Innovative derivative pricing and time series simulation techniques via machine and deep learning

Weilong Fu

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
under the Executive Committee
of the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2022

Abstract

Innovative derivative pricing and time series simulation techniques via machine and deep learning

Weilong Fu

There is a growing number of applications of machine learning and deep learning in quantitative and computational finance. In this thesis, we focus on two of them.

In the first application, we employ machine learning and deep learning in derivative pricing. The models considering jumps or stochastic volatility are more complicated than the Black-Merton-Scholes model and the derivatives under these models are harder to be priced. The traditional pricing methods are computationally intensive, so machine learning and deep learning are employed for fast pricing. In Chapter 2, we propose a method for pricing American options under the variance gamma model. We develop a new fast and accurate approximation method inspired by the quadratic approximation to get rid of the time steps required in finite difference and simulation methods, while reducing the error by making use of a machine learning technique on pre-calculated quantities. We compare the performance of our method with those of the existing methods and show that this method is efficient and accurate for practical use. In Chapters 3 and 4, we propose unsupervised deep learning methods for option pricing under Lévy process and stochastic volatility respectively, with a special focus on barrier options in Chapter 4. The unsupervised deep learning approach employs a neural network as the candidate option surface and trains the neural network to satisfy certain equations. By matching the equation and the boundary conditions, the neural network would yield an accurate solution. Special structures called singular terms are added to the neural networks to deal with the non-smooth and

discontinuous payoff at the strike and barrier levels so that the neural networks can replicate the asymptotic behaviors of options at short maturities. Unlike supervised learning, this approach does not require any labels. Once trained, the neural network solution yields fast and accurate option values.

The second application focuses on financial time series simulation utilizing deep learning techniques. Simulation extends the limited real data for training and evaluation of trading strategies. It is challenging because of the complex statistical properties of the real financial data. In Chapter 5, we introduce two generative adversarial networks, which utilize the convolutional networks with attention and the transformers, for financial time series simulation. The networks learn the statistical properties in a data-driven manner and the attention mechanism helps to replicate the long-range dependencies. The proposed models are tested on the S&P 500 index and its option data, examined by scores based on the stylized facts and are compared with the pure convolutional network, i.e. `QuantGAN`. The attention-based networks not only reproduce the stylized facts, including heavy tails, autocorrelation and cross-correlation, but also smooth the autocorrelation of returns.

Table of Contents

List of Figures

List of Tables

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my advisor Prof. Ali Hirsa. He is a talented researcher with plentiful professional experience. He always leads me to the most interesting research topics and also permits my freedom to pursue my research interests. His invaluable advice, support, and guidance are indispensable for my thesis, and also help me become a curious and independent researcher. It is my great honor to work with him closely, and I will always remember the moments when we worked together.

I would like to extend my sincere thanks to the rest of my thesis committee, Prof. Garud Iyengar, Prof. Agostino Capponi, Prof. Ciamac Moallemi and Prof. Jörg Osterrieder, for generously spending their precious time serving on my thesis committee and for providing insightful comments.

I am very grateful to Prof. Garud Iyengar and Prof. Agostino Capponi for their constructive feedback on the thesis proposal, which helped us improve the proposed procedures and findings in Chapter 3. I am also deeply grateful to Dr. Alireza Javaheri and Mehdi Sonthonnax of Credit Suisse for their time and invaluable advice on Chapter 4, and Prof. Jörg Osterrieder and Christoph Auth of Zurich University of Applied Sciences for their time and invaluable advice on Chapter 5. I really appreciate their guidance and kind help.

I am thankful to my colleagues and friends that I have met throughout my Ph.D. studies. Their company and support make my life more colorful and enjoyable. Many thanks to all the faculty and staff members at the Department of Industrial Engineering and Operations Research for creating such a supportive and lovely community.

Finally, I would like to thank my family for believing in me and their continuous support.

