Gated Deep Neural Networks for Implied Volatility Surfaces

Yu Zheng♣,♠∗   **Yongxin Yang**◇†   **Bowei Chen**♡‡

♣ ArrayStream Technologies, London, UK
♠ Department of Finance, Southwestern University of Finance and Economics, China
◇ School of Informatics, University of Edinburgh, UK
♡ Adam Smith Business School, University of Glasgow, UK

Abstract

This paper presents a framework of developing neural networks for predicting implied volatility surfaces. Conventional financial conditions and empirical evidence related to the implied volatility are incorporated into the neural network architecture design and model training including no static arbitrage, boundaries, asymptotic slope and volatility smile. They are also satisfied empirically by the option data on the S&P 500 index over twenty years. The developed neural network model and its simplified variations outperform the widely used surface stochastic volatility inspired (SSVI) model on the mean average percentage error in both in-sample and out-of-sample datasets. This study has two main methodological contributions. First, an accurate deep learning prediction model is developed and tailored to implied volatility surfaces. Second, a framework, which seamlessly combines data-driven models with financial theories, can be extended and applied to solve other related business problems.

1 Introduction

Technology has been widely used in finance to bring automation and improve services. It includes using mobile phones for online banking, creating new digital assets like cryptocurrency, and developing data-driven machine learning algorithms for asset pricing and investment. These activities have formed an emerging new multidisciplinary subject called *financial technology* (in short *fintech*), combining the fields of finance, computer science, information systems, operations research and law (Hendershott et al., 2017). According to a recent report from Ernst and Young (2018), global fintech funding has a compound annual growth rate of 44% from 2013 to 2017. It increased in 2017 with US$ 12.2 billion in the first three quarters as compared with US$ 11.7 billion in the first three quarters of 2016. Within fintech, machine learning (or broadly artificial intelligence) has become one of the hottest sectors, with

expected direct investment growth of 63% from 2016 to 2022. In this paper, we investigate an application of the state-of-the-art machine learning in fintech and develop a new deep neural network to predict implied volatility surfaces.

The research topic of this study can be tracked back to the seminal work of Black and Scholes (1973). They proposed a pricing model for European options in which an underlying asset price is driven by a geometric Brownian motion containing a drift and a volatility where the volatility term shows small fluctuations of the asset returns representing risk. The study of volatility has become popular since the crash of 1987 (Friz and Gatheral, 2005). Specifically, the *implied volatility* of an option is defined as the inverse problem of option pricing, mapping from the option price in the current market to a single value (Cont and Da Fonseca, 2002). When it is plotted against the option strike price and the time to maturity, it is referred to as the *implied volatility surface*.

The methodologies of modelling implied volatility surfaces can be classified into two major groups (Homescu, 2011). The first group is called *indirect methods*, in which an implied volatility is driven by another dynamic model such as local volatility models, stochastic volatility models and Lévy models. Notable studies include Merton (1976), Heston (1993), Kou (2002), Chockalingam and Muthuraman (2011), Kang et al. (2017) and Shiraya and Takahashi (2018). Models in this group usually have a limited number of parameters, and the volatility term is fitted by the market data along with the asset dynamics. These models are sometimes invalid empirically though they exhibit mathematical elegance. Time-dependent parameters can also be included but this will greatly increase computational time and optimisation difficulty. The second group is called *direct methods*, in which an implied volatility is specified explicitly. Direct methods can also be divided into two types. The first type specifies the dynamics of an implied volatility surface and assumes it evolves continuously over time (Cont and Da Fonseca, 2002; Carr and Wu, 2010). The second type pays attention to the static representation of implied volatility surfaces. It does not consider the evolution of the underlying asset, but uses either parametric or non-parametric methods to fit an implied volatility surface. Static models have been widely used by practitioners as they are able to provide a snapshot of the current market situation and usually fit the market data well.

∗ ✉ zhengyu@swufe.edu.cn
† ✉ yongxin.yang@ed.ac.uk
‡ ✉ bowei.chen@glasgow.ac.uk (corresponding author)

One of the most popular static models is the stochastic volatility inspired (SVI) model proposed by Gatheral (2004). It models the implied volatility slice for a fixed time to maturity. Kotzé et al. (2013) then constructed an arbitrage-free implied volatility surface by introducing a quadratic deterministic volatility function, and the arbitrage-free conditions are forced by solving two minimization problems. Gatheral and Jacquier (2014) further updated the SVI model to the surface SVI (SSVI) model, which has simpler representations than the SVI on no static arbitrage conditions and has soon been widely adopted by investors. In addition, Itkin (2015) proposed a non-parametric method to model implied volatility surfaces using polynomials of sigmoid functions. However, arbitrage-free conditions are held only at the nodes of discrete strike-expiry space. Corlay (2016) employed B-splines to construct an arbitrage-free implied volatility surface and proposed a new calibration method tailored to sparse option data.

Machine learning in asset pricing and computational investment goes back more than a decade earlier, to the late 1980's or early 1990's. Malliaris and Salchenberger (1993) demonstrated that a single hidden layer neural network can offer a valuable alternative to estimating option prices to the traditional Black-Scholes model. Malliaris and Salchenberger (1996) then used several single hidden layer neural networks to forecast the future volatility of the S&P 100 index. Gavrishchaka (2006) proposed a boosting-based framework for volatility prediction in which a collection of generalized autoregressive conditional heteroskedasticity models are trained separately and then combined to form a strong predictor. This approach is also called *ensemble learning*. Audrino and Colangelo (2010) presented a semi-parametric method for predicting implied volatility surfaces in which the base model is a regression tree and the difference between the model prediction and the actual value is sequentially minimised by adding more trees. Coleman et al. (2013) used kernel machines (Vapnik, 2000) to calibrate the volatility function for option pricing. However, all these mentioned studies only examined small datasets – Gavrishchaka (2006) targeted IBM stock options only, and Audrino and Colangelo (2010) and Coleman et al. (2013) worked on one-month data of the S&P 500 index options.

Recently, Yang et al. (2017) proposed a class of gated neural networks that can automatically learn to divide-and-conquer the problem space (Gradojevic et al., 2009) for robust and accurate pricing European options. Inspired by their work, we develop a gated neural network to predict implied volatility surfaces in this paper. Unlike many previous studies, in which machine learning algorithms were used directly as a "black box", our model is tailored to implied volatility surfaces. We design and calibrate the neural network by incorporating the related financial conditions and empirical evidence such as no static arbitrage, boundaries, asymptotic slope and volatility smile. Therefore, these heuristics should be met. From a high level perspective, our study bridges the gap between the data-driven machine learning algorithms and the existing financial theories. A methodological framework is developed to integrate both fields simultaneously, which can be applied to solve other problems in many other business fields. In addition, the proposed

model is validated with the option data on the S&P 500 index over twenty years, even including the options with a short time to maturity. Compared with related studies, the experimental settings in our study are more challenging, so the model needs to be more robust and stable. In experiments, conventional financial conditions and empirical evidence are met empirically, and our model outperforms the widely used SSVI model on the mean average percentage error in both in-sample and out-of-sample datasets. It also outperforms other similar neural network models without incorporating financial conditions and empirical evidence, which further justifies the importance of integrating financial theories in model development.

The remainder of the paper is organised as follows. Section 2 introduces the preliminaries of neural networks; Section 3 introduces our proposed neural network model; Section 4 presents our experimental results; and Section 5 concludes the paper.

