APPM 4530 Final Project: Deep Learning Volatility

Diego Alvarez
Adam Hoerger

December 2021

1 Introduction

The paper that we chose is "Deep Learning Volatility: A deep neural network perspective on pricing and calibration in (rough) volatility" [10] models by Blanka Horvath, Aitor Muguruza, and Mehdi Thomas. The paper is at the cutting edge of stochastic volatility research and brings in a litany of new quantitative methods including: rough paths, neural networks, and stochastic volatility. Although stochastic volatility is a heavily studied area within quantitative finance, many models have failed to describe accurate options pricing. Another concern that is addressed by the authors is the idea of calibration which is a common problem that practitioners encounter. The paper both provides a new framework for modeling volatility that provides a significant improvement in model calibration times compared to existing approaches.

This stochastic model allows for a more robust pricing using a relatively new topic in math. The underlying math for this model is built upon rough paths which were first introduced in 1990s pioneered by Terry Lyons who first proposed the model and later Martin Harier who developed the generalization of rough paths called regularity structures.

The authors of the paper also incorporate deep learning, which is a somewhat new area of machine learning. Although deep learning is researched more than rough paths, there are still many areas of deep learning that researchers are not sure of. Although deep learning (initially called a multilayer perceptron model) was first described in 1967 its widespread application did not begin until the late 2010s when computational processing became capable of building these models as well as easy applicable programming packages to deploy them.

1.1 Initial Motivation from Black-Scholes-Merton Model

To briefly cover the initial model of Black-Scholes-Merton [4] we first start with a function for the price of the option $V \in C^{1,2}$. The function V, takes a series of parameters $V(\tau, S_t, \sigma, r, K)$. Under the Black-Scholes-Merton model the authors assume that r, K, σ are constant (essentially making V a function of only τ and

S_t). They also assume that the stock price S_t follows this stochastic process and that the bank account follows

$$dS_t = \mu S_t dt + \sigma S_t dW_t \tag{1}$$

$$dB_t = rB_t dt \tag{2}$$

The initial model comes from applying Ito's lemma to the option pricing function $V(\tau, S_t)$

$$dV = \frac{\partial V}{\partial t}dt + \frac{\partial V}{\partial S}dS + \frac{1}{2} \cdot \frac{\partial^2 V}{\partial S^2}dS^2 \tag{3}$$

As in class, this translates to the Black-Scholes-Merton PDE

$$\frac{\partial V}{\partial t} + \frac{1}{2} \cdot \sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0$$

Looking at the Black-Scholes-Merton PDE we can see that volatility σ is a constant. Although we did not consider σ within our function V it initially arises from the fact that we assume that S_t follows the stochastic process described in equation 1.

Upon analysis it is most likely the case the volatility is not constant, as Black-Scholes-Merton fails to describe features of implied volatility surfaces such as volatility smiles which imply that volatility of options are dependent on their strike prices.

Figure 1: This is the S&P 500 (SPX) call option with February 22nd, 2022 Expiration. On the bottom axis they use "Moneyness" which measure the difference between underlying and strike price.

The models below attempt to make some advancements in the modeling volatility as function which Black-Scholes-Merton fails to describe. A majority of these models look at modifying some of the initial assumptions that the Black-Scholes-Merton model made when describing the stock prices.

1.2 Local Volatility, and Heston Volatility

After the work that Black, Scholes, and Merton created, there have been a series of other models that tried to describe options pricing under a more relaxed constraint which lets volatility vary. The earliest model was the local volatility model [7]. The local volatility model assumes that volatility σ is a function of t and S_t. The function takes the form

$$dS(t) = \mu S dt + \sqrt{\nu(t)} S dW \tag{4}$$

In this case we have the function $U \in C^{1,2,2}$ which takes the parameters S, v, and t where v is volatility of the option (this called the volatility of volatility nicknamed vol of vol). When we apply Ito's lemma and end up with a similar looking partial differential equation that takes the form

$$\begin{gathered} \frac{1}{2}\nu S^2 \frac{\partial U}{\partial S^2} + \rho\sigma v S \frac{\partial^2 U}{\partial S \partial \nu} + \frac{1}{2}\sigma^2 \nu \frac{\partial^2 U}{\partial \nu^2} + \\ rS\frac{\partial U}{\partial S} + (\kappa[\theta - \mu(t)] - \lambda(S, v, t))\frac{\partial U}{\partial \nu} - rU + \frac{\partial U}{\partial t} = 0 \end{gathered} \tag{5}$$

We used a similar approach like in the Black-Scholes-Merton model, but the main takeaway is that volatility can now vary rather than be a constant which was originally stated under Black-Scholes-Merton. We also see that volatility arises from the underlying stochastic process $S(t)$ as well as being a parameter $U(s, t, v)$.