Chapter 1: Introduction

In recent years, there has been a growing number of applications of machine learning and deep learning in finance, including algorithmic trading, price forecasting, fraud detection/prevention, portfolio management, high performance computing, risk management and financial planning. The learning techniques are generally applied with fewer model assumptions and are good at fitting high-dimensional functions.

In this thesis, we discuss two selected applications in finance. In the first one, we employ machine learning and deep learning in derivative pricing. Since the Black-Merton-Scholes (BMS) model [10], more models have been proposed to fill the gap between the theory and the market. The models under Lévy processes take jumps into consideration to better fit the large drawdowns and volatility surfaces, while the models of stochastic volatility introduce a process of volatility to fit the volatility term structure and consider the negative correlation between the underlying asset and volatility. These models are more complicated than the BMS model and the options under these models are harder to price. The traditional pricing methods are computationally intensive, thus machine learning and deep learning are employed for fast pricing. The other application is to simulate financial time series. Simulation of financial time series is challenging because of the complex statistical properties of the real financial data. Traditional simulation methods generate samples with stronger assumptions, while the deep learning models are better at simulating samples that satisfy more statistical properties. The simulated data could be used in risk management, or as the training data in machine learning and deep learning approaches in algorithmic trading and portfolio management.

In this chapter, we introduce the literature that covers the topics in this thesis. In Chapter 2, we develop a pricing method using kernel regression for American options under the variance gamma model. In Chapter 3, we describe a pricing method using unsupervised deep learning for derivative pricing under Lévy processes. In Chapter 4, we propose a pricing method using unsupervised deep learning for barrier options and test the method for the Bergomi model, a high-dimensional stochastic volatility model. In Chapter 5, we present the simulation approach of financial time series using generative adversarial networks and apply them to the S&P 500 index and its options.

1.1 Literature review of option pricing

1.1.1 Option pricing under Lévy processes

Financial models based on Lévy processes are better at describing the fat tails of asset returns and matching the implied volatility surfaces in option markets than the diffusion models, since Lévy processes take jumps into consideration in addition to the Gaussian movements. Some examples are the variance gamma (VG) model [82], the normal inverse Gaussian (NIG) model [5], the tempered stable process (also known as the CGMY model, [15]), the Merton's jump diffusion model [85] and the Kou's double exponential jump diffusion model [72].

Finite difference methods are widely used to price American options under Lévy processes. A backward partial integro-differential equation (PIDE) is solved by the implicit scheme in [54], the explicit-implicit scheme in [25] and the fixed point iteration in [104]. Aside from the backward PIDE are the forward PIDE in [51] and the fractional partial differential equation (FPDE) in [19, 84]. Besides the finite difference method, simulation is also a traditional method for pricing American options, e.g., the least square Monte Carlo (LSM) method [78].

Traditional methods are usually computationally expensive. Here are some acceleration methods:

- Transformation-based methods help to get rid of the finite difference in each time step. After the Fast-Fourier-Transform (FFT) was suggested in [17] to price European options, it is employed in the Q-FFT method [90] and CONV method [79] to price Bermudan and American options. Similarly, Fourier-cosine expansions are used in the COS method [33] and Shannon wavelet expansions in the SWIFT method [83] to evaluate the early exercise.

- Some methods derive a PDE out of the PIDE to accelerate instead of dealing with the PIDE directly. A PDE in the Fourier space is solved in the Fourier Space Time-stepping (FST) method [63]. A pseudo-parabolic equation is solved in [62] and a PDE of the integral term is solved in [18].

- Approximation formulae also accelerate computation. For American options, it is equivalent to price the early exercise premium. The quadratic approximation was proposed in [6] to price the premium for the BMS model, which was later elaborated in [65] to further reduce its error. An approximation following the quadratic approximation is given in [72] under a double exponential jump diffusion model and in [14] under a mixed-exponential jump diffusion model. The quadratic approximation is also applied to the VG process in [44], where they first find the exercise boundary of American options through a fixed point system and then solve an approximated equation. However, the approximated equation introduces errors since it cannot completely describe the surface of the premium of the American options. In [36], the authors use non-parametric regression to reduce the error of quadratic approximation.

