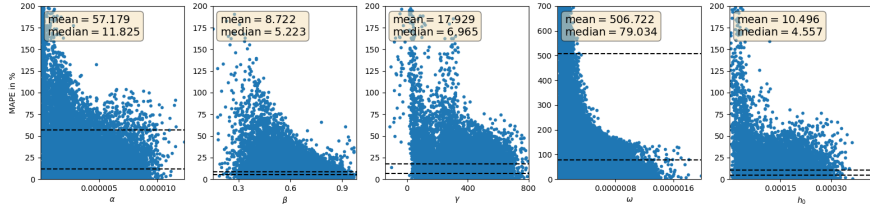


FIGURE C.6: Test Sample Performance of the Autoencoder Network  $G \circ F$ . Relative error per parameter in the testing set. Only parameter combinations which fulfill the stationarity constraint

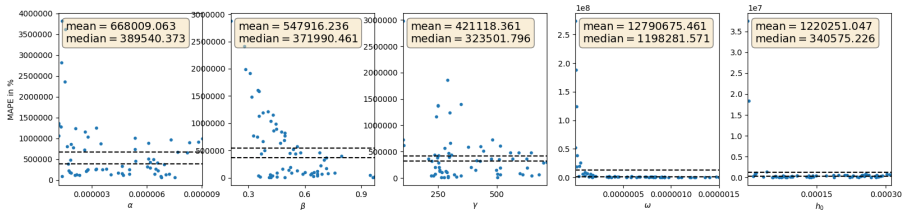


FIGURE C.7: Test Sample Performance of the Autoencoder Network  $G \circ F$ . Relative error per parameter in the testing set. Only parameter combinations which do not fulfill the stationarity constraint

## Different Datasets

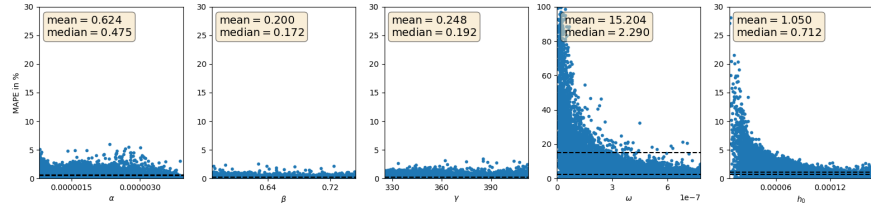


FIGURE C.8: Calibration relative error per parameter in the uniform symmetric distributed testing set of trained CNN. Only parameter combination that fulfill the stationarity constraint

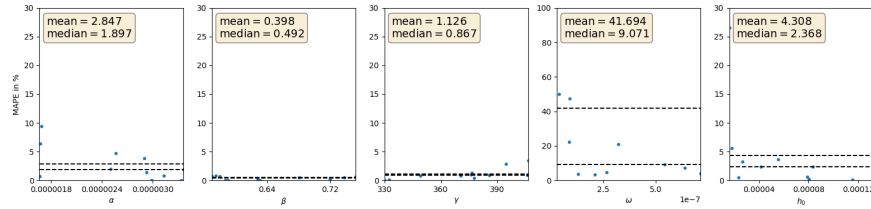


FIGURE C.9: Calibration relative error per parameter in the uniform symmetric testing set of trained CNN. Only parameter combination that do not fulfill the stationarity constraint

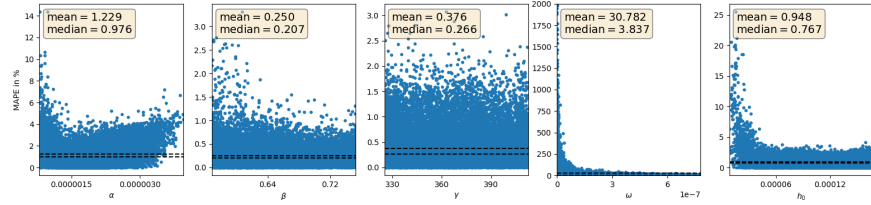


FIGURE C.10: Calibration relative error per parameter in the log-normal distributed testing set of trained CNN. Only parameter combination that fulfill the stationarity constraint

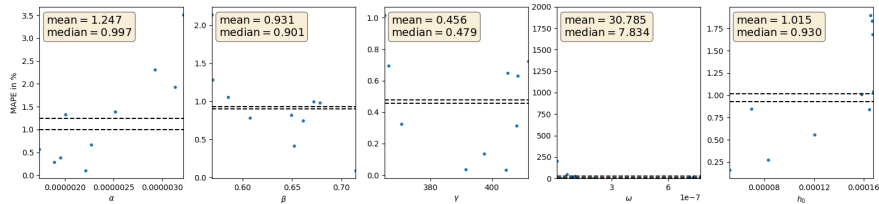


FIGURE C.11: Calibration relative error per parameter in the log-normal testing set of trained CNN. Only parameter combination that do not fulfill the stationarity constraint

