Deep Learning Volatility
A deep neural network perspective on pricing and calibration in (rough) volatility models

Blanka Horvath
Department of Mathematics, King's College London
blanka.horvath@kcl.ac.uk, b.horvath@imperial.ac.uk

Aitor Muguruza
Department of Mathematics, Imperial College London & NATIXIS
aitor.muguruza-gonzalez15@imperial.ac.uk

Mehdi Tomas
CMAP & Ladhyx, Ecole Polytechnique
mehdi.tomas@polytechnique.edu

January 25, 2019

Abstract

We present a consistent neural network based calibration method for a number of volatility models—including the rough volatility family—that performs the calibration task within a few milliseconds for the full implied volatility surface. The aim of neural networks in this work is an off-line approximation of complex pricing functions, which are difficult to represent or time-consuming to evaluate by other means. We highlight how this perspective opens new horizons for quantitative modelling: The calibration bottleneck posed by a slow pricing of derivative contracts is lifted. This brings several model families (such as rough volatility models) within the scope of applicability in industry practice. As customary for machine learning, the form in which information from available data is extracted and stored is crucial for network performance. With this in mind we discuss how our approach addresses the usual challenges of machine learning solutions in a financial context (availability of training data, interpretability of results for regulators, control over generalisation errors). We present specific architectures for price approximation and calibration and optimize these with respect different objectives regarding accuracy, speed and robustness. We also find that including the intermediate step of learning pricing functions of (classical or rough) models before calibration significantly improves network performance compared to direct calibration to data.

2010 *Mathematics Subject Classification*: 60G15, 60G22, 91G20, 91G60, 91B25
Keywords: Rough volatility, volatility modelling, Volterra process, machine learning, accurate price approximation, calibration, model assessment, Monte Carlo

The authors are grateful to Ben Wood, Jim Gatheral and Ryan McCrickerd for stimulating discussions. MT acknowledges financial support from the econophysix chair of Ecole Polytechnique.

Contents

1 Introduction

Approximation methods for option prices came in all shapes and forms in the past decades and they have been extensively studied in the literature and well-understood by risk managers. Clearly, the applicability of any given option pricing method (Fourier pricing, PDE methods, asymptotic methods, Monte Carlo, ... etc.) depends on the regularity properties of the particular stochastic model at hand. Therefore, tractability of stochastic models has been one of the most decisive qualities in determining their popularity. In fact it is often a more important quality than the modelling accuracy itself: It was the (almost instantaneous) SABR asymptotic formula that helped SABR become the benchmark model in fixed income desks, and similarly the convenience of Fourier pricing is largely responsible for the popularity of the Heston model, despite the well-known hiccups of these models. Needless to say that it is the very same reason (the concise Black Scholes formula) that still makes the Black-Scholes model attractive for calculations even after many generations of more realistic and more accurate stochastic market models have been developed. On the other end of the spectrum are rough volatility models, for which (despite a plethora of modelling advantages, see [6, 19, 24] to name a few) the necessity to rely on relatively slow Monte Carlo based pricing methods creates a major bottleneck in calibration, which has proven to be a somewhat limiting factor with respect to industrial applications. This dichotomy can become a headache in situations when we have to weigh up the objectives of accurate pricing vs. fast calibration against one another in the choice of our pricing model. It could significantly ease the headache if one had a formula that directly outputs for a choice of model parameters the corresponding vanilla option prices (as the Black-Scholes formula does) for a large range of maturities and strikes.

In fact, the idea of mapping model parameters to shapes of the implied volatility surface directly is not new. The Stochastic volatility inspired SSVI, eSSVI surfaces (see [23, 25, 29]) do just that: A given set of parameters is translated directly to different shapes of (arbitrage-free) implied volatility surfaces, bypassing the step of specifying any stochastic dynamics for the underlying asset. For certain stochastic models something similar is also possible in restricted regimes. For stochastic models that admit asymptotic expansions, such direct mappings from model parameters to (approximations of) the implied volatility surface can be obtained (one example is the famous SABR formula), but such asymptotic formulae have only limited validity along the surface as they are typically limited to certain asymptotic regimes by their very nature. Therefore, a direct mapping from different parameter combinations of stochastic models to different shapes of the (full) implied volatility surface is definitely of beneficial value for modelling: It combines the advantages of direct parametric volatility surfaces (of the SSVI family) with the possibility to link volatility surfaces to the stochastic dynamics of the underlying asset. At the same time, if such direct mappings had been easily available for any given stochastic model in the past, then numerical approximations of option prices would have been redundant. However, building on the knowledge gained from numerical approximations of option prices in different stochastic models, it is today possible to find a direct pricing functional (or an approximation thereof) for stochastic models, that can directly output for a given set of model parameters the corresponding implied volatilities for a large range of maturities and strikes along the whole surface. Clearly, for such a direct approximation map to be fully beneficial for volatility modelling in practice, it is desirable that its approximation accuracy is of the same order as the accuracy of the original numerical pricing approximations.
One contribution of this paper is to demonstrate this on a number of stochastic volatility models, including the rough Bergomi model of [6]. Our audacity of doing so is inspired by the array of pos-

sibilities to train deep neural networks to approximate multidimensional functions. Our objective was to to move the price approximation part of the calibration process (which has created a bottleneck for some (rough) volatility models) into an off-line preprocessing part. This preprocessing amounts to training and storing the approximative direct pricing map. The accuracy of this direct pricing map is demonstrated in our numerical experiments.

Why shift the pricing and calibration tasks of option pricing to a neural network? At first, the motivation is to explore if we can, but beyond that to show that we can do so in a computationally meaningful way: The potential prowess that the availability of neural networks provides readily opens the door to new dimensions of calibration efficiency of stochastic models. There have been several recent contributions on neural network calibrations of stochastic models [8, 11, 30, 44, 14]. Clearly, much depends on the finesse of the particular network design with respect to the performance of these networks. A pioneering work of Hernandez et al. [30] provided one of the first successful Neural Network based calibrations of stochastic models. In their design the network is trained to directly return calibrated (optimized) parameters of the stochastic model. One of the crucial advantages that neural network approximations brings into classical quantitative modelling is the possibility to learn the (approximative) map directly from model parameters to the shapes of implied volatility surfaces. This is possible if the learning is done via a separation of the approximation network and the calibration, as opposed to initial works of Hernandez [30]. By this separation the improvements are twofold:
(1) The training becomes more robust (with respect to generalisation errors on unseen data).
(2) The speed-up of the on-line calibration part is even more even more pronounced than in direct calibration to data.
One of the striking advantages of these improvements is that it speeds up the (on-line) calibration Rough volatility models to the realm of just a few milliseconds. Another advantage of our modelling choice is that by its very design it can be appliedet al to portfolios including multiple strikes and maturities at the same time which is the first step towards their application as hedging instruments. See for example Buehler et al. [11] a motivation.

The paper is organised as follows: In Section 2 we present a neural network perspective on model calibration and recall some stochastic models considered in this paper. In this section we also formalise our objectives about the accuracy and speed of neural network approximation of pricing functionals and the basic ideas of our training. In section 2.4 we elaborate on the advantages of neural networks as approximators of option pricing maps in the context of calibration. Section 3 recalls some background on neural networks as functional approximator and some aspects of neural network training that influenced the setup of our network architecture and training design. In Section 4 we compare different objective functions (from direct calibration to data to an image-based implicit learning approach) and motivate our choice of image-based objective function. We give details about network architectures for the approximation network and compare different optimisers for the calibration step. In Section 5 we present our numerical experiments and Section 6 points to further potential applications and outlook to future work.
Numerical experiments and codes are provided on GitHub: NN-StochVol-Calibrations , where an accessible code demo of our results can be downloaded. We also created a library of stochastic models where this approach is demonstrated to work well.

2 A neural network perspective on model calibration

In plain words, any calibration procedure is meant to fix the model parameters such that the model is as close as possible to the observed reality. In a financial context, our model represents the underlying (stocks, indices, volatility, etc.) and we are interested in calibrating the model to available market prices of financial contracts based on this underlying.
Let us first formalise this by setting the notation $\mathcal{M} := \mathcal{M}(\theta)_{\theta\in\Theta}$ which represents an abstract model with parameters θ in the set $\Theta \subset \mathrm{R}^n$, for some $n \in \mathbb{N}$. Thus the model $\mathcal{M}(\theta)$ (stochastic or parametric) and the corresponding prices of financial contracts are fully specified by the choice of the parameter combination $\theta \in \Theta$. Furthermore, we introduce a pricing map $P : \mathcal{M}(\theta, \zeta) \to R^m$, where $\zeta : (C(\mathbb{R}) \to \mathrm{R}^m)$, $m \in \mathbb{N}$ denote the financial products we aim to price, such as vanilla options for (a set of) given maturities and strikes. Let us denote the observed market data corresponding to the contracts, by $\mathcal{P}^{MKT}(\zeta) \in \mathbb{R}^m$, $m \in \mathbb{N}$.

Parameter Calibration: The parameter configuration $\hat{\theta}$ solves a δ-calibration problem for a model $\mathcal{M}(\Theta)$ for the conditions $\mathcal{P}^{MKT}(\zeta)$ if

$$\hat{\theta} = \underset{\theta\in\Theta}{\operatorname{argmin}}\, \delta(P(\mathcal{M}(\theta), \zeta), \mathcal{P}^{MKT}(\zeta)) \tag{1}$$

where $\delta(\cdot, \cdot)$ is a suitable choice of metric, defined in the corresponding topological space for the financial contract ζ at hand.
For most financial models however (1) represents an idealised form of the calibration problem as in practice there rarely exists an analytical formula for the option price $P(\mathcal{M}(\theta), \zeta)$ and for the vast majority of financial models it needs to be computed by some numerical approximation scheme.

Approximate Parameter Calibration We say that the parameter configuration $\hat{\theta} \in \Theta$ solves an *approximate δ-calibration* problem for the model $\mathcal{M}(\Theta)$ for the conditions $\mathcal{P}^{MKT}(\zeta)$ if

$$\hat{\theta} = \underset{\theta\in\Theta}{\operatorname{argmin}}\, \delta(\widetilde{P}(\mathcal{M}(\theta), \zeta), \mathcal{P}^{MKT}(\zeta)) \tag{2}$$

where $\delta(\cdot, \cdot)$ is a suitably chosen metric and $\widetilde{P}$ is a numerical approximation of the pricing map P.

In the remainder of this paper it is this second type of calibration problem that we will be concerned with in our neural network calibrations: In our numerical experiments in Section 5 we consider the numerical approximation $\widetilde{P}$ of the pricing map P as the benchmark (available truth) for generating synthetic training samples for training a neural network to approximate pricing maps. Clearly, the better the original numerical approximations, the better the network approximation will be.

2.1 A brief reminder of some (rough) models considered

We would like to emphasize that our methodology can in principle be applied to any (classical or rough) volatility model. From the classical Black Scholes or Heston models to the rough Bergomi model of [6], also to large class of rough volatility models (see Horvath, Jacquier and Muguruza [37] for a general setup). In fact the methodology is not limited to stochastic models, also parametric models of implied volatility could be used for generating training samples of abstract models, but we have not pursued this direction further.

The Rough Bergomi model
In the abstract model framework, the rough Bergomi model is represented by $\mathcal{M}^{rBergomi}(\Theta^{rBergomi})$, with parameters $\theta = (\xi_0, \nu, \rho, H) \in \Theta^{rBergomi}$. On a given filtered probability space $(\Omega, \mathcal{F}, (\mathcal{F}_t)_{t\geq 0}, \mathbb{P})$ the model corresponds to the following system

$$\begin{aligned} dX_t &= -\frac{1}{2}V_t dt + \sqrt{V_t} dW_t, \quad \text{for } t > 0, \quad X_0 = 0, \\ V_t &= \xi_0(t)\mathcal{E}\left(\sqrt{2H}\nu \int_0^t (t-s)^{H-1/2} dZ_s\right), \quad \text{for } t > 0, \quad V_0 = v_0 > 0 \end{aligned} \tag{3}$$

where $H \in (0,1)$ denotes the Hurst parameter, $\nu > 0$, $\mathcal{E}(\cdot)$ the stochastic exponential [17], and $\xi_0(\cdot) > 0$ denotes the initial forward variance curve (see [10, Section 6]), and W and Z are correlated standard Brownian motions with correlation parameter $\rho \in [-1, 1]$. To fit the model parameters into our abstract model framework $\Theta^{rBergomi} \subset \mathbb{R}^n$ for some $n \in \mathbb{N}$, the initial forward variance curve $\xi_0(\cdot) > 0$ is approximated in our numerical experiments in Sections 5.1.1 and 5.2.1 simply by a constant value, and in Sections 5.1.2, and 5.2.2 by a piecewise constant function. We refer the reader to Horvath, Jacquier and Muguruza [37] for a general setting of rough volatility models and their numerical simulation.

The Heston model
The Heston model, appearing in our numerical experiments of 6 is described by the system

$$\begin{aligned} dS_t &= \sqrt{V_t} S_t dW_t \quad \text{for } t > 0, \quad S_0 = s_0 \\ dV_t &= a(b - V_t)dt + v\sqrt{V_t} dZ_t \quad \text{for } t > 0, \quad V_0 = v_0 \end{aligned} \tag{4}$$

with W and Z Brownian motions with correlation parameter $\rho \in [-1, 1]$, $a, b, v > 0$ and $2ab > v^2$. In our framework it is denoted by $\mathcal{M}^{Heston}(\theta)$ with $\theta = (a, b, v, \rho) \in \Theta^{Heston} \subset \mathbb{R}^4$. The Heston model is considered in our numerical experiments in Section 6. It was also considered by [8, 15] in different neural network contexts.

The Bergomi model
In the general n-factor Bergomi model, the volatility is expressed as

$$V_t = \xi_0(t)\mathcal{E}\left(\eta_i \sum_{i=1}^{n} \int_0^t \exp\left(-\kappa_i(t-s)dW_s^i\right)\right) \quad \text{for } t > 0, \quad V_0 = v_0 > 0, \tag{5}$$

where $\eta_1, \ldots, \eta_n > 0$ and $(W^1, \ldots, W^n)$ is an n-dimensional correlated Brownian motion, $\mathcal{E}(\cdot)$ the stochastic exponential [17], and $\xi_0(\cdot) > 0$ denotes the initial forward variance curve, see [10, Section 6] for details. In this work we consider the Bergomi model for $n = 1, 2$ in Section 5 and in Appendix A. Henceforth, $\mathcal{M}^{1FBergomi}(\xi_0, \beta, \eta, \rho)$ represents the 1 Factor Bergomi model, corresponding to the following dynamics:

$$\begin{aligned} dX_t &= -\frac{1}{2}V_t dt + \sqrt{V_t} dW_t \quad \text{for } t > 0, \ X_0 = 0 \\ V_t &= \xi_0(t)\mathcal{E}\left(\eta \int_0^t \exp(-\beta(t-s))dZ_s\right) \quad \text{for } t > 0, \quad V_0 = v_0 > 0, \end{aligned} \tag{6}$$

where $\nu > 0$, and W and Z are correlated standard Brownian motions with correlation parameter $\rho \in [-1, 1]$. To fit the model parameters into our abstract model framework $\Theta^{1FBergomi} \subset \mathbb{R}^n$, for some $n \in \mathbb{N}$, the initial forward variance curve $\xi_0(\cdot) > 0$ is approximated in our numerical experiments in Sections 5.1.1 and 5.2.1 simply by a constant value, and by a piecewise constant function in Sections 5.1.2, and 5.2.2.