Another model proposed is the Heston model which was created by Steven Heston [8]. The Heston model assumes that the stock price follows this stochastic process

$$dS_t = \mu S_t dt + \sqrt{\nu_t} S_t dW_t \tag{6}$$

Then the ν_t is the instantaneous variance which follows a CIR process

$$d\nu_t = \kappa(\theta - \nu_t)dt + \xi\sqrt{\nu_t} dW_t^{\nu} \tag{7}$$

Where ν_0 is the initial volatility, θ is the long run average variance of the price. The expected value of ν_t tends to θ, κ is the reversion rate, and ξ is the volatility of the volatility (variance of ν_t).

1.3 Rough Paths, Hurst Parameters, and Rough Volatility modelling

Rough paths is an area of stochastic analysis that deals with the smooth paths that allow for a solution of controlled differential equations. The main goal of

rough paths is that the goal is to describe a smooth function that can exhibit highly oscillatory and non-linear systems. A major benefit to rough paths is that their values are in truncated free tensor algebras. That is a benefit because neural network and neural network packages are essentially tensor calculations. This has allowed for both the mathematical theory of rough paths and is application in neural networks to be explored simultaneously.

Below we'll give an example of building a rough path. We start with a n-tensor power of $\mathbb{R}^d$

$$(\mathbb{R}^d)^{\otimes n} := \mathbb{R}^d \otimes \mathbb{R}^d \otimes \ldots \otimes \mathbb{R}^d \tag{8}$$

Then we can define a tensor space (this is necessary because this will allow us to transform the problem into neural networks). We can define a truncated tensor algebra space $T^n(\mathbb{R}^d)$

$$T^n(\mathbb{R}^d) := \bigotimes_{i=0}^{n} (\mathbb{R}^d)^{\otimes i} \tag{9}$$

From here we can define a continuous path $X : [0, T] \to \mathbb{R}^d$ defined on the tensor space we can define the signature of that path as

$$S(X) := (1, X^1, X^2, \ldots) \in T(\mathbb{R}^d) \tag{10}$$

Then we can define our path X similar as an integral equation (similar to Lyons 1998 [13]). In this case we have a series of integrals

$$\begin{aligned} S(X)_{s,t} = \Bigg(1, \int_{s<s_1<t} dX_{s_1}, \int_{s<s_1<s_2<t} dX_{s_1} \otimes dX_{s_2}, \ldots, \\ \int_{s<s_1,\ldots<s_n<t} dX_{s_1} \otimes \ldots \otimes dX_{s_n}, \ldots \Bigg) \end{aligned} \tag{11}$$

Although this can look complex essentially the signature of each path is an integral in each tensor direction (n dimension). Although this process may look like a lot of hand-waving we can thus build a tensor-based Brownian motion. We start with a Brownian motion $(B_t)_{t\geq 0}$. We can define the Brownian motion where $\circ$ is the Stratonovich integration [12] [1]

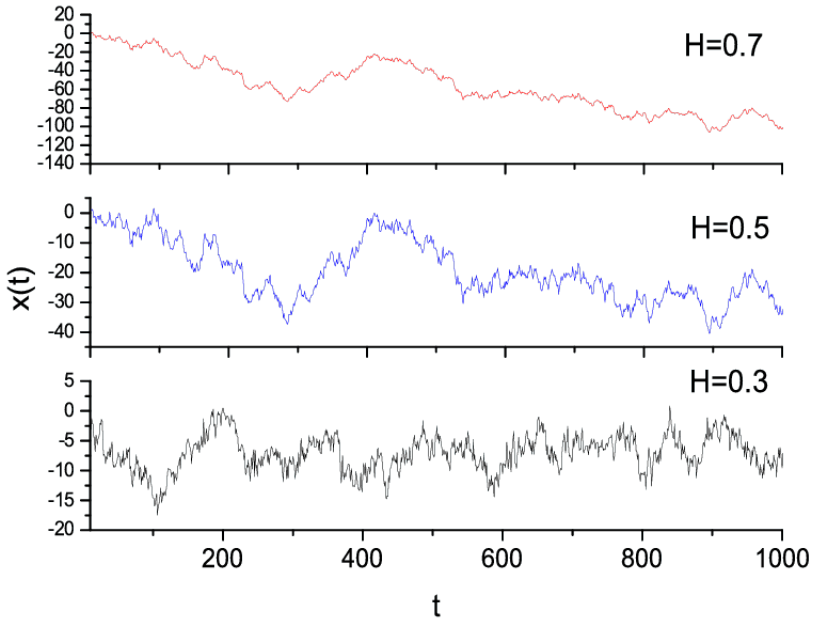
$$B_{s,t} = \left(1, \int_{s<s_1<t} \circ dB_{s_1}, \int_{s<s_1<s_2<t} \circ dB_{s_1} \otimes \circ dB_{s_2}\right) \tag{12}$$

Again the main thing to draw from equation 12 is that we are able to build a Brownian motion that allows for pathwise integration in multiple dimensions (in this case we restrict to $2 < p < 3$).