1.1.2 Pricing barrier options under stochastic volatility

Stochastic volatility models are good at replicating the volatility smiles and the correlation between the underlying asset and volatility among the pure diffusion frameworks. Some examples are the Heston model [49], the SABR model [45] and the Bergomi model [9]. The Bergomi model is more complex since it includes multiple volatility factors and is shown to be better at replicating the term structure of forward variances. However, since the stochastic volatility models define additional dynamics of volatility, option pricing under these models is generally more challenging than that under the models which only consider dynamics of the underlying asset.

Barrier options are path-dependent options whose payoff depends on whether or not the underlying asset has reached the barrier level. They are classified into up/down-and-in/out calls/puts based on the position of the barrier level, its payoff after the barrier level is reached, and the corresponding vanilla option. Traditional methods to price barrier options under the stochastic volatility models include the finite difference method in [21, 42, 68], forward evolution equations for knock-out options in [16] and the simulation method in [1, 26]. Also, an analytic approximation for barrier options under stochastic volatility models was proposed in [38].

1.1.3 Option pricing using machine and deep learning

Recently, many pricing approaches are proposed based on machine learning and deep learning.

- In [36], kernel regression is applied to pre-calculated data to price American options. The PIDE is converted into an ordinary integro-differential equation (OIDE), and kernel regression is used to calculate a correction term in the OIDE to reduce pricing errors. This work is included in Chapter 2.

- In supervised deep learning, the neural network is used as a function w.r.t. all the parameters involved in the model. The networks are trained to fit the option price surface or the volatility surface given labels generated by other pricing methods. Recently, this idea has drawn growing attention in the literature (see e.g. [8, 34, 53, 77, 61, 7, 57]). The advantage of neural network approaches is that they are fast in computing prices and volatilities once trained and thus they are a good choice for model calibration. However, in supervised learning, it is pretty costly to generate the training labels by other pricing methods, e.g. finite differences, FFT, or simulation.

- There are unsupervised deep learning approaches as well. The option price surface for a given model is a solution of a PDE or a PIDE and thus the pricing problem is reduced to solving equations. Neural networks have been used to solve PDEs in [76, 73, 74, 93], where the networks are employed as the approximated solutions and the derivatives are calculated either by finite difference or back-propagation [95]. The networks are trained to match the PDE and boundary conditions. In this way, the PDE is solved and no labels are needed for training neural networks. Additionally, several modifications are made to deal with high-dimensional problems. In [97], the second-order derivatives are estimated by Monte Carlo simulation. This approach has been applied to vanilla options but not yet to barrier options because common smooth neural networks cannot match the discontinuous boundary conditions or replicate the asymptotic behaviors of barrier options at short maturity. We extend this method in Chapter 3 and 4.

- In [46], the option prices are solved by forward-backward stochastic differential equations. Neural networks are used to approximate the diffusion term in the stochastic differential equations, which is related to the gradient of the solution. Since then, some variants have been applied to the barrier options in [110, 39].

1.2 Literature review of time series simulation via generative adversarial networks

Training and evaluation of trading strategies need lots of data. Due to the limited amount of real data, there is a growing need to be able to simulate realistic financial data which satisfies the stylized facts. There has already been a vast literature of financial time series models. The generalized autoregressive conditional heteroskedasticity (GARCH) [11] model and its variants are applied to the stock prices and indices. The BMS model [10], the Heston model [49], the VG model [82], etc. are applied to the option surfaces. The parametric models are popular for their simplicity, mathematical explicitness and robustness. However, it is difficult for a parametric model to fit all the major stylized facts.

