
NO-ARBITRAGE DEEP CALIBRATION FOR VOLATILITY SMILE AND SKEWNESS

A PREPRINT

Kentaro Hoshisashi^{*†}, Carolyn E. Phelan[†], and Paolo Barucca[†]

[†]Department of Computer Science , University College London , Gower Street, London, WC1E 6BT, UK

January 30, 2024

ABSTRACT

Volatility smile and skewness are two key properties of option prices that are represented by the implied volatility (IV) surface. However, IV surface calibration through nonlinear interpolation is a complex problem due to several factors, including limited input data, low liquidity, and noise. Additionally, the calibrated surface must obey the fundamental financial principle of the absence of arbitrage, which can be modeled by various differential inequalities over the partial derivatives of the option price with respect to the expiration time and the strike price. To address these challenges, we have introduced a Derivative-Constrained Neural Network (DCNN), which is an enhancement of a multilayer perceptron (MLP) that incorporates derivatives in the objective function. DCNN allows us to generate a smooth surface and incorporate the no-arbitrage condition thanks to the derivative terms in the loss function. In numerical experiments, we train the model using prices generated with the SABR model to produce smile and skewness parameters. We carry out different settings to examine the stability of the calibrated model under different conditions. The results show that DCNNs improve the interpolation of the implied volatility surface with smile and skewness by integrating the computation of the derivatives, which are necessary and sufficient no-arbitrage conditions. The developed algorithm also offers practitioners an effective tool for understanding expected market dynamics and managing risk associated with volatility smile and skewness.

Keywords volatility surface, neural networks, deep learning, no-arbitrage constraints, gradient-based learning, partial differential equations, Derivative-Constrained Neural Network

JEL Classification C45, C63, D40, G12

1 Introduction

Implied volatility (IV) is the volatility that is hypothesized to make sense of empirical option prices, i.e. implied volatility is the value of volatility which would result in the market price using the Black-Scholes formula. The standard calibration of the premium surface requires us to find a solution to the Black-Scholes partial differential equation (PDE). Traditional calibration mainly involves minimizing the difference between values predicted by a model and those observed, nevertheless no-arbitrage conditions should ensure that the price of any derivative is fixed at the same level as the value of a replicating portfolio, as shown in Delbaen und Schachermayer (1994). Under these conditions, it is a necessary and sufficient condition that certain derivatives of the option prices should satisfy certain derivative inequalities by Carr und Madan (2005). Taking into account the no-arbitrage conditions enhances the robustness and interpretability of the calibration of the premium surface, in particular in presence of sparse market data.

In such a situation, researchers have been investigating how effective the artificial neural networks (ANNs) approach is applied to the calibration problem in the options market, based on its use as a universal approximator as proved in Cybenko (1989); Hornik u. a. (1989, 1990). In this study, we expand the standard ANNs backpropagation with derivative

^{*}k.hoshisashi@ucl.ac.uk

terms in the loss function and thus incorporate it into the calibration process, enhancing accuracy. Specifically, this study proposes an expansion of Derivative-Constrained Neural Network (DCNN) described in Lo und Huang (2023), which is the extension of a multilayer perceptron (MLP) with Automatic Differentiation (AD) to compute the exact derivatives simultaneously. This approach ensures the network's differentiability, which is essential for representing derivative functions of the original function of MLP, with an expansion of an MLP with reverse AD that generates the first and second derivatives efficiently in Speelpenning (1980). The resulting network has a deep learning architecture that allows the efficient computation of the derivatives and can therefore introduce differential soft constraints for the generated surface.

To evaluate the representation capability of smile and skewness features in IV surfaces as seen in Rubinstein (1985); Corrado und Su (1997), we use one of the stochastic volatility models, the Stochastic Alpha Beta Rho (SABR) model introduced by Hagan u. a. (2002), often used by practitioners. We utilize sparse option premiums generated using the SABR model and evaluate the ability to reproduce premiums and risk profiles which retain the characteristics of volatility smile and skewness. Through our DCNN network, the interpolation of the premium surface is improved, benefiting from the efficient computation of derivatives and the consideration of no-arbitrage conditions.

An effective model necessitates precise calibration of observable data of option premiums. When there is a model that has explanatory power, it is possible to obtain a more accurate probability distribution using historical data, and it will also be possible to perform derivative evaluation and risk management on the same model. Empirically, it has been observed that for models with small pricing errors, the skewness and kurtosis of the unconditional probability distribution implied by the model under the risk-neutral measure are significantly different from those under historical data, as shown in Feldhüter (2016). We also demonstrate that this model serves as a powerful tool for practitioners to understand the expected market dynamics and manage risks with volatility smile and skewness.

2 Background and Literature Review

The effectiveness of ANNs in addressing function approximation problems has been thoroughly researched across several fields. The fields that bear relevance to this study are detailed below.

2.1 Volatility smile and skewness

In the early 1970s, the Black und Scholes (1973) model facilitated the pricing of options based on the assumption that the underlying asset's volatility is constant. However, empirical observations revealed that options with different strike prices actually implied varied volatilities, known as the volatility smile and skewness, as shown in Corrado und Su (1997) and Rubinstein (1985). Numerous local volatility models, such as Dupire u. a. (1994) and Derman u. a. (1996), attempted to account for the reproduction of the static pattern of the smiles. Effective prediction of their dynamics was made possible by the introduction of stochastic volatility models, such as Hull und White (1990); Heston (1993); Hagan u. a. (2002), and Gatheral (2004). One common practical approach is the SABR stochastic volatility model by Hagan u. a. (2002), which employs parameters to characterize the smile and skewness in its stochastic differential equations. However, the dynamics of the volatility smile are still not perfectly represented, necessitating frequent recalibration to align the model with market data. Related to these challenges, the potential of ANNs for solving PDEs has been explored in Barhak und Fischer (2001); Chen u. a. (2018); Khoo u. a. (2021).

2.2 Financial applications of ANNs

Previous research into finding option premiums using ANNs has primarily focused on IV and has often employed adapted global optimization methods, such as Liu u. a. (2019); Cont und Vuletić (2022); Choudhary u. a. (2023). Additionally, there have been applications of advanced network models, such as variational autoencoders in Bergeron u. a. (2022), generative Bayesian models in Jang und Lee (2019), hybrid gated NN in Cao u. a. (2021), and Vol GANs in Cont und Vuletić (2022). Several studies have addressed the calibration problems of option products by penalizing the loss using soft constraints, as shown in Itkin (2019); Ackerer u. a. (2020); Choudhary u. a. (2023); Cont und Vuletić (2022) or mapping pricing from model parameters, such as Bayer u. a. (2019); McGhee (2020); Horvath u. a. (2021). A comprehensive review of ANNs methods for option pricing was conducted in Ruf und Wang (2019).

2.3 Multi-task deep learning

Multi-task deep learning described in Zhang und Zhang (2014); Strezoski und Worring (2017) and multi-objective optimization in Gunantara (2018) address the challenge of weighing multiple loss functions to obtain better performance in Kendall u. a. (2018); Márquez-Neila u. a. (2017) with both soft and/or hard constraints for ANNs. In this context,

the introduction of derivative terms in the loss function to fit full PDEs has been explored in physics-informed neural networks (PINNs) introduced by Raissi u. a. (2019), and DCNNs in Lo und Huang (2023) where isolated derivative terms are considered by Yeh und Cheng (2010); Yao u. a. (2020); Pizarroso u. a. (2020).

3 Calibration Problem

The calibration of volatility surfaces or option prices is an important inverse problem in quantitative finance. In Dupire u. a. (1994), the author has proposed the local volatility model, in which the European options prices satisfy the PDE of the following form with the current price of an underlying asset $S_t > 0$,

$$-\frac{1}{2}\sigma^2(K, \tau)K^2\frac{\partial^2 C}{\partial K^2} + \frac{\partial C}{\partial \tau} + (r - q)K\frac{\partial C}{\partial K} + qC = 0, \quad (1)$$

with initial and boundary conditions given by

$$\begin{aligned} C(K, 0) &= (S_T - K)^+, \\ \lim_{K \rightarrow \infty} C(K, \tau) &= 0, \\ \lim_{K \rightarrow 0} C(K, \tau) &= S_t, \end{aligned} \quad (2)$$

where K is the strike price and τ is the option term to expiry T from valuation date t , and $C = C(K, \tau)$ is the value of the European call option with expiration date T , strike price K with $K, \tau \in [0, \infty)$, a risk-free rate r , and a dividend yield q . The inverse problem of the implied volatility model is that, given limited options prices, we would like to know the premium function $C(K, \tau)$ or an implied (not local) volatility surface, which gives these options prices via the Black-Scholes formula. The challenge of this inverse problem is the scarcity of options price data. To solve this, possibilities are to interpolate/extrapolate the price data or add further information relevant to the problem.

3.1 No-arbitrage constraints of European options

The calibration of option prices is limited by sparse data and, as discussed, should obey the constraints imposed by the no-arbitrage conditions. The no-arbitrage principle posits that market prices prevent guaranteed returns above the risk-free rate. We consider the necessary and sufficient conditions for no-arbitrage, reported in Carr und Madan (2005). In fact, these conditions had been implemented by Ait-Sahalia und Duarte (2003) and were later also used by Roper (2010); Fengler und Hin (2015) as the constraints that our surface needs to satisfy. This allows us to appropriately express the call option price as a two-dimensional surface. The necessary and sufficient conditions for no-arbitrage are represented as non-strict inequalities for several first and second derivatives,

$$-e^{-r\tau} \leq \frac{\partial C}{\partial K} \leq 0, \quad \frac{\partial^2 C}{\partial K^2} \geq 0, \quad \frac{\partial C}{\partial \tau} \geq 0. \quad (3)$$

In Eq. (3), no-arbitrage conditions require these derivatives to have a specific sign. The standard architecture does not automatically satisfy these conditions when calibrating with a loss function simply based on the mean squared error (MSE) for the prices.

3.2 ANNs with No-arbitrage Constraints

Here, we can consider the problem as a calibration problem of an approximated price function defined by the ANNs, $\hat{\Phi}(K, \tau)$. Having obtained each derivative constraint from $\hat{\Phi}$, we can now introduce the total cost function with no-arbitrage constraints for European (call) options to minimise,

$$E(\mathbf{C}, \hat{\Phi}) = E_{\text{MSE}} + E_{\mathcal{P}}, \quad (4)$$

$$E_{\text{MSE}} = \frac{1}{N} \sum_{i=1}^N \left\{ C(K_i, \tau_i) - \hat{\Phi}(K_i, \tau_i) \right\}^2. \quad (5)$$

Here, $C(K_i, \tau_i)$ is the observed premium for the indexed values of strike K_i and time to expiry τ_i , $i = 1, \dots, N$ from the observed dataset. The penalty term $E_{\mathcal{P}}$ represents a score of the total no-arbitrage conditions involving the first and

second derivatives,

$$E_{\mathcal{P}} = \frac{1}{M^{(K)} M^{(\tau)}} \sum_{i=1}^{M^{(K)}} \sum_{j=1}^{M^{(\tau)}} \left\{ \lambda \left(m_K, \frac{\partial \hat{\Phi}(\hat{K}_j, \hat{\tau}_j)}{\partial \hat{K}_j} \right) + \lambda \left(m_{KK}, \frac{\partial^2 \hat{\Phi}(\hat{K}_j, \hat{\tau}_j)}{\partial \hat{K}_j^2} \right) + \lambda \left(m_{\tau}, \frac{\partial \hat{\Phi}(\hat{K}_j, \hat{\tau}_j)}{\partial \hat{\tau}_j} \right) \right\}. \quad (6)$$

The set of \hat{K} and $\hat{\tau}$ values in which we evaluate the derivatives can be obtained with a mesh grid, $j = 1, \dots, M$. The h terms are sign-adjustment coefficients, which make sure that the signs of the penalty terms are correct, $h_K = 1, h_{KK} = -1, h_{\tau} = -1$. $\lambda(m, x)$ is an intensifier of derivative losses in the total cost,

$$\lambda(m, x) = \begin{cases} m \cdot g(x), & \text{if penalty} \\ 0, & \text{if not penalty} \end{cases} \quad (7)$$

where $m \in \mathbb{R}$ are constants and $g(x)$ are intensifier functions, which can be non-linear.

4 Derivative-Constrained Neural Network (DCNN)

This section introduces the Derivative-Constrained Neural Network (DCNN) as an expansion of the work in Lo und Huang (2023), which efficiently computes the partial derivatives of a neural network function with respect to its input features. We consider a simple feed-forward neural network architecture, a multilayer perceptron (MLP). Let $L \geq 2$ be an integer representing the depth of the network; we consider a neural network constructed with one input vector, L hidden layers, and one output value. Both the input values and the output variable are real numbers, i.e., $\mathbf{x} \in \mathbb{R}^n$ and $y \in \mathbb{R}$. We can consider the MLP as a multivariate function Φ depending on the variables \mathbf{x} , i.e. $\Phi : \mathbf{x} \mapsto y$,

$$y = \Phi(\mathbf{x}) = A_L \circ f_{L-1} \circ A_{L-1} \circ \dots \circ f_1 \circ A_1(\mathbf{x}), \quad (8)$$

where for $l = 1, \dots, L$, $A_l : \mathbb{R}^{d_{l-1}} \rightarrow \mathbb{R}^{d_l}$ are affine functions as $A_l(\mathbf{x}_{l-1}) = W_l^T \mathbf{x}_{l-1} + \mathbf{b}_l$, and d_l is the number of neurons in the next layer l for $\mathbf{x}_{l-1} \in \mathbb{R}^{d_{l-1}}$, with $W_l \in \mathbb{R}^{d_{l-1} \times d_l}$ and $\mathbf{b}_l \in \mathbb{R}^{d_l}$, $d_0 = n$, $d_L = 1$, and $\mathbf{x}_0 = \mathbf{x}$. f_l is an activation function which is applied component-wise. Given a dataset \mathbf{X} , which includes a set of pairs $(\mathbf{x}^{(i)}, y^{(i)})$, $i = 1, \dots, N$, and a cost function $E(\mathbf{X}, \Phi)$, the network model Φ is found by fitting the values of W and \mathbf{b} which minimize the cost function.