The reason why we included this in our paper is that we can create a fractional Brownian motion. The fractional Brownian motion includes a Hurst parameter, which adds in long-term memory of time series. It was first discovered

[1] which was the "precursor" integration method I learned before Ito's integral, also not sure if the paper is the original paper for this process.

by Harold Edwin Hurst who found the Hurst Exponent while studying the dam size of the Nile River [11]. The Hurst Parameter allows for memory within time time series. Below is an example of time series with different Hurst Parameters [1].



The interesting part of the Hurst Parameter is that with memory it allows for a neural network to "learn" the time series. We can apply the Hurst parameter into a Brownian motion creating the Fractional Brownian motion. The Fractional Brownian Motion takes the form

$$B_H^m(s,t) = \left(1, \int_{s<s_1<t} dB_H^m(s_1), \int_{s<s_1<s_2<t} dB_H^m(s_1), \int_{s<s_1<s_2<t} dB_H^m(s_1) \otimes dB_H^m(s_2), \right.$$
$$\left. \int_{s<s_1<s_2<s_3<t} dB_H^m(s_1) \otimes dB_H^m(s_2) \otimes dB_H^m(s_3)\right) \tag{13}$$

Again, the equation looks very complex but the main idea is that the space that we integrate over is fractional Brownian motion with a Hurst parameter which adds "memory" into the time series. We add this model into our paper because the authors use fractional Brownian motions into the Bergomi model. We'll first introduce the Bergomi Model which was developed by Lorezno Bergomi who is the head of quantitative research in equity derivatives at Societe Generale. The Bergomi model goal is to model the dynamic of volatility smiles. The idea

behind the Bergomi model is that they are trying to model exotics using vanilla derivatives. The problem occurs when the pricer has to hedge vega [2]. To hedge that vega we have to ensure the the model incorporates all of the dynamics of the implied volatility. The model takes this form (called the 1-factor Bergomi Model) [3]

$$\begin{cases} dX_t = -\frac{1}{2}V_t dt + \sqrt{V_t} dW_t \\ V_t = \xi_0(t)\varepsilon\left(\eta \int_0^t \exp(-\beta(t-s))dZ_s\right) \end{cases} \tag{14}$$

Here the terms get a little to complex for the scope of the class but, we'll do our best at defining them. The function $\varepsilon(\cdot)$ is the stochastic exponential, that is $Y = \varepsilon(X)$ is the solution to $dY_t/dt = Y_t dX_t/dt$. The V_t is the instantaneous spot variance process and the $\xi_t^u,\ u \geq t$ is the instantaneous forward variance. Let's first define the term spot and forward (these originally come from fixed income if you are familiar, for this case we'll use interest rates to first define them). Let's say we are at t_0 of a timeline. The spot rate of a 1 year bond is the interest rate for a bond starting at t_0 and maturing at t_1. The forward rate of a 1 year bond is the interest rate for a bond starting at t_1 and maturing at t_2 where you buy at t_0. In this case instead of using interest rates we are using volatility, and instead of securities that "mature" we are using options that "expire". In this case everything is done instantaneously which implies that intervals are infinitesimally small. The $W^1, \dots W^n$ is an n-dimensional correlated Brownian.

An extension of the Bergomi Model comes from adding "roughness" into it called the Rough Bergomi (rBergomi). [2][3] The rBergomi model takes the form

$$\begin{cases} dX_t = -\frac{1}{2}V_t dt + \sqrt{V_t} dW_t, \\ V_t = \xi_0(t)\varepsilon\left(\sqrt{2H}\nu \int_0^t (t-s)^{H-\frac{1}{2}} dZs\right) \end{cases} \tag{15}$$

The model keeps the same parameters as the one in equation 14, except $H \in (0, 1)$. Here we can see where the Hurst parameter plays in. At this level (which is at almost peak complexity in the field and the most cutting edge research of options pricing at the moment), it is not worth going into the intricacies of the model. The main idea behind this model is that it tries to understand volatility within the rough paths framework.

What Horvath, Muguruza, and Thomas attempt to do is develop a method for analyzing these rough models in a more time-efficient manner than brute force Monte Carlo methods. In particular, their method accelerates the processes of pricing options and calibrating model parameters, i.e. fitting parameter values to explain observed data.

[2] Although we didn't cover hedging-based pricing models in the class they were somewhat covered. The idea is that we can hedge the derivative by matching the volatility with an opposite bet of the underlying security. Think of pricing a call option, to do that we had to short the stock. In other words we hedged the volatility with the same stochastic process. Also vega is a "Greek" which are different options sensitives. The "Greeks" come from taking partial derivatives of the Black Scholes model. Vega is $\frac{\partial V}{\partial \sigma}$

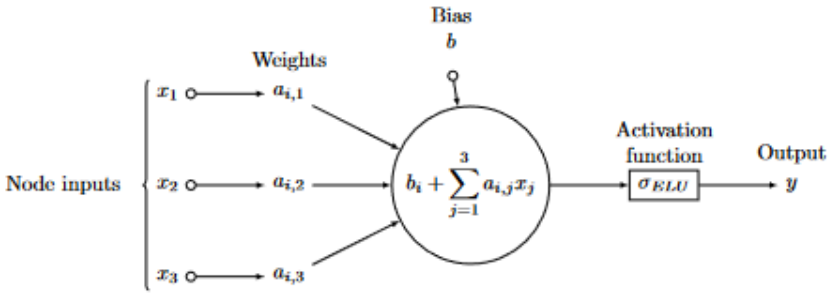
[3] Also an co-author of the rBergomi model paper is Jim Gatheral, a co-author of this paper.