2 Neural Networks Preliminaries

As a special machine learning paradigm, neural networks (also called *connectionist systems* or *representation learning*) are inspired by ideas from psychology and neuroscience (Schmidhuber, 2015; Lecun et al., 2015). Simply, a neural network is a set of algorithms, modelled loosely to mimic the human brain, that is designed to recognise patterns in data. Unlike conventional machine learning techniques, which are limited in their ability to process natural data in their raw form and require careful engineering and considerable domain expertise, neural networks allow a machine to be fed with raw data and can automatically discover the needed data representations for pattern recognition such as detection and classification. In other words, neural networks can learn and recognise patterns from data by examples, generally without being programmed with any task-specific rules from domain knowledge. For example, in image recognition, a neural network can learn to identify images that contain dogs by analysing example images that have been manually labelled as "dog" or "not a dog" and using these to identify dogs in other images. They can automatically generate identifying characteristics from the raw data that they process without any prior knowledge about dogs. Therefore, neural networks are representation learning methods with multiple levels of representation, obtained by composing simple but non-linear modules that each transforms the representation at one level (starting with the raw input) into a representation at a higher, slightly more abstract level (Lecun et al., 2015). It is also widely believe that a neural network can approximate any continuous functions through different architecture designs of representation layers. This is known as the universal approximation theorem (Cybenko, 1989; Hornik, 1991).

In addition to finance, neural networks have also been used in other business fields for decades, ranging from operations research to marketing and business analytics, to provide computing capabilities in solving prediction, classification and optimisation problems (Mangasarian, 1993; Smith, 1999). For example, Gupta et al. (2000) discussed the application of neural networks to select the best heuristic algorithm to solve a given

scheduling problem. Belloni et al. (2009) developed an extension of the rescaled perceptron algorithm for solving a homogeneous linear inequality system. Agarwal et al. (2006) proposed an augmented neural-network to integrate greedy and non-greedy heuristics for task scheduling. Franklin (2019) investigated several specific recurrent neural networks for computer music reproduction and generation. Neural networks have distinguishing characteristics, such as type of network topology, number of hidden layers in the network, and the learning rules of the network. In the following, we introduce some neural network fundamentals which are related to our proposed architecture in this paper.

The simplest feedforward neural network architecture contains two representation layers: an input layer where the data flows in, and an output layer where the predictions are produced. Suppose that the input data is of N-dimension, and the output is of M-dimension, a two-layer neural network can be expressed as follows:

$$\hat{y} = x^T W + b, \tag{1}$$

where W is a matrix of size $N \times M$ and b is a vector of length M. In machine learning literature, W is usually referred to as the *weight*, and b as the *bias*. Let $\Theta = \{W, b\}$ denote a set of parameters. The process of training a neural network (also called the *model calibration* or *parameter estimation*) is to solve the following optimisation problem:

$$\underset{\Theta}{\mathrm{argmin}} \frac{1}{N} \sum_{n=1}^{N} \ell(y_n, \hat{y}_n)$$
$$= \underset{W,b}{\mathrm{argmin}} \frac{1}{N} \sum_{n=1}^{N} \ell(y_n, x_n^T W + b),$$

where $\{x_n, y_n\}_{n=1}^{N}$ is called the *training data* and $\ell(y, \hat{y})$ is called the *loss function* which measures the difference between the ground-truth y in data and the model prediction $\hat{y}$. For example, $\ell(y, \hat{y}) = (y - \hat{y})^2$ calculates the squared difference of these two values and the objective function here is called the *mean squared error* or *squared L2 norm* (Bishop, 2006).

The advanced neural network models are different to Eq. (1) from two architecture design settings: (i) the hidden layer; and (ii) the activation function. The hidden layer is an extra degree of computation. For example, a three-layer neural network can be written as follows:

$$\hat{y} = (x^T \bar{W} + \bar{b}) \tilde{W} + \tilde{b}, \tag{2}$$

where $\bar{W}, \bar{b}, \tilde{W}, \tilde{b}$ are matrices with dimensions $N \times K$, $K \times 1$, $K \times M$, $M \times 1$, respectively. Here K is a hyper-parameter[1] that indicates how many neurons in the hidden layer and we could simply denote $\Theta = \{\bar{W}, \bar{b}, \tilde{W}, \tilde{b}\}$. If a neural network with more than one hidden layer, it can be called the *deep* neural network. Here the term deep generally refers to neural networks which use sophisticated mathematical modelling to process data in complex architecture (Goodfellow et al., 2016).

---

[1]Hyper-parameter is not learned through training data, but assigned before training.

Activation functions can add non-linearity to neural networks. For example, a sigmoid function $\sigma(\cdot)$ can be added into Eq. (2) for the hidden layer, and then it becomes

$$\hat{y} = \sigma(x^T \bar{W} + \bar{b}) \tilde{W} + \tilde{b}. \tag{3}$$

Scalar and vector functions are two broad classes of activation functions. The former acts on neurons in the element-wise fashion so that neurons do not affect each other. The common choices are the sigmoid function, the hyperbolic tangent function, the softplus function, and the rectified linear unit (ReLU) function. Vector functions treat neurons as a vector, thus the value of one neuron may affect others, such as the softmax function.

There is no unique receipt of developing a neural network to solve specific problems. Usually, domain knowledge and empirical evidence can be incorporated into neural networks through activation functions, weight constraints, pseudo training data and loss functions. If a three-layer neural network needs to ensure positive output values, the activation function and the weight constraint can be added into Eq. (3) as follows:

$$\hat{y} = \sigma(x^T \bar{W} + \bar{b}) e^{\tilde{W}} + e^{\tilde{b}}. \tag{4}$$

Here $\sigma(\cdot)$ can be either the sigmoid function or the softplus function as both functions always produce positive values. If the neural network outputs need to be monotonically increasing or decreasing with respect to one of the input variable x, it can be rewritten

$$\hat{y} = \begin{cases} \sigma(x e^{\bar{W}} + \bar{b}) e^{\tilde{W}} + \tilde{b}, & \text{if } \frac{\partial \hat{y}}{\partial x} > 0, \\ \sigma(-x e^{\bar{W}} + \bar{b}) e^{\tilde{W}} + \tilde{b}, & \text{if } \frac{\partial \hat{y}}{\partial x} < 0. \end{cases} \tag{5}$$

Generating a set of pseudo data that softly constrains the loss function was first proposed by Abu-Mostafa (1993). For example, N pseudo data can be generated and be added into the following loss function if it is assumed to be monotonically increase:

$$\ell = \sum_{n=1}^{N} \max\{0, \hat{y}(x) - \hat{y}(x + \epsilon)\}, \tag{6}$$

where ϵ is a small positive value (e.g., 0.001). It is not difficult to see the violation of the monotonicity property for the generated values will cause loss. Therefore, the optimisation will tend to find an appropriate function that has a such property. This method is flexible and easy to implement. However, how well the property is satisfied needs to be checked with data empirically. The existing financial theories can be properly defined as different loss functions based on pseudo training data and be incorporated into the neural network. We will explain this in detail in Section 3.

3 Model

Let $(S_t)_{t \geq 0}$ be the spot price of an asset at time t, defined on a filtered probability space $(\Omega, \mathcal{F}, (\mathcal{F}_t)_{t \geq 0}, \mathbb{P})$, where Ω is the sample space, $\mathcal{F}$ is a sigma-field, $(\mathcal{F}_t)_{t \geq 0}$ is a filtration and $\mathbb{P}$ is the probability space. The market is assumed to be arbitrage-free and the time to maturity of a financial product is always

finite. To avoid dealing with interest rates and dividends, the forward measure can be used. Let $(F_{t,T})_{t\geq 0}$ be the forward price of the asset with maturity date T. It can be calculated by $\frac{S_t}{B(t,T)}$ where $B(t,T)$ is the price at time t of a zero-coupon bond paying one unit at time T. The no-arbitrage assumption ensures there exists an equivalent martingale measure in which $(F_{t,T})_{t\geq 0}$ is a martingale (Cont and Da Fonseca, 2002). The log forward moneyness m can be obtained by $\log\{\frac{K}{F_{t,T}}\}$, where K is the strike price. The annualized time to maturity τ is then defined as $\frac{T-t}{A}$, where A is the annualization factor. Therefore, the implied volatility $v(m,\tau)$ can be rewritten as a function of m and τ, and its ground truth value can be obtained by inverting the BlackScholes option pricing formula.