Recently, more data-driven approaches based on generative adversarial networks (GANs) [41] are proposed to deal with the problem. The GAN includes a generator, which is used to generate samples, and a discriminator, which is responsible for judging whether the generated samples are similar enough to the real data. The applications of GANs to financial time series range from the underlying asset price prediction [113, 112, 71] and simulation [100, 28, 69, 35, 106] to the option surface simulation [105]. Some more GANs of time series are proposed in [101, 87, 32, 20, 109, 70] and some more generative models of time series are in [56, 89, 107]. However, the network structures of the GANs in financial time series simulation are mostly limited in convolutional networks [75] and recurrent networks [55, 23].

There have been various different GANs which employ the attention mechanism [4, 80] to improve their performance, e.g., the convolutional networks with attention [111, 12, 27], and the transformer networks [64, 59]. But the convolutional networks with attention and the transformer networks have not been tested on financial time series. We have developed and tested the attention-based GANs on the financial data in Chapter 5.

The regularized GAN, which was proposed in [29], employs a pre-trained network as a second discriminator, which could be used to emphasize selected stylized facts, e.g. high kurtosis. However, this approach does not work as we expected, and it is shown that the discriminator trained on one dataset may not be suitable to distinguish the real data in another dataset.

Chapter 2: Pricing American options under variance gamma using non-parametric regression

2.1 Introduction

In this chapter, we aim to find a new method for pricing American options under the pure jump model, with a good balance between speed and accuracy. We will focus on the variance gamma (VG) model for simplicity, while it can be generalized to other pure jump models. First, the method avoids dealing with time steps through similar steps described by the quadratic approximation in [6]. Second, we add a correction term to the approximated equation to reduce the error caused by the approximation step. Third, we employ kernel regression, which is a nonparametric machine learning technique, to estimate the correction term using pre-calculated data. The method is thus called Quadratic Approximation with Kernel Regression (`QAKR`). When compared with the methods of learning the price surface or the volatility surface directly like those by neural networks, the `QAKR` method does not need as much data. Also, the price surface learned by neural networks might be irregular due to the complex structure of neural networks, while the `QAKR` method avoids that problem since it approximates the option premium by a smooth exponential function. When compared with the finite difference methods and the transformation-based methods, the `QAKR` method is more favorable if we need to compute many option prices, e.g., model calibration.

The chapter is organized as follows: In Section 2.2 we do a quick review of the VG model and pricing of European and American options under VG. In Section 2.3, we find a simple way to apply the Ju-Zhong method [65] to VG. Even though we do not expect this simple approach would yield an acceptable solution, we think it is worth examining it. Our numerical results show that the error can be within the bid-ask spread but often beyond it. In Section 2.4, we elaborate the development of the `QAKR` method, summarize the algorithm and give some high-level intuitions.

In Section 2.5 we present the results of numerical experiments and show that the `QAKR` method performs well in both speed and error. In Section 2.6, we conclude the chapter and discuss some possible future research.

2.2 The variance gamma model

Let $b(t;\theta,\sigma) = \theta t + \sigma W(t)$ be a Brownian motion with drift rate θ and volatility σ, where $W(t)$ is a one-dimensional standard Brownian motion. Meanwhile, let $\gamma(t;1,\nu)$ be a gamma process with mean rate 1 and variance rate ν. The gamma process $\gamma(t;1,\nu)$ has independent gamma increments with mean h and variance νh over time intervals of length h.

Then the three-parameter VG process $X(t;\sigma,\theta,\nu)$ is defined by

$$X(t;\sigma,\theta,\nu) = b(\gamma(t;1,\nu),\theta,\sigma).$$

The compound process $X(t;\sigma,\theta,\nu)$ is a time-changed Brownian motion with drift and its increments have a fat-tailed distribution.