We consider optimization problems which include losses, not only associated with the network function $\Phi(\mathbf{x})$, but also its derivatives. The total cost includes several terms, which account for the derivatives,

$$E(\mathbf{X}, \hat{\mathbf{X}}, \Phi) := \frac{1}{N} \sum_{i=1}^N \left(y^{(i)} - \Phi(\mathbf{x}^{(i)}) \right)^2 + \frac{1}{M} \sum_{j=1}^M \left\{ \lambda_{m_1}(h_1 \nabla \Phi(\hat{\mathbf{x}}^{(j)})) + \lambda_{m_2}(h_2 \nabla^2 \Phi(\hat{\mathbf{x}}^{(j)})) \right\}, \quad (9)$$

where, $\hat{\mathbf{x}}^{(j)}$ is input features for derivative losses in computational grids $\hat{\mathbf{X}}$ for $j = 1, \dots, M$, and h are vectored signs. ∇ and ∇^2 are partial derivative vectors respectively,

$$\nabla \Phi(\mathbf{x}) = \left[\frac{\partial \Phi}{\partial x_1}, \frac{\partial \Phi}{\partial x_2}, \dots, \frac{\partial \Phi}{\partial x_n} \right]^T, \quad \nabla^2 \Phi(\mathbf{x}) = \left[\frac{\partial^2 \Phi}{\partial x_1^2}, \frac{\partial^2 \Phi}{\partial x_2^2}, \dots, \frac{\partial^2 \Phi}{\partial x_n^2} \right]^T. \quad (10)$$

Using Eq. (9) as the cost function, the optimizer uses gradients with respect to the parameters (i.e., W, \mathbf{b}) for updates. A challenge lies in that Eq. (9) involves derivatives with respect to \mathbf{x} , also functions of the parameters. When numerical approximation of derivatives is used, it could result in slow or inaccurate solutions. To solve this, this study utilizes DCNN, an extended backpropagation algorithm in Rumelhart u. a. (1986) with Automatic Differentiation, for the gradient in Eq. (9) through exact derivative formulations. Following Yeh und Cheng (2010) and Pizarroso u. a. (2020), the first derivatives of a layer in Φ is

$$\nabla_l := \frac{\partial \mathbf{x}_l}{\partial \mathbf{x}} = \{ f'_l \circ A_l(\mathbf{x}_{l-1}) * W_l^T \} \nabla_{l-1}, \quad (11)$$

with $*$ denoting tensor broadcasting in Van Der Walt u. a. (2011). We then compute the first derivative of Φ , i.e., $\nabla \Phi$, by sequentially applying the chain rule in Eq. (11). Note, f' of the last layer and ∇_{l-1} of the first layer aren't applied in

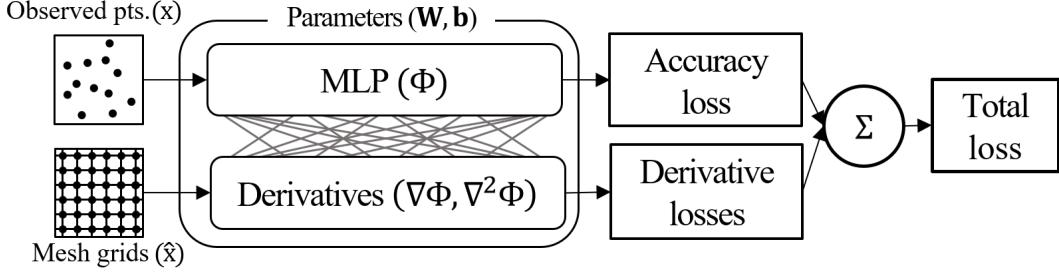


Figure 1: The whole network architecture of Derivative-Constrained Neural Network (DCNN). It is an expansion of a multilayer perceptron (MLP) with additional networks that simultaneously generate its first and second derivatives.

Eq. (11). For the second derivative of Φ , we use the definition $\partial^2\Phi/\partial x^2 = \partial(\partial\Phi/\partial x)/\partial x$. The second derivative of each layer is described as

$$\begin{aligned} \nabla_l^2 := \frac{\partial^2 \mathbf{x}_l}{\partial \mathbf{x}^2} &= f_l'' \circ A_l(\mathbf{x}_{l-1}) * \{W_l^\top \nabla_{l-1}\}^{\circ 2} \\ &\quad + \{f_l' \circ A_l(\mathbf{x}_{l-1}) * W_l^\top\} \nabla_{l-1}^2. \end{aligned} \quad (12)$$

and $\{\cdot\}^{\circ 2}$ is the operation of the Hadamard product for element-wise as described in Reams (1999). Subsequently, $\nabla^2\Phi(\mathbf{x})$ is obtained using Eqs. (11) and (12) with the chain rule. These formulations require MLP activation functions to be second-order differentiable or higher. It is noted that functions like Relu or Elu need slight additional consideration at non-differentiable singular points. If all activation functions are second-order differentiable, the same is true for the whole network as shown in Hornik u. a. (1990).

4.1 Algorithms

The Derivative-Constrained Neural Network (DCNN) algorithm efficiently computes the partial derivatives of a neural network function with respect to its input features. Algorithm 1 exhibits characteristics that set it apart from conventional learning methods. First, the computation points $\hat{\mathbf{X}}$ for the derivatives of the MLP do not correspond with the points of the training dataset \mathbf{X} , which is typically sparse and unbalanced. The algorithm adjusts the derivatives to fit mesh grids, hence capturing derivative data across a wide array of input features. Secondly, the cost function E does not depend only on the MLP's direct output but also on its derivatives as specified in Eq.(9), all of which depend on identical network parameters. DCNN facilitates accurate calibration and gradient computation of the parameters through precise formulations, which consist of a linear transformation and the activation function's derivatives, as described in the previous section.

5 Testing with Synthetic Data

The developed algorithm (i.e. DCNN) for evaluating volatility smile and skewness was first tested on simulated values in a parameterized two-dimensional case of the surface interpolation problem. We took up the well-known Stochastic Alpha Beta Rho (SABR) model introduced by Hagan u. a. (2002) and prepared a sparse two-dimensional dataset to test our methodology. As an empirical experiment, we applied DCNN for the surface interpolation with real market data, which are sparse, and examined the efficiency of the solution for the surface with no-arbitrage constraints.

5.1 The SABR model

The SABR model in Hagan u. a. (2002) is a typical parametric model, which can capture the market volatility smile and skewness and reasonably depict market structure. When F_t is defined as the forward price of an underlying asset at time t , the SABR model is described as

$$\begin{aligned} dF_t &= \alpha_t F_t^\beta dW_t^1, \\ d\alpha_t &= \nu \alpha_t dW_t^2, \\ \langle dW_t^1, dW_t^2 \rangle &= \rho dt. \end{aligned} \quad (13)$$

Here, W_t^1, W_t^2 are standard Wiener processes, α_t is the model volatility, ρ is the correlation between the two processes, and ν is analogous to vol of vol in the Heston model defined in Heston (1993), these are parameters corresponding

Algorithm 1 Derivative-Constrained Neural Network Algorithm

Require: Training dataset of size N $\mathbf{X} : (\mathbf{x}^{(i)}, y^{(i)}), i = 1, \dots, N$

- 1: Mesh grid points of size M $\hat{\mathbf{X}} : \hat{\mathbf{x}}^{(j)}, j = 1, \dots, M$
- 2: An MLP Φ with L layers, $l = 1, \dots, L$, weights W , bias b
- 3: Cost function $E(\mathbf{X}, \hat{\mathbf{X}}, \Phi_{W,b})$
- 4: Derivative function by backpropagation δ
- 5: Optimizer function of the gradient $g(x)$
- 6: Max number of epochs I_{\max}
- 7: Initialize $W, b \leftarrow$ random numbers
- 8: **for** $k \leftarrow 1$ to I_{\max} **do**
- 9: /* Propagate forward MLP in training dataset */
- 10: **for all** training data $(\mathbf{x}, y) \in \mathbf{X}$ **do**
- 11: $e_1 \leftarrow y - \Phi_{W,b}(\mathbf{x})$
- 12: **end for**
- 13: /* Calculate the derivatives of MLP in mesh grid points */
- 14: **for all** mesh grid point $\hat{\mathbf{x}} \in \hat{\mathbf{X}}$ **do**
- 15: $e_2, e_3 \leftarrow \nabla \Phi(\hat{\mathbf{x}}), \nabla^2 \Phi(\hat{\mathbf{x}})$
- 16: **end for**
- 17: /* Calculate the total cost */
- 18: $e_4 \leftarrow E(\mathbf{X}, \hat{\mathbf{X}}, \Phi_{W,b})$
- 19: /* Calculate partial derivatives with respect to parameters
- 20: of the cost function by backpropagation */
- 21: $\frac{\partial E}{\partial W}, \frac{\partial E}{\partial b} \leftarrow \delta_W E, \delta_b E$
- 22: /* Update the weights and bias */
- 23: $W, b \leftarrow W - g\left(\frac{\partial E}{\partial W}\right), b - g\left(\frac{\partial E}{\partial b}\right)$
- 24: **end for**
- 25: **return** Φ, e

to skewness and smiles. The additional parameter β describes the slope of the skewness. A significant feature of the SABR model is that the price of the European option can be formulated in closed form, as shown in Hagan u. a. (2002), up to the accuracy of a series expansion. Essentially, it is shown there that the IV in the SABR model is given by the appropriate formula in Eq (16) from Black (1976). For given α, β, ρ, v and $F = S_t e^{r\tau}$ with a fixed risk-free rate under the risk-neutral measure in Hull (1993), this volatility is given by:

$$\sigma_{\text{SABR}}(K, \tau) = \frac{\alpha \left(1 + \left(\frac{(1-\beta)^2}{24} \frac{\alpha^2}{(FK)^{1-\beta}} + \frac{1}{4} \frac{\rho \beta v \alpha}{(FK)^{(1-\beta)/2}} + \frac{2-3\rho^2}{24} v^2 \right) \tau \right)}{(FK)^{(1-\beta)/2} \left[1 + \frac{(1-\beta)^2}{24} \ln^2 \frac{F}{K} + \frac{(1-\beta)^4}{1920} \ln^4 \frac{F}{K} \right]} \frac{z}{\chi(z)}, \quad (14)$$

$$z = \frac{v}{\alpha} (FK)^{(1-\beta)/2} \ln \frac{F}{K}, \quad \chi(z) = \ln \left(\frac{\sqrt{1-2\rho z + z^2} + z - \rho}{1-\rho} \right).$$

Note as in Hagan u. a. (2002) that if $K = F$ then the z and $\chi(z)$ terms are removed from the equation, as then $\frac{z}{\chi(z)} = 1$ in the sense of a limit, and so

$$\sigma_{\text{SABR}}(F, \tau) = \frac{\alpha \left(1 + \left(\frac{(1-\beta)^2}{24} \frac{\alpha^2}{F^{2-2\beta}} + \frac{1}{4} \frac{\rho \beta v \alpha}{F^{1-\beta}} + \frac{2-3\rho^2}{24} v^2 \right) \tau \right)}{F^{1-\beta}}. \quad (15)$$

Discussion and analysis of parameters setting methodologies for the SABR model with limited input data are also discussed in West (2005).

Finally, option premiums for the experiment are computed from the volatility σ_{SABR} by the Black formula introduced by Black (1976). This is similar to the Black-Scholes formula by Black und Scholes (1973) for valuing options, except that the underlying spot price is replaced by a discounted forward price F . Then, the Black formula states the price for a European call option is

$$C(K, \tau) = e^{-r\tau} [FN(d_1) - KN(d_2)], \quad (16)$$

where

$$d_1 = \frac{\ln(F/K) + (\sigma_{\text{SABR}}^2/2)\tau}{\sigma \sqrt{\tau}}, \quad d_2 = d_1 - \sigma_{\text{SABR}} \sqrt{\tau}, \quad (17)$$

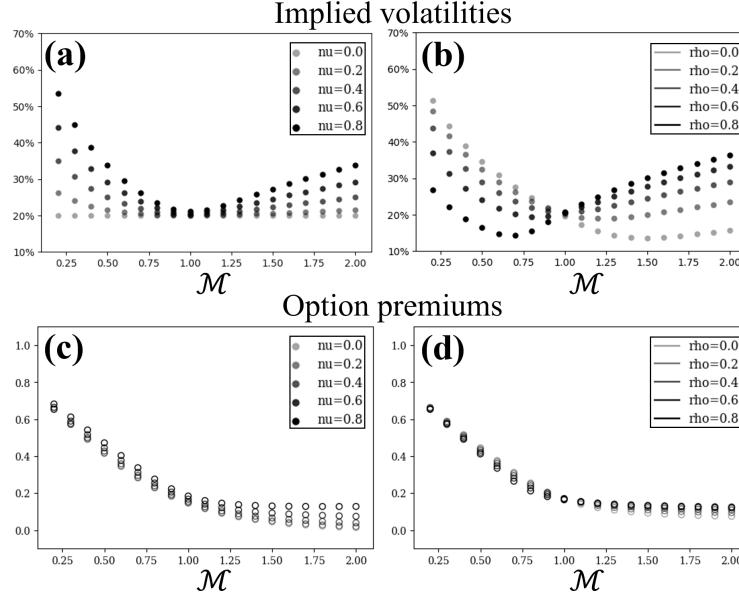


Figure 2: Sparse data of implied volatility (IV) and option (call) premiums using the Stochastic Alpha Beta Rho (SABR) model by moneyness $\mathcal{M} = K/F$. **(a)** The volatilities by smile parameters (ν) with fixed $\rho = 0$. **(b)** By skewness parameters (ρ) with fixed $\nu = 0.6$. **(c-d)** shows the option (call) premiums with corresponding to **(a-b)**.

and $N(\cdot)$ is the cumulative normal distribution function.

It is noted that option characteristics are often encapsulated in IV, but the market variables are option premiums. Figure 2 shows the volatilities and premiums under fixed parameters by $F = 1, \alpha = 0.2, \beta = 1, r = 0.04, \tau = 1$, and $\mathcal{M} = K/F$. By changing smile and skewness parameters, we observe a smaller difference in the numerical values on a premium basis in Figure 2 **(c-d)** than that of volatility in Figure 2 **(a-b)**. Therefore, the calibration task in the actual situation becomes a challenging problem to capture smile and skewness characteristics based on the observed premiums.