3.1 Setup

Implied volatility surfaces have been well studied in financial literature. Theorem 1 presents the conditions for an implied volatility surface, in which conditions 1-5 ensure the absence of arbitrage (Gulisashvili, 2012); conditions 6-7 specify the boundaries (Carr and Wu, 2007); and condition 8 describes the log forward moneyness behaviour (Lee, 2004).

Theorem 1 *Let* $d_{\pm}(m,\tau) = -\frac{m}{\sqrt{\tau}v(m,\tau)} \pm \frac{\sqrt{\tau}v(m,\tau)}{2}$, $n(\cdot)$ *and* $N(\cdot)$ *be the probability density and cumulative functions of a standard normal distribution, respectively. The following conditions should hold for an implied volatility surface* $v(m,\tau)$:

1. *(**Positivity**)* $v(m,\tau) > 0$ *for all* $(m,\tau) \in \mathbb{R} \times \mathbb{R}^+$.

2. *(**Twice Differentiability**)* *For every* $\tau > 0$, $m \to v(m,\tau)$ *is twice differentiable on* $\mathbb{R}$.

3. *(**Monotonicity**)* *For every* $m \in \mathbb{R}$, $\tau \to \sqrt{\tau}v(m,\tau)$ *is increasing on* $\mathbb{R}^+$, *therefore*

$$v(m,\tau) + 2\tau\partial_\tau v(m,\tau) \geqslant 0.$$

4. *(**Absence of Butterfly Arbitrage**)* *For all* $(m,\tau) \in \mathbb{R}\times\mathbb{R}^+$,

$$\left(1 - \frac{m\partial_m v(m,\tau)}{v(m,\tau)}\right)^2 - \frac{(v(m,\tau)\tau\partial_m v(m,\tau))^2}{4}$$
$$+ \tau v(m,\tau)\partial_{mm}v(m,\tau) \geqslant 0.$$

5. *(**Limit Condition**)* *For every* $\tau > 0$, *then*

$$\lim_{m\to+\infty} d_+(m,\tau) = -\infty.$$

6. *(**Right Boundary**)* *If* $m \geq 0$, *then*

$$N(d_-(m,\tau)) - \sqrt{\tau}\partial_m v(m,\tau)n(d_-(m,\tau)) \geqslant 0.$$

7. *(**Left Boundary**)* *If* $m < 0$, *then*

$$N(-d_-(m,\tau)) + \sqrt{\tau}\partial_m v(m,\tau)n(d_-(m,\tau)) \geqslant 0.$$

8. *(**Asymptotic Slope**)* *For every* $\tau > 0$, *then*

$$2|m| - v^2(m,\tau)\tau > 0.$$

In addition to Theorem 1, volatility smile has been considered an important empirical evidence, which will be included in our model. For a given time to maturity, when the implied volatility is plotted against the strike price, it creates a line that slopes upward on either end, looking like a "smile". In order to include volatility smile in our model, the following smile function $\phi(\cdot)$ for the log forward moneyness is defined:

$$\phi(z) = \sqrt{z\tanh(z+\frac{1}{2}) + \tanh(-\frac{1}{2}z+\epsilon)}, \quad z \in \mathbb{R}, \quad (7)$$

where $\tanh(\cdot)$ is the hyperbolic tangent function and ϵ is a small value to ensure numerical stability. The function exhibits not only a skew pattern like volatility smile but also meets the second condition of Theorem 1 as it is smoothly twice differentiable with the co-domain $(0,\infty)$.

3.2 Architecture Design

Fig. 1 presents a schematic view of our neural network architecture design. Our model is called the *multi-model* because it uses several single models as building blocks and their weights are specified by another neural network. The input of the multi-model is the log forward moneyness m and the time to maturity τ; and the output is the predicted implied volatility $\hat{v}(m,\tau)$. Simply, a neural network is based on a collection of connected units called *neurons*. Each connection can transmit a signal from one neuron to another. In our design, additional arithmetic operations on signal transmission are performed at some stages in terms of gate operators. For example, $\otimes$ is the multiplication gate operator that multiplies the signals it receives, and $\oplus$ is the addition gate operator, which sums up the signals it receives. For the reader's convenience, a brief description of model parameters is given in Table 1.

As depicted in Fig. 1, each single model can be used to model an implied volatility surface. Mathematically, a single model can be expressed as follows:

$$\hat{v}(m,\tau) = y(m,\tau)$$
$$= \sum_{j=1}^{J} \phi(m\bar{W}_{1,j} + \bar{b}_j)\psi(\tau\tilde{W}_{1,j} + \tilde{b}_j)e^{\hat{W}_{j,1}} + e^{\hat{b}},$$
$$(8)$$

where $\phi(\cdot)$ is the smile function to activate neurons related to m and $\psi(\cdot)$ is the sigmoid function to activate neurons related to τ, J is the number of neurons in the hidden layer, and $\bar{b}$, $\bar{W}$, $\tilde{b}$, $\tilde{W}$, $\hat{W}$, $\hat{b}$ are network parameters. Each of $\bar{b}$, $\bar{W}$, $\tilde{b}$, $\tilde{W}$, $\hat{W}$ has J elements and $\hat{b}$ has one element. Therefore, there is a total of $5J + 1$ parameter values. It should be noted that the predicted implied volatility from Eq. (8) is always positive, satisfying the first condition of Theorem 1.

Single models are used as building blocks and are combined to construct a deeper and more complex architecture. The multi-
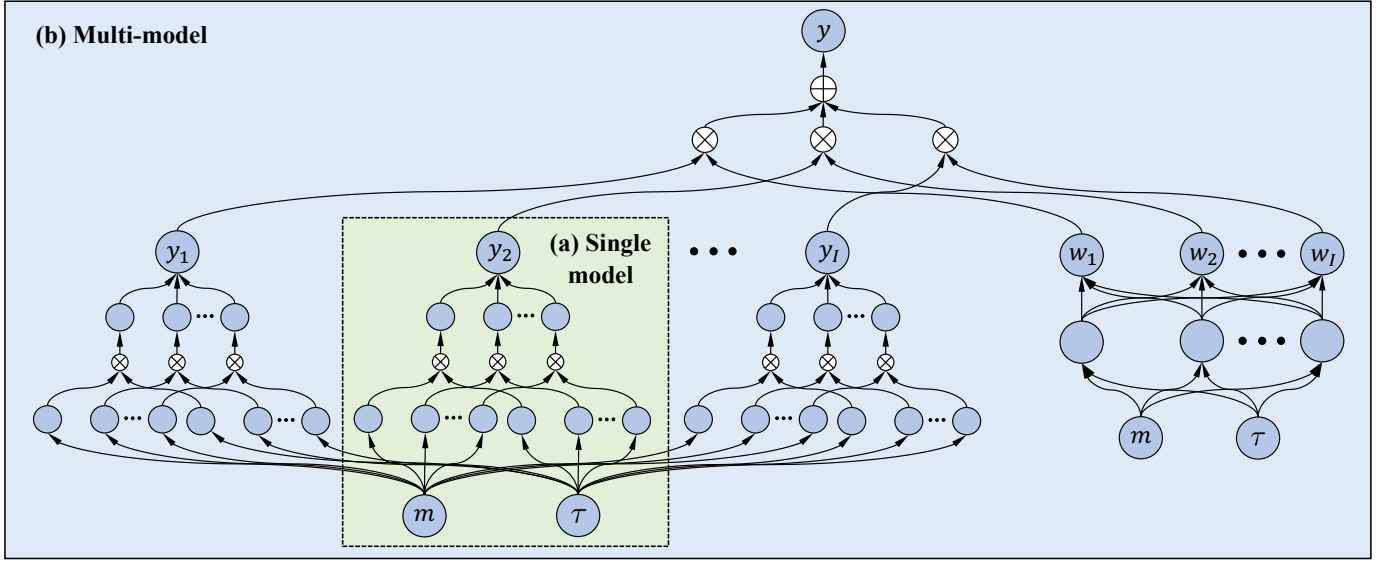
Figure 1: Schematic view of neural network architecture design: (a) the single model; (b) the multi-model. The multi-model consists of several single models and their weights are given by another neural network. Bias terms are omitted; $\otimes$ is the multiplication gate operator; $\oplus$ is the addition gate operator.