The Lévy density of the VG process is given by

$$k(x) = \frac{e^{-\lambda_p x}}{\nu x} 1_{x>0} + \frac{e^{-\lambda_n |x|}}{\nu |x|} 1_{x<0}, \tag{2.1}$$

where

$$\lambda_p = \left(\frac{\theta^2}{\sigma^4} + \frac{2}{\sigma^2 \nu}\right)^{\frac{1}{2}} - \frac{\theta}{\sigma^2} \tag{2.2}$$

and

$$\lambda_n = \left(\frac{\theta^2}{\sigma^4} + \frac{2}{\sigma^2 \nu}\right)^{\frac{1}{2}} + \frac{\theta}{\sigma^2}. \tag{2.3}$$

Also, the characteristic exponent of the VG process is given by

$$\phi(\xi) = -\frac{1}{\nu}\ln\left(1 + \frac{\sigma^2\nu\xi^2}{2} - \mathrm{i}\theta\nu\xi\right)$$

such that $\ln\mathbb{E}\left(e^{\mathrm{i}\xi X(t)}\right) = t\phi(\xi)$ holds.

The risk neutral process of the stock price under the VG model is given by

$$S(t) = S(0)\exp((r-q)t + X(t) + \omega t), \tag{2.4}$$

where r is the risk-free interest rate, q is the dividend rate of the stock, and $\omega = \frac{1}{\nu}\ln(1-\sigma^2\nu/2-\theta\nu)$; ω is calculated such that $\mathbb{E}(S(t)) = S(0)\exp((r-q)t)$, i.e., the discounted price process $e^{-(r-q)t}S(t)$ is a martingale. The martingale property of the discounted price is equivalent with the no-arbitrage condition.

2.2.1 European options under VG

Let $\Theta = \{r, q, T, \sigma; \nu, \theta\}$ be the parameter set in the VG model. Then the price of a European put option with the strike K and the maturity T under the parameter Θ is given by

$$p(S(t), t; K, \Theta) = e^{-r(T-t)}\mathbb{E}_t((K - S(T))^+).$$

According to [81], the price of a European put option on a stock following Equation (2.4) is given by

$$\begin{aligned} p(S(0), 0; K, \Theta) &= Ke^{-rT}\Psi\left(-d\sqrt{\frac{1-c_2}{\nu}}, -\alpha\sqrt{\frac{\nu}{1-c_2}}, \frac{T}{\nu}\right) \\ &\quad -S(0)e^{-qT}\Psi\left(-d\sqrt{\frac{1-c_1}{\nu}}, -(\alpha+s)\sqrt{\frac{\nu}{1-c_1}}, \frac{T}{\nu}\right), \end{aligned} \tag{2.5}$$

where

$$d = \frac{1}{s}\left(\ln\frac{S(0)}{K} + (r-q)T + \frac{T}{\nu}\ln\left(\frac{1-c_1}{1-c_2}\right)\right),$$

$c_1 = v(\alpha+s)^2/2$, $c_2 = v\alpha^2/2$, $\alpha = \xi s$, $\xi = \theta/\sigma^2$, and $s = \sigma/\sqrt{1+\frac{\theta^2 v}{2\sigma^2}}$ and the function Ψ is defined in terms of the modified Bessel function of the second kind and the degenerate hyper-geometric function of two variables (see [81]).

Other than the explicit formula, having the characteristic function of the VG process, one can price European options under VG by the Fast-Fourier-Transform (FFT) technique (see [17]).

2.2.2 American options under VG

When the risk neutral stock price is $S(t)$, by its Markov property, the American option is given by

$$P(S(t), t; K, \Theta) = \sup_{t \le \tau \le T} \mathbb{E}_t(e^{-r(\tau-t)}(K - S(\tau))^+),$$

where the supremum is taken over all stopping times τ defined on the probability space with regard to the filtration generated by the stock price $S(t)$. For American put options, at each t, there exists a critical stock price $S^\star(t) \le K$, such that if $S(t) > S^\star(t)$, the value of the option is greater than the immediate exercise value and the optimal action is to wait, while if $S(t) \le S^\star(t)$, the value of the option is the same as the immediate exercise value and the optimal action is to exercise the option. In the first quadrant of a two-dimensional space, $\{(S,t) : S > S^\star(t), 0 \le t \le T\}$ is called the continuation region and $\{(S,t) : S \le S^\star(t), 0 \le t \le T\}$ the exercise region.