The SABR model
The stochastic alpha beta rho model of Hagan et al. [31, 32] is denoted in our setting as $\mathcal{M}^{SABR}(\alpha, \beta, \rho)$ and is defined as

$$\begin{aligned} dS_t &= V_t S_t^{\beta} dW_t \quad \text{for } t > 0, \quad S_0 = s_0. \\ dV_t &= \alpha V_t dZ_t \quad \text{for } t > 0, \quad V_0 = v_0 \end{aligned} \tag{7}$$

where $v_0, s_0, \alpha > 0$ and $\beta \in [0, 1]$. The SABR model is considered by McGhee in [50] in a neural network context (see also Section 2.3).

2.2 Calibration bottlenecks in volatility modelling and deep calibration

Whenever for a stochastic volatility model the numerical approximate calibration procedures (2) are computationally slow, a bottleneck in calibration time can leave the model outside of the scope of applicability for industrial production irrespective of other desirable features the model might have. This is the case in particular for the family rough volatility models, where the rough fractional Brownian motion in the volatility dynamics rules out usual Markovian pricing methods such as finite differences. So far this calibration bottleneck for has been a major limiting factor for the class of rough volatility models, whose overwhelming modelling advantages have been explored and highlighted in rapidly expanding number of academic articles [2, 22, 24, 6, 5, 7, 9, 18, 24, 39, 36, 40, 1] in the past years. Other examples include models with delicate degeneracies (such as the SABR model around zero forward) which for a precise computation of arbitrage-free prices require time consuming numerical pricing methods such as Finite Element Methods [38], Monte Carlo [12, 47] or the evaluation of multiple integrals [3].

In fact, model calibration in a broad sense also covers the training of deep neural networks: fitting (calibrating) a number of parameters (nodes) of a model (network) to a given dataset $\mathcal{P}^{MKT}(\zeta)$. Indeed, it is natural to explore deep learning techniques for calibration. By doing so one can potentially enhance calibration in comparison to standard methods: By training (calibrating) a neural network to some version of the mapping (1) one may be able to bypass calibration bottleneck posed by traditional numerical pricing methods, or perhaps bypass traditional stochastic models altogether.
The task we address here is to identify the best modelling architectures in doing so, where we manoeuvre between objectives of (i) a robust calibration method that performs well on out of sample data, (ii) fast and accurate (on-line) calibration, and (iii) interpretability of parameters for regulators and risk management purposes.

Calibration to market data directly: Arguably, learning the map (2) from available price data to the calibrated parameters of a model (either a neural network directly or a traditional stochastic model) is the most direct link to the data. While some see the distant future of quantitative finance in the second approach (calibrating a neural network to data directly and fully

bypassing traditional models) this approach leaves the meaning of calibrated network parameters unexplained, not to mention the ambiguity about the choice of the number of network parameters and network design. This can cause major challenges towards today's regulatory requirements. A second, more model based approach was proposed in the pioneering work of Hernandez [30], followed by several other authors as Stone[59], Dimitroff, Röder and Fries[15] and many others. A main characteristic of the neural network proposed by [30] is that option price approximation and parameter calibration are done in one step within the same network. We include a brief reminder of this approach and its advantages and drawbacks in Section 4.1. In our work, we advocate a seperation of these two tasks.

Two Step Approach (i) Learn a model and (ii) Calibrate to data: The two step approach is somewhere mid-way between a sole reliance on traditional pricing methods (Monte Carlo, Finite Elements, Finite Differences, Fourier Methods, Asymptotic Methods etc.) and the direct approach described above that calibrate directly to the price data. Here, one separates the calibration procedure described in (2) (resp. (2)) into two parts: **(i)** We first learn (approximate) the pricing map by a neural network that maps parameters of a stochastic model to pricing functions (or implied volatilities, cf. sections (2.1 and 4.2) and we store this map during an off-line training procedure. In a second step **(ii)** we calibrate (on-line) the now deterministic approximative learned price map, which speeds up the on-line calibration by orders of magnitude. To formalise the two step approach, we write for a payoff ζ and a model $\mathcal{M}$ with parametes $\theta \in \Theta$

$$\textbf{(i) Learn: } \widetilde{F}(\Theta,\zeta) = \widetilde{P}(\mathcal{M}(\Theta,\zeta)) \qquad \textbf{(ii) Calibrate: } \hat{\theta} = \operatorname*{argmin}_{\theta\in\Theta} \delta(\widetilde{F}(\theta,\zeta), \mathcal{P}^{MKT}(\zeta)). \tag{8}$$

Note that in part **(ii)** of (8) we essentially replaced $\widetilde{P}(\mathcal{M}(\Theta,\zeta))$ in equation (2) by its learned (deterministic) couterpart $\widetilde{F}(\Theta,\zeta)$ (which will be a Neural Network see Section 4.3) from **(i)**. Therefore, this second calibration is–by its deterministic nature–considerably faster than calibration of all those traditional stochastic models, which involve numerical simulation of the expected payoff $P(\mathcal{M}(\theta,\zeta)) = \mathbb{E}[\zeta(X(\theta))]$ for some underlying stochastic process X^θ. The first part **(i)** in (8) denotes an approximation of the pricing map through a neural network, which is calibrated in a supervised training procedure using the original (possibly slow) numerical pricing maps for training (see sections 4.3 and 5 for details in specific examples).

In the following sections we elaborate on the objectives and advantages of this two step calibration approach and present examples of neural network architectures, precise numerical recipes and training procedures to apply the two step calibration approach to a family of stochastic volatility models. We also present some numerical experiments (corresponding codes are available on GitHub: NN-StochVol-Calibrations) and report on learning errors and on calibration times.

2.3 Challenges in neural network approximations of pricing functionals

In general problem (1) (resp. (2)) is solved using suitable numerical optimisation techniques such as gradient descent [28], specific methods for certain metrics (such as Lavenberg-Marquadnt [48] for L^2), neural networks, or tailor-made methods to the complexity of the optimisation problem and objective function at hand[1]. But irrespective of their level of sophistication all optimisers for calibration share a common property: repeated (iterative) evaluation of the pricing map $\theta \mapsto P(\mathcal{M}(\theta),\zeta)$

[1]For details and an overview on calibration methods see [28].

(resp. an approximation $\widetilde{P}$ thereof) on each instance θ of consecutive parameter combinations until a sufficiently small distance $\delta(\widetilde{P}(\mathcal{M}(\theta),\zeta),\mathcal{P}^{MKT}(\zeta)$ between model prices and observed prices is achieved. Consequently, the pricing map is arguably the computational cornerstone of a calibration algorithm. Main differences between specific calibration algorithms effectively lie in the way the specific choice of evaluated parameter combinations $\{\theta_1,\theta_2\ldots\}$ are determined, which hence determines the total number N of functional evaluations of the pricing function $\big(P(\mathcal{M}(\theta_i),\zeta)\big)_{i=1\ldots N}$ used in the calibration until the desired precision $\delta(\widetilde{P}(\mathcal{M}(\hat{\theta}),\zeta),\mathcal{P}^{MKT}(\zeta))$ is achieved. In case the pricing map

$$\begin{aligned} P(\mathcal{M}(\cdot),\zeta) \;&: \Theta \longrightarrow P(\mathcal{M}) \\ &\quad \theta \mapsto P(\mathcal{M}(\theta),\zeta) \end{aligned}$$

involved in (1) is available in closed form, and can be evaluated instantaneously, the calibration (2) is fast even if a high number N of functional evaluations is used. If the pricing map is approximated numerically, calibration time depends strongly on the time needed to generate a functional evaluation of the numerical approximation

$$\theta_i \mapsto \widetilde{P}(\mathcal{M}(\theta_i),\zeta), \qquad \theta_i \in \{\theta_1,\ldots\theta_N\} \tag{9}$$

at each iteration $i=1,\ldots,N$ of the calibration procedure. Slow functional evaluations potentially cause substantial bottlenecks in calibration time. This is where we see the most powerful use of the prowess of neural network approximation:

A neural network is constructed to replace in **(i)** of (8) the pricing map, that is to approximate (for a given financial contract ζ) the pricing map from the full set[2] of model parameters Θ of the model to the corresponding prices $P(\mathcal{M}(\theta,\zeta))$ (see Section 4.2 for details). The *first challenge* for the neural network approximator of pricing functionals is to speed up this process and enable us to obtain *faster functional evaluations* and thereby lift the bottleneck of calibration. The *second challenge* is to do so with an accuracy that remains within the error bounds of the original numerical pricing discretisation:

$$\begin{aligned} \widetilde{F} : \Theta &\longrightarrow \widetilde{P}(\mathcal{M}) \\ \theta &\mapsto \widetilde{F}(\theta,\zeta) \end{aligned} \tag{10}$$

More precisely (motivated by (2)), for any parameter combination $\theta\in\Theta$ we aim to approximate the numerical approximation $\widetilde{P}$ of the true option price P with the neural network $\widetilde{F}$ up to the same order of precision $\epsilon>0$ up to which $\widetilde{P}$ approximates P. That is, for any $\theta\in\Theta$

$$\begin{aligned} &\widetilde{F}(\theta) = \widetilde{P}(\mathcal{M}(\theta),\zeta)+\mathcal{O}(\epsilon) \quad \text{whenever} \quad \widetilde{P}(\mathcal{M}(\theta),\zeta) = P(\mathcal{M}(\theta),\zeta)+\mathcal{O}(\epsilon), \text{ so that} \\ &\widetilde{F}(\theta) = P(\mathcal{M}(\theta),\zeta)+\mathcal{O}(\epsilon). \end{aligned} \tag{11}$$

In our numerical experiments in Section 5 we demonstrate that our approximation network achieves this approximation accuracy and yields a substantial speedup in terms of functional evaluations.

[2]Note that the set $\theta_1,\ldots,\theta_N$ in (9) is extended to the full set of possible parameter combinations Θ in (10).

2.4 Advantages of applying neural networks as approximators of pricing functionals

There are several advantages of seperating the tasks of pricing and calibration. Here we list some of the most convincing resons to do so. Above all, the most appealing reason is that it allows us to build upon the knowledge we have gained about the models in the past decades, which is of crucial importance from a risk management perspective. By its very design, deep learning the *price approximation* **(i)** combined with **(ii)** deterministic calibration does not cause more headache to risk managers and regulators than the corresponding stochastic models do. Designing the training as described above demonstrates how deep learning techniques can successfully extend the toolbox of financial engineering, without making compromises on any of our objectives.

1. The interpretability of model parameters $\theta \in \Theta$ remains the same as for traditional stochastic models and decades worth of knowledge is available for risk managemet. The knowledge gathered in many years of experiance with traditional models remains useful and risk management libraries of models remain valid. The neural network is only used as a computational enhancement of models. Network parameters do not need interpretation in this appraoch, this leaves some flexibility in the network architecture (number of nodes and layers) to optimise performance (see Section 4.3). The link between the architecture (number of nodes and layers) of a neural network and the number of training samples needed for fitting the network such as the trade-off between the number of nodes of a neural network and over- and underfitting and the corresponding training- and generalisation errors is discussed in the following general Section 3.

2. The availability of training data for training the deep neural network is not an issue as it is synthetically generated by traditional numerical methods. Since the training can be done off-line, this leaves us some flexibility with time needed for training or with the number of training samples we need to generate. At the same time, if we want to use this method to price large portfolios of options, we want to keep the training time within reasonable limits even for a large number of contracts. Hence we do want to keep the number samples needed for training as low as possible and construct the training in such a way that the number of training samples we need to generate does not explode with the number of derivative contracts we consider. See Section 3 for more details.

3. This approach opens up new horizons for accurate and consistent arbitrage free pricing: By learning the pricing map from model parameters to expected payoffs we relocate the time-consuming numerical simulation and evaluation procedure into an off-line pre-preocessing part of the calibration and speed up on-line calibration time for any stochastic volatility model for which some precise numerical pricing method (for example Monte Carlo simulation) is available. This instantaneously extends the scope of models that can be used in production way beyond the scope that was thinkable with traditional technologies. With this approach, rough volatility models can be calibrated within milliseconds (see Section 5). In theory, this can be extended beyond the considered models: Whenever a consistent numerical pricer exists for a model, it can be approximated and replaced by a deep neural network that provides fast numerical evaluations of the pricing map.

3 Deep neural networks as functional approximators

3.1 A reminder of some relevant properties of neural networks

Deep feed forward[3] neural networks are the most basic deep neural networks, originally designed to approximate some function F^*, which is not available in closed form but only through sample pairs of given input data x and output data $y = F^*(x)$. In a nutshell, a feed forward network defines a mapping $y = F(x, w)$ and the training determines (calibrates) the optimal values of network parameters $\widehat{w}$ that result in the best function approximation[4] $F^*(\cdot) \approx F(\cdot, \widehat{w})$ of the unknown function $F^*(\cdot)$ for the given pairs of input and output data (x, y), cf. [28, Chapter 6].
To formalise this, we introduce some notation and recall some basic definitions and principles of function approximation via (feedforward) neural networks:

Definition 1 (Neural network). Let $L \in \mathbb{N}$ and the tuple $(N_1, N_2 \cdots, N_L) \in \mathbb{N}^L$ denote the number of layers (depth) and the number of nodes (neurons) on each layer respectively. Furthermore, we introduce the affine functions

$$\begin{aligned} w^l : \ \mathbb{R}^{N_l} &\longrightarrow \mathbb{R}^{N_{l+1}} \text{ for } 1 \leq l \leq L-1 \\ x &\mapsto A^{l+1}x + b^{l+1} \end{aligned} \tag{12}$$

acting between layers for some $A^{l+1} \in \mathbb{R}^{N_{l+1} \times N_l}$. The vector $b^{l+1} \in \mathbb{R}^{N_{l+1}}$ denotes the *bias term* and each entry $A^{l+1}_{(i,j)}$ denotes the *weight* connecting node $i \in N_l$ of layer l with node $j \in N_{l+1}$ of layer $l+1$. For the the collection of affine functions of the form (12) on each layer we fix the notation $w = (w^1, \ldots, w^L)$. We call the tuple w the *network weights* for any such collection of affine functions. Then a Neural Network $F(w, \cdot) : \mathbb{R}^{N_0} \to \mathbb{R}^{N_L}$ is defined as the composition:

$$F := F_L \circ \cdots \circ F_1 \tag{13}$$

where each component is of the form $F_l := \sigma_l \circ W^l$. The function $\sigma_l : \mathbb{R} \to \mathbb{R}$ is referred to as the *activation function*. It is typically nonlinear and applied component wise on the outputs of the affine function W^l. The first and last layers, F_1 and F_L, are the *input* and *output* layers. Layers in between, $F_2 \cdots F_{L-1}$, are called *hidden layers*.

Remark 1 (Number of network weights). The set of affine functions $W^l : \mathbb{R}^{N_l} \to \mathbb{R}^{N_{l+1}}$ for $1 \leq l \leq L-1$ is characterised by the set of parameters $w \in \mathbb{R}^{\sum_{i=1}^{L-1} N_i(1+N_{i+1})} =: \Omega$, hence the number of network weights (or neural network parameters) to calibrate is $\sum_{i=1}^{L-1} N_i(1 + N_{i+1})$.

The close-up behaviour of a single neuron is summarised in Figure 3.1 below. For any $i \in N_l$, and any input vector $x \in \mathbb{R}^{N_l}$, the output of the node i of layer l is

$$x \mapsto \sigma_l(a_i^T \ x + b_i),$$

where the input $x \in \mathbb{R}^{N_l}$ is given by the output of all nodes in the previous layer, the term $b_i^l \in \mathbb{R}$ denotes the bias term and $a_i \in \mathbb{R}^{N_l}$ the set of weights connecting node i to the next layer. Note that $a_i = A^l_{(i,\ .)}$.
The following central result of Hornik justifies the use of neural networks as approximators for multivariate functions and their derivatives.