model can be expressed as follows:

$$\hat{v}(m,\tau) = \sum_{i=1}^{I} y_i(m,\tau) w_i(m,\tau), \tag{9}$$

$$y_i(m,\tau) = \sum_{j=1}^{J} \phi(m\bar{W}_{1,j}^{(i)} + \bar{b}_j^{(i)})\psi(\tau\tilde{W}_{1,j}^{(i)} + \tilde{b}_j^{(i)})e^{\hat{W}_{j,1}^{(i)}} + e^{\hat{b}^{(i)}}, \tag{10}$$

$$w_i(m,\tau) = \frac{e^{\sum_{k=1}^{K} \psi(m\dot{W}_{1,k} + \tau\dot{W}_{2,k} + \dot{b}_k)\ddot{W}_{k,i} + \ddot{b}_i}}{\sum_{i=1}^{I} e^{\sum_{k=1}^{K} \psi(m\dot{W}_{1,k} + \tau\dot{W}_{2,k} + \dot{b}_k)\ddot{W}_{k,i} + \ddot{b}_i}}, \tag{11}$$

where $\dot{W}, \dot{b}, \ddot{W}, \ddot{b}$ are the newly added parameter terms of the network for weighting single models. The dimensions of $\dot{W}$, $\dot{b}, \ddot{W}, \ddot{b}$ are $2 \times K$, $K \times 1$, $K \times I$, and $I \times 1$, respectively. Therefore, the total number of parameter values in the multi-model is $(5J + K + 2)I + 3K$.

3.3 Embedding Constraints in the Optimisation

The designed neural network needs to be calibrated with the market data so that it can be used for prediction. The aim is to minimise the in-sample difference between the predicted $\hat{v}$ and the ground truth v. Mathematically, the model training can be expressed by minimising the following loss function ℓ:

$$\ell = \ell_0 + \gamma\ell_1 + \delta\ell_2 + \eta\ell_3 + \rho\ell_4 + \omega\ell_5, \tag{12}$$

where ℓ_0 is the data loss function, $\ell_1, \ldots, \ell_4$ are the loss functions that incorporate financial conditions discussed in Theorem 1, ℓ_5 is the regularization term to avoid over-fitting, and $\gamma, \delta, \eta, \rho, \omega$ are the hyper-parameters controlling the weights of $\ell_1, \ldots, \ell_5$.

The data loss ℓ_0 is a joint loss function which combines the Mean Squared Log Error (MSLE) and the Mean Squared Per-

centage Error (MSPE), defined as follows:

$$\ell_0 = \alpha\left[\underbrace{\frac{1}{N}\sum_{n=1}^{N}(\log(v_n) - \log(\hat{v}_n))^2}_{\text{MSLE}}\right] + \beta\left[\underbrace{\frac{1}{N}\sum_{n=1}^{N}(\frac{v_n - \hat{v}_n}{v_n})^2}_{\text{MSPE}}\right], \tag{13}$$

where α and β are hyper-parameters. In machine learning, a joint loss is often used to deal with sensitive data or high-dimensional feature spaces (Goodfellow et al., 2016).

The loss function ℓ_1 specifies the monotonicity condition in Theorem 1. Let $a(m,\tau) := v(m,\tau) + 2\tau\partial_\tau v(m,\tau)$ and the objective of ℓ_1 is to push $a(m,\tau)$ to be non-negative. This can be achieved by randomly sampling P unique values from the domain of m and Q unique values from the domain of τ. Therefore, ℓ_1 is defined as follows:

$$\ell_1 = \sum_{p=1}^{P}\sum_{q=1}^{Q}\max\{0, -a(m_p, \tau_q)\}. \tag{14}$$

It is not difficult to see that ℓ_1 adds a penalty if negative values are produced by $a(m,\tau)$ for a certain set of (m,τ) pairs. Therefore, if an infinite number of samples is generated, ℓ_1 can be reduced to zero during the optimisation and the condition would be met.

The loss function ℓ_2 specifies the absence of butterfly arbitrage condition in Theorem 1. Let $b(m,\tau) := (1 - \frac{m\partial_m v(m,\tau)}{v(m,\tau)})^2 - \frac{(v(m,\tau)\tau\partial_m v(m,\tau))^2}{4} + \tau v(m,\tau)\partial_{mm}v(m,\tau)$ and the objective is to push $b(m,\tau)$ to be non-negative. This can be achieved using the same way as ℓ_1, by randomly sampling P unique values

Table 1: Summary of neural network parameters.

Notation	Description	Shape	Number in single model	Number in multi-model
$\bar{W}$	Weight term for moneyness	$1 \times J$	1	I
$\bar{b}$	Bias term for moneyness	$J \times 1$	1	I
$\tilde{W}$	Weight term for time to maturity	$1 \times J$	1	I
$\tilde{b}$	Bias term for time to maturity	$J \times 1$	1	I
$\hat{W}$	Weight term for final pricing	$J \times 1$	1	I
$\hat{b}$	Bias term for final pricing	1	1	I
$\dot{W}$	Weight term for weighting model hidden	$2 \times K$	0	1
$\dot{b}$	Bias term for weighting model hidden	$K \times 1$	0	1
$\ddot{W}$	Weight term for weighting model output	$K \times I$	0	1
$\ddot{b}$	Bias term for weighting model output	$I \times 1$	0	1

from the domain of m and Q unique values from the domain of τ. Then ℓ_2 is defined as follows:

$$\ell_2 = \sum_{p=1}^{P} \sum_{q=1}^{Q} \max\{0, -b(m_p, \tau_q)\}. \qquad (15)$$

The loss function ℓ_3 specifies the left and right boundary conditions in Theorem 1. Let $c_1(m, \tau) := N(d_-(m, \tau)) - \sqrt{\tau}\partial_m v(m, \tau)n(d_-(m, \tau))$ and $c_2(m, \tau) := N(-d_-(m, \tau)) + \sqrt{\tau}\partial_m v(m, \tau)n(d_-(m, \tau))$. The objective of ℓ_3 is to push both functions to be non-negative. To achieve this, P_1 and P_2 unique non-negative values can be sampled from the domain of m, and Q unique values from the domain of τ. Then ℓ_3 is defined as follows:

$$\ell_3 = \sum_{p_1=1}^{P_1} \sum_{q=1}^{Q} \max\{0, -c_1(m_{p_1}, \tau_q)\}$$
$$+ \sum_{p_2=1}^{P_2} \sum_{q=1}^{Q} \max\{0, -c_2(m_{p_2}, \tau_q)\}. \qquad (16)$$

The loss function ℓ_4 specifies the asymptotic condition in Theorem 1. Let $g(m, \tau) := 2|m| - v^2(m, \tau)\tau$. Similar to $\ell_0, \ldots, \ell_3$, P unique values can be sampled from the domain of m and Q unique values can be sampled from the domain of τ. Then ℓ_4 is defined as follows:

$$\ell_4 = \sum_{p=1}^{P} \sum_{q=1}^{Q} \max\{0, -(g(m_p, \tau_q) - \epsilon)\}, \qquad (17)$$

where $\epsilon = 10^{-5}$ is a small value which ensures $g(m, \tau)$ to be positive.

To prevent over-fitting, the regularization term ℓ_5 is added into the loss function for all weight terms:

$$\ell_5 = \begin{cases} ||\bar{W}||_F^2 + ||\tilde{W}||_F^2 + ||\hat{W}||_F^2, & \text{if single model,} \\ \sum_{i=1}^{I} ||\bar{W}^{(i)}||_F^2 + \sum_{i=1}^{I} ||\tilde{W}^{(i)}||_F^2 & \text{if multi-model.} \\ + \sum_{i=1}^{I} ||\hat{W}^{(i)}||_F^2 + ||\dot{W}||_F^2 + ||\ddot{W}||_F^2, & \end{cases}$$
$$(18)$$

where $|| \cdot ||_F^2$ is the square of Frobenius norm, e.g., $||W||_F^2 = \frac{1}{2}\sum_{i=1}^{I}\sum_{j=1}^{J} W_{i,j}^2$ for an $I \times J$ weight matrix W.