It is shown in [54] that the price of a European option $p(S, t; K, \Theta)$ and the price of an American option $P(S, t; K, \Theta)$ in the continuation region satisfy this PIDE:

$$\begin{aligned} \int_{-\infty}^{\infty} \left[V(Se^x, t) - V(S,t) - \frac{\partial V}{\partial S}(S,t) S(e^x - 1) \right] k(x) dx & \\ + \frac{\partial V}{\partial t}(S,t) + (r-q) S \frac{\partial V}{\partial S}(S,t) - rV(S,t) &= 0. \end{aligned} \tag{2.6}$$

Here $V(S,t)$ is the price and $k(x)$ is the Lévy density given by Equation (2.1).

By making changes of the variables, $x = \ln S$, $\tau = T - t$ and $w(x,\tau) = V(S,t)$, we get

$$\begin{aligned}
\frac{\partial w}{\partial x}(x,\tau) &= S\frac{\partial V}{\partial S}(S,t),\\
\frac{\partial w}{\partial \tau}(x,\tau) &= -\frac{\partial V}{\partial t}(S,t),\\
w(x+y,\tau) &= V(Se^{y},t),
\end{aligned}$$

and the following equation

$$\begin{aligned}
\int_{-\infty}^{\infty}\left[w(x+y,\tau) - w(x,\tau) - \frac{\partial w}{\partial x}(x,\tau)(e^{y}-1)\right]k(y)dy & \\
-\frac{\partial w}{\partial \tau}(x,\tau) + (r-q)\frac{\partial w}{\partial x}(x,\tau) - rw(x,\tau) &= 0. \qquad (2.7)
\end{aligned}$$

This equation can be utilized by the finite difference method to price American options under VG (see e.g. [54]).

2.3 A simple approach for pricing under VG

Before we propose the `QAKR` method, we want to test whether the methods for pricing under the BMS model can be borrowed for the VG model. In this section, we give a simple approach. The idea is to approximate the VG premium (American minus European) with the BMS premium with suitable parameters and the key step is to approximate Equation (2.7) with the BMS equation. The derivation of the approach is described in Appendix A.1 and the steps are as follows:

- First, we calculate the difference of American and European options of the BMS model with the volatility replaced by $\sqrt{\sigma^2(\epsilon)}$ and the dividend rate replaced by $q - \omega(\epsilon)$ where

$$\begin{aligned}
\sigma^2(\epsilon) &= \int_{|y|\le\epsilon} y^2 k(y)dy,\\
\omega(\epsilon) &= \int_{|y|>\epsilon} (1-e^{y})k(y)dy.
\end{aligned}$$

Here $k(x)$ is the Lévy density of the VG process in Equation (2.1). The price of American options is given by the Ju-Zhong method [65] and the price of European options is given by the B-S formula. A detailed description of the Ju-Zhong method is given in Appendix A.2.

- Then we add the difference from the last step to the VG European price to get an approximated VG American price. The VG European price can be given either by the explicit expression (2.5) or the FFT technique. We use FFT in this chapter.

The approach is referred to as the simple approach hereafter. We set ϵ to 0.65 based on empirical tests. The simple approach is very fast thanks to the Ju-Zhong method, but our empirical results show it is not "always" within the bid-ask spread. So there is a need to develop a more accurate and efficient method.

2.4 Development of the QAKR method

We need a more accurate methodology than the simple approach that was introduced in Section 2.3. In this section, we propose and develop the QAKR method for pricing American options under VG.

