Deep learning volatility

A deep neural network perspective on pricing and calibration in rough volatility models

Mario Correddu

July 8, 2022

Introduction

- goal: option pricing
- Problem: we need regularity properties of stochastic models in order to apply any common option pricing method
- common used models trade accuracy for applicability
- rough volatility suffer from applicability issues (Montecarlo pricing is slow)
- Possible workaround: approximate pricing function using neural networks

Calibration problem

- let's set $\mathcal{M} = \mathcal{M}(\theta)_{\theta\in\Theta}$ an abstract model parametrized by $\theta \in \mathbb{R}^n$
- let $\mathcal{P} : \mathcal{M}(\theta, \zeta) \to \mathbb{R}^m$, $\zeta \in Z'$ be the pricing map, where Z' denotes the financial products we aim to price, such as vanilla options for (a set of) given maturities and strikes.
- given a distance δ, $\hat{\theta} \in$ *Theta* solves a δ-calibration problem for a model $\mathcal{M}(\Theta)$ for the conditions $\mathcal{P}^{MKT}(\zeta)$ if

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

- given a distance δ, $\hat{\theta} \in$ *Theta* solves an approximate δ-calibration problem for a model $\mathcal{M}(\Theta)$ for the conditions $\mathcal{P}^{MKT}(\zeta)$ if

$$\hat{\theta} = \operatorname*{argmin}_{\theta\in\Theta} \delta(\tilde{\mathcal{P}}(\mathcal{M}(\theta), \zeta), \mathcal{P}^{MKT}(\zeta))$$

where $\tilde{\mathcal{P}}$ is numerical approximation of $\mathcal{P}$

Neural Networks

Definition: Neural network

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

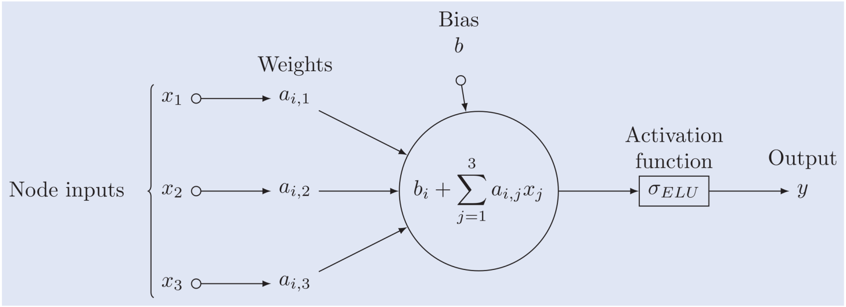
$$w_l : x \to A_{l+1}x + b_{l+1}$$

for $1 \leq l \leq L - 1$, acting between layers N_l and N_{l+1} of layer $l + 1$. Then a Neural Network

$$F(w, \cdot) = F((w_1, ..., w_L), \cdot) : \mathbb{R}^{N_0} \to \mathbb{R}^{N_L}$$

is defined as the composition: $F := F_L \circ \cdots \circ F_1$ 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.

Neural Networks



Neural Networks

Theorem 1
Universal approximation theorem (Hornik et al. 1989)

Let $NN^{\sigma}_{d_0,d_1}$ be the set of neural networks with activation function $\sigma : \mathbb{R} \to \mathbb{R}$, input dimension $d_0 \in \mathbb{N}$ and output dimension $d_1 \in \mathbb{N}$. Then, if σ is continuous and non-constant, $NN^{\sigma}_{d_0,d_1}$ is dense in $L^p(\mu)$ for all finite measures μ.

Neural Networks

Theorem 2
Universal approximation theorem for derivatives (Hornik et al.1990)

Let $F^* \in C^n$ and $F : \mathbb{R}^{d_0} \to \mathbb{R}$ and $NN^{\sigma}_{d_0,1}$ be the set of single-layer neural networks with activation function $\sigma : \mathbb{R} \to \mathbb{R}$, input dimension $d_0 \in N$ and output dimension 1. Then, if the (non-constant) activation function is $\sigma \in C^n(\mathbb{R})$, then $NN^{\sigma}_{d_0,1}$ arbitrarily approximates f and all its derivatives up to order n.