It is worth noting that, in Eqs. (14)-(17), we sample m and τ from specific intervals rather than the training data (see Section 4.2 for details). In the practice of machine learning model training, if the training data have limited observations of input variables, creating synthetic data by sampling values from their domains or specific intervals is often used (Choe et al., 2017; Liu et al., 2017). If their values are sampled from the training data, the calibrated neural network may fail to meet the conditions in the case when their values in prediction are out of the scope of the training data.

4 Experiments

In this section, we describe the used option data, explain the experimental settings, present and discuss our experimental results.

4.1 Data

Our option data on the S&P 500 index is obtained from OptionMetrics. It includes a total of 5,116 trading days, covering the period from 04/01/1996 to 29/04/2016. OptionMetrics also provides data on the zero-coupon yield curve, which is constructed based on the London Inter-Bank Offered Rate (LIBOR). However, because the traditional LIBOR-based zero curve is not risk-free after the 2008 financial crisis (Ametrano and Bianchetti, 2013), we extract the Overnight Index Swap (OIS) rates from Bloomberg and bootstrap the zero rate curve from the OIS for the period from 01/01/2008 to 14/10/2016. The zero rate curve provided by OptionMetrics is used for the data prior to 01/01/2008. The risk-free rates are interpolated using a cubic spline to match the option maturity. Forward price is estimated by the put-call parity (Bilson et al., 2015).

4.2 Experimental Settings

The original option data is further filtered before model training. Option quotes which are less than 3/8 are excluded because they are close to tick size, which might be misleading. The
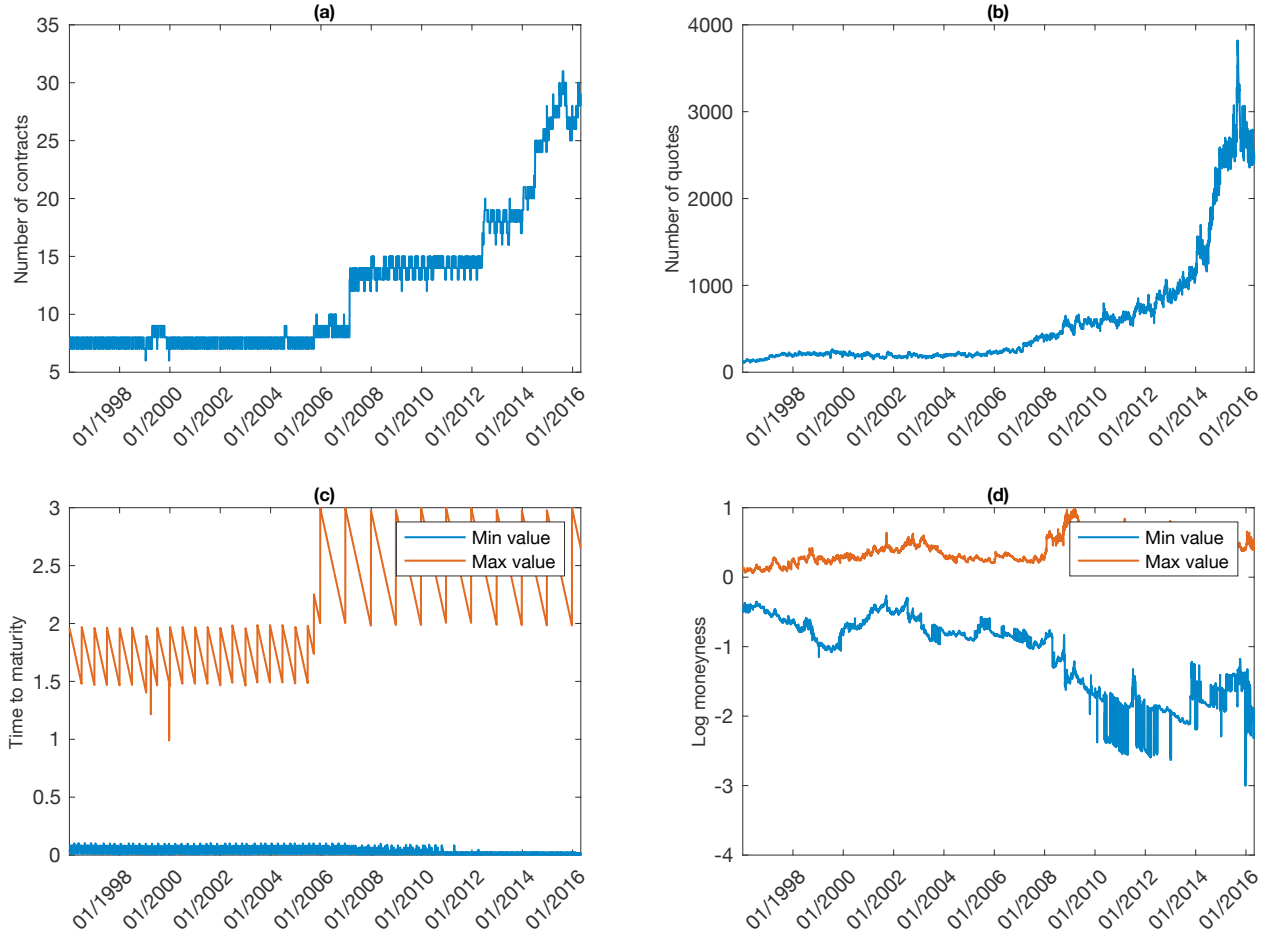
Figure 2: Time series plots of options from 04/01/1996 to 29/04/2016: (a) the number of contracts; (b) the number of quotes; (c) the time to maturity; (d) the log moneyness.

bid-ask mid-point price is calculated as a proxy for the closing price. In-the-money option quotes are excluded because of small transaction volume (Bliss and Panigirtzoglou, 2005). Scholars usually do not analyse option contracts with time to maturity of less than 7 days (Andersen et al., 2017). However, as these options are getting popular recently, e.g., weekly index options, we here analyse option contracts with a short time to maturity and only exclude the contracts with maturity of less than 2 days. Analysing options with a short time to maturity is challenging because it requires a model with high robustness and stability. Our prepared data finally contains 63,338 option contracts with 2,986,754 valid quotes. The quotes are then used to calculate implied volatility values by inverting the BlackScholes option pricing formula.

Fig. 2 provides a descriptive summary of the prepared data. As shown in Fig. 2(a), the number of option contracts doubled from 2007 to 2012, and it reached more than 30 contracts for each day in 2016. Therefore, in Fig. 2(b), the number of quotes increases exponentially. Fig. 2(c)(d) present the range of the time to maturity and the log moneyness, which are $[1, 3]$ and $[-3, 1]$, respectively.

Our model is the multi-model. It is compared with the single model and the SSVI for the benchmark. We also compare it

with a neural network with the simplest architecture, which has a single hidden layer using the sigmoid activation function and only one constraint that ensures positive output. For simplicity, we call it the *vanilla model*. In addition, to justify the importance of embedding financial conditions in the constrained optimisation, all neural network models are further trained under a setting where $\ell_1, \cdots, \ell_4$ are removed from the loss function in Eq. (12), called the *incomplete constraints* setting. Finally, seven models are examined in experiments, including six different versions of neural networks, the hyper-parameter settings of which are summarised in Table 2. To avoid the effect of model size on model performance, neural networks with the same architecture design are specified with the same model size. Synthetic data are generated to meet the constraints specified by Eqs. (14)-(17). The ratio of real market data and synthetic data is 1/6. m is sampled in $[-6, -3] \cup [3, 6]$ for the asymptotic condition and in $[-3, 3]$ for other conditions; τ is sampled in $[0.002, 3]$. These values are set based on the observations from historical data, as shown in Fig. 2 (c)(d).