From Section 2.4.1 to 2.4.3, we explain the development of the method from the partial integro-differential equation (PIDE) of the VG model, including accelerating by the quadratic approximation and reducing errors by nonparametric regression. Section 2.4.4 introduces a property that simplifies calculation. Section 2.4.5 summarizes the method into an algorithm. Section 2.4.6 gives some insights of the method and explains why it works well.

2.4.1 From PIDE to OIDE

Denote $\omega = -\int_{-\infty}^{\infty}(e^y - 1)k(y)dy$, and then Equation (2.7) can be simplified to

$$\begin{aligned}\int_{-\infty}^{\infty}\left[w(x+y,\tau) - w(x,\tau)\right]k(y)dy - \frac{\partial w}{\partial \tau}(x,\tau) & \\ +(r-q+\omega)\frac{\partial w}{\partial x}(x,\tau) - rw(x,\tau) &= 0. \end{aligned} \tag{2.8}$$

The early exercise premium is given by

$$w(x,\tau;K,\Theta) = P(e^x, T-\tau;K,\Theta) - p(e^x, T-\tau;K,\Theta),$$

which is the difference between an American option price and a European option price. It satisfies Equation (2.8) in the continuation region, and is equal to $K - e^x - p(e^x, T-\tau;K,\Theta)$ in the exercise region. The continuation region $S(t) > S^\star(t)$ becomes $x > \ln(S^\star(T-\tau))$ and the exercise region $S(t) \le S^\star(t)$ becomes $x \le \ln(S^\star(T-\tau))$ since $x = \ln(S)$ and $t = T-\tau$.

The finite difference method makes use of the PIDE (2.8) and divides the time interval into many steps that have to be solved at each time step. The key idea to accelerate is to get rid of the time steps and just focus on the last step. So we want to approximate the PIDE by an ordinary integro-differential equation (OIDE).

We approximate $w(x,\tau)$ in the same way as the quadratic approximation in [6]. Let $w(x,\tau) = h(\tau)f(x,h(\tau))$, where $h(\tau) = 1 - e^{-r\tau}$, then Equation (2.8) becomes

$$\begin{aligned}\int_{-\infty}^{\infty}\left[f(x+y,h(\tau)) - f(x,h(\tau))\right]k(y)dy + (r-q+\omega)\frac{\partial f}{\partial x}(x,h(\tau))&\\ -\frac{r}{h(\tau)}f(x,h(\tau)) - r(1-h(\tau))\frac{\partial f}{\partial h(\tau)}(x,h(\tau)) &= 0.\end{aligned}$$

According to [6], $(1-h(\tau))f_h(x,h)$ is close to 0 but not exactly 0. We replace the term $r(1-h(\tau))h(\tau)f_h(x,h)$ with a correction term $\mathcal{E}(x;K,\Theta)$ to remove the derivative with respect to time. Then we have the OIDE

$$\begin{aligned}\int_{-\infty}^{\infty}\left[w(x+y,T) - w(x,T)\right]k(y)dy - \frac{r}{1-e^{-rT}}w(x,T)&\\ +(r-q+\omega)\frac{\partial w}{\partial x}(x,T) &= \mathcal{E}(x;K,\Theta), \qquad (2.9)\end{aligned}$$

where $\mathcal{E}(x;K,\Theta)$ is close to 0 compared with the other terms on the l.h.s.

Let $x^\star = \ln(S^\star(0))$ be the exercise boundary at maturity. The premium $w(x,T;K,\Theta)$ should satisfy Equation (2.9) on $x > x^\star$ (the continuation region) and be equal to $K - e^x - p(e^x, 0;K,\Theta)$

on $x \le x^\star$ (the exercise region).

2.4.2 Solving the OIDE by parameterization

In this part, we aim to solve Equation (2.9) given $\mathcal{E}(x; K, \Theta)$. We leave it to Section 2.4.3 to determine the value of $\mathcal{E}(x; K, \Theta)$.