[3] The network is called feed forward if there are no feedback connections in which outputs of the model are fed back into itself.

[4] In our case y is a 8×11-point grid on the implied volatility surface and x are model parameters $\theta \in \Theta$, for details see Section 4.

Theorem 1 (Universal approximation theorem (Hornik, Stinchcombe and White [34])). *Let $\mathcal{NN}^{\sigma}_{d_0,d_1}$ be the set of neural networks with activation function $\sigma : \mathbb{R} \mapsto \mathbb{R}$, input dimension $d_0 \in \mathbb{N}$ and output dimension $d_1 \in \mathbb{N}$. Then, if σ is continuous and non-constant, $\mathcal{NN}^{\sigma}_{d_0,1}$ is dense in $L^p(\mu)$ for all finite measures μ.*

There is a rapidly growing literature on approximation results with neural networks, see [33, 35, 51, 58] and the references therein. Among these we would like to single out one particular result:

Theorem 2 (Universal approximation theorem for derivatives (Hornik, Stinchcombe and White [35])). *Let $F^* \in \mathcal{C}^n$ and $F : \mathbb{R}^{d_0} \to \mathbb{R}$ and $\mathcal{NN}^{\sigma}_{d_0,1}$ be the set of single-layer neural networks with activation function $\sigma : \mathbb{R} \mapsto \mathbb{R}$, input dimension $d_0 \in \mathbb{N}$ and output dimension 1. Then, if the (non-constant) activation function is $\sigma \in \mathcal{C}^n(\mathbb{R})$, then $\mathcal{NN}^{\sigma}_{d_0,1}$ arbitrarily approximates f and all its derivatives up to order n.*

Remark 2. Theorem 2 highlights that the smoothness properties of the activation function are of significant importance in the approximation of derivatives of the target function F^*. In particular, to guarantee the convergence of l-th order derivatives of the target fuction, we choose an activation function $\sigma \in C^l(\mathrm{R})$. Note that the ReLu activation function , $\sigma_{ReLu}(x) = (x)^+$ is not in $\mathcal{C}^l(\mathrm{R})$ for any $l > 0$, while $\sigma_{Elu}(x) = \alpha(e^x - 1)$ is smooth.

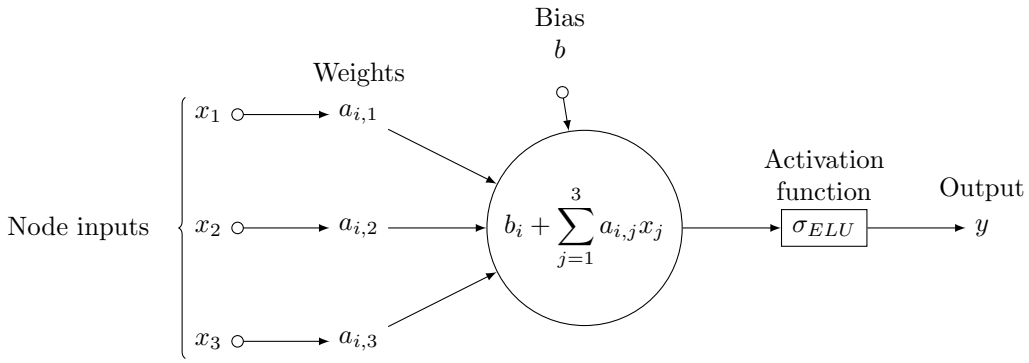

Figure 1: In detail neuron behaviour

3.2 Neural network training and the interplay between network capacity and size of the training dataset

Definition 2 (Training and test sets). Let $F^* : \mathbb{R}^{N_0} \to \mathbb{R}^{N_L}$ be a generic function. Then we say that the labelled set $X^{train} = \{x_i, F^*(x_i)\}_{i=1,\dots,M}$ is a *training set* corresponding to F^*. A part of the training set X^{test} is set aside to test the network on unseen data.

Definition 3 (Neural Network Training/Calibration). Let $F^* : \mathbb{R}^{N_0} \to \mathbb{R}^{N_L}$ be a generic function that is only available though a set of input-output pairs, and let $X^{train} = \{x_i, F^*(x_i)\}_{i=1,\dots,M}$ denote its corresponding training set. Let $F(w, \cdot) : \mathbb{R}^{N_0} \to \mathbb{R}^{N_L}$ be a neural network and $w \in \Omega$, where Ω denotes the set of parameters that characterises the network. Then, Training F is done solving

$$\hat{w} = \underset{w \in \Omega}{\operatorname{argmin}} \mathcal{L}\left(\{F(w, x_i\}_{i=1}^{M}, \{F^*(x_i)\}_{i=1}^{M}\right) \tag{14}$$

for a given training set X^{train} and where $\mathcal{L}$ is a loss function, corresponding to an objective function of our choice.

The choice of the objective function plays a particularly important role for the training and performance of the network. Here, we elaborate on this briefly and give full details in Section 4.3.2.

Remark 3. In our experiments in Section 4.3 we use the mean squared error (denoted as L^2 error) some approximation of the implied volatlity surface as objective function. The associated optimisation problem is given by:

$$\hat{w} = \underset{w\in\Omega}{\operatorname{argmin}} \frac{1}{M}\sum_{i=1}^{M} \left(F(w,x_i) - F^*(x_i)\right)^2 \tag{15}$$

Clearly, we have the objective of approximating $F^* \approx P^{\mathcal{M}}(\theta,\zeta)$ for a given model $\mathcal{M}$ and payoff characterised by ζ. Though this theory can be set up for general payoffs and pricing functionals $\theta \to P^{\mathcal{M}}(\theta,\zeta)$, we will focus in our experiments on (some well-chosen) approximation of the implied volatilities[5] ($F^* = \widetilde{\sigma}_{BS}(\theta)$). More precisely, in the calibration we seek to solve the minimisation calibration problem

$$\hat{\theta} := \underset{\theta\in\Theta}{\operatorname{argmin}}\, d(\Sigma_{BS}^{\mathcal{M}(\theta)}, \Sigma_{BS}^{MKT}) \tag{16}$$

for some metric $d : \mathbb{R}^{n\times m}\times\mathbb{R}^{n\times m}\to\mathbb{R}^+$, where $\Sigma_{BS}^{\mathcal{M}(\theta)} := \{\sigma_{BS}^{\mathcal{M}(\theta)}(k_i,T_j)\}_{i=1,..,n,\ j=1,...,m}$ denotes a (finite) set of values on the implied volatility surface generated by the chosen approximative model pricing function[6] $P(\mathcal{M}(\theta),k,T)$ for a given stochastic model $\mathcal{M}(\Theta)$ with parameters $\theta\in\Theta$, and where $\Sigma_{BS}^{MKT} := \{\sigma_{BS}^{MKT}(k_i,T_j)\}_{i=1,..,n,\ j=1,...,m}$ denotes the corresponding market implied volatilities. A natural approach to this problem is Gradient Descent:

Definition 4 (Gradient Descent Algorithm). A Gradient Descent algorithm (GD algorithm) associated to the objective function $\mathcal{L}$, corresponds to the iterative update rule

$$w_0\in\Omega \quad \text{and} \quad w_n = w_{n-1} - \alpha(\nabla^w\mathcal{L})\left(\{F(w,x_i\}_{i=1}^M, \{F^*(x_i)\}_{i=1}^M\right) \quad \alpha>0. \tag{17}$$

The standard practice (see [20] or [28]) is to perform Stochastic Gradient Descent. The advantage of Stochastic Gradient Descent algorithms over regular Gradient Descent is (at least) threefold; First, one obtains faster iterations by reducing the number of data points used to compute the gradiend ∇^w. Second, by randomly sampling from the training set one prevents overfitting to singular data points. Third, one bypasses local minima through random sampling.

Definition 5 ((Mini-)Batch size). Let X^{train} denote a given set training data in form of input-output pairs $\{x,F^*(x)\}$. We call the subset $\{x_j,F^*(x_j)\}_{j\in\mathcal{U}\{1,...,m\}} := X_{i,m}^{\text{batch}} \subseteq X^{train}$ the i-th Batch of the training dataset, where $\mathcal{U}\{0,1,...,n\}$ is the discrete uniform distribution. Then, m is termed as the (mini-)Batch size.

Definition 6 (Stochastic Gradient Descent Algorithm). A stochastic gradient descent algorithm (SGD algorithm) with Batch size m associated to the objective function $\mathcal{L}$ corresponds to the

[5] To set the notation for this, recall $BS(\sigma_{BS}(k,T),S_0,k,T) = P(k,T)$. where BS denotes the Black-Scholes Call pricing function $BS(\sigma,S_0,k,T) := S_0\mathcal{N}(d_+) - K\mathcal{N}(d_-), \quad d_\pm := \frac{\log(S_0)-k}{\sqrt{T}\sigma} \pm \frac{\sqrt{T}\sigma}{2}$, in terms of log-strike k, initial spot $S_0>0$, maturity T and volatility $\sigma>0$ for the Gaussian cumulative distribution function $\mathcal{N}(\cdot)$.

[6] Here we mean a numerical or asymptotic approximation of the option price or implied volatility.

iterative update rule

$$w_0 \in \Omega \quad \text{and} \quad w_n = w_{n-1} - \alpha(\nabla^w \mathcal{L}) \left(F(w_{n-1}, X^{\text{batch}}_{n,m}), F^*(X^{\text{batch}}_{n,m})\right) \quad \alpha > 0 \tag{18}$$

where $F^*(X^{\text{batch}}_{n,m})$ (respectively F) is defined as $\{F^*\left((X^{\text{batch}}_{n,m})_i\right)\}_{i=1}^m$ for notation convenience.

The choice of batch sizes in the gradient descent is a part of the choice of the training design. Small batch sizes make for noisier gradient estimates and may reduce the accuracy of the parameter update. Very large batch sizes increase computation time because they require holding in memory and computing a larger number of gradients at a time, though they should reduce the variance of the gradient estimate. The full process of running through all batches by the optimization algorithm is an epoch [28]. A common feature in the training of deep learning models is the U-shape of the validation error as a function of epochs, as reported in Goodfellow, Bengio and Courville [28]. Indeed, as the number of epochs grows too large, the model risks overfitting the data because identical inputs will have appeared many times. On the other hand, too few epochs do not allow the algorithm to run through sufficient iterations and there will still be room to decrease the error in the end of the training.

In training a neural network, we have two goals in mind:

1. To keep the training error (the $\mathcal{L}$ error in (14) on the training set) X^{train} as small as possible.
2. To keep the generalisation error of the calibrated network $F(\hat{w}, \cdot)$ on unseen data (i.e. $F(\hat{w}, x) \approx F^*(x)$ for $x \in X^{test}$) is as small as possible.

These goals corresponds to the two central challenges; 1) underfitting and 2) overfitting. In this work our objective is to determine the appropriate architecture (number of layers and nodes of the neural network). This creates a dichotomy, between model complexity and approximation error which has been termed in the literature as Bias-Variance tradeoff (see [20] for instance). This tradeoff can be in some cases controlled by the so-called *capacity* of the model, that is, the model's ability to fit a wide variety of functions. In some relevant examples model capacity is related to the so-called *Vapnik-Chervonenkis dimension* (VC-dimension) [61]. The concept of VC-dimension may be extended to more general situations (see Vapnik [61]), but in specific cases it is hard to control. Nevertheless one rule of thumb is that the number of parameters of the network is directly proportional to the capacity (flexibility of the model). Hence it is inversely proportional to the training error (the more nodes the better the fit to training data) and directly proportional to the generalisation error (the more nodes the worse it generalises). Also, network capacity and the number of nodes is directly proportional to the data needed to train the network (it is desirable to have at least one data point per network weight). Motivated by this, we aim to keep the number of nodes in network as small as possible.

The following Theorem provides theoretical bounds for the above rule of thumb and establishes a connection between the number of nodes in a network and the number of training samples needed to train it.

Theorem 3 (Estimation bounds for Neural Networks (Barron [4]))**.** *Let $\mathcal{NN}^{\sigma}_{d_0,d_1}$ be the set of single-layer neural networks with Sigmoid activation function $\sigma(x) = \frac{e^x}{e^x+1}$, input dimension $d_0 \in \mathbb{N}$ and output dimension $d_1 \in \mathbb{N}$. Then:*

$$\mathbb{E}\|F^* - \hat{F}\|_2^2 \leq \mathcal{O}\left(\frac{C_f^2}{n}\right) + \mathcal{O}\left(\frac{nd_0}{N}\log N\right)$$

where n is the number of nodes, N is the training set size and C_{F^*} is the first absolut moment of the Fourier magnitude distribution of F^*.

Remark 4. Barron's [4] insightful result gives a rather explicit decomposition of the error in terms of bias (model complexity) and variance:

- $\mathcal{O}\left(\frac{C_{F^*}^2}{n}\right)$ represents the model complexity, i.e. the larger n (number of nodes) the smaller the error
- $\mathcal{O}\left(\frac{nd_0}{N}\log N\right)$ represents the variance, i.e. a large n must be compensated with a large training set N in order to avoid overfitting.

A clear lower bound on the number of samples needed is given by the number of neural network parameters to train.

Finally, we motivate the use of multi layer networks and the choice of network depth. Even though a single layer might theoretically suffice to arbitrarily approximate any continuous function,in practice the use of multiple layers dramatically improves the approximation capacities of network. We informally recall the following Theorem due to Eldan and Shamir [16] and refer the reader to the original paper for details.

Theorem 4 (Power of depth of Neural Networks (Eldan and Shamir [16]))**.** *There exists a simple (approximately radial) function on $\mathbb{R}^d$, expressible by a small 3-layer feedforward neural networks, which cannot be approximated by any 2-layer network, to more than a certain constant accuracy, unless its width is exponential in the dimension.*

Remark 5. In spite of the specific framework by Eldan and Shamir [16] being restrictive, it provides a theoretical justification to the power of "deep" neural networks (multiple layers) against "flatter" networks (i.e. few layers) as in [50] with a larger number of neurons. On the other hand, multiple findigs indicate [8, 42] that adding hidden layers beyond 4 hidden layers does not significantly improve network performance.

4 Pricing and calibration with neural networks: Optimising network and training

In this section we compare different objective functions (direct calibration to data to an image-based implicit learning approach) and motivate our choice of image-based objective function. We give details about network architectures for the approximation network and compare different optimisers for the calibration step.

4.1 One-step appraoach: Deep calibration by the inverse map

In his pioneering work [30] Hernandez proposes to consider the inverse map

$$\mathcal{P}^{-1}(\Sigma_{BS}^{MKT}, \{k_i\}_{i=1}^n, \{T_j\}_{j=1}^m) \to \hat{\theta} \tag{19}$$

that performs the calibration task directly via Neural Networks. In spite of the promising results by Hernandez [30] the main drawback of this approach, as Hernandez observes, is the lack of control

on the function $\mathcal{P}^{-1}$. Furthermore, from a risk management prespective one has no guarantee how well the learned mapping of $\mathcal{P}^{-1}$ will solve the original problem (16), when exposed to unseen data. In fact, this is the behaviour observed in Hernandez [30], since the out of sample performance tends to differ from the in sample one, suggesting a not fully satisfactory generalisation of the learned map. We recover the same behaviour of the inverse map in our own experiments, which we included in Appendix A.

4.2 Two-step approach: Pointwise training, image-based and implicit training and the role of the objective funciton

A different approach is to first approximate the pricing map via neural network F,

$$\widetilde{P}(\mathcal{M}(\theta), \zeta) \approx F(\theta, \zeta, w)$$

and then solve the calibration problem 2 in a second step. We present multiple approaches to tackle this task, the *Image-based implicit learning* being our method of choice for our numerical experiments in Section 5.