5.2 Experimental design

To investigate the ability of ANNs to recreate the volatility smile and skewness, we prepared sparse option premiums for in-sample testing and dense ones for out-of-sample testing based on the SABR model. Sparse option premiums used for in-sample testing are calculated using mesh grids by 25 points for moneyness $\mathcal{M} = K/F \in [0.1, 2.5]$ and 7 for $T \in [0.1, 0.5, 1, 2, 3, 4, 5]$ with varying smile (ν) and skewness (ρ) parameters. Additionally, samples contain the boundary conditions with 200 additional points with $\mathcal{M} \in [0, 2.5]$ and $\tau = 0$ or in $T \in [0, 5]$ and $K = 0$. After that, these premiums are fitted into various models, such as the cubic spline method, MLP, Arbitrage-free smoothing (AFS) by Fengler (2009), and DCNN. For comparison, we evaluate the errors from the predictions using dense mesh girds as out-of-sample, which are equally distributed by 126 points in $\mathcal{M} \in [0, 2.5]$ and 101 points in $T \in [0, 5]$. We use an evaluation metric as followability of smile and skewness as similar to Eq (5),

$$E_{\text{MSE}}^{(\sigma)} = \frac{1}{N} \sum_{i=1}^N \left\{ \sigma_{\text{SABR}}(K_i, \tau_i) - \sigma_{\text{Black}}\left(\hat{\Phi}(K_i, \tau_i), F, K_i, r, \tau_i\right) \right\}^2, \quad (18)$$

where $\sigma_{\text{Black}}(\cdot)$ is the function which computes IV based on the Black model corresponding to the predicted premium $\hat{\Phi}(K_i, \tau_i)$. To fit a more realistic market situation, the following experiment utilized sparse (and uneven) grid data referring to actual historical traded grids from 10th to 14th July 2023 and generates synthetic option premiums on the grids by Eq. 14 as in-sample training data.

The base MLP architecture used in this study consists of two fully connected hidden layers with Softplus activation functions, whose derivatives are $f' : (1 + e^{-x})^{-1}$, $f'' : e^{-x}(1 + e^{-x})^{-2}$ in Eqs. (11) and (12), with an output layer with a linear function (i.e. no activation function). Each layer has 16 neurons. We set the number of epochs as 10,000, and Adam, introduced by Kingma und Ba (2014), as the optimizer using gradient-based training with normally randomised weight initialization. In the cubic spline model, we use the `SmoothBivariateSpline` function with

Table 1: A summary of intraday prices for S&P 500 options.

	10-Jul-23	11-Jul-23	12-Jul-23	13-Jul-23	14-Jul-23
count	2,066	1,930	2,261	1,757	1,963
τ_{\min}	0.005	0.000	0.000	0.000	0.000
τ_{\max}	5.444	4.438	5.433	5.430	5.427
\mathcal{M}_{\min}	0.022	0.022	0.041	0.043	0.042
\mathcal{M}_{\max}	2.312	2.310	2.297	1.939	1.926

three degrees in the Scipy package (Virtanen u. a. (2020)) in Python. Our numerical experiments were run using Pytorch (Paszke u. a. (2019)) and JAX (Bradbury u. a. (2018)) packages for efficient automatic differentiation on Google Colaboratory (Google (2023)) with 36 GB of RAM and a dual-core CPU of 2.3 GHz. We consistently used the same random seeds across different conditions for the statistical analysis and changed these seed values ten times. We set $m_1 = m_3 = 0.001$, $m_2 = 0.01$, $g(x) = x$ in Eq. (6), and $\alpha = 0.2$, $\beta = 1.0$, $q = 0$, and $r = 0.04$ as the parameters of the SABR model. In the implementation of AFS proposed in Fengler (2009), we utilized the Matlab codes provided by the authors for our calculations. Note that AFS is available for a rectangular mesh grid, although empirical data has scattered grids, and does not define interpolation along the τ dimension in the paper. However, we adapted presmoother function from their codebase for interpolation along τ , enabling implementation on our rectangular mesh grid per AFS requirements by eliminating grids at $\tau = 0$ and reducing grids at $K = 0$ compared to other models. The visualizations in the figures of the results section were based on the model outputs when $\nu = 0.6$ and $\rho = -0.4$.

5.3 The S&P 500 dataset

The empirical experiments were conducted using intraday prices for S&P 500 call options from 10th to 14th July 2023; we obtained about two thousand points on a daily basis via the Yahoo finance library as Table 1. We added the synthesized points corresponding to boundary conditions to training, and all other setups are the same as in the previous section. We also conducted the backtests summarized in Appendix D.

5.4 Results

We obtained sparse in-sample data for training and dense out-of-sample data to compare the effectiveness of interpolation with volatility smile and skewness by models: cubic spline, MLP, and DCNN.

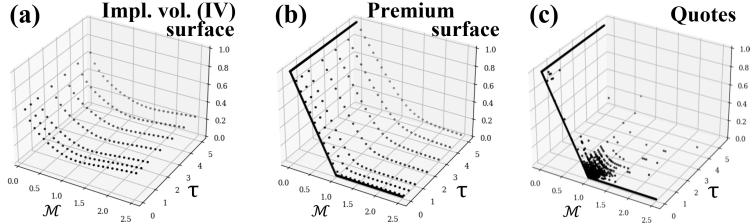


Figure 3: Sparse samples for the training set. (a) The IVs with the fixed SABR parameter ($\nu = 0.6, \rho = -0.4$). (b) the option premiums corresponding to (a) with initial and boundary conditions in Eq.(2). (c) S&P 500 option premiums traded intra-day as of 12th July 2023, with boundary conditions.

At first, we show the issue associated with generating the IV surface near boundary conditions. As shown in 3 (a), we can prepare the whole volatility surface using the SABR model; however, the original option premiums in Figure 3 (c) largely comprise of at-the-money options with a short time to expiry which makes it is hard to identify a complete implied volatility surface from premiums.

Figure 4 (a-d) illustrates the predicted results for option premiums from trained models derived from out-of-sample data. In Figure 4(e-f), we compare the profiles of the IV surface created by the cubic spline method, MLP, AFS, and DCNN. Training times were approximately 30 seconds for a simple MLP and 50 for DCNN. Note again that calibration provided option premium surfaces, which were converted to the implied volatility surface. As a result, the cubic spline method, a simple interpolation approach, reveals the largest area of invalid volatility, signifying a higher IV error. In contrast, more sophisticated techniques, like AFS and particularly DCNN, display fewer errors. Additionally,

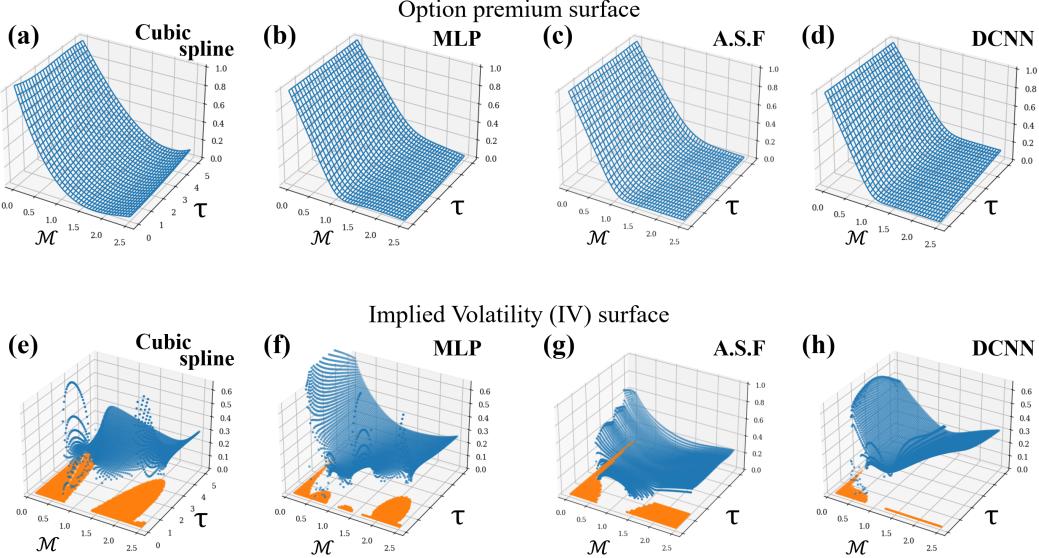


Figure 4: Calibrated models fitted by synthesized option premiums via SABR model (**a-d**), and IV surface completed by those premiums (**e-h**). Orange points show the invalid IVs computed by corresponding to option premiums (**a-c**).

Table 2: Premium, penalty and volatility errors in out-of-sample. **Bold values** indicate lower (better) values among the models.

Conditions		Out-of-sample Errors											
SABR param.		Cubic Spline			MLP			AFS ^a			DCNN		
ν	ρ	E_{MSE} $\times 10^{-4}$	$E_{\mathcal{P}}$ $\times 10^{-3}$	$E_{\text{MSE}}^{(\sigma)}$ $\times 10^{-2}$	E_{MSE} $\times 10^{-4}$	$E_{\mathcal{P}}$ $\times 10^{-3}$	$E_{\text{MSE}}^{(\sigma)}$ $\times 10^{-2}$	E_{MSE} $\times 10^{-4}$	$E_{\mathcal{P}}$ $\times 10^{-3}$	$E_{\text{MSE}}^{(\sigma)}$ $\times 10^{-2}$	E_{MSE} $\times 10^{-4}$	$E_{\mathcal{P}}$ $\times 10^{-3}$	$E_{\text{MSE}}^{(\sigma)}$ $\times 10^{-2}$
0.0	0.0	6.25	10.65	1.49	0.71	0.83	1.66	0.10	0.37	1.14	1.03	0.14	1.21
0.2	0.0	6.97	10.92	2.31	0.91	0.80	1.44	0.11	0.37	1.56	0.65	0.11	0.75
0.4	0.0	7.21	10.82	2.21	1.26	0.65	1.10	0.21	0.23	2.33	0.79	0.08	0.62
0.6	0.0	7.90	10.68	2.06	0.83	0.75	0.70	0.66	0.10	2.90	0.82	0.08	0.63
0.8	0.0	9.38	10.55	2.14	2.37	1.05	0.79	1.86	0.08	3.02	1.26	0.07	0.73
0.6	-0.8	9.90	10.97	3.17	1.13	0.79	1.16	0.99	0.07	3.59	0.75	0.06	0.87
0.6	-0.4	8.58	10.75	2.53	1.50	0.65	0.80	0.91	1.10	3.13	0.67	0.05	0.66
0.6	0.4	7.72	10.72	1.64	1.97	0.83	1.03	0.43	0.90	2.77	0.65	0.10	0.58
0.6	0.8	8.43	10.86	1.32	2.05	0.94	1.17	0.31	5.10	2.22	0.90	0.13	0.67

^a Arbitrage-free smoothing (AFS) refers to the method proposed in Fengler (2009).

DCNN gives the most stable results for IV, indicating that it effectively generates a surface that can represent smile and skewness features, thus characterizing the IV surface more precisely.

To corroborate the assumption, we tabulated the prediction errors in Eqs. (5) and (18), which signify the predicted option premiums and their IV on the mesh grid for out-of-sample data, in contrast with the ideal values calculated by Eq. (14) in the SABR model. These errors are summarized in Table 2. It is clear that the cubic spline method displays the highest errors in both metrics in all cases. In contrast, MLP and DCNN perform similarly with regard to the premium, while DCNN performs better for volatility. This variation in MLP and DCNN performance across different metrics suggests that DCNN is more efficient in recognizing features on the IV surface, likely due to its integration of no-arbitrage constraints.

Next, in Table 3, we also analyzed the errors in Eqs. (5) and (6), representing calibration performance of option premiums and derivatives implied by the no-arbitrage conditions. While MLP offered the best results in the premium metric, DCNN excelled in the risk metric. This difference in the performance of MLP and DCNN across different metrics substantiates the idea that DCNN's superior capability for identifying features on the IV surface is likely due to its integration of no-arbitrage constraints during learning.

Table 3: In-sample errors in accuracy and penalty losses. **Bold values** indicate lower (better) values among the models.

Conditions SABR param.	ν	ρ	In-sample Errors							
			Cubic Spline		MLP		AFS ^a		DCNN	
			E_{MSE} $\times 10^{-4}$	$E_{\mathcal{P}}$ $\times 10^{-3}$						
0.0	0.0	0.0	8.44	415.0	0.47	0.38	0.07	0.60	1.85	0.13
0.2	0.0	0.0	8.44	414.8	0.39	0.32	0.07	0.68	1.64	0.12
0.4	0.0	0.0	8.45	414.1	0.46	0.34	0.16	0.39	1.65	0.11
0.6	0.0	0.0	8.48	412.6	0.45	0.36	0.06	0.39	1.57	0.11
0.8	0.0	0.0	8.55	410.2	0.46	0.35	1.63	1.70	1.48	0.10
0.6	-0.8	0.0	8.59	407.9	0.45	0.34	0.85	1.50	1.42	0.09
0.6	-0.4	0.0	8.59	406.6	0.43	0.33	0.80	1.40	1.36	0.09
0.6	0.4	0.0	8.60	408.4	0.43	0.33	0.36	1.20	1.38	0.09
0.6	0.8	0.0	8.65	411.3	0.46	0.34	0.27	1.50	1.46	0.10
Quotes 10Jul23			5.75	125.9	0.30	0.67	— ^b	— ^b	3.99	0.34
Quotes 11Jul23			5.67	114.2	0.48	0.64	— ^b	— ^b	3.92	0.34
Quotes 12Jul23			5.52	119.4	0.50	0.90	— ^b	— ^b	3.91	0.33
Quotes 13Jul23			6.01	150.2	0.35	0.78	— ^b	— ^b	3.86	0.34
Quotes 14Jul23			6.05	166.6	1.46	1.15	— ^b	— ^b	5.38	0.35

^a Arbitrage-free smoothing (AFS) refers to the method proposed in Fengler (2009).

^b AFS requires a rectangular mesh grid, although empirical data has scattered grids.