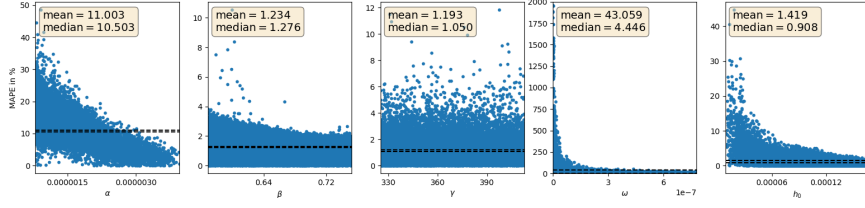


FIGURE C.12: Calibration relative error per parameter in the uniform symmetric distributed testing set of pre-trained CNN. Only parameter combination that fulfill the stationarity constraint

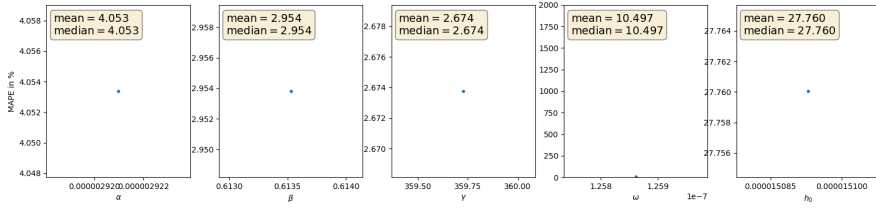


FIGURE C.13: Calibration relative error per parameter in the uniform symmetric distributed testing set of pre-trained CNN. Only parameter combination that do not fulfill the stationarity constraint

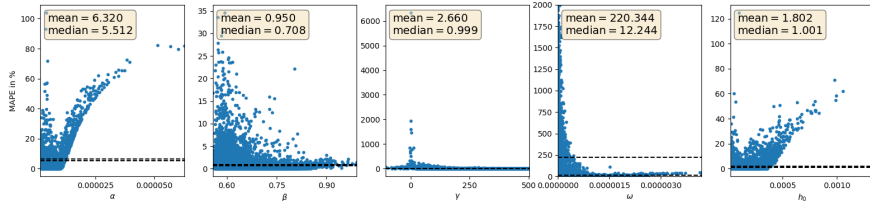


FIGURE C.14: Calibration relative error per parameter in the log-normal distributed testing set of pre-trained CNN. Only parameter combination that fulfill the stationarity constraint

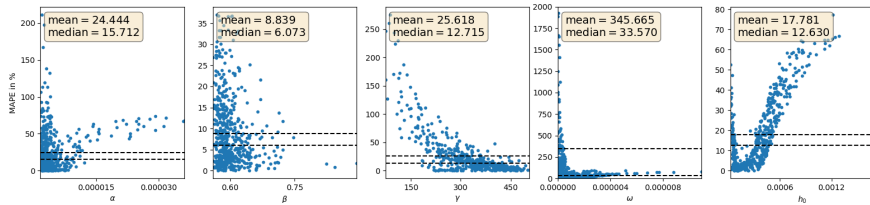


FIGURE C.15: Calibration relative error per parameter in the log-normal distributed testing set of pre-trained CNN. Only parameter combination that do not fulfill the stationarity constraint



# Bibliography

- Ahmend, N.K. et al. (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](https://doi.org/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](https://doi.org/10.1080/13504860210132897).
- Anderson, G. (1996). “Nonparametric Tests of Stochastic Dominance in Income Distributions”. In: *Econometrica* 64.5, pp. 1183–93. JSTOR: [2171961](https://www.jstor.org/stable/2171961).
- Badescu, A. et al. (2020). “How robust is GARCH option pricing calibration?” Working Paper.
- Beran, J. (2018). *Mathematical foundations of time series analysis: a concise introduction*. Springer.
- BIS, Bank for International Settlements (2019). *Statistical release: OTC derivatives statistics at end-June 2019*. URL: [bis.org/publ/otc\\_hy1911.pdf](https://bis.org/publ/otc_hy1911.pdf) (visited on 01/27/2020).
- 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](https://www.jstor.org/stable/1831029).
- 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](https://doi.org/10.1016/j.physa.2018.11.061).
- 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](https://doi.org/10.1109/HICSS.2000.926755).
- EFMA, European Financial Management Association (2018). *Machine learning in European financial institutions*. URL: [efma.com/study/detail/28154](https://efma.com/study/detail/28154) (visited on 04/20/2020).

- 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](#).
- FIA (2019). *Global Futures and Options Trading Reaches Record Level in 2019*. URL: [fia.org/node/5542](http://fia.org/node/5542) (visited on 01/27/2020).
- Föllmer, H. and A. Schied (2016). *Stochastic Finance: An Introduction in Discrete Time*. Walter de Gruyter, 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](#).
- 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](#).
- Horvath, B., A. Muguruza, and A. Tomas (2019). *Deep Learning Volatility*. DOI: [10.2139/ssrn.3322085](#).
- Hull, J.C. (2001). *Fundamentals of Futures and Options Markets*. Prentice Hall, 2011.
- 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. DOI: [10.1016/j.eswa.2018.03.002](#).
- Kirilenko, A. et al. (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: [q-fin/1901.08943](https://arxiv.org/abs/q-fin/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](https://www.jstor.org/stable/3003143).
- Roch, A. (2017). "Asymptotic asset pricing and bubbles". In: *Mathematics and Financial Economics* 2018.2. DOI: [10.1007/s11579-017-0204-1](https://doi.org/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.