Remark 6. By design, this approach overcomes all problems encountered in Section 4.1. Notice that, when the model is exposed to a new out-of-sample environment the solution will still solve the original problem 2 as log as the approximation F is accurate. This approach then, is by construction consistent with (2).

Pointwise learning

In one way or another the uderlying principle of this approach appears in several related contributions Bayer and Stemper [8], De Spiegeleer, Madan, Reyners and Schoutens[14], Ferguson and Green [21] and McGhee [50] and its advantage is the ability to asses the quality of $\hat{P}$ using Monte Carlo or PDE methods, and indeed it is superior training in terms of robustness. More precisely, in the implied volatility problem one needs to:

1. Learn the map $F^*(\theta, T, k) = \sigma_{BS}^{\mathcal{M}(\theta)}(T, k)$ via neural network $\widetilde{F}(\theta, T, k) := F(\theta, \hat{z}, T, k)$ where

$$\begin{aligned} F^* : \Theta \times [0, T_{max}] \times [k_{min}, k_{max}] &\longrightarrow \mathbb{R} \\ (\theta, T, k) &\mapsto F^*(\theta, T, k) \end{aligned} \tag{20}$$

   where

$$\hat{z} = \underset{z \in \mathbb{R}^n}{\operatorname{argmin}} \sum_{u=1}^{N_{Train}} \left(F(\theta_u, z, T_u, k_u) - F^*(\theta_u, T_u, k_u)\right)^2 \tag{21}$$

   where $\theta_u \in \Theta,\ T_u \in [0, T_{max}]$ and $k_u \in [k_{min}, k_{max}]$ for all $u = 1, ..., N_{Train}$.

2. Solve

$$\hat{\theta} := \underset{\theta \in \Theta}{\operatorname{argmin}} \sum_{i=1}^{n} \sum_{j=1}^{m} (\widetilde{F}(\theta, T_i, k_j) - \sigma_{BS}^{MKT}(k_i, T_j))^2.$$

More explicitly, the second step corresponding to the rough Bergomi model is given by

$$\hat{\theta} = (\hat{\xi_0}, \hat{\nu}, \hat{\rho}, \hat{H}) = \underset{(\xi_0, \nu, \rho, H) \in \Theta}{\operatorname{argmin}} \sum_{i=1}^{n} \sum_{j=1}^{m} (\widetilde{F}(\xi_0, \nu, \rho, H, T_i, k_j) - \sigma_{BS}^{MKT}(k_i, T_j))^2.$$

This approach is much in line with the spirit of research and today's options pricing, where one (arbitrarily chosen) point indexed by (k, T) is of interest at a time. However, it requires a large network and consequently a large number of training sets is needed. For instance, Bayer and Stemper report $4(4096^2) \approx 67.000.000$ parameters to perform the infinite dimensional optimisation task (see Bayer and Stemper [8]) with the associated computational challenge. This number can be reduced by adjusting the objective function in (21) as McGhee does in [50] for the SABR model 7. There, the inputs are $(\theta^{SABR}, T, k_1, \ldots, k_{10})$ and there are ten volatility outputs $\sigma_1, \ldots, \sigma_{10}$ per maturity T.

$$\begin{aligned} F^* : \Theta \times [0, T_{max}] \times [k_{min}, k_{max}]^{10} &\longrightarrow \mathbb{R}^{10} \\ (\theta, T, k_1, ..., k_{10}) &\mapsto F^*(\theta, T, k_1, ..., k_{10}). \end{aligned} \tag{22}$$

Though the array of strikes $(k_1, ..., k_{10})$ is denoted in [50] as input vector, it is in fact computed implicitly: In [50] the strike range is sampled dynamically as a function of maturity T and SABR parameters θ^{SABR}. This approach by McGhee [50], is built on a single layer neural network ranging from 250 to 1000 nodes, yielding roughly 6.000 to 24.000 parameters to be calibrated. Note in particular that in (22) the dimensionality of the output has increased to 10 compared to (20)

Image-based implicit learning

We take this idea further and design an implicit form of the pricing map that is based on storing the implied volatility surface as an image given by a grid of "pixels". This image-based representation has a formative contribution in the performance of the network we present in Section 5. We present our contribution here; Let us denote by $\Delta := \{k_i, T_j\}_{i=1,\, j=1}^{n,\, m}$ a fixed grid of strikes and maturities, then we propose the following two step approach:

1. Learn the map $F^*(\theta) = \{\sigma_{BS}^{\mathcal{M}(\theta)}(T_i, k_j)\}_{i=1,\, j=1}^{n,\, m}$ via neural network $\widetilde{F}(\theta) := F(\theta, \hat{w})$ where

$$\begin{aligned} F^* : \Theta &\longrightarrow \mathbb{R}^{n \times m} \\ \theta &\mapsto F^*(\theta) \end{aligned} \tag{23}$$

   where the input is a parameter combination $\theta \in \Theta$ of the stochastic model $\mathcal{M}(\Theta)$ and the output is a $n \times m$ grid on the implied volatility surface $\{\sigma_{BS}^{\mathcal{M}(\theta)}(T_i, k_j)\}_{i=1,\, j=1}^{n,\, m}$ where $n, m \in \mathbb{N}$ are chosen appropriately (see Section 4.3). Then,

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

2. Solve

$$\hat{\theta} := \underset{\theta \in \Theta}{\operatorname{argmin}} \sum_{i=1}^{n} \sum_{j=1}^{m} (\widetilde{F}(\theta)_{ij} - \sigma_{BS}^{MKT}(T_i, k_j))^2.$$

Remark 7. Notice that $\hat{w}(\Delta)$ depends on Δ implicitly, consequently so does $\widetilde{F}(\theta) = F(\theta, \hat{w}(\Delta))$ (hence the name implicit learning). This setting is similar to that of image recognition and exploits the structure of the data to reduce the complexity of the Network (see Section 5 for details).

Remark 8. In our experiments we chose $n = 8$ and $m = 11$. At first, a criticism of mapping (23) might be the inability to extrapolate/interpolate between maturities/strikes outside the grid Δ. However, one is free to choose the grids Δ as fine as needed. In addition, one may use standard (arbitrage free) uni/bi-variate splines techniques to extrapolate/interpolate across strikes and maturities, as with traditional market data observable only at discrete points.

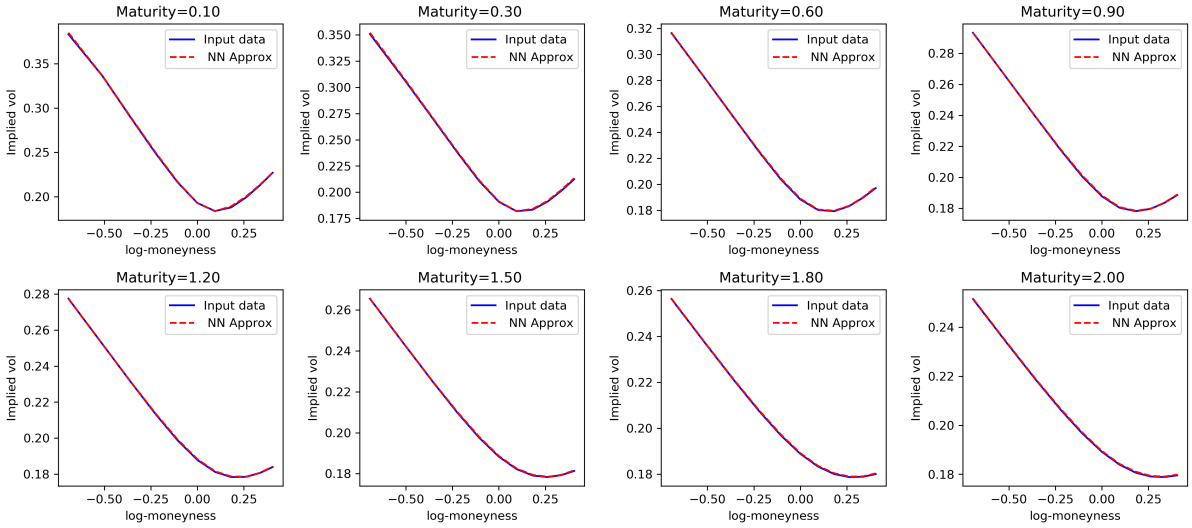

Figure 2: Volatility surface generated by the neural network approximator and the corresponding original counterpart on a grid given by 8 maturities and 11 strikes.

4.2.1 Advantages of the image-based implicit training

By evaluating the values of implied volatility surface along 8×11 gridpoints with 40.000 (80.000 for non-constant forward variances) different parameter combinations we effectively evaluate the "fit" of the surface to numerically generated ones across the same number of points. By moving the evaluation of the implied volatilities into the objective function we improve the learning in many aspects:

- The first advantage of implicit training is that it does exploit the structure of the data. Already in (22) updates in neighbouring volatility points σ_{n-1} and σ_n can be incorporated in the learning process. If the output is a full grid as in (23) this effect is further enhanced.
- A further advantage of the image based implicit training is, that by evaluating the objective function on a larger set of (grid) points, injectivity of the mapping can be more easily guaranteed than in the pointwise training: Two distinct parameter combinations are less likely to yield the same value across a set of gridpoints, then if evaluated only on a single point.
- The number of training samples only amounts to the number of different parameter combinations we sample and not to the points sampled on the implied volatility surface.

- The inifinite dimensional optimisation problem of learning the shape (of the implied volatility surface for every possible parameter combination) becomes a finite dimesnional one: While for each parameter combination $\theta \in \Theta$ the objective in [8] is that at any arbitrary point $(k,t) \in [k_{min}, k_{max}] \times [0,T]$ the (squared) error between the neural network approximation and the numerical approximation of the implied volatility remians small $(\widetilde{F}(\theta,k,t) - F^*(\theta,k,t))^2$, in our objective funtion we only require this along a fixed grid $\{\sigma_{BS}^{\mathcal{M}(\theta)}(T_i,k_j)\}_{i=1,\,j=1}^{8,\;\;11}$.

- We do not need sophisticated sampling methods on the volatility surface (as in BS) nor do we need to make sure to sample more points on the implied volatility surface in the more liquidly traded regions. By sampling always the same gridpoints on the implied volatility surface we speed up the training of the neural network: updates of the network on each gridpoint also imply additional information for updates of the network on neighbouring gridpoints. One can say that we regard the implied volatility surface as an image with a given number of pixels.

- In this training we store the generated 60.000 sample paths for the training data and use them to evaluate multiple maturities (here 8) and strikes (here 11) at the same time. Therefore we can easily add and evaluate additional maturities and strikes to the same set of paths and do not limit ourselves to one specific grid on the implied volatility surface. Note in particular that in this training design we can refine the grid on the implied volatility surface without increasing the number of training samples needed and without significantly increasing the computational time for training as the portfolio of vanilla options on the same underlying grows with different strikes and maturities.

4.3 Network architecture and training

Motivated by the above analysis, we choose to set up the calibration in the implicit two-step approach. This involves a separation of the calibration procedure into (i) "Deep approximation" an approximation network with an implicit training and (ii) "Calibration" a calibration layer on top. We first start by describing the approximation network in the implicit image-based training and discuss the calibration in Section 4.4 below. Im addition, we will highlight specific techniques that contribute to the robustness and efficiency of our design.

4.3.1 Network architecture of the implied volatility map approximation

Here we motivate our choice of network architecture for the following numerical experiments which were inspired by the analysis in the previous sections. Our network architecture is summarised in the graph 4.3.1 below.

1. A fully connected feed forward neural network with 3 hidden layes (due to Theorem 4) and 30 nodes on each layers (see Figure 4.3.1 for a detailed representation)

2. Input dimension = n, number of model parameters

3. Output dimension = 11 strikes$\times$ 8 maturities for this experiment, but this choice of grid can be enriched or modified.

4. The three inner layers have 30 nodes each, which adding the corresponding biases results on a number
$$(n+1) \times 30 + 3 \times (1+30) \times 30 + (30+1) \times 88 = 30n + 5548$$

of network parameters to calibrate (see Section 3 for details).

5. Motivated by Threorem 2 we choose the Elu $\sigma_{Elu} = \alpha(e^x - 1)$ activation function for the network.

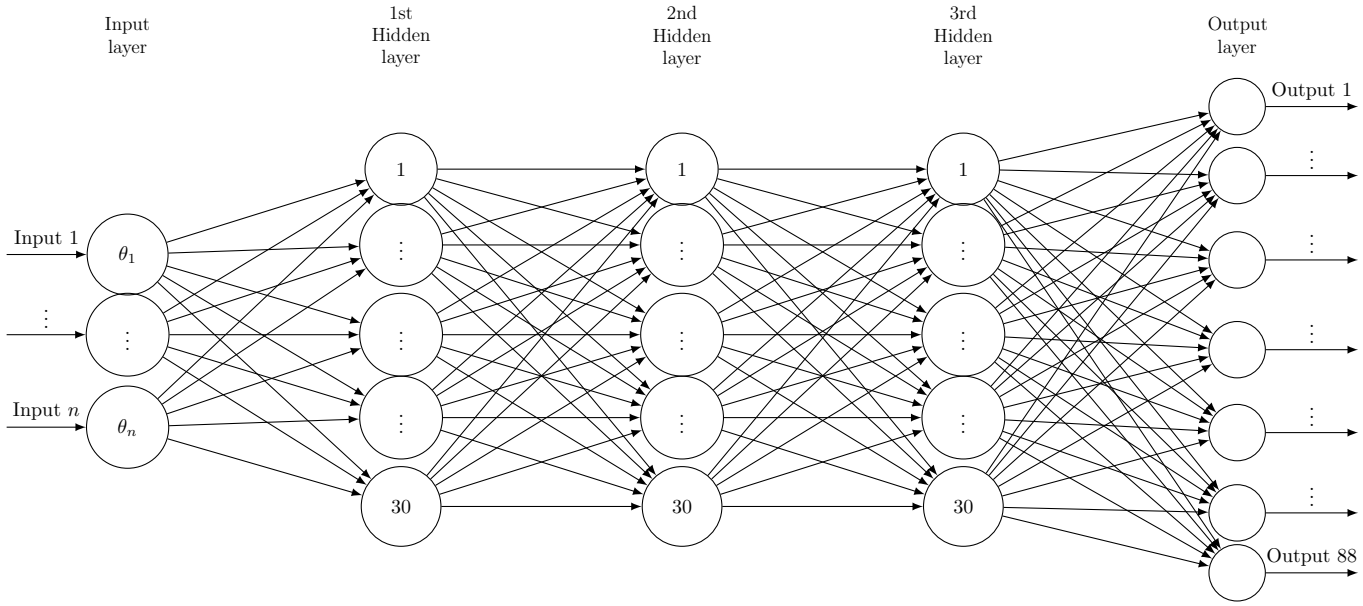

Figure 3: Our neural network architecture with 3 hidden layers and 30 neurons on each hidden layer, with the model parameters of the respective model on the input layer and with the 8×11 implied volatility grid on the output layer.