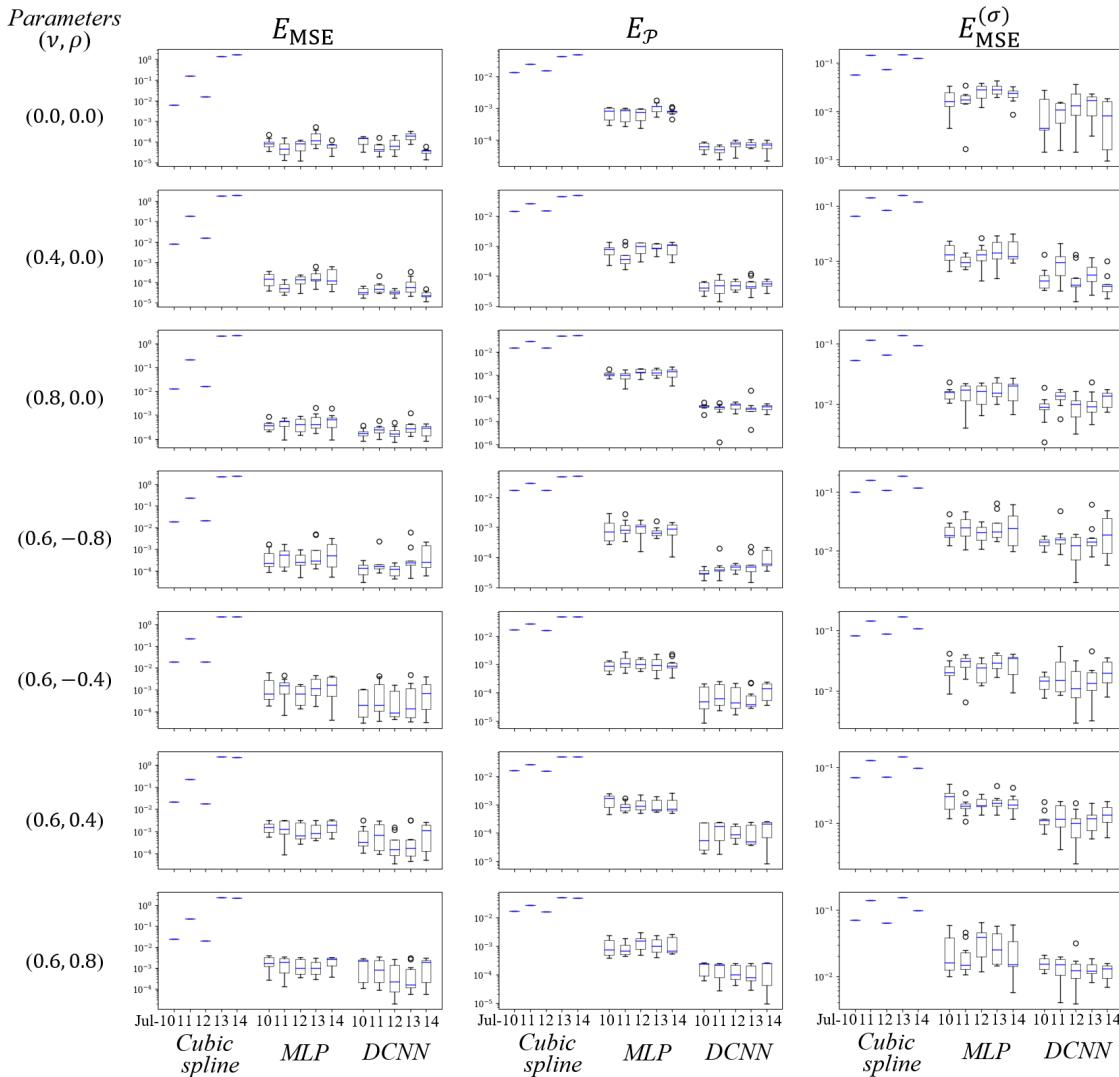


Figure 5: Premium, derivatives penalty and volatility errors with boxplots in out-of-sample of synthetic option premiums on the sparse (and uneven) grids aligned with real market data from 10th to 14th on Jul 2023.

To evaluate the capability in a more realistic market situation, the following experiment in Figures 5 and 6 utilized sparse (and uneven) grid data referring to actual historical traded grids from five days and generated synthetic option premiums on the grids as in-sample training data. In Figure 5 tabulated the prediction errors same as that of Table 2, which signify the predicted option premiums and their IV on the sparse (and uneven) grids for out-of-sample data with dense grids, in contrast with the ideal values in the SABR model. In all test cases, DCNN again showed the best performance with the lowest errors for derivative penalty and implied volatilities across changes in smile and skewness parameters.

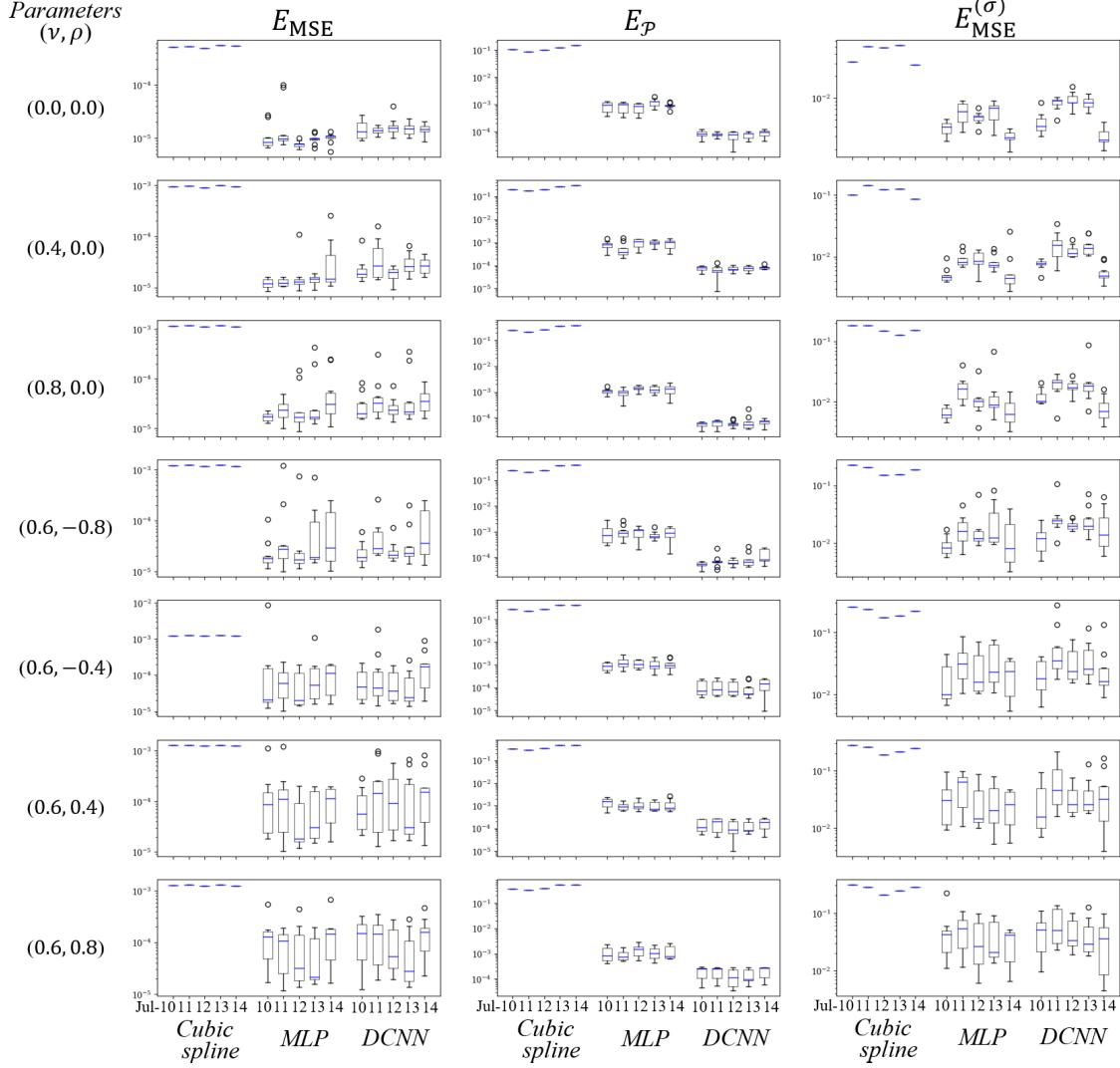


Figure 6: Premium, derivatives penalty and volatility errors with boxplots in in-sample of synthetic option premiums on the sparse (and uneven) grids aligned with real market data from 10th to 14th on Jul 2023.

In Figure 6, Training on these sparse, in-sample premiums resulted in higher losses compared to training on even mesh grids. Among the models, DCNN showed the lowest derivative penalties for all test cases, while MLP was superior in terms of accuracy for premium and volatility predictions, different from out-of-sample results.

These results also suggest that DCNN have a greater capability for interpolating premiums from sparse, real-world trading data compared to the other methods. DCNN provided the most stable results for implied volatility, indicating it can effectively generate a surface capturing the smile and skewness features on the predictions. This demonstrates the DCNN's ability to characterize the implied volatility surface more precisely.

Furthermore, Figure 7 depicts the MSE loss and the penalty loss due to derivative terms across various epochs. A trade-off between solution accuracy and the penalty from derivative terms is visible throughout the learning process. In

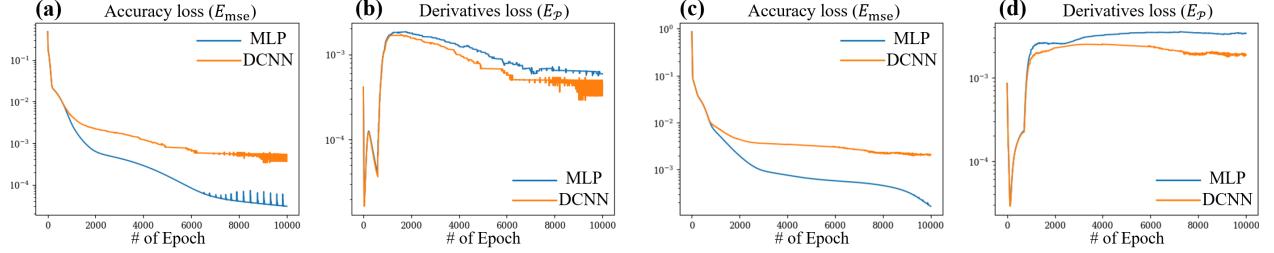


Figure 7: The accuracy (MSE) and loss due to the derivatives in the experiments on synthetic data **(a-b)** and real market data **(c-d)** with a logarithmic scale by learning epochs.

Figure 7**(b)** and **(d)**, a decrease in penalty related to the derivative term is seen for DCNNs. This trend shows a trade-off between accuracy and penalty from derivative terms, brought about by the effort to comply with no-arbitrage constraints during learning.

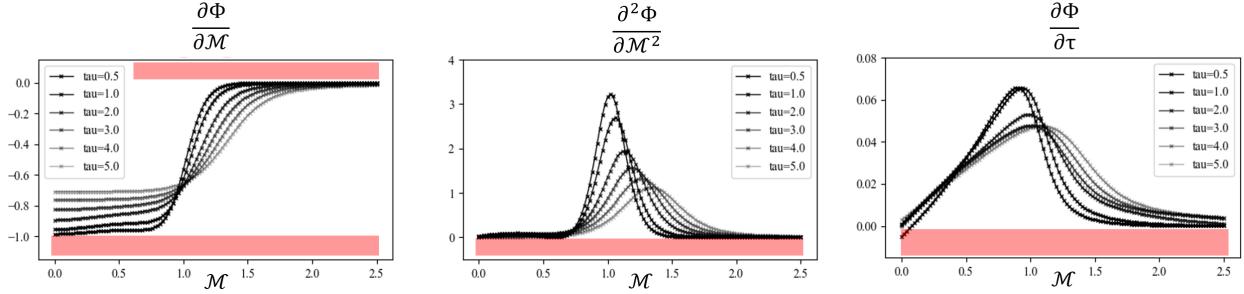


Figure 8: Derivative profiles by DCNN. It shows the first and second differential values of \mathcal{M} (left and centre) and the first differential values of τ (right), and each line with sliced in $\tau \in [0.5, 1.0, 2.0, 3.0, 5.0]$. The colored area indicates breaking the no-arbitrage condition aligned with each inequality in the no-arbitrage constraints in Eq. (3).

In order to know the characteristics of the function surface in detail, Figure 8 shows the first and second differential values of \mathcal{M} (up and center) and the first differential values of τ of the surface in each model surface sliced at $\tau \in [0.5, 1.0, 2.0, 3.0, 5.0]$. In financial terms, each derivative corresponds to risk parameters in orders: *Dual Delta*, *Dual Gamma*, and *Dual Theta* as described in Wystup (2002). We can observe that DCNN successfully prevents the results from breaking conditions for the derivative values, as represented by the shaded areas in Figure 8. Note that DCNN is still a soft constraint approach and does not secure fully no-arbitrage conditions.

Lastly, Figure 9 shows the first and second differentials of predicted values in \mathcal{M} and τ in Eq. (3). DCNN effectively ensures that the results do not infringe on the conditions for derivative values, each of which aligns with an inequality constraint in Eq. (3). In Figure 9, we see that the plots for DCNN have fewer areas in red, denoting the regions where the no-arbitrage conditions on the derivatives are broken. Consequently, these findings prove that DCNN is a powerful tool that aids practitioners not only in understanding market volatility dynamics but also in managing risks effectively.

6 Discussion

The specific contributions of this paper are as follows:

1. The development of a new calibration framework for option premiums: This study utilized the DCNN algorithm, powered by deep learning, which efficiently computes derivatives with automatic differentiation and introduces differential constraints, allowing convergence to the appropriate function surface with accurate detection of volatility characteristics whilst adhering to no-arbitrage conditions.
2. Effectiveness of DCNN in capturing the features of the IV surface: The study utilized the SABR model to assess the ability of DCNN to represent the capability of smile and skewness features in IV surfaces. DCNN significantly improved the interpolation of the premium surface for volatility smile and skewness due to deep learning with no-arbitrage constraints.