Neural Networks

Theorem 3
Estimation bounds for Neural Networks (Barron 1994)

Let $NN^{\sigma}_{d_0,d_1}$ be the set of single-layer neural networks with Sigmoid activation function $\sigma(x) = \frac{e^x}{e^x+1}$, input dimension $d_0 \in \mathcal{N}$ and output dimension $d_1 \in \mathcal{N}$. Then:

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

where n is the number of nodes, N is the training set size and C_{F^*} is the first absolute moment of the Fourier magnitude distribution of $F*$ meaning if $F^*(x) = \int_{\mathbb{R}^d} e^{i\omega x}\tilde{f}(\omega)d\omega$, then:

$$C_f = \int_{\mathbb{R}^d} |\omega|_1 |\tilde{f}| d\omega$$

Neural Networks

Theorem 4
Power of depth of Neural Networks (Eldan and Shamir 2016)

There exists a simple (approximately radial) function on 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.

Neural networks in finance

- first used a data driven approach train a Neural network to return the price of options given market data
- Problems: meaning of calibrated network parameters are unexplained and issues of generalizability, and traditional paradigms of finance such as no arbitrage are hard to guarantee in the absence of a model.
- second Hernandez proposed to learn an inverse map from the market prices to the model parameters

$$\Pi^{-1} : (\mathcal{P}(\zeta))_{\zeta \in Z'} \to \hat{\theta}$$

- no control on the inverse function and accuracy degrades for out samples out of the training set

Two step approach

We present another approach:

1. We first learn (approximate) the pricing map by a neural network that maps parameters of a stochastic model to pricing functions (or implied volatilities) and we store this map during an off-line training procedure. In other words we learn:

$$\tilde{\Phi}_{NN}(\Theta, \zeta) = \tilde{P}(\mathcal{M}(\Theta, \zeta))$$

2. we calibrate (on-line) the now deterministic approximative learned price map:

$$\theta = \underset{\theta \in \Theta}{\operatorname{argmin}} \, \delta(\tilde{\Phi}_{NN}(\theta, \zeta), P^{MKT}(\zeta))$$

Challenges of the two step approach

Mainly two:

- approximate the pricing function between model error bounds
- need for fast functional evaluation do perform calibration online

Features of the two step approach with NN

- Evaluations of $\tilde{\Phi}_{NN}$ are almost instantaneous and automatic differentiation of $\tilde{\Phi}_{NN}$ with respect to the model parameters returns fast and accurate approximations of the Jacobians.
- the neural network is only used as a computational enhancement of models therefore can understand and interpret the output as a function of model parameters against traditional numerical methods and existing risk management libraries of models remain valid with minimal modification.
- training becomes more robust (with respect to generalisation errors on unseen data).
- training of the network is done once and offline thus can be done without worrying about time and computing resources generating as much synthetical data as needed.
- this approach is applicable to any model that allows a consistent numerical pricer

Pointwise and grid-based training

We train the network to learn the implied volatility surface instead of the price, and using the mean squared error as objective function. We can:

- train the neural network as a function of the strikes and the expiration date, giving the objective function:
  Thus the output of the network is always a real number. The training set is generated by sampling according to a suitable distributions the values of strikes and maturities.
- the grid-based approach consists in learning the entire volatility surface σ_{K_i,T_i} giving the objective function written as
  Thus the output of the network can be interpreted as an image whose entries are indexed by the strikes and expiration times taken into consideration.

Pointwise and grid-based training

We prefer the grid-based approach, justified by the following properties:

- while interpolation in pointwise is done by the network and in the grid-based it has do be done manually, interpolation of the volatility surface is very well understood.
- grid-based approach provides a variance reduction simply because there are less parameters in input
- in the grid-based approach we can generate data by simulating a path and use it for all the different strikes and maturities.