4.3.2 Training of the approximation network

We train the neural network using gradient descent, the so-called 'Adam' minibatch training scheme due to Kingman and Ba [43], which is a version of the Stochastic Gradient Descent algorithm (introduced in Definition 18). Given parameters $0 \leq \beta_1, \beta_2 < 1, \epsilon, \alpha$, initial iterates $u_0 := 0, v_0 := 0, w_0 \in \Omega$, the Adam scheme has the following iterates:

$$\begin{aligned} g_n &:= \nabla^w \sum_{i=1}^{m} \mathcal{L}\left(F(w_{n-1}, X_{n,m}^{\text{batch}}), F^*(X_{n,m}^{\text{batch}})\right) \\ u_{n+1} &:= \beta_1 u_n + (1-\beta_1) g_n \\ v_{n+1} &:= \beta_2 v_n + (1-\beta_2) g_n^2 \\ w_{n+1} &:= w_n - \alpha \frac{u_{n+1}}{1-\beta_1^{n+1}} \frac{1}{\sqrt{v_n/(1-\beta_2^{n+1})} + \epsilon}. \end{aligned}$$

We follow the common features of optimization techniques and choose mini-batches, as described in Goodfellow, Bengio and Courville [28]. Typical batch size values range from around 10 to 100. In our case we started with small batch sizes and increased the batch size until training performance consistently reached a plateau. Finally, we chose batch sizes of 32, as performance is similar for batch sizes above this level, and larger batch sizes increase computation time by computing a larger number of gradients at a time.

In our training design, we use a number of regularisation techniques to speed up convergence of the training, to avoid overfitting and improve the network performance.

1) Early stopping: The choice of the number of 'epochs', i.e. the number of times the full process of 'the optimization algorithm running through all batches' can strongly influence the performance of the network [28]. Indeed, as the number of epochs grows too large, the model risks overfitting the data because identical inputs will have appeared many times. We choose the number of epochs as 200 and stop updating network parameters if the error has not improved in the test set for 25 steps.

2) Normalisation of model parameters: Usually, model parameters are restricted to a given domain i.e. $\theta \in [\theta_{min}, \theta_{max}]$. Then, we perform the following normalisation transform:

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

This, facilitates the Neural Network's learning process, since there is no need to comprehend the particular magnitude and domain of each parameter.

3) Normalisation of implied volatilities: The normalisation of implied volatilities is a more delicate matter, since $\sigma_{BS}(T, k, \theta^{train}) \in [0, \infty)$, for each T and k. Therefore, we choose to normalise the surface subtracting the sample empirical mean $\bar{\sigma}_{BS}^{Train}(T, k) := \frac{1}{|\Theta^{train}|} \sum_{\theta \in \Theta^{train}} \bar{\sigma}_{BS}(T, k, \theta^{train})$ where the average is taken over all samples $\theta \in \Theta^{train}$ in the training dataset, and dividing by the sample standard deviation $\text{std}^{train}(\sigma_{BS}(T, k))$ (defined analogously) of the training set i.e.

$$\frac{\sigma_{BS}(T, k, \theta) - \bar{\sigma}_{BS}^{train}(T, k)}{\text{std}(\sigma_{BS}(T, k, \theta^{train})}.$$

Even though the normalised variable is not restricted in a domain as in the previous case, this also markedly helps to reduce complexity of the data.

4.4 The calibration step

Once the pricing map approximator $\widetilde{F}$ for the implied volatility is found, only the calibration step in (2) is left to solve. In general, for financial models the pricing map F^* is assumed to be smooth (at least C^1 differentiable) with respect to all its input parameters θ. However, $\widetilde{F}$ is not guaranteed to be C^1 and two different optimization approaches are considered.

Gradient-based optimizers

A standard necessary first order condition for optimality in 2 is that

$$\nabla^{\theta}\delta\left(\widetilde{F}(\mathcal{M}(\theta),\zeta),\mathcal{P}^{MKT}(\zeta)\right)=0, \tag{24}$$

provided that the objective function is smooth. Then, a natural update rule is to move along the gradient via Gradient Descent i.e.

$$\theta_{i+1}=\theta_i-\lambda\nabla^{\theta}\delta\left(\widetilde{F}(\mathcal{M}(\theta_i),\zeta),\mathcal{P}^{MKT}(\zeta)\right),\quad \lambda>0. \tag{25}$$

A common feature of gradient based optimization methods building on (25) is the use of the gradient $\nabla^{\theta}\delta\left(\widetilde{F}(\mathcal{M}(\theta),\zeta),\mathcal{P}^{MKT}(\zeta)\right)$, hence its correct and precise computation is crucial for subsequent success. Examples of such algorithms, are Levenberg-Marquardt [48, 49], Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm [53], L-BFGS-B [62] and SLSQP [45]. The main advantage of the aforementioned methods is the quick convergence towards condition (24). However, (24) only gives necessary and not sufficient conditions for optimality, hence special care must be taken with non-convex problems.

Remark 9. Notably, making use of Theorem 2 we use a smooth activation functions in order to guarantee $\nabla^{\theta}\widetilde{P}\approx\nabla^{\theta}\widetilde{F}$

Gradient-free optimizers

Gradient-free optimization algorithms are gaining popularity due to the increasing number of high dimensional nonlinear, non-differentiable and/or non-convex problems flourishing in many scientific fields such as biology, physics or engineering. As the name suggests, gradient-free algorithms make no C^1 assumption on the objective function. Perhaps, the most well known example is the Simplex based Nelder-Mead [52] algorithm. However, there are many other methods such as COBYLA [55] or Differential Evolution [60] and we refer the reader to [56] for an excellent review on gradient-free methods. The main advantage of these methods is the ability to find global solutions in 2 regardless of the objective function. In contrast, the main drawback is a higher computational cost compared to gradient methods.

To conclude, we summarise the advantages of each approach in Table 1.

	Gradient-based	**Gradient-free**
Convergence Speed	Very Fast	Slow
Global Solution	Depends on problem	Always
Smooth activation function needed	Yes to apply Theorem 2	No
Accurate gradient approximation needed	Yes	No

Table 1: Comparison of Gradient vs. Gradient-free methods.

5 Numerical experiments

In our numerical experiments we demonstrate that the accuracy of the approximation network indeed remains within the accuracy of the Monte Carlo error bounds and proclaimed in the introductory sections' objectives. For this we first compute the benchmark Monte Carlo errors in Figures 4-5 and compare this with the neural network approximation errors in Figures 6, 7, 8 and Figure 9. For this seperation into steps (i) and (ii) to be computationally meaningful, the neural network approximation has to be a reasonably accurate approximation of the true pricing functionals and each functional evaluation (i.e. evaluation an option price for a given price and maturity) should have a considerable speed-up in comparison to the original numerical method. In this section we demonstrate that our network achieves both of these goals.

5.1 Numerical accuracy and speed of the price approximation network

As mentioned in Section 4.2 one crucial improvement win in comparison with direct neural network approaches, as pioneered by Hernandez [30], is the separation of (i) the implied volatility approximation function, mapping from parameters of the stochastic volatility model to the implied volatility surface–thereby bypassing the need for expensive Monte-Carlo simulations—and (ii) the calibration procedure, which (after this separation) becomes a simple deterministic optimisation problem.

1. Approximation accuracy: here we compare the error of the approximation network error to the error of Monte Carlo evaluations. We compute Monte Carlo prices with 60.000 paths as reference at the nodes where we compute the implied volatility grid using Algorithm 3.5 in Horvath, Jacquier and Muguruza [37]. In Figures 4 and5 the approximation accuracy of the Monte Carlo method for the full implied volatility surface is computed using pointwise relative error with respect to the 95% Monte Carlo confidence interval. Figures 6, 7, 8 and 9 demonstrate that we indeed achieved our aim of the the same approximation accuracy for the neural network as for the Monte Carlo approximation (i.e. within a few basis points) that we announced in Section 2.4. For reference, the spread on options is around 0.2% in implied volatility terms for the most liquid and those below a year. This translates into 1% relative error for a implied volatility of 20%.

2. Approximation speed: Table 2 shows the CPU computation time for functional evaluation of a full surface under two different models; rBergomi 3 and 1 Factor Bergomi 6. In the first line of Table 2 we take the forward variance ξ_0 (see Section 5.1.1 for details) as constant whereas in the second line we consider a piecewise constant forward variance $\xi_0(\cdot)$ (see Section 5.1.2 for details).

	MC Pricing 1F Bergomi Full Surface	MC Pricing rBergomi Full Surface	NN Pricing Full Surface	NN Gradient Full Surface	Speed up NN vs. MC
Flat forward variance	300.000 μs	500.000 μs	$14,3$ μs	47 μs	$21.000 - 35.000$
Piecewise constant forward variancce	300.000 μs	500.000 μs	$30,9$ μs	113 μs	$9.000 - 16.000$

Table 2: Computational time of pricing map (entire implied volatility surface) and gradients via Neural Network approximation and Monte Carlo (MC)

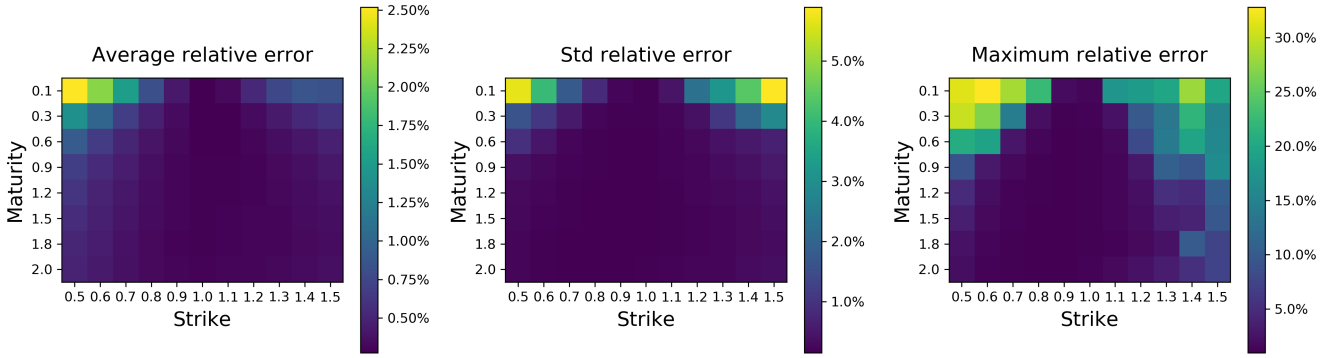

Figure 4: As benchmark we recall average relative errors of Monte Carlo prices computed across 40.000 random parameter combinations of the Rough Bergomi model. Relative errors are given in terms of Average-Standard Deviation-Maximum (Left-Middle-Right) on implied volatility surfaces in the Rough Bergomi model, computed using 95% confidence intervals.

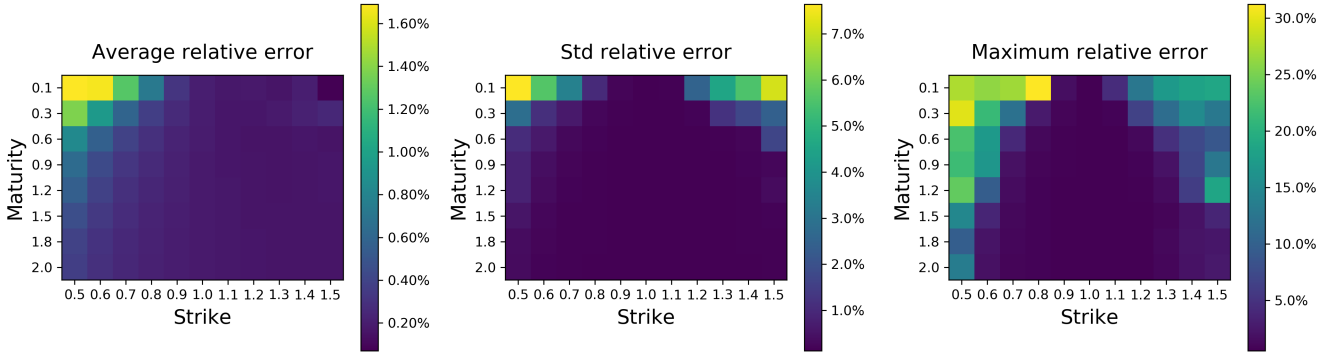

Figure 5: As benchmark we recall average relative errors of Monte Carlo prices computed across 40.000 random parameter combinations of the 1 Factor Bergomi model. Relative errors are given in terms of Average-Standard Deviation-Maximum (Left-Middle-Right) on implied volatility surfaces in the 1 Factor Bergomi model, computed using 95% confidence intervals.

5.1.1 Flat forward variances

In this section we consider $\xi_0(t) = \xi_0$ to be constant and analyse the performance of our approach. We will consider the rough Bergomi 3 and 1 Factor Bergomi models 6 from Section 2.1.

- Normalized parameters as input and normalised implied volatilities as output
- 3 hidden layers with 30 neurons and *Elu* activation function
- Output layer with *Linear* activation function
- Total number of parameteres: 5.668
- Train Set: 34.000 and Test Set: 6.000
- Rough Bergomi sample:$(\xi_0, \nu, \rho, H) \in \mathcal{U}[0.01, 0.16] \times \mathcal{U}[0.5, 4.0] \times \mathcal{U}[-0.95, -0.1] \times \mathcal{U}[0.025, 0.5]$
- 1 Factor Bergomi sample:$(\xi_0, \nu, \rho, \beta) \in \mathcal{U}[0.01, 0.16] \times \mathcal{U}[0.5, 4.0] \times \mathcal{U}[-0.95, -0.1] \times \mathcal{U}[0, 10]$
- strikes=$\{0.5, 0.6, 0.7, 0.8, 0.9, 1, 1.1, 1.2, 1.3, 1.4, 1.5\}$
- maturities=$\{0.1, 0.3, 0.6, 0.9, 1.2, 1.5, 1.8, 2.0\}$
- Training data samples of Input-Output pares are computed using Algorithm 3.5 in Horvath, Jacquier and Muguruza [37] with 60.000 sample paths and the spot martingale condition i.e. $\mathbb{E}[S_t] = S_0,\ t \geq 0$ as control variate.

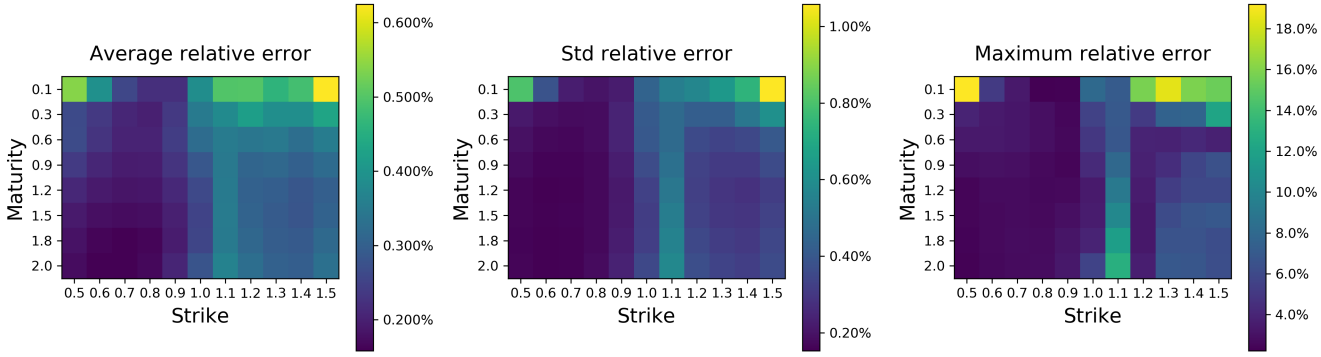

Figure 6: We compare surface relative errors of the neural network approximator against the Monte Carlo benchmark across all training data (34.000 random parameter combinations)in the rough Bergomi model. Relative errors are given in terms of Average-Standard Deviation-Maximum (Left-Middle-Right).