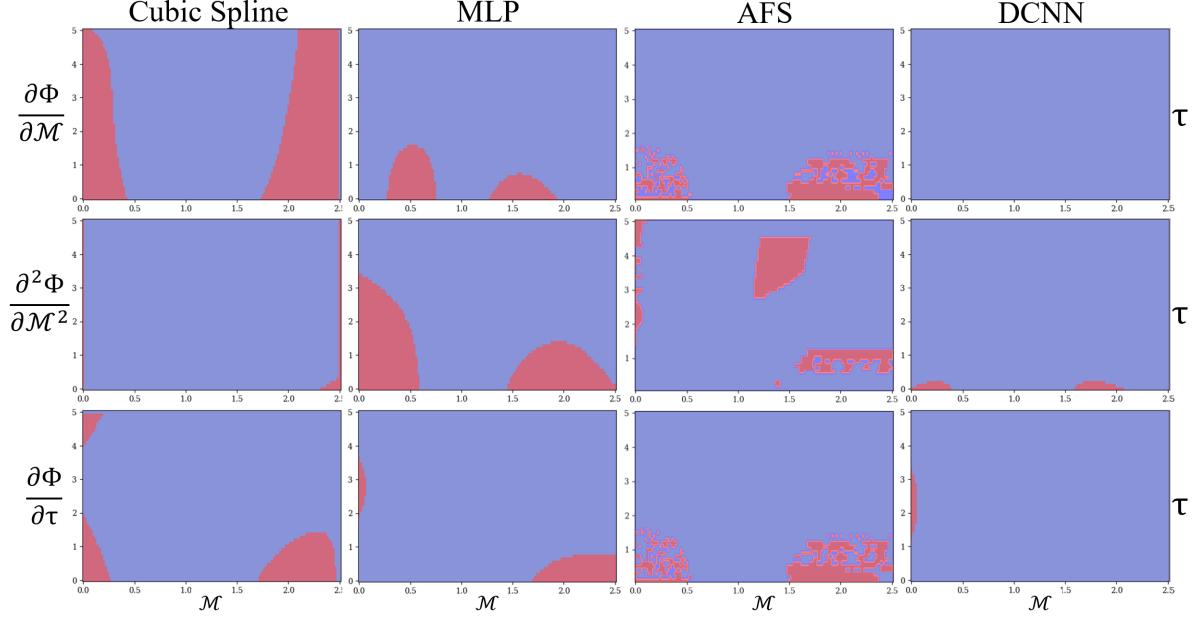


Figure 9: Risk profiles (i.e. derivative terms) comparison by option premium surface. The red-coloured area indicates breaking no-arbitrage conditions.

3. Trade-offs between price accuracy and derivative penalty: The findings demonstrate that as the learning process proceeds, there's a decrease in the penalty associated with the derivative term, reflecting a balance between accuracy and penalty from derivative terms. The results emphasize the network's efforts to comply with no-arbitrage constraints during learning, enhancing the model's overall effectiveness and robustness.

Limitations DCNNs have inherent methodological limitations rooted in their fundamental definitions. Specifically, DCNNs are designed for simple MLPs, which are fully connected feedforward networks based on the universal approximation theorem. DCNNs have not yet been implemented for other network architectures where this theorem's security is not fully guaranteed. Additionally, the main text was unable to definitively prescribe weightings for each loss term in multi-task deep learning contexts. However, we offer one solution using a soft-weighting approach for self-adaptive DCNNs, detailed in Appendix E.

7 Conclusion

This paper presents Derivative-Constrained Neural Networks (DCNN), a neural network algorithm for approximating solutions to partial differential equations. It employs derivative values obtained through automatic differentiation, enhancing the overall accuracy and interpretability of the solution. This includes meeting constraints such as no-arbitrage, capturing volatility smile and skewness effects, and addressing the limitations of traditional calibration methods in scenarios with sparse training data. In our experimental setup, DCNN shows better results in identifying features of smile and skewness on the implied volatility surface, likely due to its incorporation of no-arbitrage constraints during the learning process. This study uses the SABR Stochastic Volatility model to demonstrate the improved interpolation of the premium surface and risk profiles. The findings on both simulated and real data highlight the potential of DCNN as a tool for understanding market dynamics and managing risk. It provides a data-driven solution to calibration problems, allowing for an accurate representation of both option premium and volatility surfaces.

Acknowledgement

The authors are thankful to the Department of Computer Science, University College London, for providing us with the resources to perform this case study.

Supplemental material

The conceptual codes for this study are available for peer review at a private link. The codes would be made publicly available on GitHub prior to the publication of the paper. The datasets generated and/or analyzed during the current study are not publicly available due to privacy and data security considerations but are available from the corresponding author upon reasonable request.

References

- [Ackerer u. a. 2020] ACKERER, Damien ; TAGASOVSKA, Natasa ; VATTER, Thibault: Deep smoothing of the implied volatility surface. In: *Advances in Neural Information Processing Systems* 33 (2020), S. 11552–11563
- [Ait-Sahalia und Duarte 2003] AIT-SAHALIA, Yacine ; DUARTE, Jefferson: Nonparametric option pricing under shape restrictions. In: *Journal of Econometrics* 116 (2003), Nr. 1-2, S. 9–47
- [Barhak und Fischer 2001] BARHAK, Jacob ; FISCHER, Anath: Parameterization and reconstruction from 3D scattered points based on neural network and PDE techniques. In: *IEEE Transactions on visualization and computer graphics* 7 (2001), Nr. 1, S. 1–16
- [Bayer u. a. 2019] BAYER, Christian ; HORVATH, Blanka ; MUGURUZA, Aitor ; STEMPER, Benjamin ; TOMAS, Mehdi: On deep calibration of (rough) stochastic volatility models. In: *arXiv preprint arXiv:1908.08806* (2019)
- [Bergeron u. a. 2022] BERGERON, Maxime ; FUNG, Nicholas ; HULL, John ; POULOS, Zissis ; VENERIS, Andreas: Variational autoencoders: A hands-off approach to volatility. In: *The Journal of Financial Data Science* 4 (2022), Nr. 2, S. 125–138
- [Black 1976] BLACK, Fischer: The pricing of commodity contracts. In: *Journal of financial economics* 3 (1976), Nr. 1-2, S. 167–179
- [Black und Scholes 1973] BLACK, Fischer ; SCHOLES, Myron: The pricing of options and corporate liabilities. In: *Journal of political economy* 81 (1973), Nr. 3, S. 637–654
- [Bradbury u. a. 2018] BRADBURY, James ; FROSTIG, Roy ; HAWKINS, Peter ; JOHNSON, Matthew J. ; LEARY, Chris ; MACLAURIN, Dougal ; NECULA, George ; PASZKE, Adam ; VANDERPLAS, Jake ; WANDERMAN-MILNE, Skye ; ZHANG, Qiao: *JAX: composable transformations of Python+NumPy programs*. 2018. – URL <http://github.com/google/jax>
- [Cao u. a. 2021] CAO, Yi ; LIU, Xiaoquan ; ZHAI, Jia: Option valuation under no-arbitrage constraints with neural networks. In: *European Journal of Operational Research* 293 (2021), Nr. 1, S. 361–374
- [Carr und Madan 2005] CARR, Peter ; MADAN, Dilip B.: A note on sufficient conditions for no arbitrage. In: *Finance Research Letters* 2 (2005), Nr. 3, S. 125–130
- [Chen u. a. 2018] CHEN, Ricky T. ; RUBANOVA, Yulia ; BETTENCOURT, Jesse ; DUVENAUD, David K.: Neural ordinary differential equations. In: *Advances in neural information processing systems* 31 (2018)
- [Choudhary u. a. 2023] CHOUDHARY, Vedant ; JAIMUNGAL, Sebastian ; BERGERON, Maxime: FuNVol: A Multi-Asset Implied Volatility Market Simulator using Functional Principal Components and Neural SDEs. In: *arXiv preprint arXiv:2303.00859* (2023)
- [Cont und Vučetić 2022] CONT, Rama ; VULETIĆ, Milena: Simulation of arbitrage-free implied volatility surfaces. In: *Available at SSRN* (2022)
- [Corrado und Su 1997] CORRADO, Charles J. ; SU, Tie: Implied volatility skews and stock index skewness and kurtosis implied by S&P 500 index option prices. In: *Journal of Derivatives* 4 (1997), Nr. 4, S. 8–19
- [Cybenko 1989] CYBENKO, George: Approximation by superpositions of a sigmoidal function. In: *Mathematics of control, signals and systems* 2 (1989), Nr. 4, S. 303–314
- [Delbaen und Schachermayer 1994] DELBAEN, Freddy ; SCHACHERMAYER, Walter: A general version of the fundamental theorem of asset pricing. In: *Mathematische annalen* 300 (1994), Nr. 1, S. 463–520
- [Derman u. a. 1996] DERMAN, Emanuel ; KANI, Iraj ; ZOU, Joseph Z.: The local volatility surface: Unlocking the information in index option prices. In: *Financial analysts journal* 52 (1996), Nr. 4, S. 25–36
- [Dupire u. a. 1994] DUPIRE, Bruno u. a.: Pricing with a smile. In: *Risk* 7 (1994), Nr. 1, S. 18–20
- [Feldhütter 2016] FELDHÜTTER, Peter: Can affine models match the moments in bond yields? In: *Quarterly Journal of Finance* 6 (2016), Nr. 02, S. 1650009
- [Fengler 2009] FENGLER, Matthias R.: Arbitrage-free smoothing of the implied volatility surface. In: *Quantitative Finance* 9 (2009), Nr. 4, S. 417–428

- [Fengler und Hin 2015] FENGLER, Matthias R. ; HIN, Lin-Yee: Semi-nonparametric estimation of the call-option price surface under strike and time-to-expiry no-arbitrage constraints. In: *Journal of Econometrics* 184 (2015), Nr. 2, S. 242–261
- [Gatheral 2004] GATHERAL, Jim: A parsimonious arbitrage-free implied volatility parameterization with application to the valuation of volatility derivatives. In: *Presentation at Global Derivatives & Risk Management, Madrid* (2004), S. 0
- [Google 2023] GOOGLE: Frequently Asked Questions. Available online:. In: <https://research.google.com/colaboratory/faq.html> (accessed on 24th Jul 2023) (2023)
- [Gunantara 2018] GUNANTARA, Nyoman: A review of multi-objective optimization: Methods and its applications. In: *Cogent Engineering* 5 (2018), Nr. 1, S. 1502242
- [Hagan u. a. 2002] HAGAN, Patrick S. ; KUMAR, Deep ; LESNIEWSKI, Andrew S. ; WOODWARD, Diana E.: Managing smile risk. In: *The Best of Wilmott* 1 (2002), S. 249–296
- [He u. a. 2015] HE, Kaiming ; ZHANG, Xiangyu ; REN, Shaoqing ; SUN, Jian: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: *Proceedings of the IEEE international conference on computer vision*, 2015, S. 1026–1034
- [Heston 1993] HESTON, Steven L.: A closed-form solution for options with stochastic volatility with applications to bond and currency options. In: *The review of financial studies* 6 (1993), Nr. 2, S. 327–343
- [Hornik u. a. 1989] HORNIK, Kurt ; STINCHCOMBE, Maxwell ; WHITE, Halbert: Multilayer feedforward networks are universal approximators. In: *Neural networks* 2 (1989), Nr. 5, S. 359–366
- [Hornik u. a. 1990] HORNIK, Kurt ; STINCHCOMBE, Maxwell ; WHITE, Halbert: Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. In: *Neural networks* 3 (1990), Nr. 5, S. 551–560
- [Horvath u. a. 2021] HORVATH, Blanka ; MUGURUZA, Aitor ; TOMAS, Mehdi: Deep learning volatility: a deep neural network perspective on pricing and calibration in (rough) volatility models. In: *Quantitative Finance* 21 (2021), Nr. 1, S. 11–27
- [Hull 1993] HULL, John: *Options, futures, and other derivative securities*. Bd. 7. Prentice Hall Englewood Cliffs, NJ, 1993
- [Hull und White 1990] HULL, John ; WHITE, Alan: Pricing interest-rate-derivative securities. In: *The review of financial studies* 3 (1990), Nr. 4, S. 573–592
- [Itkin 2019] ITKIN, Andrey: Deep learning calibration of option pricing models: some pitfalls and solutions. In: *arXiv preprint arXiv:1906.03507* (2019)
- [Jang und Lee 2019] JANG, Huisu ; LEE, Jaewook: Generative Bayesian neural network model for risk-neutral pricing of American index options. In: *Quantitative Finance* 19 (2019), Nr. 4, S. 587–603
- [Kendall u. a. 2018] KENDALL, Alex ; GAL, Yarin ; CIPOLLA, Roberto: Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, S. 7482–7491
- [Khoo u. a. 2021] KHOO, Yuehaw ; LU, Jianfeng ; YING, Lexing: Solving parametric PDE problems with artificial neural networks. In: *European Journal of Applied Mathematics* 32 (2021), Nr. 3, S. 421–435
- [Kingma und Ba 2014] KINGMA, Diederik P. ; BA, Jimmy: Adam: A method for stochastic optimization. In: *arXiv preprint arXiv:1412.6980* (2014)
- [Laue u. a. 2020] LAUE, Sören ; MITTERREITER, Matthias ; GIESEN, Joachim: A simple and efficient tensor calculus. In: *Proceedings of the AAAI Conference on Artificial Intelligence* Bd. 34, 2020, S. 4527–4534
- [Li u. a. 2018] LI, Yang ; FAN, Chunxiao ; LI, Yong ; WU, Qiong ; MING, Yue: Improving deep neural network with multiple parametric exponential linear units. In: *Neurocomputing* 301 (2018), S. 11–24
- [Liu u. a. 2019] LIU, Shuaiqiang ; BOROVYKH, Anastasia ; GRZELAK, Lech A. ; OOSTERLEE, Cornelis W.: A neural network-based framework for financial model calibration. In: *Journal of Mathematics in Industry* 9 (2019), S. 1–28
- [Lo und Huang 2023] LO, KaiChieh ; HUANG, Daniel: *On Training Derivative-Constrained Neural Networks*. 2023
- [Márquez-Neila u. a. 2017] MÁRQUEZ-NEILA, Pablo ; SALZMANN, Mathieu ; FUÀ, Pascal: Imposing hard constraints on deep networks: Promises and limitations. In: *arXiv preprint arXiv:1706.02025* (2017)
- [McClenny und Braga-Neto 2020] MCCLENNY, Levi ; BRAGA-NETO, Ulisses: Self-adaptive physics-informed neural networks using a soft attention mechanism. In: *arXiv preprint arXiv:2009.04544* (2020)