2 Approach

The primary contribution of this paper is to separate the approximation of the pricing map into two steps: 1) learn the pricing map onto a grid-based output space under arbitrary model parameters, then 2) calibrate the model to market parameters. In particular, this approach was devised in an attempt to accelerate the calibration of parameters in the aforementioned (rough) stochastic volatility models. The bulk of the computation time is concentrated in the training of a neural network in step 1, which allows for rapid computation of pricing maps under various parameter values. Once the neural network is trained, the second step involves the calibration of parameter values based on observed data, a step which is accelerated due to the rapid pricing map evaluation.

2.1 Neural Networks

The first step in the method devised by Horvath et al. is centered on the use of feed-forward neural networks. The authors use these neural networks not as a means of prediction, as in certain "black box" applications, but rather as high-dimensional functional approximators to pricing maps, or a function mapping a set of model parameter values to a grid of option prices over various maturities and strike prices. A neural network is composed of many neurons, each of which multiply its inputs by weights, add a bias, and apply what is termed an "activation function" to produce its output:



These neurons can be chained together in layers, with outputs of prior neurons being used as inputs to other neurons, in order to form a feed-forward neural network. Note that the layers in between the input layer and the layer that produces the final output are referred to as "hidden" layers. See section 2.2 for a visual representation of a neural network with four hidden layers. The neural network can then be made to approximate some function $f(x)$ by adjusting or "training" all weights in the network so as to minimize the error of the approximation (on a set of training data). Of course, one may call into question whether a neural network, whose computations are based entirely on linear combinations and possibly non-linear activation functions has enough expressiveness to approximate an arbitrary pricing function. However, in 1989,

Hornik et al. established the following result [9]:

Theorem 1 (Universal approximation theorem (Hornik, Stinchcombe and White [38])). *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.*

This theorem establishes that a single-layer neural network which can approximate any measurable function arbitrarily well, which justifies the use of neural networks in the approximation of pricing maps. Another result (which Horvath et al. call Theorem 2) establishes that if the activation function σ is in C^n, then the neural network additionally approximates up to the n-th derivative of the function. Specifically, if σ is chosen to be smooth, then a single-layer neural network can additionally be used to approximate a function and its first derivative. This is beneficial later, when using a gradient-based optimizer in the calibration step. The authors discuss additional proven results to further justify their choices in neural network architecture. The first establishes that a greater number of nodes which will improve the accuracy of a neural network approximation, so long as the number of training data points is sufficiently large to offset the effects of overfitting[4]. The second establishes that increasing the number of layers allows for greater approximation accuracy than would be possible without a drastic increase in the number of neurons per layer.

2.2 Solution Architecture

Recall that the authors split the problem of price map approximation into two steps. In the first step, they train a neural network to learn the pricing map, solving the problem

$$\hat{w} = \arg\min_{w \in \mathbb{R}^n} \sum_{u=1}^{N_{\text{train}}} \sum_{i=1}^{n} \sum_{j=1}^{m} (F(\theta_u, w)_{ij} - F^{\star}(\theta_u)_{ij})^2$$

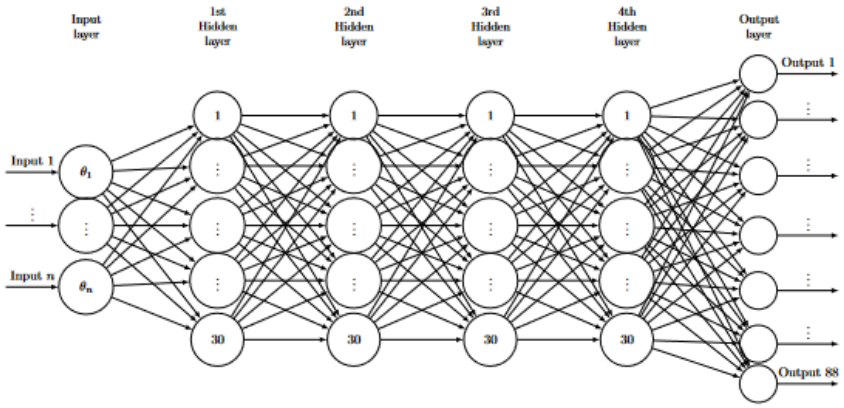
That is, they wish to adjust the weights in the neural network to minimize the total squared error between the approximation provided by the neural network, $F(\theta, w)$, and the true pricing map $F^{\star}(\theta)$ across a full $n \times m$ grid of maturities and strike prices under N_{train} different sample parameters θ. The output of the trained neural network approximator is defined for input parameters θ to be $\widetilde{F}(\theta) = F(\theta, \hat{w})$. Note that any evaluation of $\widetilde{F}$ under parameters θ is essentially a series of weighted sums and additions, which can be reduced to a series of matrix multiplications and additions (a very optimized task, nowadays).