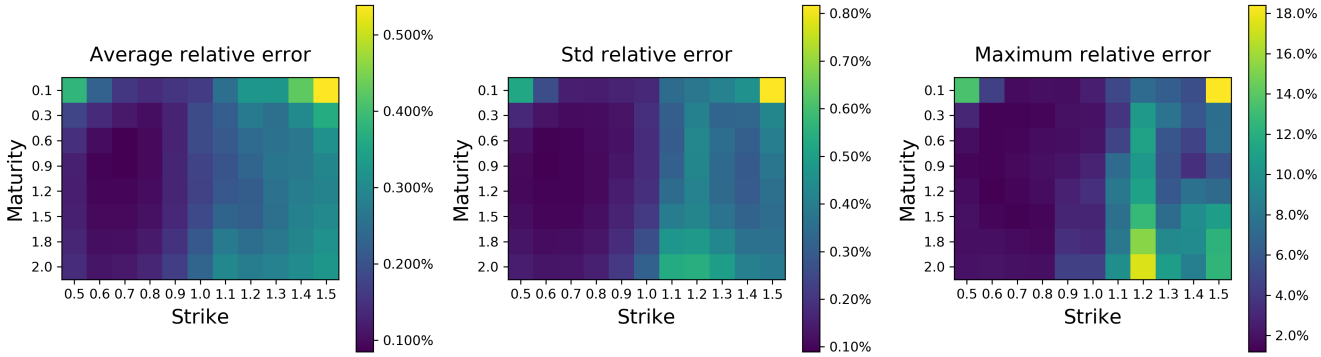

Figure 7: We compare surface relative errors of the neural network approximator against the Monte Carlo benchmark across all training data (34.000 random parameter combinations)in the 1 Factor Bergomi model. Relative errors are given in terms of Average-Standard Deviation-Maximum (Left-Middle-Right).

Figures 6 and 7 show that the average (across all parameter combinations) relative error[7] between neural network and Monte Carlo approximations is far less than 0.5% consistently (left image in Figures 6 and 7) with a standard deviation of less than 1% (middle image in Figures 6 and 7). Nevertheless, the maximum relative error goes as far as 25%. As previously stated, the beauty of this approach is the ability to asses wether the approximation is suitable and if not, where exactly

[7]Relative here is computed here as $|\sigma^{NN}(T,k) - \sigma^{MC}(T,k)|/|\sigma^{MC}(T,k)|$.

fails or is more delicate. In this case, we observe that the approximation is less precise for short maturities and deep out-of-the-money/in-the-money options. Theses errors are consistent with the errors of the Monte Carlo training set (see Figures 4-5).

5.1.2 General forward variances

In this section we consider a piecewise constant forward variance curve $\xi_0(t) = \sum_{i-1}^{n} \xi_i \mathbf{1}_{\{t_{i-1}<t<t_i\}}$ where $t_0 = 0 < t_1 < ... < t_n$ and $\{t_i\}_i = 1, .., n$ are the option maturity dates ($n = 8$ in our case). This is the modelling approach suggested by Bergomi [10]. We will consider again the rough Bergomi 3 and 1 Factor Bergomi models 6

- Normalized parameters as input and normalised implied volatilities as output
- 4 hidden layers with 30 neurons and *Elu* activation function
- Output layer with *Linear* activation function
- Total number of parameteres: 5.878
- Train Set: 68.000 and Test Set: 12.000
- Rough Bergomi sample:$(\xi_0, \nu, \rho, H) \in \mathcal{U}[0.01, 0.16]^8 \times \mathcal{U}[0.5, 4.0] \times \mathcal{U}[-0.95, -0.1] \times \mathcal{U}[0.025, 0.5]$
- 1 Factor Bergomi sample:$(\xi_0, \nu, \rho, \beta) \in \mathcal{U}[0.01, 0.16]^8 \times \mathcal{U}[0.5, 4.0] \times \mathcal{U}[-0.95, -0.1] \times \mathcal{U}[0, 10]$
- strikes=$\{0.5, 0.6, 0.7, 0.8, 0.9, 1, 1.1, 1.2, 1.3, 1.4, 1.5\}$
- maturities=$\{0.1, 0.3, 0.6, 0.9, 1.2, 1.5, 1.8, 2.0\}$
- Training data samples of Input-Output pares are computed using Algorithm 3.5 in Horvath, Jacquier and Muguruza [37] with 60.000 sample paths and the spot martingale condition i.e. $\mathbb{E}[S_t] = S_0,\ t \geq 0$ as control variate.

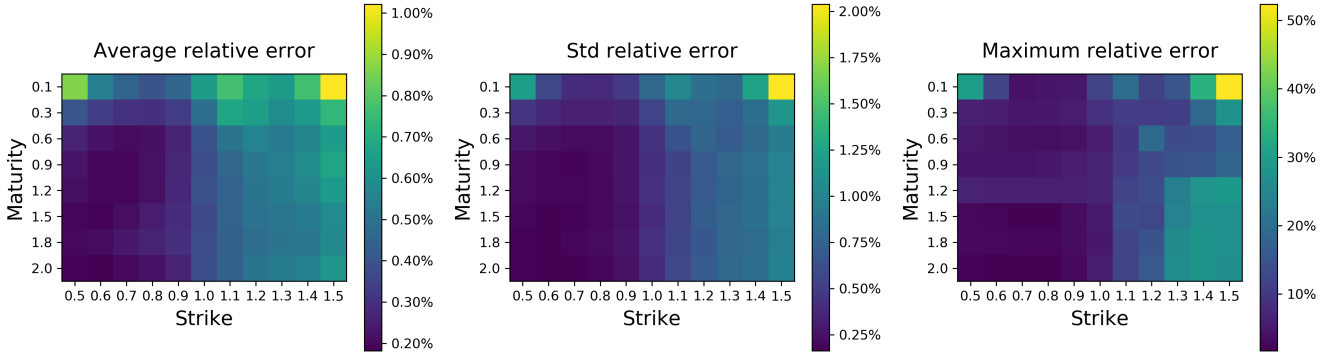

Figure 8: We compare surface relative errors of the neural network approximator against the Monte Carlo benchmark across all training data (68.000 random parameter combinations)in the rough Bergomi model. Relative errors are given in terms of Average-Standard Deviation-Maximum (Left-Middle-Right).

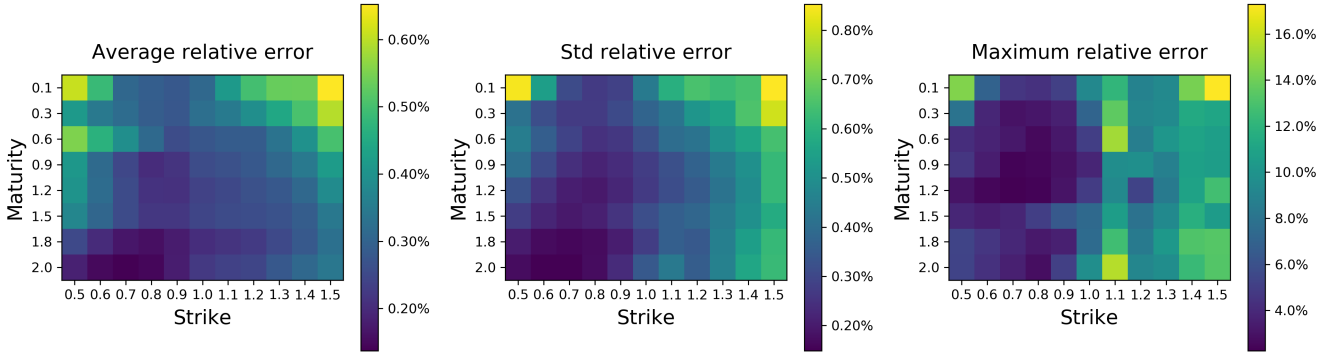

Figure 9: We compare surface relative errors of the neural network approximator against the Monte Carlo benchmark across all training data (68.000 random parameter combinations)in the 1 Factor Bergomi model. Relative errors are given in terms of Average-Standard Deviation-Maximum (Left-Middle-Right).

Figures 8 and 9 show that the average (across all parameter combinations) relative error between neural network and Monte Carlo approximations is far less than 0.5% consistently (left image in Figures 8 and 9) with a standard deviation of less than 1% (middle image in Figures 8 and 9). As before in the flat forward variance case (see Figures 6 and 7), the maximum relative error goes as far as 25%. We conclude that the methodology generalises adequately to the case of non-constant forward variances, by showing the same error behaviour.

5.2 Calibration accuracy and speed

Figure 10 and 11 reports average calibration times on test set for different parameter combinations on each of the models analysed. We conclude that gradient-based optimizers outperform in terms of speed gradient-free ones. Moreover, in Figure 11 one observes that computational times in gradient-free methods are heavily affected by the dimension of the parameter space, i.e. flat forward variances are much quicker to calibrate than piecewise constant ones. We find that Lavenberg-Marquardt is the most balanced optimizer in terms of speed/convergence and we choose to perform further experiments with this optimizer. The reader is encouraged to keep in mind that a wide range of optimizers is available for the calibration and the optimal selection of one is left for future research.

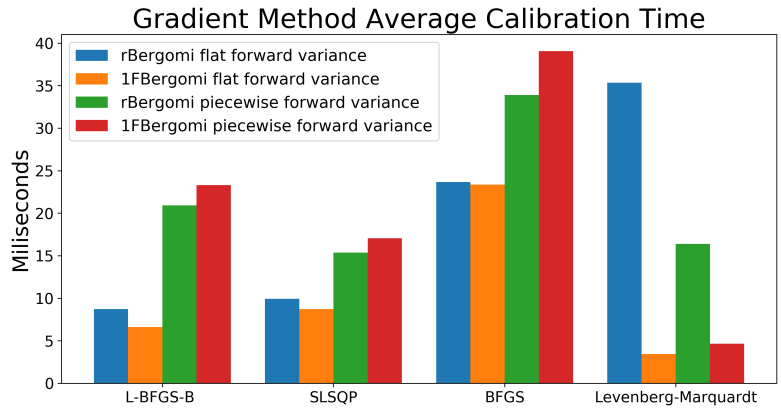

Figure 10: Average calibrations times for all models using gradient-based optimizers.

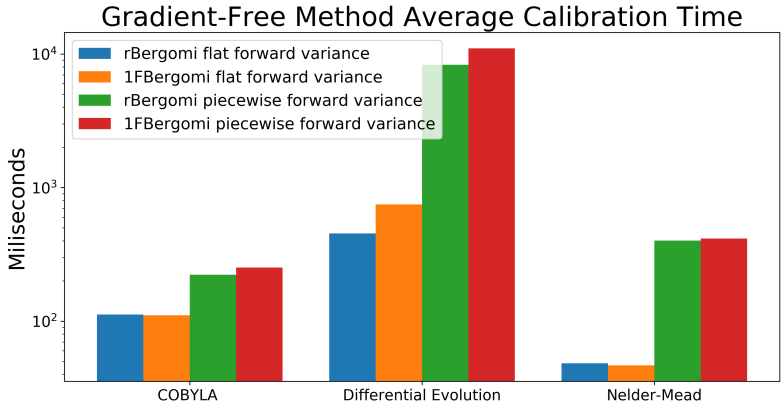

Figure 11: Average calibrations times for all models using gradient-free optimizers.

In order to asses the accuracy, we report the calibrated model parameters $\widehat{\theta}$ compared to the synthetically generated data with the set of parameters $\overline{\theta}$ that was chosen for the generation of our synthetic data. We measure the accuracy of the calibration via parameter relative error i.e.

$$E_R(\widehat{\theta}) = \frac{|\widehat{\theta} - \overline{\theta}|}{|\overline{\theta}|}$$

as well as the root mean square error (RMSE) with respect to the original surface i.e.

$$\text{RMSE}(\widehat{\theta}) = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m}(\widetilde{F}(\widehat{\theta})_{ij} - \sigma_{BS}^{MKT}(T_i, k_j))^2}.$$

Therefore, on one hand a measure of good calibration is a small RMSE. On the other hand, a measure of parameter sensitivity on a given model is the combined result of RMSE and parameter relative error.

5.2.1 Flat forward variances

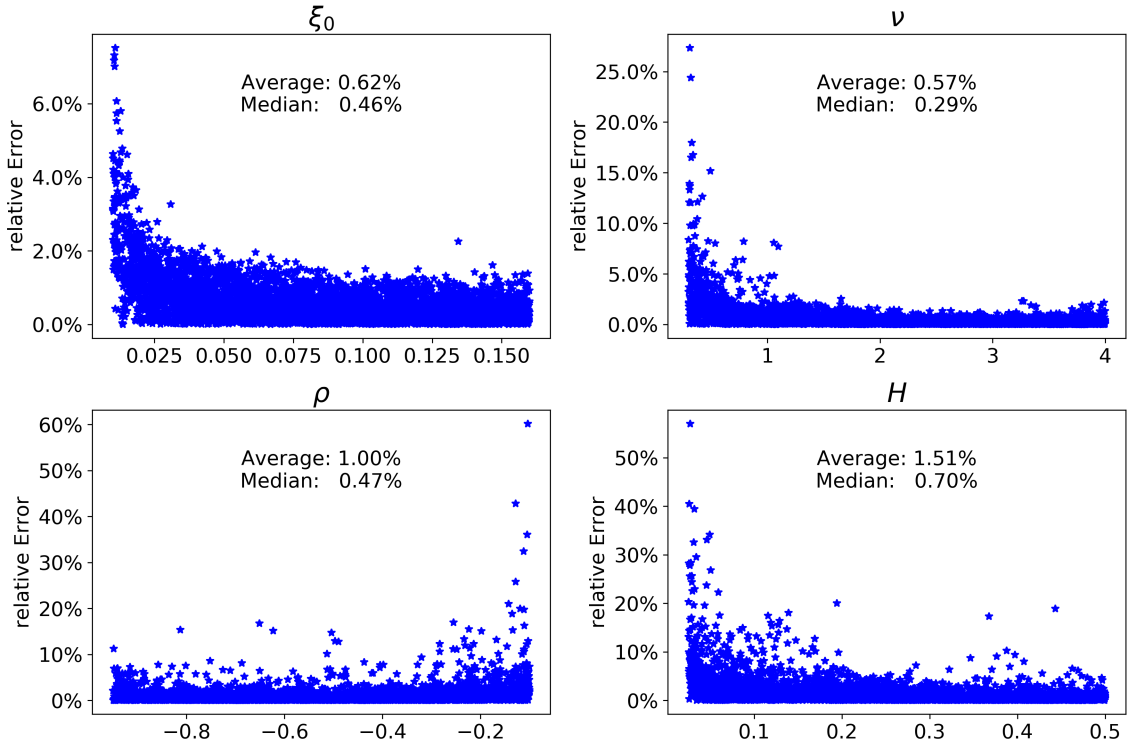

Figure 12: Calibration relative error per parameter in the test set in the rough Bergomi model

Figures 12 shows relative errors after calibration via Levengerg-Marquardt in the rough Bergomi model. We observe that largest errors are concentrated for small H or small vol of vol ν situations. Naturally, the relative error is more sensitive around 0 as well. Once again, we emphasise that by understanding the error zones of the pricing function P (see Figure 8) along with parameter relative errors in Figure 12, we are able to asses its quality and detect parameter configurations that might yield a lower performance of the calibration process.