- [McGhee 2020] McGHEE, William: An artificial neural network representation of the SABR stochastic volatility model. In: *Journal of Computational Finance* 25 (2020), Nr. 2
- [Paszke u. a. 2019] PASZKE, Adam ; GROSS, Sam ; MASSA, Francisco ; LERER, Adam ; BRADBURY, James ; CHANAN, Gregory ; KILLEEN, Trevor ; LIN, Zeming ; GIMELSHEIN, Natalia ; ANTIGA, Luca ; DESMAISON, Alban ; KOPF, Andreas ; YANG, Edward ; DEVITO, Zachary ; RAISON, Martin ; TEJANI, Alykhan ; CHILAMKURTHY, Sasank ; STEINER, Benoit ; FANG, Lu ; BAI, Junjie ; CHINTALA, Soumith: PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: *Advances in Neural Information Processing Systems* 32. Curran Associates, Inc., 2019, S. 8024–8035. – URL <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>
- [Pizarroso u. a. 2020] PIZARROSO, Jaime ; PORTELA, José ; MUÑOZ, Antonio: NeuralSens: sensitivity analysis of neural networks. In: *arXiv preprint arXiv:2002.11423* (2020)
- [Raissi u. a. 2019] RAISSI, Maziar ; PERDIKARIS, Paris ; KARNIADAKIS, George E.: Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. In: *Journal of Computational physics* 378 (2019), S. 686–707
- [Reams 1999] REAMS, Robert: Hadamard inverses, square roots and products of almost semidefinite matrices. In: *Linear Algebra and its Applications* 288 (1999), S. 35–43
- [Roper 2010] ROPER, Michael: Arbitrage free implied volatility surfaces. In: *preprint* (2010)
- [Rubinstein 1985] RUBINSTEIN, Mark: Nonparametric tests of alternative option pricing models using all reported trades and quotes on the 30 most active CBOE option classes from August 23, 1976 through August 31, 1978. In: *The Journal of Finance* 40 (1985), Nr. 2, S. 455–480
- [Ruf und Wang 2019] RUF, Johannes ; WANG, Weiguan: Neural networks for option pricing and hedging: a literature review. In: *arXiv preprint arXiv:1911.05620* (2019)
- [Rumelhart u. a. 1986] RUMELHART, David E. ; HINTON, Geoffrey E. ; WILLIAMS, Ronald J.: Learning representations by back-propagating errors. In: *nature* 323 (1986), Nr. 6088, S. 533–536
- [Speelpenning 1980] SPEELPENNING, Bert: *Compiling fast partial derivatives of functions given by algorithms*. University of Illinois at Urbana-Champaign, 1980
- [Srivastava u. a. 2014] SRIVASTAVA, Nitish ; HINTON, Geoffrey ; KRIZHEVSKY, Alex ; SUTSKEVER, Ilya ; SALAKHUTDINOV, Ruslan: Dropout: a simple way to prevent neural networks from overfitting. In: *The journal of machine learning research* 15 (2014), Nr. 1, S. 1929–1958
- [Strezoski und Worring 2017] STREZOSKI, Gjorgji ; WORRING, Marcel: Omniart: multi-task deep learning for artistic data analysis. In: *arXiv preprint arXiv:1708.00684* (2017)
- [Van Der Walt u. a. 2011] VAN DER WALT, Stefan ; COLBERT, S C. ; VAROQUAUX, Gael: The NumPy array: a structure for efficient numerical computation. In: *Computing in science & engineering* 13 (2011), Nr. 2, S. 22–30
- [Virtanen u. a. 2020] VIRTANEN, Pauli ; GOMMERS, Ralf ; OLIPHANT, Travis E. ; HABERLAND, Matt ; REDDY, Tyler ; COURNAPEAU, David ; BUROVSKI, Evgeni ; PETERSON, Pearu ; WECKESSER, Warren ; BRIGHT, Jonathan u. a.: SciPy 1.0: fundamental algorithms for scientific computing in Python. In: *Nature methods* 17 (2020), Nr. 3, S. 261–272
- [West 2005] WEST, Graeme: Calibration of the SABR model in illiquid markets. In: *Applied Mathematical Finance* 12 (2005), Nr. 4, S. 371–385
- [Wystup 2002] WYSTUP, Uwe: Vanilla options. In: *Foreign Exchange Risk* (2002), S. 3–14
- [Yao u. a. 2020] YAO, Zhewei ; GHOLAMI, Amir ; KEUTZER, Kurt ; MAHONEY, Michael W.: Pyhessian: Neural networks through the lens of the hessian. In: *2020 IEEE international conference on big data (Big data)* IEEE (Veranst.), 2020, S. 581–590
- [Yeh und Cheng 2010] YEH, I-Cheng ; CHENG, Wei-Lun: First and second order sensitivity analysis of MLP. In: *Neurocomputing* 73 (2010), Nr. 10-12, S. 2225–2233
- [Ying 2019] YING, Xue: An overview of overfitting and its solutions. In: *Journal of physics: Conference series* Bd. 1168 IOP Publishing (Veranst.), 2019, S. 022022
- [Zhang und Zhang 2014] ZHANG, Cha ; ZHANG, Zhengyou: Improving multiview face detection with multi-task deep convolutional neural networks. In: *IEEE Winter Conference on Applications of Computer Vision* IEEE (Veranst.), 2014, S. 1036–1041

Appendices

A The first and second derivatives of Multi-layer Perceptron (MLP)

This section extends the work on the first and second derivatives of MLP to cross derivatives. In second derivative calculations, Eq. (12) can be applied for the selections of second-order partial derivatives, but not for the mixed partial derivatives. The formulations in this section demonstrate how cross-derivatives can be incorporated to extend derivative-constrained neural networks (DCNNs).

Following the notation in Section 4, we can redefine ∇^2 in Eq. (10) with cross derivatives using the Hessian matrix \mathbf{H} ,

$$\nabla^2 \Phi_{W,b}(\mathbf{x}) := \mathbf{H}_\Phi = \begin{bmatrix} \frac{\partial^2 \Phi}{\partial x_1^2} & \frac{\partial^2 \Phi}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 \Phi}{\partial x_1 \partial x_n} \\ \frac{\partial^2 \Phi}{\partial x_2 \partial x_1} & \frac{\partial^2 \Phi}{\partial x_2^2} & \cdots & \frac{\partial^2 \Phi}{\partial x_2 \partial x_n} \\ \vdots & \ddots & \ddots & \vdots \\ \frac{\partial^2 \Phi}{\partial x_n \partial x_1} & \frac{\partial^2 \Phi}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 \Phi}{\partial x_n^2} \end{bmatrix}. \quad (19)$$

Here, the entry of the i -th row and the j -th column is

$$(\mathbf{H}_f)_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}. \quad (20)$$

Similarly, we reconsider the second derivative of each layer $\nabla_l^2 \in \mathbb{R}^{d_l \times n \times n}$ at the l -th layer defined in Eq. (12) with cross derivatives,

$$\nabla_l^2 := \frac{\partial^2 \mathbf{x}_l}{\partial \mathbf{x}^2} = \left[\mathbf{H}_{x_1^{(l)}}, \dots, \mathbf{H}_{x_{d_l}^{(l)}} \right] \quad (21)$$

where, the elements of the Hessian matrix $\mathbf{H}_{x_i^{(l)}}$ are

$$\mathbf{H}_{x_i^{(l)}} = \begin{bmatrix} \frac{\partial^2 x_i^{(l)}}{\partial x_1^2} & \cdots & \frac{\partial^2 x_i^{(l)}}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 x_i^{(l)}}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 x_i^{(l)}}{\partial x_n^2} \end{bmatrix}. \quad (22)$$

Calculating the derivatives of tensor equations, a process known as tensor calculus, is crucial in machine learning. One significant aspect to consider is the efficiency of evaluating these equations and their derivatives, which depends on the way these expressions are represented. Here, Eqs. (11) and (12) - which contain cross derivatives and are characterized by tensor computations in an automatic differentiation framework.

Tensor calculation In tensor calculus Laue u. a. (2020), for tensors A , B , and C the multiplication of A and B can be written as

$$C[s_3] = \sum_{(s_1 \cup s_2) \setminus s_3} A[s_1] \cdot B[s_2] = A *_{(s_1, s_2, s_3)} B \quad (23)$$

where C is the result tensor and s_1 , s_2 , and s_3 are the index sets of the left argument, the right argument, and the result tensor, respectively. The summation index is excluded from the index set of the result tensor $s_3 \subseteq (s_1 \cup s_2)$ explicitly represents the index set of C , which is always a subset of the union of s_1 and s_2 .

Based on the definition provided, tensor multiplication can be described succinctly with fewer summation symbols. Furthermore, this notation closely resembles the tensor multiplication `einsum` found in Python packages. As an illustration, the inner product of matrices A and B can be expressed as

$$A *_{(ik, kj, ij)} B = \sum_k A_{ik} \cdot B_{kj} = A \cdot B. \quad (24)$$

Subsequently, we present a formulation for the first and second derivatives of the function Φ . Note that the product chain rule can be employed between the l -th and $(l-1)$ -th layers, given that each layer is fully connected in the feedforward network.

Initially, we examine the first-order derivatives of each layer's output \mathbf{x}_l with respect to the input features \mathbf{x} . Note that the input features are $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{x} = x_1, \dots, x_n$. The outputs of the l -th layer are denoted as $\mathbf{x}^{(l)} \in \mathbb{R}^{d_l}$, where $\mathbf{x}^{(l)} = x_1^{(l)}, \dots, x_{d_l}^{(l)}$. Consequently, each element of the first derivative at the first layer is formulated as

$$\frac{\partial x_j^{(1)}}{\partial x_i} = (f'_1 \circ A_1(\mathbf{x}))_j \times (W_1^\top)_{ji}. \quad (25)$$

By employing tensor calculation, we can succinctly represent $(\nabla_1)_{ji} = \frac{\partial x_j^{(1)}}{\partial x_i}$ as

$$\nabla_1 = f'_1 \circ A_1(\mathbf{x}) *_{(j,ji,ji)} W_1^\top, \quad (26)$$

For $l = 2, \dots, L-1$, we adhere to the linear algebra notations found in Laue u. a. (2020), and rewrite Eq. (11) using tensor notation with the chain product rule

$$\nabla_l = \{(f'_l \circ A_l(\mathbf{x}_{l-1})) *_{(j,ji,ji)} W_l^\top\} \nabla_{l-1}. \quad (27)$$

∇_L is expressed because the last layer lacks an activation function,

$$\nabla_L = \{A_L(\mathbf{x}_{L-1}) *_{(j,ji,ji)} W_L^\top\} \nabla_{L-1}. \quad (28)$$

Correspondingly, leveraging the above equations, we can formulate the second derivatives with cross derivatives. At this juncture, each element of the second derivative at the first layer is formulated as

$$\mathbf{H}_{x_j^{(1)}} = (f''_1 \circ A_1(\mathbf{x}) *_{(j,ji,ji)} (W_1^\top))_j \otimes (W_1^\top)_j, \quad (29)$$

where \otimes indicates the dyadic product of vectors. Utilizing the tensor calculation, we can represent $(\nabla_1^2)_j = \mathbf{H}_{x_j^{(1)}}$ as

$$\nabla_1^2 = \{f''_1 \circ A_1(\mathbf{x}) *_{(j,ji,ji)} (W_1^\top)\} *_{(ji,jk,jik)} (W_1^\top). \quad (30)$$

When $l = 2, \dots, L-1$, the partial derivatives of the outputs of the l -th layer with respect to \mathbf{x} is calculated using the product rule, i.e. $(u \cdot v)' = u' \cdot v + u \cdot v'$, for Eq. (27) with ∇_{l-1} and ∇_{l-1}^2 as

$$\begin{aligned} \nabla_l^2 &= \{f''_l \circ A_l(\mathbf{x}_{l-1}) *_{(j,ji,ji)} (W_l^\top \nabla_{l-1})\} *_{(ji,jk,jik)} (W_l^\top \nabla_{l-1}) \\ &\quad + \{f'_l \circ A_l(\mathbf{x}_{l-1}) *_{(j,ji,ji)} W_l^\top\} *_{(jm,mik,jik)} \nabla_{l-1}^2, \end{aligned} \quad (31)$$

where f'_l and f''_l are the first and second-order derivatives of the activation function at the l -th layer, respectively. Lastly, the second derivative for the last layer, which is equivalent to $\nabla^2 \Phi(\mathbf{x})$, is obtained as

$$\nabla^2 \Phi(\mathbf{x}) = \nabla_L^2 = \{A_L(\mathbf{x}_{L-1}) *_{(j,ji,ji)} W_L^\top\} *_{(jm,mik,jik)} \nabla_{L-1}^2. \quad (32)$$

The second-order derivatives with cross derivatives of an MLP, as delineated in Eq. (32), can be seamlessly integrated into Algorithm 1. As a natural extension of DCNNs for financial applications, additional cross-derivative sensitivities such as Vanna, Charm, and cross-Gamma could be incorporated for multi-asset products.

B The selection of Network configurations

B.1 Differentiability of activation functions

This section summarizes the network's requirement of the introduced DCNN and compares network configurations to performance for guiding principles for selecting appropriate configurations for its models and applications.

First of all, consideration of the differentiability of the activation function is essential in selecting the activation function of the DCNN. To adapt DCNN, the function of the network requires at least a second differentiable because The derivative formula of the activation function's derivatives, f' , and f'' , are required.

In addition to that, C2 continuity is usually required for almost all problems by DCNN because of the requirement of continuity of the first and second derivative surfaces. In Table 4, The derivatives of the Rectified Linear Unit (ReLU) and Exponential Linear Unit (ELU) functions are undefined at $x = 0$. These are typically set to constant values to prevent errors during the backpropagation process in the vanilla process, which doesn't use DCNN approach. In Table 5, the high-order derivatives with the output of MLP (y) are enumerated for typical activation functions. It is noted that

Table 4: Derivatives functions of the typical activation function. PReLU in He u. a. (2015) and MPELU in Li u. a. (2018) are generalized and unified for ReLU and ELU.