4.3 Results

Neural network models are trained using TensorFlow in Python (Roy et al., 2019) and we use the method proposed

Table 2: Experimental settings of hyper parameters.

Parameter	Multi	Multi[†]	Single	Single[†]	Vanilla	Vanilla[†]
I	4	4	1	1	1	1
J	8	8	32	32	32	32
K	5	5	-	-	-	-
Initial learning rate	0.1	0.1	0.1	0.1	0.1	0.1
Number of iterations	20000	20000	20000	20000	20000	20000
α	1	1	1	1	1	1
β	1	1	1	1	1	1
γ	10	0	10	0	10	0
δ	1	0	1	0	1	0
η	10	0	10	0	10	0
ρ	1	0	1	0	1	0
ω	0.00005	0.00005	0.00005	0.00005	0.00005	0.00005

[†] Model with incomplete constraints.

Table 3: Mean and standard deviation (Std) of the MAPEs for all models.

Model	Implied volatility				Option price			
	Training set		Test set		Training set		Test set	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std
Multi	1.74	0.50	3.34	2.18	5.97	1.86	10.64	6.72
Multi[†]	1.76	0.50	3.35	2.17	6.03	1.86	10.67	6.70
Single	2.15	0.67	3.60	2.12	7.38	2.57	11.64	6.68
Single[†]	1.82	0.52	3.38	2.16	6.20	1.91	10.77	6.67
Vanilla	3.21	0.98	4.46	2.07	11.31	3.57	14.61	6.42
Vanilla[†]	2.87	0.80	4.18	2.04	10.53	3.34	14.17	6.60
SSVI	2.59	0.85	3.73	2.18	8.71	2.72	12.74	6.74

[†] Model with incomplete constraints

by Kingma and Ba (2015) for stochastic optimisation. The selected option quotes available on each trading day are used to compute the mean average percentage error (MAPE) of implied volatilities. The latter can be used to compute option prices; therefore, the MAPE of option prices can be obtained. In the following, we compare our proposed model with the benchmarked models, check if the financial conditions are satisfied, and then investigate the effects of constraints in the optimisation.

Table 3 provides an overall summary of the mean and the standard deviation of MAPEs of implied volatility and option price for each examined model. Our proposed multi-model outperforms other models: (i) on both implied volatility and option price; and (ii) in both training and test sets. The training set result shows the in-sample error representing how good the model fits the data in calibration, while the test set result shows the out-of-sample error representing the prediction power of the model. Except for vanilla models, other neural network models achieve better performance than the widely used SSVI, affirming that our network architecture design has a great advantage of modelling implied volatility surfaces. Models with incomplete constraints are slightly behind the models with full constraints. Fig. 3 further plots the MAPEs of each quarter to compare the (complete) models with the SSVI in both training and test sets. The multi-model can dynamically capture the data patterns and has the smallest and the most stable moving MAPE over time.

Fig. 4 checks whether the financial conditions set in Theorem 1 are satisfied by neural network models over time, including monotonicity, left boundary, right boundary, absence of arbitrage and asymptotic slope. These conditions are met by the multi-model because the violation percentages are less than 0.1%. Overall, the complete models are more robust than models with incomplete constraints. This justifies the importance of incorporating financial conditions. Figs. 5-6 show why regularization is needed, check if the limit condition is met and compare the implied volatility surface from the multi-model and the multi-model without regularization. As described earlier, regularization is usually used to avoid over-fitting (Goodfellow et al., 2016), and we can see that the implied volatility surface generated by the multi-model without regularization is not smooth with the "smile" pattern. Fig. 6 further plots the risk-neutral density extracted from the multi-model and the multi-model without regularization for the forward returns with 11, 32, 109 and 704 days duration. The densities of the forward returns with 109 and 704 days in the multi-model without regularization look strangely like a Gaussian mixture model. The limit condition is also verified by Fig. 6(c).
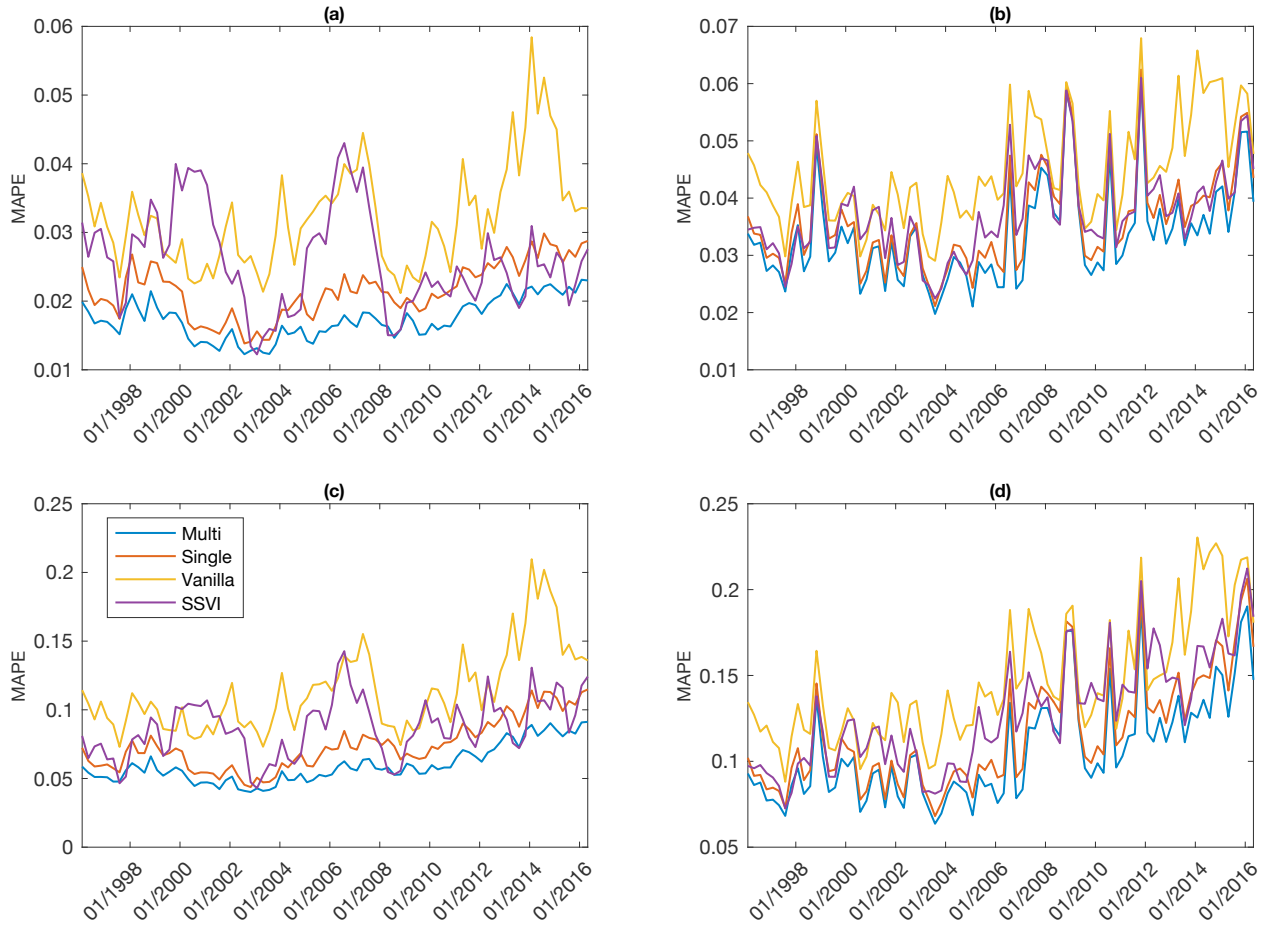
Figure 3: Plots of the MAPEs in each quarter for the neural network models and the SSVI of: (a) implied volatilities in the training set; (b) implied volatilities in the test set; (c) option prices in the training set; (d) option prices in the test set.