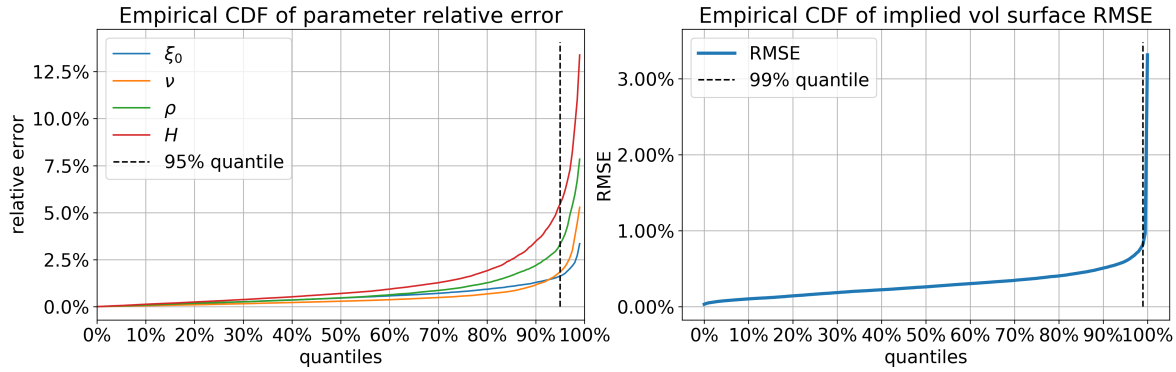

Figure 13: Cumulative Distribution Function (CDF) of Rough Bergomi parameter relative errors (left) and RMSE (right) after Levengerg-Marquardt calibration across test set random parameter combinations.

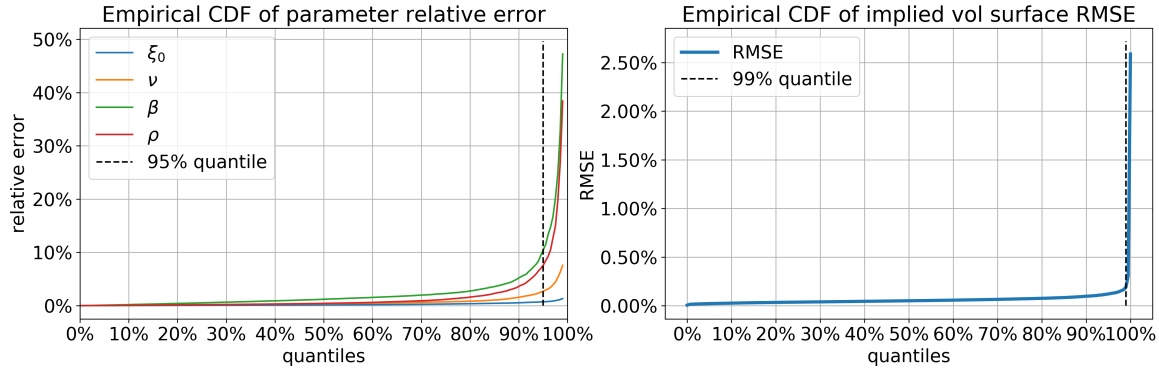

Figure 14: Cumulative Distribution Function (CDF) of 1 Factor Bergomi parameter relative errors (left) and RMSE (right) after Levengerg-Marquardt calibration across test set random parameter combinations.

To finalise our analysis, Figures 13 and 14 show that the 99% quantile of the RMSE is below 1%, even though parameter relative errors might be higher (see 12 as well), particularly when the parameters are close to 0. Notably, the maximum RMSE in both models is below 4%, which suggests a surprisingly good accuracy.

5.2.2 General forward variances

We perform a similar analysis with a piecewise constant term-structure of forward variances. Figures 15 and 16 show that the 99% quantile of the RMSE is below 1% and shows that the Neural Network approach generalises properly to the piecewise constant forward variance. Again, we find that the

largest relative errors per parameter are concentrated around 0, consequence of using the relative error as measure. This suggests a successful generalisation to general forward variances, which to our knowledge has not been addressed before by means of neural networks or machine learning techniques.

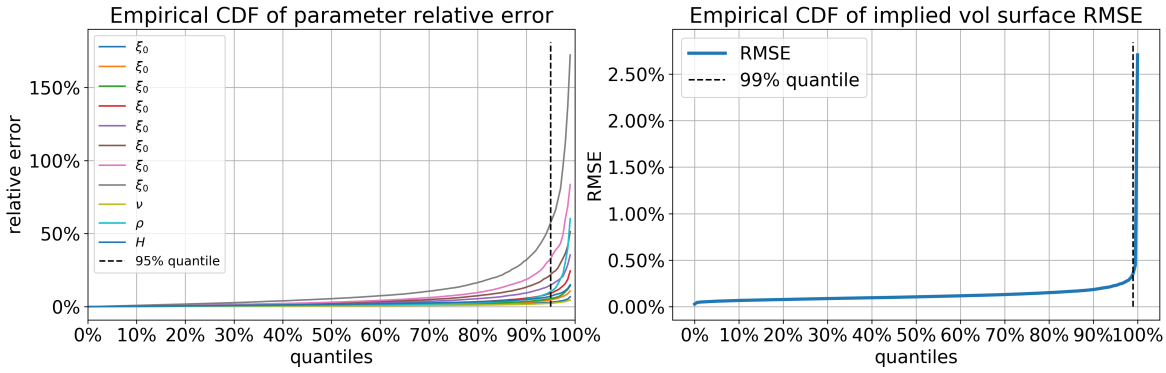

Figure 15: Cumulative Distribution Function (CDF) of Rough Bergomi parameter relative errors (left) and RMSE (right) after Levengerg-Marquardt calibration across test set random parameter combinations.

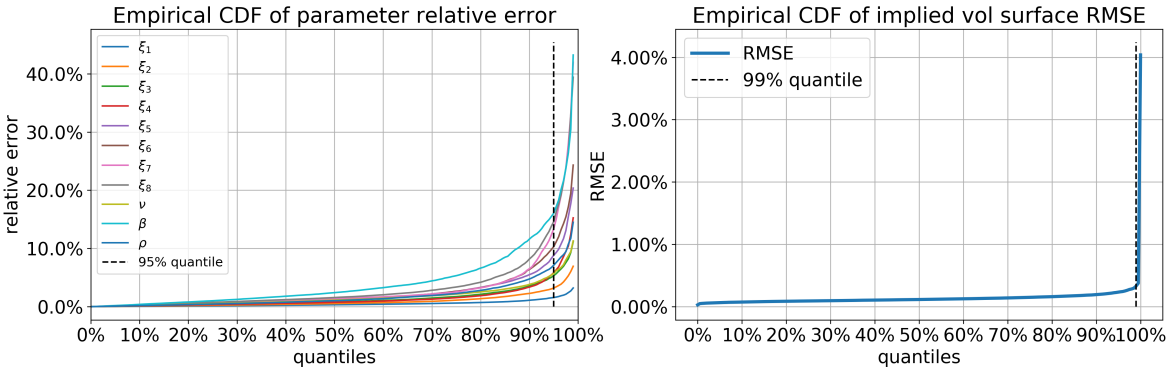

Figure 16: Cumulative Distribution Function (CDF) of 1 Factor Bergomi parameter relative errors (left) and RMSE (right) after Levengerg-Marquardt calibration across test set random parameter combinations.

6 Conclusions and potential applications: Assessment of "best-fit" models

To sum up, 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, as we will do here, 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. In our numerical sections we provide details on neural network approximations of the implied volatility mapping. We present present specific architectures that achieve this and discuss their respective performances.

Potential applications and outlook towards mixture of "expert" models: In the previous sections we set up a powerful approximation method to closely approximate implied volatilities under different stochastic models and highlighted that the choice of the objective function (evaluation of the surface on a grid, inspired by pixels of an image) was crucial for the performance of the network. Now we are interested in the inverse task and ask whether a neural network—trained by this objective function to multiple stochastic models simulatanously—can identify which stochastic model a given set of data comes from. By doing so potential applications we have in mind are twofold:
(1) Ultimately we are interested in which model (or what mixture of existing stochastic models) describes the market best.
(2) From a more academic and less practival perspective, we are interested whether and to what extent is it possible to "translate" parameters of one stochastic model to parameters of another.

We set up a further experiment with the aim to identify the stochastic volatility model which produced a given implied volatility surface, among a list of possible models. We worked under a classification setting: a neural network was trained to identify which stochastic volatility model generated an implied volatility surface. To do so, implied volatility surfaces were generated by the Heston, Bergomi and rough Bergomi models, as in Section 2.1. For each volatility surface, a "flag" corresponding to the model was assigned (eg: 1 for Heston, 2 for Bergomi and 3 for rough Bergomi). The training set is thus of the form: $(\Sigma_{BS}^{\mathcal{M}(\theta)}, I)$, where $\mathcal{M}$ is one of the three models $\mathcal{M}^{\text{Heston}}$, $\mathcal{M}^{\text{Bergomi}}$, $\mathcal{M}^{\text{rBergomi}}$ and θ is in their respective sets of admissible parameters Θ^{Heston}, Θ^{Bergomi}, Θ^{rBergomi} and I is a flag to represent the model which generated the data ($I = 1$ if $\mathcal{M} = \mathcal{M}^{\text{Heston}}$, $I = 2$ if $\mathcal{M} = \mathcal{M}^{\text{Bergomi}}$ and $I = 3$ if $\mathcal{M} = \mathcal{M}^{\text{rBergomi}}$).

We used 20.000 implied volatility surfaces for each model and generated them as in Section 5. We set aside 20% for validation and set 20% of the remaining set for testing to avoid overfitting when training the classifier. Thus the network was trained on 38400 samples, tested on 9600 and validated using 12000 points. We split the data to keep an even distribution of all models in the training and validation data.

We chose a small, fully connected feedforward network to avoid overfitting and with exponentially linear activation functions for the same reasons as those outlined in section 4.3. The network is composed of 2 hidden layers (of 100 and 50 output nodes respectively) with exponentially linear activation functions and an output layer with a softmax activation function. The output of

the network represents the probabilities of a given surface belonging to a particular model. We used stochastic gradient descent to minimize cross-entropy and the number of epochs was set to 20. The cross-entropy of two discrete distributions (p, q) with K possible distinct values is $H(p, q) := -\sum_{1 \leq i \leq K} p_i \log q_i$.

We report the classifiers' effectiveness and comment on the results in Figure 6.

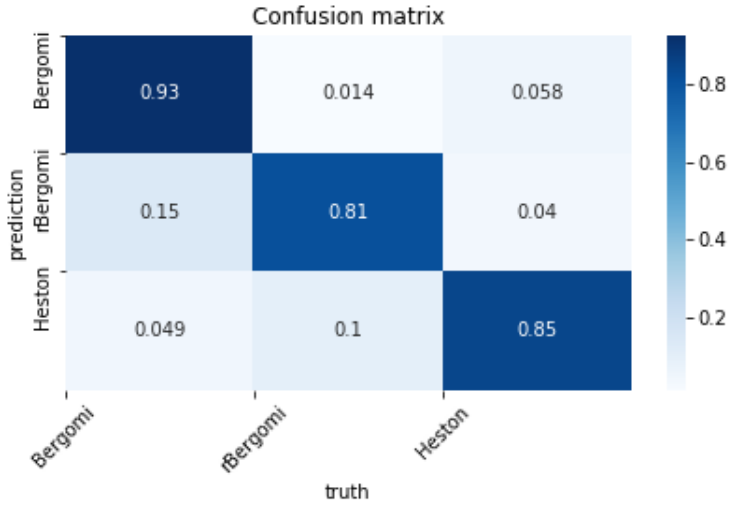

Figure 17: Confusion matrix of the neural network classifier. Entry (i, j) of the matrix corresponds to the relative number of elements of class j classified as class i by the neural network. Elements of the diagonal thus correspond to correctly classified points and off-diagonal elements to misclassifications. While a significant amount of surfaces from the rough Bergomi model are misclassified as Bergomi surfaces, around 10% of validation set surfaces from the rough Bergomi model have a Hurst exponent between 0.45 and 0.55. For those parameters, the two models are similar, thus explaining around 10% out of 15% of the misclassifications from our sample.

The previous experiment fits into the larger scope of recognizing *mixtures*, or combinations, of models. We present here one approach to this problem. Given three surfaces from the Heston, Bergomi and rough Bergomi models, $\Sigma^{\mathcal{M}^{\text{Heston}}}$, $\Sigma^{\mathcal{M}^{\text{Bergomi}}}$, $\Sigma^{\mathcal{M}^{\text{rough Bergomi}}}$, define the mixture surface as $\Sigma^{\mathcal{M}^{\text{Mixture}((a,b,c))}} := a\Sigma^{\mathcal{M}^{\text{Heston}}} + b\Sigma^{\mathcal{M}^{\text{Bergomi}}} + c\Sigma^{\mathcal{M}^{\text{rough Bergomi}}}$, where $a, b, c \geq 0$ and $a + b + c = 1$. We could map each mixture surface to a tuple (a, b, c) such that $a+b+c = 1$. a, b and c would thus represent the respective probabilities of the mixture surface belonging to a particular model. Our previous approach concerned only surfaces from a single model: combinations (a, b, c) such that $a = 0, b = 0$ (rough Bergomi model), $a = 0, c = 0$ (Bergomi model) or $b = 0, c = 0$ (Heston model). This is only one of many possible settings and we leave this problem for further research.

A A numerical experiment with the inverse map

To motivate the main drawbacks of this approach we calibrate rough Bergomi model i.e. consider the simple map

$$\mathcal{P}^{-1}(\Sigma_{BS}^{rBergomi}) \to (\hat{\xi_0}, \hat{\nu}, \hat{\rho}, \hat{H})$$

where $\Sigma_{BS}^{rBergomi} \in \mathbb{R}^{n \times m}$ is a rBergomi implied volatility surface and $(\hat{\xi_0}, \hat{\nu}, \hat{\rho}, \hat{H})$ the optimal solution to (16).

Remark 10. For simplicity we consider the strikes and maturities to be fixed for all implied volatility surfaces.

Inverse Map Architecture

- 1 convolutional layer with 16 filters and 3×3 sliding window
- MaxPooling layer with 2×2 sliding window
- 50 Neuron Feedforward Layer with *Elu* activation function
- Output layer with *linear* activation function
- Total number of parameters: 10.014
- Train Set: 34.000 and Test Set: 6.000
- $(\xi_0, \nu, \rho, H) \in \mathcal{U}[0.01, 0.16] \times \mathcal{U}[0.3, 4.0] \times \mathcal{U}[-0.95, -0.1] \times \mathcal{U}[0.025, 0.5]$
- strikes=$\{0.5, 0.6, 0.7, 0.8, 0.9, 1, 1.1, 1.2, 1.3, 1.4, 1.5\}$
- maturities=$\{0.1, 0.3, 0.6, 0.9, 1.2, 1.5, 1.8, 2.0\}$
- Implied vols computed using Algorithm 3.5 in Horvath, Jacquier and Muguruza [37] with 60.000 sample paths and the spot martingale condition i.e. $\mathbb{E}[S_t] = S_0,\ t \geq 0$ as control variate.