Name	Activation function(f)	First derivatives(f')	Second derivatives(f'')
Sigmoid	$(1 + e^{-x})^{-1}$	$e^{-x}(1 + e^{-x})^{-2}$	$e^{-2x}(1 - e^{-x})(1 + e^{-x})^{-3}$
Softplus	$\ln(e^x + 1)$	$(1 + e^{-x})^{-1}$	$e^{-x}(1 + e^{-x})^{-2}$
Hyperbolic Tangent	$(e^x - e^{-x})(e^x + e^{-x})^{-1}$	$4(e^x + e^{-x})^{-2}$	$-8(e^x - e^{-x})(e^x + e^{-x})^{-3}$
ReLU (PReLU)	$\begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ ax & \text{if } x < 0 \end{cases}$	$\begin{cases} 1 & \text{if } x > 0 \\ \text{not defined} & \text{if } x = 0 \\ a & \text{if } x < 0 \end{cases}$	$\begin{cases} 0 & \text{if } x \neq 0 \\ \text{not defined} & \text{if } x = 0 \end{cases}$
ELU (MPELU)	$\begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ \alpha(e^{\beta x} - 1) & \text{if } x < 0 \end{cases}$	$\begin{cases} 1 & \text{if } x > 0 \\ \text{not defined} & \text{if } x = 0 \\ \alpha\beta^2 e^{\beta x} & \text{if } x < 0 \end{cases}$	$\begin{cases} 0 & \text{if } x > 0 \\ \text{not defined} & \text{if } x = 0 \\ \alpha\beta^2 e^{\beta x} & \text{if } x < 0 \end{cases}$

Table 5: Derivatives functions of the output in the typical activation function. The output is defined as $y = f(x)$ in Table 4. PReLUHe u. a. (2015) and MPELULi u. a. (2018) are generalized and unified for well-known ReLU and ELU.

Name	First derivatives(f')	Second derivatives(f'')	Third derivatives
Sigmoid	$y(1 - y)$	$y(1 - y)(1 - 2y)$	$y(1 - y)(1 - 6y + 6y^2)$
Softplus	$1 - e^{-y}$	$e^{-y}(1 - e^{-y})$	$e^{-2y}(1 - e^{-y})(2 - e^y)$
Hyperbolic Tangent	$1 - y^2$	$-2y(1 - y^2)$	$(1 - y^2)(6y^2 - 2)$
ReLU (PReLU)	$\begin{cases} 1 & \text{if } y > 0 \\ \text{not defined} & \text{if } y = 0 \\ a & \text{if } y < 0 \end{cases}$	$\begin{cases} 0 & \text{if } y \neq 0 \\ \text{not defined} & \text{if } y = 0 \end{cases}$	$\begin{cases} 0 & \text{if } y \neq 0 \\ \text{not defined} & \text{if } y = 0 \end{cases}$
ELU (MPELU)	$\begin{cases} 1 & \text{if } y > 0 \\ \text{not defined} & \text{if } y = 0 \\ \beta(y + \alpha) & \text{if } y < 0 \end{cases}$	$\begin{cases} 0 & \text{if } y > 0 \\ \text{not defined} & \text{if } y = 0 \\ \beta^2(y + \alpha) & \text{if } y < 0 \end{cases}$	$\begin{cases} 0 & \text{if } y > 0 \\ \text{not defined} & \text{if } y = 0 \\ \beta^3(y + \alpha) & \text{if } y < 0 \end{cases}$

the backpropagation with DCNN requires additional settings, which is a higher-order differentiable activation function by reverse-mode of automatic differentiation as shown in the third derivatives in Table 5.

Finally, adapting DCNN to include derivative information necessitates two careful considerations. First, we need to ensure the representational capability of the derivative surface, which implies that the selected activation function should be adept at approximating the shape and range of the derivative surface using its own derivative function. Second, we must confirm the existence of a higher-order derivative of the derivative surfaces for the backpropagation in gradient-based learning. One of our findings suggests that DCNN shows a trade-off in the efficiency and accuracy of the learning, inferring that the efficient changes in performance are reflected by alterations in the property of higher-order derivatives of activation functions.

B.2 Comparison of network configurations

The architecture and hyperparameters used to configure neural networks impact their performance on machine learning tasks. Building on the property analysis, this section provides a comparative functional assessment focused on model accuracy, i.e., parameterization impact of networks on predictive performance.

Table 6: Performance comparison of each error by the configurations of neural networks and packages, displayed as mean values with standard deviations. The other settings are similar to that in Section 5.2^a.

Network configuration (# of)				MLP				DCNN			
Act. func. (f)	Layers (L)	Neurons (d_l)	Param. (W, b)	E_{MSE} $\times 10^{-4}$	$E_{\text{MSE}}^{(\sigma)}$ $\times 10^{-2}$	$E_{\mathcal{P}}$ $\times 10^{-3}$	E_{MSE} $\times 10^{-4}$	$E_{\text{MSE}}^{(\sigma)}$ $\times 10^{-2}$	$E_{\mathcal{P}}$ $\times 10^{-3}$		
elu	3	16	337	0.17 (0.06)	0.43 (0.13)	0.24 (0.14)	0.33 (0.18)	0.37 (0.15)	0.03 (0.01)		
elu	5	16	881	0.16 (0.16)	0.38 (0.23)	0.45 (0.15)	0.19 (0.07)	0.37 (0.18)	0.03 (0.01)		
elu	9	16	1,969	0.18 (0.13)	0.49 (0.23)	1.64 (0.37)	0.93 (1.35)	0.63 (0.36)	0.04 (0.02)		
elu	3	64	4,417	0.25 (0.10)	0.44 (0.09)	0.29 (0.13)	0.37 (0.32)	0.49 (0.31)	0.03 (0.01)		
elu	5	64	12,737	18.48 (44.52)	2.68 (4.51)	0.98 (0.82)	4.17 (7.85)	0.85 (0.74)	0.05 (0.03)		
elu	3	128	17,025	3.16 (5.58)	1.07 (1.15)	0.34 (0.16)	1.12 (1.30)	0.70 (0.43)	0.06 (0.04)		
elu	9	64	29,377	1231.99 (671.56)	51.60 (23.49)	0.46 (0.90)	782.23 (743.31)	34.62 (26.99)	0.51 (0.83)		
elu	5	128	50,049	167.39 (460.94)	8.23 (17.56)	0.89 (0.97)	37.20 (62.49)	4.44 (4.93)	0.26 (0.31)		
elu	9	128	116,097	1279.89 (484.98)	53.70 (19.79)	24.44 (69.17)	1420.40 (527.32)	59.24 (23.09)	2.06 (6.03)		
softplus	3	16	337	2.04 (3.68)	0.99 (0.72)	0.85 (0.69)	1.45 (2.11)	0.73 (0.44)	0.07 (0.04)		
softplus	5	16	881	0.04 (0.01)	0.15 (0.09)	0.84 (0.15)	0.14 (0.09)	0.35 (0.22)	0.02 (0.00)		
softplus	9	16	1,969	0.49 (0.54)	0.44 (0.37)	1.35 (0.27)	0.69 (0.48)	0.76 (0.27)	0.03 (0.01)		
softplus	3	64	4,417	0.08 (0.01)	0.31 (0.17)	0.24 (0.04)	0.30 (0.39)	0.33 (0.17)	0.03 (0.01)		
softplus	5	64	12,737	0.63 (1.75)	0.38 (0.48)	1.46 (1.21)	0.09 (0.04)	0.25 (0.16)	0.02 (0.01)		
softplus	3	128	17,025	0.63 (1.14)	0.52 (0.36)	0.16 (0.06)	0.27 (0.08)	0.36 (0.11)	0.02 (0.00)		
softplus	9	64	29,377	1240.29 (618.95)	48.65 (23.80)	0.54 (1.09)	1236.91 (618.34)	48.43 (23.92)	0.01 (0.01)		
softplus	5	128	50,049	0.63 (1.07)	0.48 (0.40)	2.52 (0.84)	0.42 (0.62)	0.50 (0.39)	0.04 (0.02)		
softplus	9	128	116,097	1549.45 (0.75)	60.53 (0.03)	0.00 (0.00)	1548.61 (1.09)	60.49 (0.06)	0.00 (0.00)		
tanh	3	16	337	0.19 (0.22)	0.47 (0.27)	0.82 (0.42)	0.14 (0.09)	0.37 (0.18)	0.03 (0.01)		
tanh	5	16	881	0.16 (0.31)	0.47 (0.49)	1.22 (0.21)	0.17 (0.10)	0.54 (0.42)	0.02 (0.01)		
tanh	9	16	1,969	0.21 (0.18)	0.48 (0.25)	2.89 (0.69)	1.05 (1.12)	1.05 (0.76)	0.05 (0.03)		
tanh	3	64	4,417	1.38 (2.34)	0.85 (0.79)	1.58 (0.46)	0.17 (0.12)	0.52 (0.38)	0.02 (0.01)		
tanh	5	64	12,737	161.35 (462.66)	7.57 (17.72)	232.18 (333.81)	174.40 (491.32)	8.21 (18.66)	0.27 (0.26)		
tanh	3	128	17,025	175.21 (516.90)	6.51 (15.99)	3.42 (1.75)	4.32 (8.50)	1.07 (0.71)	0.07 (0.05)		
tanh	9	64	29,377	1423.25 (246.61)	55.63 (13.42)	74.81 (224.43)	1606.31 (125.70)	62.60 (4.59)	0.00 (0.00)		
tanh	5	128	50,049	902.03 (737.56)	35.80 (27.55)	51.63 (119.59)	906.51 (738.74)	36.15 (27.83)	0.20 (0.45)		
tanh	9	128	116,097	1701.27 (303.26)	68.96 (16.89)	0.00 (0.00)	1459.35 (138.41)	56.34 (6.72)	0.00 (0.00)		

^a Experimental conditons are $N = 575$, $M = 286$ and 10,000 epochs repeated 10 times in the synthesized data with $\rho = -0.4$ and $\nu = 0.6$ in the SABR model.

^b In the case of the ELU function, $\alpha = \beta = 1$ as applied in Table 4.

Table 6 compares the predictive performance of MLP and DCNN models under various architectural configurations. Increasing depth and width generally decreases MSE up to a point before overfitting occurs. Softplus activation tends to provide the lowest errors for shallow networks. Tanh produces the lowest errors for medium depth/width ELU activation, resulting in some instability for deeper models. The analysis shows the importance of tuning network configuration and activation functions jointly to optimize performance and stability. Overall, guidelines indicate that 2-4 layers with 16-64 units per layer work well across functions; deeper models require more careful activation selection.

Overfitting manifests in very high volume parameter settings, an inherent challenge posing barriers to generalization. As explored by Ying (2019), some regularization methods can give improvements to prevent units from co-adapting too much in such overfitting. One idea of the solution for the overfitting is to randomly drop units (along with their connections) from the neural network during training, introduced in Srivastava u. a. (2014).

The contrasts along dimensions of properties and functional performance provide valuable insights to inform appropriate activation function selection for different model architectures and machine learning problem settings. Overall there remain ample opportunities to refine activation choices and improve generalization across network complexities when applying neural networks to financial pricing.

C Computational complexity

To evaluate the stability of the DCNN under various configurations, we conducted a comparative analysis of computational complexity across different model dimensions and software implementations. Specifically, we assessed how factors including layer depth, neuron width, activation functions, and framework choice impact processing time. We contrasted the DCNN, which incorporates analytical derivative calculations, against a standard MLP baseline without derivative consideration.

Table 7: Comparison of computational time by the configurations of neural networks and packages, displayed as mean values with standard deviation in seconds. The other settings is similar to that in Section 5.2^a.

Layers (L)	Network configuration (# of)			MLP(Pytorch)		DCNN(Pytorch)		MLP(JAX)		DCNN(JAX)	
	Neurons (d_l)	Act. func. (f)	Param. (W, b)	mean	(std)	mean	(std)	mean	(std)	mean	(std)
3	16	softplus	337	35.21	(0.34)	69.57	(0.67)	9.23	(0.45)	11.75	(0.59)
3	16	tanh	337	29.76	(0.59)	63.12	(0.37)	9.54	(1.74)	12.16	(0.53)
3	16	elu	337	33.35	(0.12)	68.90	(1.61)	9.79	(1.69)	12.40	(0.46)
3	64	softplus	4,417	62.10	(0.18)	152.36	(0.55)	9.32	(0.69)	11.97	(0.48)
3	64	tanh	4,417	47.85	(0.48)	137.84	(1.97)	9.52	(1.45)	12.00	(0.39)
3	64	elu	4,417	59.47	(2.41)	159.02	(8.75)	9.74	(1.47)	12.14	(0.68)
3	128	softplus	17,025	102.56	(0.78)	368.57	(5.96)	9.15	(0.61)	12.14	(0.54)
3	128	tanh	17,025	85.68	(0.96)	341.05	(5.38)	9.43	(1.29)	11.94	(0.62)
3	128	elu	17,025	98.83	(0.52)	372.14	(2.94)	9.62	(1.48)	12.14	(0.60)
5	16	softplus	881	68.13	(0.19)	148.75	(0.38)	11.76	(0.57)	17.90	(0.61)
5	16	tanh	881	56.19	(0.21)	135.05	(0.29)	12.05	(0.38)	17.15	(0.40)
5	16	elu	881	63.75	(0.30)	143.70	(0.56)	12.17	(1.44)	18.02	(0.47)
5	64	softplus	12,737	133.15	(0.78)	381.98	(2.24)	12.57	(1.54)	16.97	(0.42)
5	64	tanh	12,737	102.64	(0.30)	344.19	(2.87)	12.29	(1.46)	17.06	(0.48)
5	64	elu	12,737	125.98	(1.60)	385.76	(2.51)	12.42	(1.38)	17.28	(0.40)
5	128	softplus	50,049	245.30	(2.50)	1,021.43	(7.08)	12.44	(1.39)	17.51	(0.51)
5	128	tanh	50,049	212.71	(3.85)	948.21	(9.75)	11.80	(1.58)	17.62	(0.50)
5	128	elu	50,049	248.08	(4.03)	1,020.47	(5.92)	12.27	(0.55)	17.56	(0.49)
9	16	softplus	1,969	133.77	(0.26)	305.85	(0.30)	16.36	(0.42)	28.26	(0.59)
9	16	tanh	1,969	109.51	(0.21)	278.17	(0.35)	16.51	(1.62)	27.99	(0.60)
9	16	elu	1,969	124.08	(0.50)	294.36	(0.36)	15.17	(0.47)	25.28	(0.83)
9	64	softplus	29,377	287.31	(12.13)	897.94	(41.08)	17.06	(0.45)	26.80	(0.87)
9	64	tanh	29,377	212.87	(1.70)	762.05	(5.53)	17.14	(1.42)	27.22	(0.93)
9	64	elu	29,377	269.09	(8.48)	871.80	(26.58)	15.91	(0.87)	26.67	(0.91)
9	128	softplus	116,097	541.15	(16.32)	2,252.24	(28.87)	17.50	(1.52)	29.21	(0.59)
9	128	tanh	116,097	485.25	(24.02)	2,260.50	(106.55)	16.72	(1.77)	28.30	(0.45)
9	128	elu	116,097	554.98	(30.61)	2,436.35	(135.17)	15.88	(0.61)	28.22	(0.84)

^a Experimental conditons are $N = 575$, $M = 286$ and 10,000 epochs repeated 10 times in the synthesized data with $\rho = -0.4$ and $\nu = 0.6$ in the SABR model.