For this step, they use the following network architecture consisting of four hidden layers, with each hidden layer made up of 30 neurons. They utilize the $\text{Elu} = \alpha(e^x - 1)$, $\alpha > 0$ activation function[5] for this network, as its smoothness means that the neural network can approximate not only a function, but also

[4] Overfitting is the machine learning term for when a model remembers rather than learns

[5] Exponential Linear Unit

its first derivative, which is necessary to guarantee the efficacy of gradient-based parameter optimization methods.



The output consists of 88 values representing an 8x11 grid of option prices over a range of strikes and maturities, though this choice of grid size was arbitrary and could be modified. The neural network aims to approximate this pricing grid as a function of model parameters, which are used as inputs to the neural network. Note that for vanilla options, they instead use a grid of implied volatilities rather than prices, but this framework is also capable of approximating pricing maps for more exotic options. Their model, under the described number of layers, layer size, and output size, contains $30n + 6478$ weights and biases to train[6]. Neural networks typically train these weights using some variation of gradient descent.

Throughout the training process, the authors additionally employ several tools to combat the effects of overfitting and accelerate convergence. The first is an early stopping condition, in which the network has a maximum number of training iterations, but will additionally end training if the parameters cease to update significantly for a certain number of iterations. They additionally normalize the model parameters to constrain them within a designated domain, for which they specify minimum and maximum values. The implied volatility surface is additionally normalized by subtracting the mean and dividing by the standard deviation. These steps ensure that any gradients are more well-behaved and that the eventual weights generalize properly to out of sample data.

Once the neural network is trained, in the second step, they calibrate the parameters to either synthetically generated or observed market data, solving the problem

$$\hat{\theta} = \arg\min_{\theta \in \Theta} \sum_{i=1}^{n} \sum_{j=1}^{m} (\widetilde{F}(\theta)_{ij} - P(T_i, K_j))^2$$

[6] n = number of parameters

across the parameter space Θ. For vanilla options, we can replace the pricing map P with a volatility surface σ. This calibration step involves the use of either a gradient-based or gradient-free optimizer, whose benefits and drawbacks are summarized in the following table:

	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

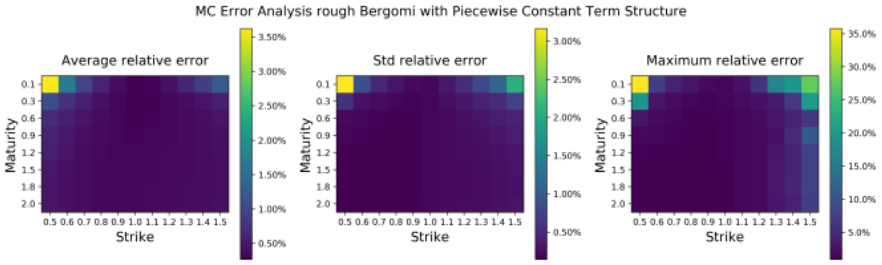
In either class of optimizer, the optimization process requires many evaluations of the pricing map under various parameter values θ. Under the traditional Monte Carlo approach, each evaluation would itself require many simulations, resulting in an overall slow process[7]. Using a neural network, each evaluation of the pricing map, though it relies on an approximation, requires only one forward pass through the neural network, which is much quicker.

[7]Some processes use a queue of Monte Carlo simulations which may lead to loss of generality

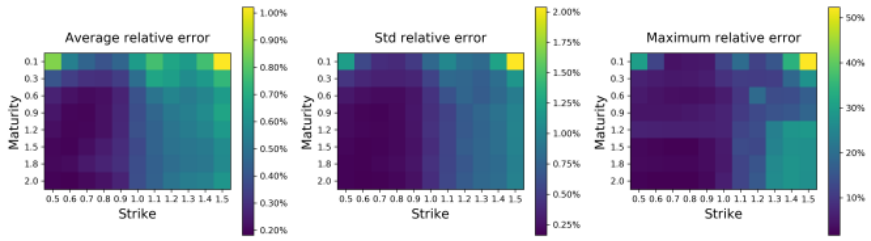
3 Numerical Results

3.1 Pricing Map Approximation

The authors establish a numerical experiment to verify 1) the accuracy of their proposed method and 2) the acceleration in the speed at which pricing map approximations can be generated. For both the 1-factor Bergomi and rough Bergomi models under a piecewise constant forward-variance curve, they generate implied volatility surfaces, equivalent to the pricing map for a vanilla option, using both Monte Carlo methods as well as their proposed deep learning approach. We only show results for the rough Bergomi models, as the 1-factor Bergomi model produces similar results with respect to the accuracy of their new method compared to the baseline Monte Carlo. In the Monte Carlo approach, for each of 80,000 sets of random parameters, they simulate 60,000 paths to generate the volatility surface under those parameters. The same Monte Carlo approach was then used to generate parameter input and volatility surface output pairs, which are then used as training and testing data for the first step of the neural network. A 95% confidence interval of the Monte Carlo method corresponds to the following relative approximation errors, averaged across all sample parameter combinations:



Meanwhile, the neural network exhibits the following relative errors, relative to the Monte Carlo benchmark:



Notably, the maximum errors are somewhat high, but the average error in the neural network approximation of the volatility surface is generally well under

1% and within the bounds suggested by a 95% confidence interval, suggesting that the authors' approximation is sufficiently accurate for vanilla options. This is further supported by the relatively small standard deviations.

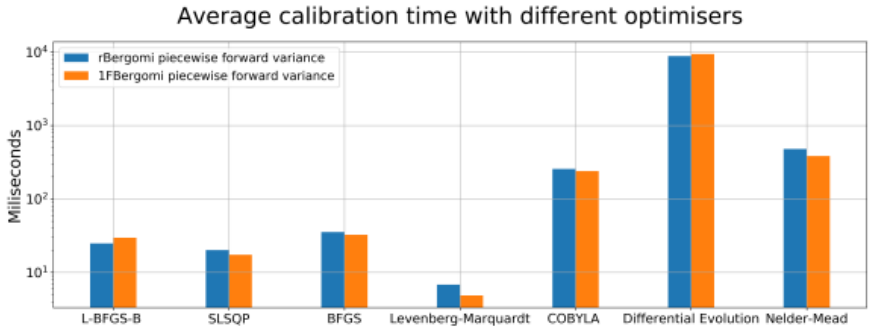
A perhaps more significant result once the initial neural network is trained, however, is that the evaluation of the volatility surface is accelerated dramatically relative to the Monte Carlo approach.

	MC Pricing 1F Bergomi Full Surface	MC Pricing rBergomi Full Surface	NN Pricing Full Surface	NN Gradient Full Surface	Speed up NN vs. MC
Piecewise constant forward variance	$300,000\ \mu s$	$500,000\ \mu s$	$30.9\ \mu s$	$113\ \mu s$	$9,000 - 16,000$

Because the network does not need to simulate paths every time the parameters are updated, the time required to update the pricing map is reduced from over 5 minutes under the Monte Carlo method to under a second. This is important because parameter calibration requires the evaluation of the pricing map under various parameters, meaning the neural network approach can be used to calibrate parameters much more rapidly.

3.2 Parameter Calibration

Once the neural network in the first step is trained, they examine a series of potential optimizers to use in the calibration of model parameters. Their discussion includes four gradient-based methods (L-BFGS-B, SLSQP, BFGS, and Levenberg-Marquardt) as well as three gradient-free methods (COBYLA, Differential Evolution, and Nelder-Mead). Using synthetically generated data, they analyzed the parameter calibration step under each optimizer.



Their results reinforce the notion that gradient-based methods are generally faster than gradient-free ones, and they proceed through the rest of the error analysis using the Levenberg-Marquardt algorithm in particular.

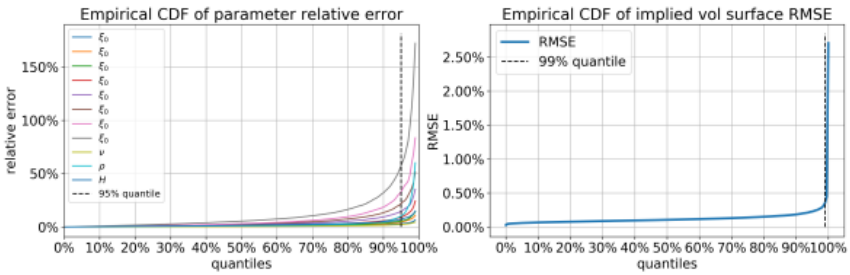
In evaluating the accuracy of the calibration step, they consider two error measures. The first is the relative error of the estimated parameters $\hat{\theta}$ with

respect to the actual parameters $\bar{\theta}$ used to synthetically create the data, and the second is the relative root mean square error (RMSE) of the resulting estimated surface compared to the Monte Carlo-generated surface σ_{mkt}. These are defined as follows:

$$\text{err}_R(\hat{\theta}) = \frac{|\hat{\theta} - \bar{\theta}|}{|\bar{\theta}|}$$

$$\text{RMSE}_R(\hat{\theta}) = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m}(\widetilde{F}(\hat{\theta})_{ij} - \sigma_{\text{mkt}}(T_i, K_j))^2}$$

Note that the RMSE is the square root of the measure being minimized in the parameter calibration. The CDF of their observed errors is shown here:



The fact that the 99% quantile of the RMSE is below 1% suggests that the calibration was quite successful, although the 99% quantiles of the parameter errors were less convincing. That being said, the massive acceleration achieved in the neural network approach, combined with the low RMSE, suggests that the neural network approach should still be quite effective in parameter calibration in practical applications.