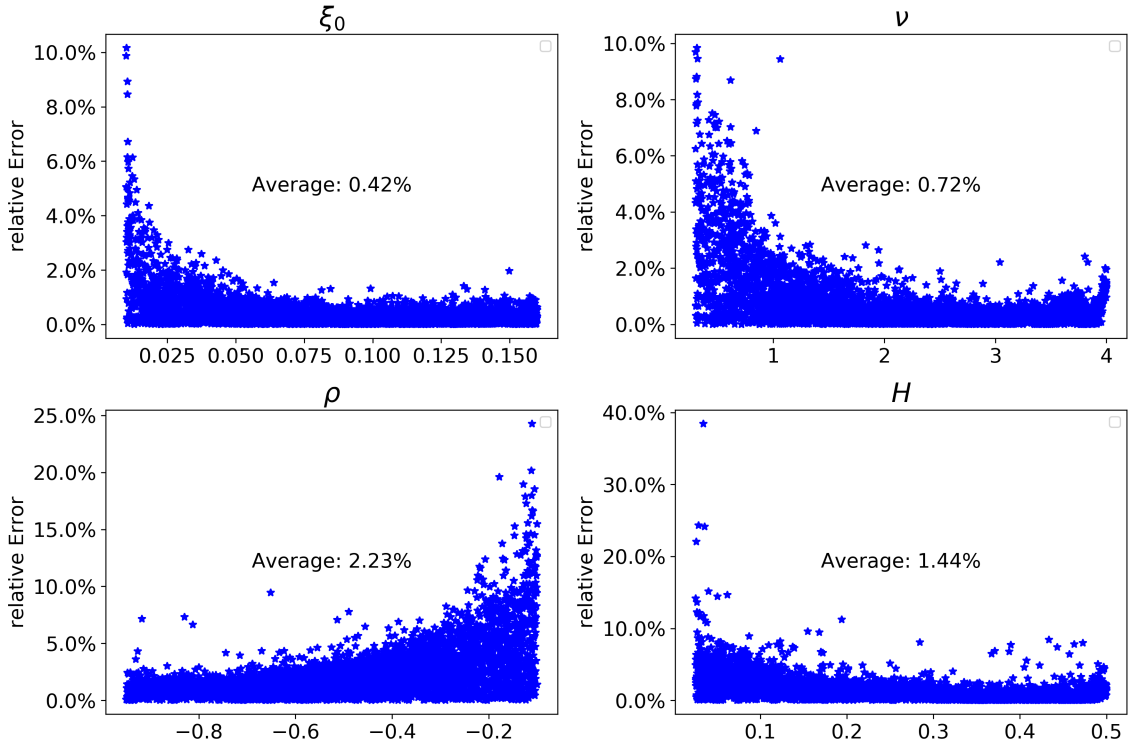

Figure 18: Out of sample relative errors per parameter calibration

Figure 18 shows that, indeed it is possible to approximate the inverse map and very sharply calibrate model parameters with a relatively small network. Convolutional networks make sense in this context, since a implied volatility surface has many features both in the strike and maturity direction that can be extracted, similar to image recognition problems. Notice also that the biggest error come from parameter configurations where the Monte Carlo input is more delicate i.e. very small H or very small volatility. Hence, the shape of the errors is intuitively natural and expected beforehand.

Black-Box function and "Real" out of sample performance

Let us now consider "Real" out of sample data, in the sense that it has not been generated by the rough Bergomi model itself. We generate implied volatility surfaces using the 2 Factor Bergomi given by

$$dX_t = -\frac{1}{2}V_t dt + \sqrt{V_t}dW_t$$
$$V_t = \xi_0(t)\mathcal{E}\left(\nu\left((1-\theta)\int_0^t \exp(-\kappa_X(t-s))dZ_s + \theta\int_0^t \exp(-\kappa_Y(t-s))dY_s\right)\right)$$

where W, Y and Z are correlated standard Brownian motions with correlation matrix. We feed smiles from this model into our Neural Network to obtain the corresponding optimal rough Bergomi parameters. We then compare this values with a brute force Monte Carlo calibration via Lavenberg-Marquandt [48, 49] algorithm. Figure 19 shows that the Neural Network does not generalize properly out of sample and concludes that the Brute Force MC method clearly beats the Inverse map approach. The results by Hernandez [30] also support this conclusion, since his out of sample performance (based on different historical period) is reasonaly worse than the in-sample one. However, we must enphasize that when the Neural Network is exposed to famiiar situations i.e. surfaces close to the ones generated by the rBergomi model it may work better than the MC approach (see points below the dashed red line in Figure 19). This is likely due to delicate parameter configurations i.e. very low variance, where MC suffers to obtain accurate estimates whereas the Network does not struggle that much.

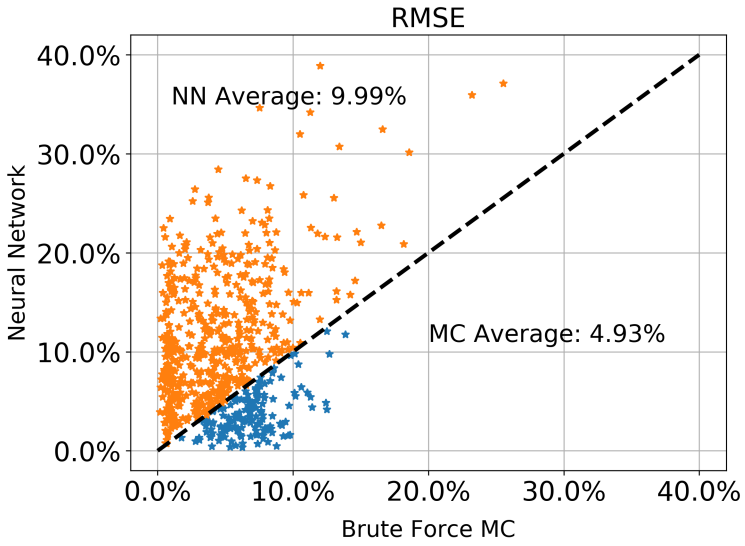

Figure 19: Stars represent the out of sample RMSE via Neural Network (NN) and Brute force Monte Carlo (MC). Dashed black line represents the indentity function.

Our conclusion is that the inverse map approach does not generalise the problem (16) for all possible settings, since by design is impossible to train $\mathcal{P}^{-1}$ on all posibble (arbitage-free) market scenarios[8]. Moreover, there is a lack of understanding in the unknown function $\mathcal{P}^{-1}$, hence from a risk-managing perspective is very difficult to justify the use of such aproach. Finally, as we have proven, the model fails to generalise to a situation that has not seen before and does not give a consistent answer with respect to the benchmark problem (16).

[8]One might technically be able to generate a large number of market scenarios and thereupon obtain corresponding optimal model parameterers. This approach, however would rely on the capacity of the market simulator to cover all possible scenario configurations, in order to avoid inconsistencies with the brute force approach.

References

[1] E. Alòs, D. García-Lorite and A. Muguruza. On smile properties of volatility derivatives and exotic products: understanding the VIX skew. arXiv:1808.03610, 2018.

[2] E. Alòs, J. León and J. Vives. On the short-time behavior of the implied volatility for jump-diffusion models with stochastic volatility. *Finance and Stochastics*, 11(4), 571-589, 2007.

[3] A, Antonov, M. Konikov and M. Spector. SABR spreads its wings. Risk (August issue), pp. 58-63, 2013.

[4] A.R. Barron. Approximation and estimation bounds for artificial neural networks. *Machine Learning*. Vol.14:1,1994 Pages 115-133 .

[5] C. Bayer, P. Friz, P. Gassiat, J. Martin and B. Stemper. A regularity structure for rough volatility. arXiv:1710.07481, 2017.

[6] C. Bayer, P. Friz and J. Gatheral. Pricing under rough volatility. *Quantitative Finance*, 16(6): 1-18, 2015.

[7] C. Bayer, P. Friz, A. Gulisashvili, B. Horvath and B. Stemper. Short-time near the money skew in rough fractional stochastic volatility models. arXiv:1703.05132, 2017.

[8] C. Bayer and B. Stemper. Deep calibration of rough stochastic volatility models. *Preprint*, arXiv:1810.03399

[9] M. Bennedsen, A. Lunde and M.S. Pakkanen. Hybrid scheme for Brownian semistationary processes. *Finance and Stochastics*, 21(4): 931-965, 2017.

[10] L. Bergomi. Stochastic Volatility Modeling. Chapman & Hall/CRC financial mathematical series. *Chapman & Hall/CRC*, 2015.

[11] H. Buehler, L. Gonon, J. Teichmann and B. Wood. Deep Hedging. *Preprint*, arXiv:1802.03042, 2018.

[12] B. Chen, C. W. osterlee and H. Van Der Weide. Efficient unbiased simulation scheme for the SABR stochastic volatility model, 2011.

[13] D. Clevert, T. Unterthiner, S. Hochreiter. Fast and accurate deep network learning by exponential linear units (ELUs). *Preprint*, arXiv:1511.07289, 2015.

[14] J. De Spiegeleer, D. Madan, S. Reyners and W. Schoutens. Machine learning for quantitative finance: Fast derivative pricing, hedging and fitting.SSRN:3191050, 2018.

[15] G. Dimitroff, D. Röder and C. P. Fries. Volatility model calibration with convolutional neural networks. *Preprint*, SSRN:3252432, 2018.

[16] R. Eldan and O. Shamir. The power of depth for feedforward neural neworks. *JMLR: Workshop and Conference Proceedings* Vol 49:1-34, 2016.

[17] C. Doléans-Dade. Quelques applications de la formule de changement de variables pour les semimartingales. *Z. Wahrscheinlichkeitstheorie verwandte Gebiete*, Vol 16: 181-194, 1970.

[18] O. El Euch and M. Rosenbaum. The characteristic function of rough Heston models. To appear in *Mathematical Finance*.

[19] O. El Euch and M. Rosenbaum. Perfect hedging in rough Heston models, to appear in *The Annals of Applied Probability*, 2018.

[20] J. Friedman, R. Tibshiran and T. Hastie. The Elements of Statistical Learning. *Springer New York Inc*, 2001.

[21] R. Ferguson and A. Green. Deeply learning derivatives. Preprint arXiv:1809.02233, 2018.

[22] M. Fukasawa. Asymptotic analysis for stochastic volatility: martingale expansion. *Finance and Stochastics*, 15: 635-654, 2011.

[23] J. Gatheral. A parsimonious arbitrage-free implied volatility parameterization with application to the valuation of volatility derivatives, *Presentation at Global Derivatives*, 2004.

[24] J. Gatheral, T. Jaisson and M. Rosenbaum. Volatility is rough. *Quantitative Finance*, 18(6): 933-949, 2018.

[25] J. Gatheral, A. Jacquier. Arbitrage-free SVI volatility surfaces. *Quantitative Finance*, 14(1): 59-71, 2014

[26] A. Gulisashvili, B.Horvath and A. Jacquier. Mass at zero in the uncorrelated SABR model. *Quantitative Finance*, 18(10): 1753-1765, 2018.

[27] A. Gulisashvili, B.Horvath and A. Jacquier. On the probability of hitting the boundary for Brownian motions on the SABR plane. *Electronic Communications in Probability*, 21(75): 1-13, 2016.

[28] I. Goodfellow, Y. Bengio and A. Courville. Deep Learning. *MIT Press*, 2016.

[29] S. Hendriks, C. Martini The Extended SSVI Volatility Surface. *Preprint*, SSRN:2971502, 2017.

[30] A. Hernandez. Model calibration with neural networks. *Risk*, 2017.

[31] P. Hagan, D. Kumar, A. Lesniewski, and D. Woodward. Managing smile risk. *Wilmott Magazine*, September issue: 84-108, 2002.

[32] P. Hagan, A. Lesniewski, and D. Woodward. Probability distribution in the SABR model of stochastic volatility. Large Deviations and Asymptotic Methods in Finance, *Springer Proceedings in Mathematics and Statistics*, 110, 2015.

[33] K.Hornik. Approximation capabilities of multilayer feedforward networks.*NeuralNetworks*, 4(2):251-257, 1991.

[34] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359-366, 1989.

[35] K. Hornik. M. Stinchcombe and H. White. Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural Networks*. Vol. 3:11, 1990.

[36] B. Horvath, A. Jacquier and C. Lacombe. Asymptotic behaviour of randomised fractional volatility models. arXiv:1708.01121, 2017.

[37] B. Horvath, A. Jacquier and A. Muguruza. Functional central limit theorems for rough volatility. arXiv:1711.03078, 2017.

[38] B. Horvath, O. Reichmann. Dirichlet Forms and Finite Element Methods for the SABR Model. *SIAM Journal on Financial Mathematics*, p. 716-754(2), May 2018.

[39] A. Jacquier, C. Martini and A. Muguruza. On VIX futures in the rough Bergomi model. *Quant. Finance*, 18(1): 45-61, 2018.

[40] A. Jacquier, M. Pakkanen and H. Stone. Pathwise large deviations for the rough Bergomi model. *Journal of Applied Probability*, 55(4): pp.1078-1092, 2018.

[41] A. Jentzen, B. Kuckuck, A. Neufeld, P. von Wurstemberger. Strong error analysis for stochastic gradient descent optimization algorithms, Preprint arXiv:1801.09324 ,2018.

[42] S. Ioffe and C. Szegedy. Batch normalisation: Accelerating deep network training by reducing internal covariate shift. *Preprint*, arXiv:1502.03167, 2015.

[43] D.P. Kingman and J. Ba, Adam: A Method for Stochastic Optimization. *Conference paper*, 3rd International Conference for Learning Representations, 2015.

[44] A. Kondratyev. Learning curve dynamics with artificial neural networks. *Preprint*, SSRN:3041232, 2018.

[45] D. Kraft. A Software Package for Sequential Quadratic Programming. *DFVLR-FB* pp.88-28, 1988.

[46] G. Kutyniok, H. Bölcskei, P. Grohs and P. Petersen, Optimal approximation with sparsely connected deep neural networks, *Preprint* arXiv:1705.01714, 2017.

[47] A. Leitao Rodriguez, L.A. Grzelak and C.W. Oosterlee. On an efficient multiple time step Monte Carlo simulation of the SABR model. *Quantitative Finance*, 17(10), pp.1549-1565, 2017.

[48] K. Levenberg. A Method for the Solution of Certain Non-Linear Problems in Least Squares. *Quarterly of Applied Mathematics*. 2: pp. 164-168, 1944.

[49] D. Marquardt. An Algorithm for Least-Squares Estimation of Nonlinear Parameters. *SIAM Journal on Applied Mathematics*. 11 (2): pp. 431-441,1963.‘

[50] W. A. McGhee. An artificial neural network representation of the SABR stochastic volatility model. *Preprint*, SSRN:3288882, 2018.

[51] H. N. Mhaskar. Approximation properties of a multilayered feedforward artificial neural network. *Advances in Computational Mathematics*, 1(1): 61-80, 1993.

[52] J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*. 7: pp. 308-313, 1965.

[53] J. Nocedal and S. Wright. Numerical Optimization. Springer Series in Operations Research and Financial Engineering. *Springer-Verlag New York*, 2006.

[54] D. Pedamonti. Comparison of non-linear activation functions for deep neural networks on MNIST classification task. *Preprint*, arXiv:1804.02763

[55] M. J. D. Powell. A direct search optimization method that models the objective and constraint functions by linear interpolation. Advances in Optimization and Numerical Analysis, eds. S. Gomez and J-P. Hennart, *Kluwer Academic (Dordrecht)*, pp. 51-67, 1994.

[56] L. M. Rios and N. V. Sahinidis. Derivative-free optimization: a review of algorithms and comparison of software implementations. *Journal of Global Optimization.* Vol. 56: 3 pp. 1247-1293, 2013.

[57] L. Setayeshgar, and H. Wang. Large deviations for a feed-forward network, *Advances in Applied Probability*, 43: 2, pp. 545571, 2011.

[58] U. Shaham, A. Cloninger, and R. R. Coifman. Provable approximation properties for deep neural networks. *Appl. Comput. Harmon. Anal.*, 44(3): 537-557, 2018.

[59] H. Stone. Calibrating rough volatility models: a convolutional neural network approach. *Preprint*, arXiv:1812.05315, 2018.

[60] R. Storn and K. Price. A Simple and Efficient Heuristic for global Optimization over Continuous Spaces. *Journal of Global Optimization.* Vol.11:4, pp341-359, 1997.

[61] V. N. Vapnik. Statistical Learning Theory. *Wiley-Interscience*, 1998.

[62] C. Zhu, R. H. Byrd and J. Nocedal. L-BFGS-B: Algorithm 778: L-BFGS-B, FORTRAN routines for large scale bound constrained optimization. *ACM Transactions on Mathematical Software*, 23: 4, pp. 550-560, 1997.