Calibration algorithm

We assume that the pricing map is at least C^1 wrt to its input parameters θ. Thus we can calculate the gradient of the pricing function and deploy standard optimizations algorithms.
We will use the Levenberg–Marquardt method that turned out to outperform every other gradient based method. LM combines the gradient descent method and the Gauss-Newton method, whose parameter update are given by:

$$h_{gd} = \alpha J^T(y - \hat{y})$$

$$\left[J^T J\right] h_{gn} = J^T(y - \hat{y})$$

in the equation:

$$\left(J^T J + \lambda I\right) h_{lm} = J^T(y - \hat{y})$$

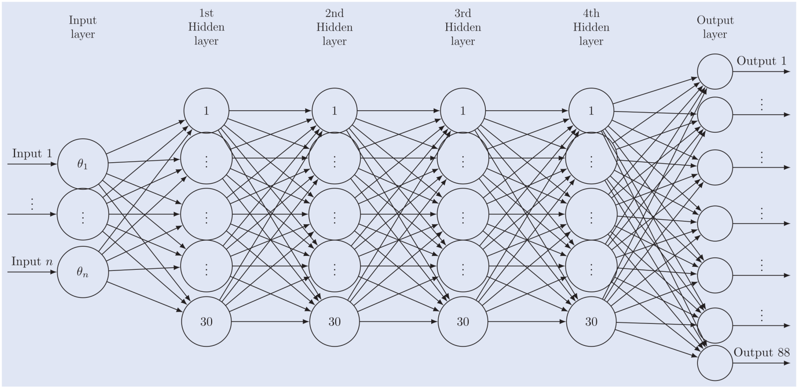
where J is the local Jacobian of the model function $\hat{y}(x; p)$ of an independent variable x and parameters p

Network Architecture

The network consists in:

- A fully connected feed forward neural network with four hidden layers and 30 nodes on each layers
- Input dimension = n, number of model parameters
- Output dimension = 11 strikes $\times$ 8 maturities for this experiment, but this choice of grid can be enriched or modified.
- The four inner layers have 30 nodes each, which adding the corresponding biases results in $30n + 6478$ network parameters.
- $\sigma_{Elu} = \alpha(e^x - 1)$ as the activation function for the network.

Network Architecture



rBergomi model

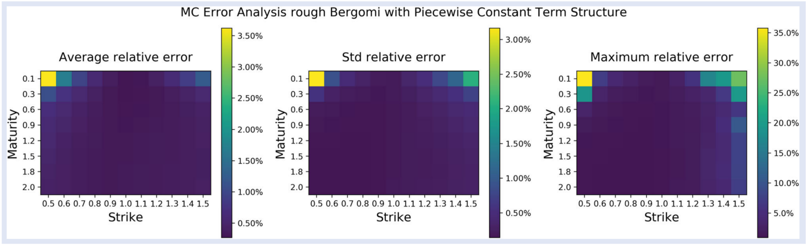
We apply the machine we built to the rBergomi model which is given by the following equation:

$$dX_t = -\frac{1}{2} v_t dt + \sqrt{v_t} dW_t$$

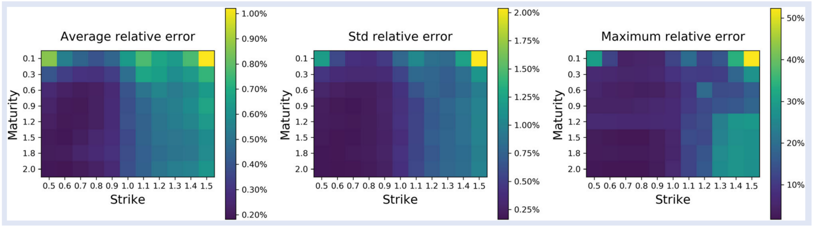
$$v_t = \xi_0(t)\mathcal{E}\left(\sqrt{2H}\nu \int_0^t (t-s)^{H-1/2} dZ_s\right)$$

Where $t > 0$, $X_0 = 0$, $\mathcal{E}()$ is the stochastic exponential, W and Z are two correlated Brownian motions with correlation parameter $o \in [-1, 1]$, H is the Hurst parameter, and $\sqrt{2H}\nu$ denotes the volatility of variance. $\xi_0(t)$ is the forward variance curve, that for our experiments we will approximate with a piecewise constant function.