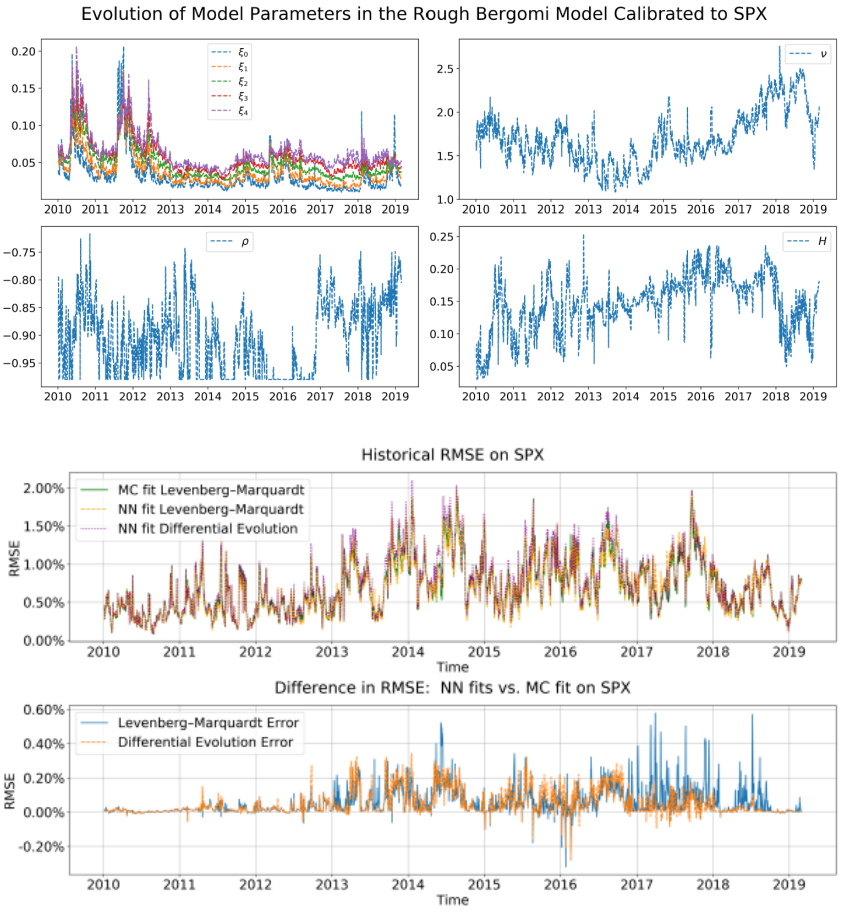
The authors now seek to apply their methods to real world data, and they attempt to calibrate the parameters of a rough Bergomi model to the SPX using historical volatility smiles from January 2010 into March 2019. At each point in time, they calibrate the model parameters using the volatility surface along the grid defined by

$$(T_1, T_2, T_3, T_4, T_5) = \frac{1}{12} \times (1, 3, 6, 9, 12)$$
$$k_i = 0.85 + (i-1) \times (0.05) \quad \text{for } i = 1, ..., 9$$

Note that the time is measured in years, so the maturities in the grid represent one month until expiration, then expirations at the end of each of the next four quarters.

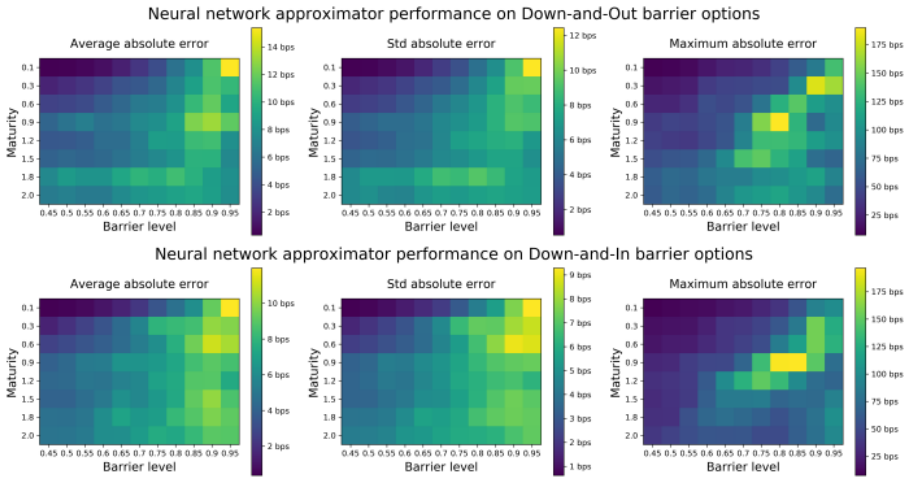
The parameter calibration is done using both a Monte Carlo approach as well as a neural network approach for two different optimizers, the Levenberg Marquardt optimizer and differential evolution. The Monte Carlo results are

used as a baseline for error evaluation of the neural network approaches. The evolution of the model parameters over time and the RMSE of each optimizer relative to the Monte Carlo calibration are shown below:



Note that the Hurst parameter H is generally less than 0.5, which is indicative of a rough path. As expected, differential evolution, a gradient-free approach, obtains generally superior results than the gradient-based Levenberg-Marquardt optimizer. However, one must keep in mind that the Levenberg-Marquardt optimizer was over 1000 times faster than differential evolution in the authors' calibration timing tests. Regardless, either optimizer produces RMSEs that are generally below 0.4%, except for a relatively small number of spikes (which do not exceed 0.6%), suggesting that the high accuracy observed with the synthetic data extrapolates sufficiently well to real market data.