5 Conclusion

In this paper, a gated neural network model is developed to predict implied volatility surfaces. Unlike many previous studies where machine learning techniques were mainly used as a "black box" or were less connected with the existing financial theories, our model has taken into account the related important financial conditions and empirical evidence. To the best of our knowledge, this is one of the very first studies which discuss a methodological framework that can integrate the data-driven machine learning algorithms (particularly deep neural networks) with the existing financial theories. The proposed model framework can be easily extended and applied to solve other similar business problems. In addition to methodological contributions, we validate the proposed model empirically with the option data on the S&P 500 index. Compared with the existing studies, our experimental settings are more challenging because the used option data is over twenty years and the options with a short time to maturity are examined. Therefore, our model needs to be robust in order to produce convincing results. As presented in Section 4, the conventional financial conditions and empirical evidence are met empirically; our model outperforms the widely used SSVI model; and it also outperforms its simplified variations – other similar neural network models without incorporating financial conditions and empir-

ical evidence. The last point also justifies the importance of integrating domain knowledge into the model.

Acknowledgement

References

Y. Abu-Mostafa. A method for learning from hints. *Advances in Neural Information Processing Systems*, pages 73–80, 1993.

A. Agarwal, V. Jacob, and H. Pirkul. An improved augmented neural-network approach for scheduling problems. *INFORMS Journal on Computing*, 18(1):119–128, 2006.

F. Ametrano and M. Bianchetti. Everything you always wanted to know about multiple interest rate curve bootstrapping but were afraid to ask. SSRN,
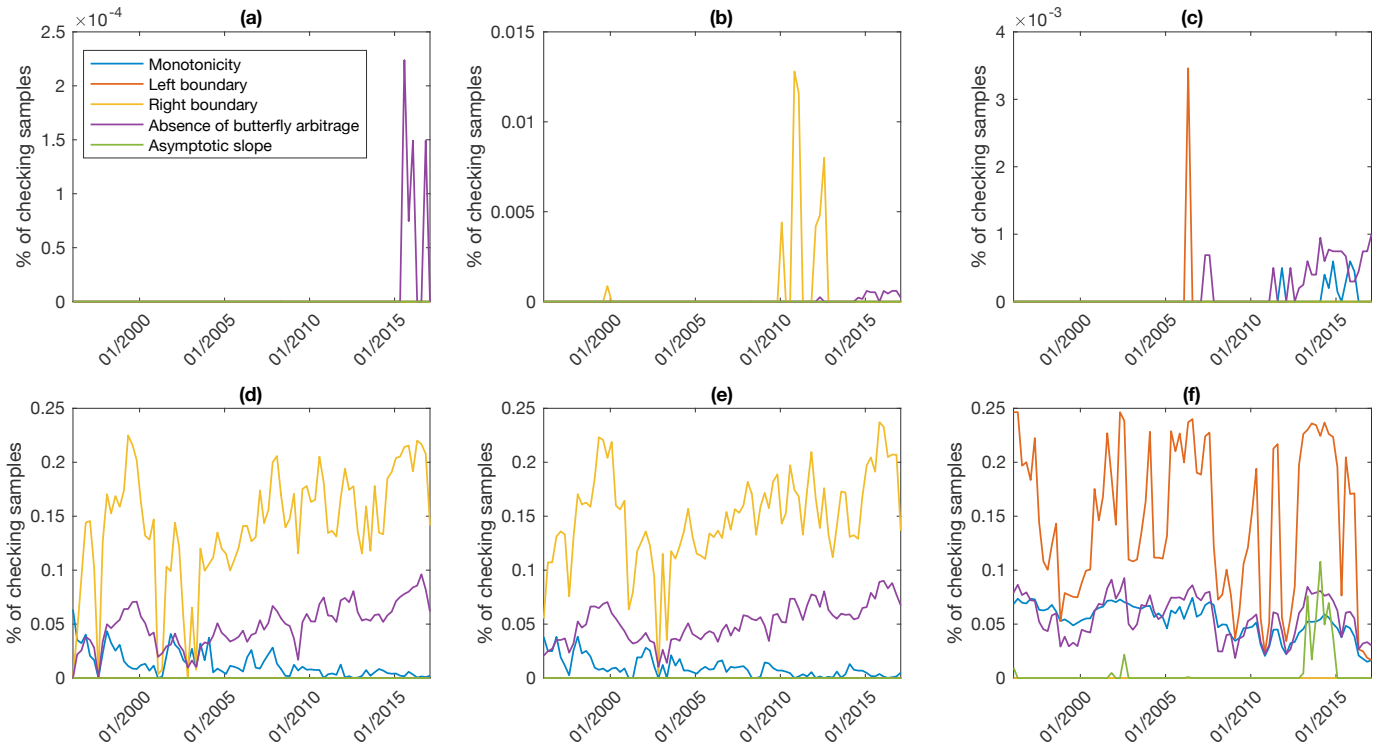
Figure 4: Plots of condition checks of: (a) the multi-model; (b) the single model; (c) the vanilla model; (d) the multi-model with incomplete constraints; (e) the single model with incomplete constraints; (f) the vanilla model with incomplete constraints.

2013. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2219548.

T. Andersen, N. Fusari, and V. Todorov. Short-term market risks implied by weekly options. *Journal of Finance*, 72(3): 1335–1386, 2017.

F. Audrino and D. Colangelo. Semi-parametric forecasts of the implied volatility surface using regression trees. *Statistics and Computing*, 20(4):421–434, 2010.

A. Belloni, R. Freund, and S. Vempala. An efficient rescaled perceptron algorithm for conic systems. *Mathematics of Operations Research*, 34(3):621641, 2009.

J. Bilson, S. Kang, and H. Luo. The term structure of implied dividend yields and expected returns. *Economics Letters*, 128:9–13, 2015.

C. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

F. Black and M. Scholes. The pricing of options and corporate liabilities. *Journal of Political Economy*, 81(3):637–654, 1973.

R. Bliss and N. Panigirtzoglou. Option-implied risk aversion estimates. *Journal of Finance*, 59(1):407–446, 2005.

P. Carr and L. Wu. Stochastic skew in currency options. *Journal of Financial Economics*, 86(1):213–247, 2007.

P. Carr and L. Wu. A new simple approach for constructing implied volatility surfaces. SSRN, 2010. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=1701685.

A. Chockalingam and K. Muthuraman. American options under stochastic volatility. *Operations Research*, 59(4):793–809, 2011.

J. Choe, S. Park, K. Kim, J. H. Park, D. Kim, and H. Shim. Face generation for low-shot learning using generative adversarial networks. *IEEE International Conference on Computer Vision Workshops*, pages 1940–1948, 2017.

T. Coleman, Y. Li, and C. Wang. Stable local volatility function calibration using spline kernel. *Computational Optimization and Applications*, 55(3):675–702, 2013.

R. Cont and J. Da Fonseca. Dynamics of implied volatility surfaces. *Quantitative Finance*, 2(1):45–60, 2002.

S. Corlay. B-spline techniques for volatility modeling. *Journal of Computational Finance*, 19(3):97–135, 2016.

G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2 (4):303–314, 1989.

Ernst and Young. The future of FinTech and financial services: whats the next big bet? *Technical Report*, 2018.

J. Franklin. Recurrent neural networks for music computation. *INFORMS Journal on Computing*, 18(3):321338, 2019.

P. Friz and J. Gatheral. Valuation of volatility derviatives as an inverse problem. *Quantitative Finance*, 5(6):531–542, 2005.