Numerical experiments

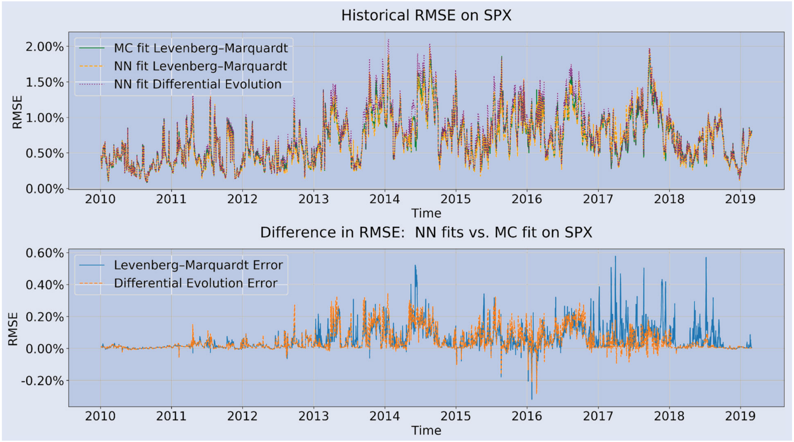


NN Error Analysis rough Bergomi with Piecewise Constant Term Structure



numerical experiments

	MC pricing rBergomi full surface	NN pricing full surface	NN gradient full surface
Piecewise constant forward variance	$500,000\,\mu s$	$30.9\,\mu s$	$113\,\mu s$



Barrier options

- Vanilla options where we parametrize the pricing function using as output the implied volatility surface.
- Digital Barrier level: binary, path-dependent options. The 'out' barrier option pays off at expiry unless at any time previously the underlying asset reaches a level B. While 'in' options give a payoff if the level is reached. We consider down options meaning that the barrier B is below the initial asset value S_0. Then the payoff is given by:

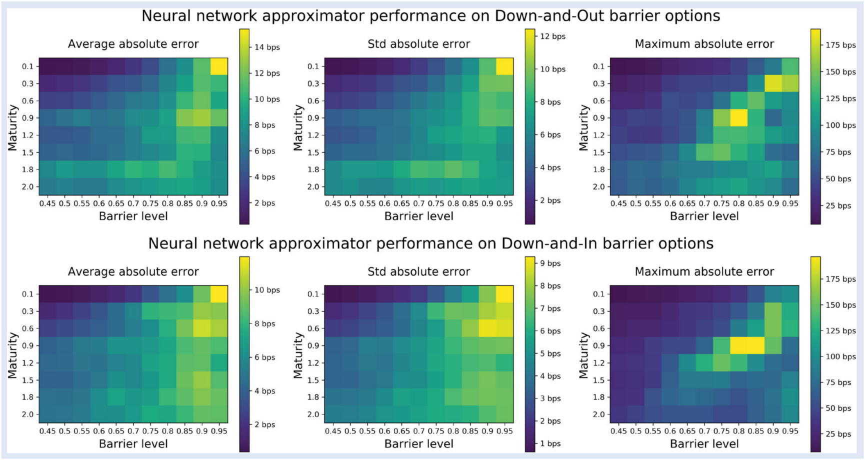
$$\mathcal{P}^{down-and-in} = E[\mathbb{1}_{\{\tau_B \leq T\}}]$$

$$\mathcal{P}^{down-and-out} = E[\mathbb{1}_{\{\tau_B \geq T\}}]$$

where $\tau_B = \inf\{S_t = B\}$. The grid used as output is given by:

$$\Delta^{Barrier} = \{B_i, T_j\} i = 1 \ldots n, j = 1 \ldots m$$

Numerical experiments


Neural network approximator performance on Down-and-Out barrier options
Average absolute error
Std absolute error
Maximum absolute error
Maturity
Barrier level
2 bps 4 bps 6 bps 8 bps 10 bps 12 bps 14 bps
2 bps 4 bps 6 bps 8 bps 10 bps 12 bps
25 bps 50 bps 75 bps 100 bps 125 bps 150 bps 175 bps
Neural network approximator performance on Down-and-In barrier options
Average absolute error
Std absolute error
Maximum absolute error
2 bps 4 bps 6 bps 8 bps 10 bps
1 bps 2 bps 3 bps 4 bps 5 bps 6 bps 7 bps 8 bps 9 bps
25 bps 50 bps 75 bps 100 bps 125 bps 150 bps 175 bps