^b In the case of the ELU function, $\alpha = \beta = 1$ as applied in Table 4.

The results in Table 7 demonstrate substantially stable overhead for the DCNN versus MLP, with the incorporation of derivative information consistently increasing computation by approximately 3-4 times. This aligned with the additional mathematical operations required for derivative computations per Eqs. (9) and (10). Meanwhile, standard neural network stacking entailed exponential complexity growth with expanding width and depth.

The comparative assessment also revealed pronounced performance advantages conferred by the JAX framework leveraging just-in-time compilation and GPU acceleration over PyTorch. Across most configurations, JAX reduced execution time by three to four times. For larger models, speedups reached up to 20 times, likely due to superior numerical stability from optimized code generation targeting parallel hardware. Diverging convergence behaviour on expansive models also suggests JAX may confer superior numerical stability. The consistent Derivative-Constrained Neural Network profile with model size makes this architecture appealing for extensible large-scale machine learning applications compared to standard multilayer perceptron performance degradation.

Based on the analysis, mid-sized DCNN models with 2-4 layers and 16-64 units balance predictive accuracy and feasible computation in under five seconds. Larger models should leverage JAX for optimal efficiency and stability. Overall, DCNN presents a realistic tool for practitioners through automatic differentiation, enabling controlled derivative computation alongside the utilization of a recent, efficient programming package.

D Backtests of empirical data from 1 October 2022 to 30 September 2023

This section conducts robust backtesting on extensive intraday options data for DCNN with the dataset on S&P 500 options spanning the period from 2022 to 2023. To investigate the efficiency of the DCNN for the historical data, the backtests were conducted using intraday traded prices of S&P 500 options for 248 days from 1 October 2022 to 30 September 2023; we obtained about 500,000 points on a daily basis via CBOE DataShop² as summarized in Table 8. We added the synthesized points corresponding to boundary conditions to training (note not for a statistics analysis), and all other setups are the same as in 5.2.

Table 8: A statistics of intraday prices of S&P 500 options for 248 business days from 1 October 2022 to 30 September 2023.

[per day]	Mean	(std. dev.)	Median	(min.)	(max.)
all count of intraday traded price	527,646	(71,116)	528,771	(183,418)	(734,441)
# of unique grids	6,516	(530)	6,458	(4,580)	(8,269)
% of call options	40.20 %	(2.02 %)	40.01 %	(35.16 %)	(47.09 %)
% of short term ($\tau < 1M$)	95.47 %	(0.93 %)	95.61 %	(91.40 %)	(97.07 %)
% of long term ($\tau > 1Y$)	2.69 %	(0.52 %)	2.62 %	(1.42 %)	(4.32 %)
% of near ATM ($M \in [0.9, 1.1]$)	71.97 %	(2.66 %)	72.10 %	(64.43 %)	(76.95 %)
% of far OTM ($M \notin [0.5, 1.5]$)	2.00 %	(0.33 %)	1.99 %	(1.17 %)	(3.05 %)

Analysis of the data presented in Table 8 reveals that intraday traded data exhibits an uneven distribution in the input variables. Approximately 95% of trades occur in short-dated options (within 1 month), while over 70% are concentrated around at-the-money (ATM) strikes. This introduces a relatively challenging calibration problem compared to more general cases. Effective modelling needs flexible interpolation ability for sparsely populated points alongside the incorporation of additional mesh information encoding derivative constraints within the DCNN framework.

Results in Figure 10 demonstrate the flexibility of DCNNs to accurately fit both the cross-section of market prices as well as fluctuations over time. The requirements for an efficient calibration are reconciling the differing distributions implied under risk-neutral and real-world measures while capturing higher moments like skewness and kurtosis.

When there is a model that has explanatory power for both probability measures, it is possible to obtain a more accurate probability distribution using historical data, and it will also be possible to perform derivative evaluation and risk management in a consistent manner. However, in the formulation based on historical data, there is a degree of freedom in determining the so-called market price of risk. In the empirical backtests in this section, we observed the DCNNs show stability and efficiency in the errors of whole periods compared with other models. The results support that DCNNs surgically incorporate derivative information in the loss function to enhance pricing and risk estimations without historical statistical assumptions, or parameters of stochastic models.

²CBOE DataShop. (2023). SPX options. <https://datashop.cboe.com/option-trades>. [dataset]

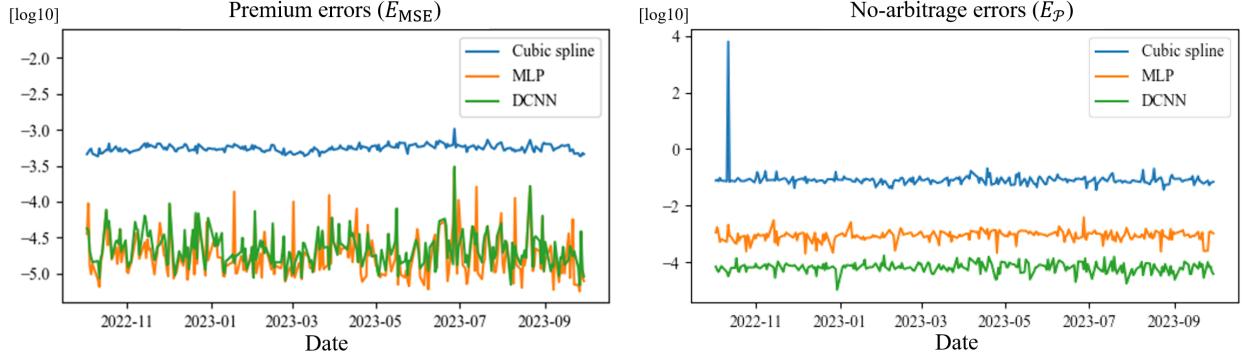


Figure 10: Backtests of intraday traded prices of S&P 500 options for 248 days from 1 October 2022 to 30 September 2023.

E Self-Adaptive weighting for Derivative-Constrained Neural Network

DCNNs with weighted loss terms have improved stability and accuracy over MLPs, yet remain rigid and non-adaptive to the weighting terms. In alignment with the neural network philosophy of self-adaptation, this section proposes a straightforward procedure applying fully trainable weights to generate multiplicative soft-weighting and attention mechanisms, following in McCleny und Braga-Neto (2020). Rather than hard-coding weights at specific parts of the loss, this proposed self-adaptive DCNN updates the loss function weights via gradient descent concurrently with the network weights. Self-adaptive DCNN employs the following weighting function in loss function based on Eq (9):

$$\lambda(m, \mathbf{x}) = \begin{cases} \gamma(m) \cdot g(\mathbf{x}), & \text{if penalty} \\ 0, & \text{if not penalty} \end{cases}, \quad (33)$$

where $m \in \mathbb{R}$ are still constants and $\gamma(x)$ are intensifier functions, which is monotonically increasing. The objective is minimized the total cost with respect to the network weights and bias but also is maximized with respect to the self-adaptation weights m . Considering the updates of a gradient ascent approach of learning with k -th weight vector of i -th derivative $m_i(k)$, $k = 1, \dots, I_{max}$,

$$m_i^{(j)}(k+1) = m_i^{(j)}(k) + \eta_{m_i^{(j)}} \nabla_{m_i^{(j)}} E(\mathbf{X}, \hat{\mathbf{X}}, \Phi), \quad (34)$$

where $\eta > 0$ is the learning rate for self-adaption weights. In the learning step, the derivatives with respect to self-adaption weights are increased when it breaks the constraints of derivative terms respectively,

$$\nabla_{m_i^{(j)}} E(\mathbf{X}, \hat{\mathbf{X}}, \Phi) = \begin{cases} \gamma'(m_i^{(j)}) \{g(h_1 \nabla \Phi(\hat{\mathbf{x}}^{(j)})) + g(h_2 \nabla^2 \Phi(\hat{\mathbf{x}}^{(j)}))\}, & \text{if penalty} \\ 0, & \text{if not penalty} \end{cases}. \quad (35)$$

Rather than being selected a priori, the self-adaptation weights and mask values produce penalty costs that increase adaptively. These penalty costs are not prescribed beforehand but are updated dynamically through the neural network training procedure.

As a testing result in this section, we applied x^2 as an intensifier function $\gamma(x)$, $m_1, m_3 = 0.001$, and $m_2 = 0.01$. Other experimental setups are similar to that in Section 5.2 ($N = 575$, $M = 286$ and 10,000 epochs, in the synthesized data with $\rho = -0.4$ and $\nu = 0.6$ in the SABR model). We also observed weight distributions after training in Figure 11.

The proposed self-adaptive DCNN architecture was implemented and evaluated using the previously described datasets. Performance metrics over training epochs reveal smooth weight convergence without instability. Learning dynamics are visualized in Figures 11 and 12. Results demonstrate the soft-weighting mechanism successfully reduces derivative losses, although some trade-off with accuracy loss is exhibited. Additionally, analysis shows that all derivative components are equally influenced by the soft-weighting rather than only subsets of the elements.

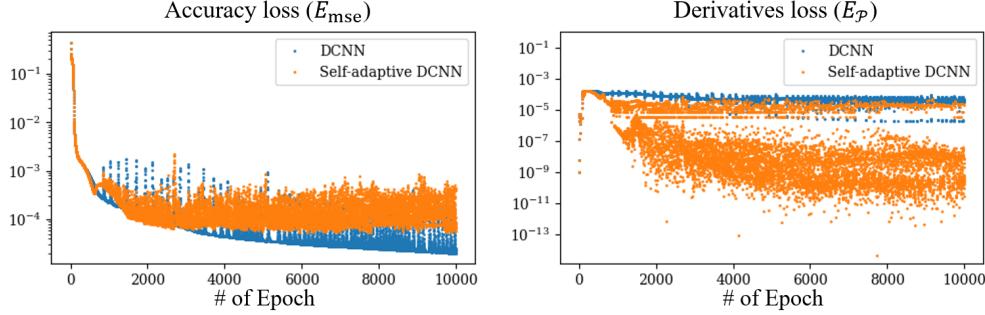


Figure 11: The accuracy (E_{MSE}) and derivative loss (E_p) in self-adaptive DCNN with a logarithmic scale by learning epochs.

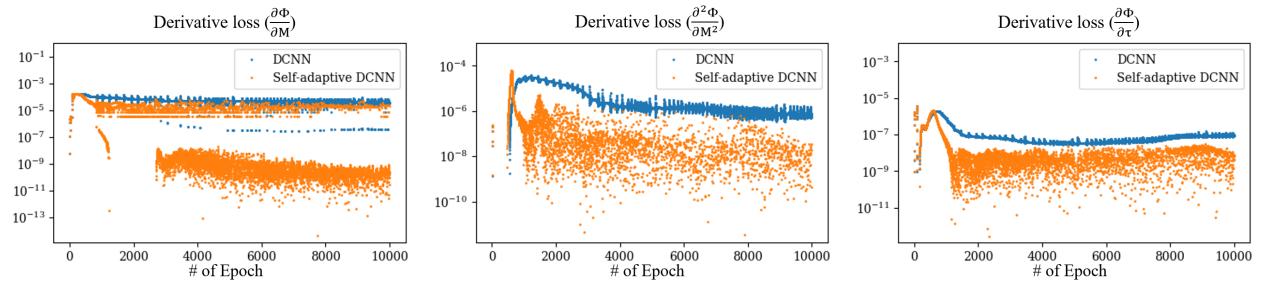


Figure 12: The derivatives losses that are elements to the derivatives (E_p) in Figure 11 with a logarithmic scale by learning epochs.

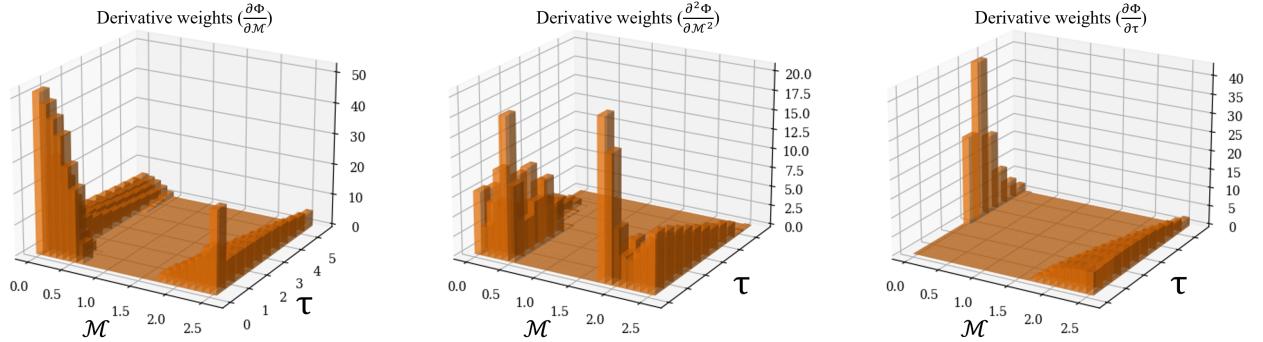


Figure 13: The distributions of weights for each derivative loss after learning in self-adaptive DCNN.

Figure 13 illustrates the distributions of loss weights for each derivative after training in the self-adaptive DCNN. High weight values indicate areas where no-arbitrage conditions were more likely to be violated during learning. Analysis reveals clustered high-loss weighting near boundary conditions and low-value short-dated options.

These results demonstrate the benefits of incorporating dynamic, trainable loss weights. By enabling the network to automatically learn optimal weightings alongside normal connection weights, adherence to derivative constraints improves without requiring pre-determined weight configurations. The performance gains validate that allowing neural networks to self-direct appropriate data-dependent weighting strategies outperforms manual loss function engineering. This aligns with core neural network principles. Just as base connections adapt during training, letting loss contribution weights calibrate automatically is advantageous. Making the weight variables differentiable parameters learned via backpropagation allows the network to holistically optimize all components in unison. This confers greater flexibility to find more accurate solutions with fewer parameters.