J. Gatheral. A parsimonious arbitrage-free implied volatility parameterization with application to the valuation of volatility derivatives. *Presentation at Global Derivatives & Risk Management*, Madrid, 2004.
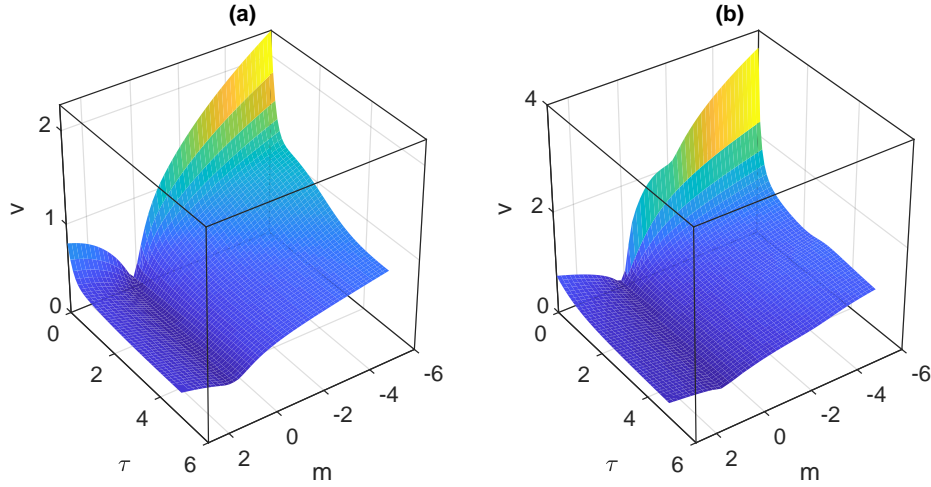
Figure 5: Plots the implied volatility surface on 11/01/2016 for: (a) the multi-model; and (b) the multi-model without regularization.
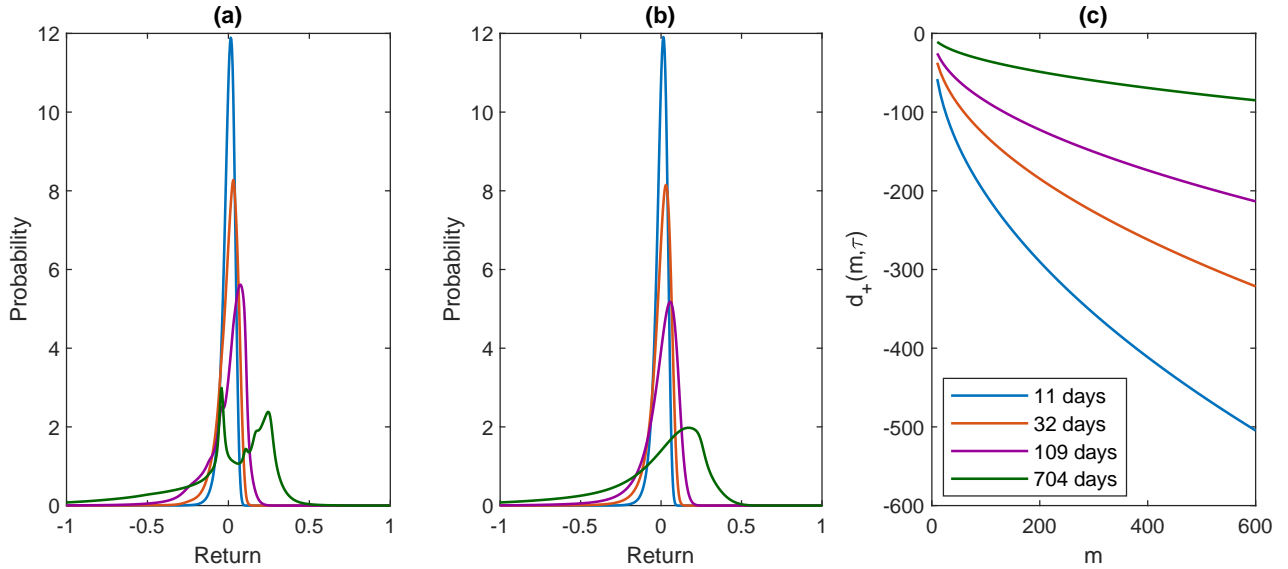

Figure 6: Plots for the forward with 11, 32, 109 and 704 days duration: (a) the density of returns from the multi-model without regularization; (b) the density of returns from the multi-model; and (c) the limit condition $d_+(m, \tau)$ in Theorem 1 for the multi-model on 11/01/2016.

J. Gatheral and A. Jacquier. Arbitrage-free svi volatility surfaces. *Quantitative Finance*, 14(1):59–71, 2014.

V. Gavrishchaka. *Econometric Analysis of Financial and Economic Time Series*, chapter Boosting-based Frameworks in financial modeling: application to symbolic volatility forecasting, pages 123–151. Emerald, 2006.

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

N. Gradojevic, R. Gencay, and D. Kukolj. Option pricing with modular neural networks. *IEEE Transactions on Neural Networks*, 20(4):626–637, 2009.

A. Gulisashvili. *Analytically Tractable Stochastic Stock Price Models*. Springer, 2012.

J. Gupta, R. Sexton, and E. Tunc. Selecting scheduling heuris-

tics using neural networks. *INFORMS Journal on Computing*, 12(2):150–162, 2000.

T. Hendershott, M. Zhang, L. Zhao, and E. Zheng. Special issue of information systems research fintech innovating the financial industry through emerging information technologies. *Information Systems Research*, 28(4):885–886, 2017.

S. Heston. A closed-form solution for options with stochastic volatility with applications to bond and currency options. *Review of Financial Studies*, 6(2):327–343, 1993.

C. Homescu. Implied volatility surface: construction methodologies and characteristics. arXiv, 2011. https://arxiv.org/abs/1107.1834.

K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.

A. Itkin. To sigmoid-based functional description of the volatil-

ity smile. *The North American Journal of Economics and Finance*, 31:264–291, 2015.

C. Kang, W. Kang, and J. Lee. Exact simulation of the Wishart multidimensional stochastic volatility model. *Operations Research*, 65(5):1190–1206, 2017.

D. Kingma and J. Ba. Adam: a method for stochastic optimization. *International Conference on Learning Representations*, pages 1–13, 2015.

A. Kotzé, C. Labuschagne, M. Nair, and N. Padayachi. Arbitrage-free implied volatility surfaces for options on single stock futures. *The North American Journal of Economics and Finance*, 26:380–399, 2013.

S. Kou. A jump-diffusion model for option pricing. *Management Science*, 48(8):1086–1101, 2002.

Y. Lecun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

R. Lee. The moment formula for implied volatility at extreme strikes. *Mathematical Finance*, 14(3):469–480, 2004.

T. Liu, Y. Cui, Q. Yin, W. Zhang, S. Wang, and G. Hu. Generating and exploiting large-scale pseudo training data for zero pronoun resolution. *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pages 102–111, 2017.

M. Malliaris and L. Salchenberger. A neural network model for estimating option prices. *Applied Intelligence*, 3(3):193–206, 1993.

M. Malliaris and L. Salchenberger. Using neural networks to forecast the S&P 100 implied volatility. *Neurocomputing*, 10 (2):183–195, 1996.

O. Mangasarian. Mathematical programming in neural networks. *ORSA Journal on Computing*, 5(4):327–434, 1993.

R. Merton. Option pricing when underlying stock returns are discontinuous. *Journal of Financial Economics*, 3:125–144, 1976.

A. Roy, S. Qureshi, K. Pande, D. Nair, K. Gairola, P. Jain, S. Singh, K. Sharma, A. Jagadale, Y.-Y. Lin, S. Sharma, R. Gotety, Y. Zhang, J. Tang, T. Mehta, H. Sindhanuru, N. Okafor, S. Das, C. Gopal, S. Rudraraju, and A. Kakarlapudi. Performance comparison of machine learning platforms. *INFORMS Journal on Computing*, 31(2):207225, 2019.

J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

K. Shiraya and A. Takahashi. Pricing average and spread options under local-stochastic volatility jump-diffusion models. *Mathematics of Operations Research*, 4(11), 2018.

K. Smith. Neural networks for combinatorial optimization: a review of more than a decade of research. *INFORMS Journal on Computing*, 11(1):15–34, 1999.

V. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 2nd edition, 2000.

Y. Yang, Y. Zheng, and T. Hospedales. Gated neural networks for option pricing: rationality by design. *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

Y. Zheng. *Machine Learning and Option Implied Information*. PhD thesis, Imperial College London, 2018.