The authors conclude their numerical analysis by computing the error in the pricing map of two barrier options.



Keep in mind that unlike the previous surface errors, this is calculated for an actual pricing map rather than a volatility surface, a consequence of the more exotic nature of this option. An average error that is generally less than 10 basis points (0.1%) with under 10 basis points of standard deviation suggests that the accuracy of this approach extends to other more exotic option types.

4 Conclusion & Applications

Towards the end of the paper the authors point at a series of future areas of research. Throughout the paper the authors were trying to use neural networks to approximate the implied volatilities of different stochastic models. They place a great emphasis on the objective function, but something that they are interested in is looking at the neural network models and finding the objective function used. The application of this allows for a mixture models (similar to ensemble learning). The applications that the "reverse modelling" is to see if parameters of different stochastic models are "translatable". Another consideration that the authors make is the future of optimizers for neural network calibration. Along with the paper is a GitHub repository. All of the neural networks that they build use the "adam" optimizer with keras / tensorflow framework [8]. Adam is a stochastic gradient descent optimizer, and it is one of the most common optimizers, but there is a lot of room left for more robust and complex stochastic gradient descent optimizers [9].

4.1 Applications

An interesting take on this paper is that a lot of the research and future research is a mix of combinations and practitioners. At the moment there is some disconnect between academics and practitioners. Risk Magazine which is one of the foremost news publications that covers quantitative finance and has their own in-house journal covers the research of rough paths. In one of their publications they state that Societe Generale quants believe that rough volatility will allow practioners to calibrate their volatility surfaces so fast that bid / offer spreads for VIX Futures and options will shrink by 15 - 20 %[6].

The model is still in its infancy. Russell Barker at Morgan Stanley states "For rough volatility, traders cannot yet map in their heads what the model says and how to hedge it or use it to make a profit" [5] Only a handful a firms have began using rough volatility. Philippe Dumont a global equities portfolio manager at Caisse de Dépôt et Placement du Québec (CDPQ) says that there is opportunity to make arbitrage trades using this model, and it Shanghai Luoshu Investment Company, a high frequency multi-strategy quant fund says that profits have increased by 20% when implementing rough volatility. Although they don't go into their process, the most likely scenario is that using the rough paths model they are able to calibrate their volatility surfaces faster than the market maker giving them a small window of price dislocation. Nataxis, a French investment bank backtested a rough volatility arbitrage model across 15 different options market using delta hedging, the model had a Sharpe ratio above 2, which indicates a level of risk-adjusted returns similar to the S&P 500.

[8] The neural network is built in Keras

[9] Stochastic Gradient Descent optimizers are part of a bigger optimizer class of gradient descents that look to find minimums by moving in the direction of the gradient

References

[1] Bao-Quan Ai, Yafeng He, and Wei-Rong Zhong. Transport in periodic potentials induced by fractional gaussian noise. *Physical Review E*, 82:061102, 12 2010.

[2] Christian Bayer, Peter Friz, and Jim Gatheral. Pricing under rough volatility. *Quantitative Finance*, 16(6):887–904, 2016.

[3] Lorenzo Bergomi. Smile dynamics i. *Risk*, 4 2004.

[4] Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. *Journal of Political Economy*, 81(3):637–654, 1973.

[5] Mauro Cesa and Rob Mannix. The volatility paradigm that's stirring up options pricing, Mar 2021.

[6] Kris Devasabai. Rough volatility's steampunk vision of future finance, Aug 2021.

[7] Bruno Dupire, The Black–scholes Model (see Black, and Gives Options. Pricing with a smile. *Risk Magazine*, pages 18–20, 1994.

[8] Steven Heston. A closed-form solution for options with stochastic volatility with applications to bond and currency options. *Review of Financial Studies*, 6:327–343, 1993.

[9] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[10] Blanka Horvath, Aitor Muguruza, and Mehdi Tomas. Deep learning volatility, 2019.

[11] H. E. Hurst. Long-term storage capacity of reservoirs. *Transactions of the American Society of Civil Engineers*, 116(1):770–799, 1951.

[12] Zhongmin Qian Laure Coutin. Stochastic analysis, rough path analysis and fractional brownian motions. *Probability Theory and Related*.

[13] Terry J. Lyons. Differential equations driven by rough signals. *Revista Matemática Iberoamericana*, 14(2):215–310, 1998.