Although we get the approximation equation (2.9), we cannot solve it explicitly due to the integral term. So we consider to solve it numerically. We use an exponential function as an approximation of $w(x, T; K, \Theta)$ in the continuation region, which coincides with the explicit solution of the approximation function in [6]:

$$w(x, T; K, \Theta) = \begin{cases} K - e^x - p(e^x, 0; K, \Theta) & x \le x^\star \\ \exp(\lambda(x - x^\star) + b) & x > x^\star \end{cases}, \tag{2.10}$$

where $w(x, T; K, \Theta)$ is set to be continuous at $x = x^\star$. There are three parameters in Equation (2.10), but by continuity at $x = x^\star$ we can solve it for b and would get $b = \ln(K - e^{x^\star} - p(e^{x^\star}, 0; K, \Theta))$. Thus there are two independent parameters in the approximation function.

After parameterizing the premium $w(x, T; K, \Theta)$, we parameterize the l.h.s. of Equation (2.9). Define

$$\begin{aligned} g(x; K, \lambda, x^\star, \Theta) \quad &= \quad (r - q + \omega)\frac{\partial w}{\partial x}(x, T; K, \Theta) - \frac{r}{1 - e^{-rT}} w(x, T; K, \Theta) \\ &\quad + \int_{-\infty}^{\infty} (w(x + y, T; K, \Theta) - w(x, T; K, \Theta))k(y)dy. \end{aligned} \tag{2.11}$$

The parametrized OIDE is

$$g(x; K, \lambda, x^\star, \Theta) = \mathcal{E}(x; K, \Theta).$$

We attempt to make $g(x; K, \lambda, x^\star, \Theta)$ close to $\mathcal{E}(x; K, \Theta)$ at every x on the region $x > x^\star$ by minimizing the loss function w.r.t. λ and $x^\star$:

$$\ell(\lambda, x^\star; \Theta) = \sum_{i=0}^{N} (g(x_i; K, \lambda, x^\star, \Theta) - \mathcal{E}(x_i; K, \Theta))^2. \tag{2.12}$$

We choose $N = 6$ in our numerical experiments based on the empirical results. We also choose $x_i = x^\star + \frac{2i}{N}(\ln(K) - x^\star)$ for $i = 0, 1, \ldots, N$, which are symmetric w.r.t. $\ln(K)$. The choice of x_i is to make Equation (2.9) hold for both in-the-money options and out-of-the-money options. Note that $x^\star < \ln(K)$ always holds for put options so the choice of x_i is valid whatever the value of $x^\star$ is.

After we solve parameters λ and $x^\star$ that minimize the loss function (2.12), the approximated price of the American put is given by

$$P(e^x, 0; K, \Theta) \approx \begin{cases} K - e^x & x \le x^\star \\ p(e^x, 0; K, \Theta) + \exp(\lambda(x - x^\star) + b) & x > x^\star \end{cases}.$$

2.4.3 Choosing the correction term $\mathcal{E}(x; K, \Theta)$

The approximation in Equation (2.10) calculates the premium given the parameters λ and $x^\star$. The minimization problem of the loss function (2.12) gives the solution of λ and $x^\star$ provided we have a suitable $\mathcal{E}(x_i; K, \Theta)$. If we can find the suitable $\mathcal{E}(x_i; K, \Theta)$ as a function of Θ, we can calculate the premium given Θ.

We find the value of $\mathcal{E}(x_i; K, \Theta)$ in the following way. First, we need to determine λ and $x^\star$ given Θ. We can calculate the price of American options $P(e^x, 0; K, \Theta)$ by the finite difference method or the CONV method (in fact any valid method for pricing American options) and the price of European options $p(e^x, 0; K, \Theta)$ by the explicit expression or FFT, and then obtain the true value of $x^\star$ from the American option pricing method and λ by regressing $\ln(P(e^x, 0; K, \Theta) - p(e^x, 0; K, \Theta))$, $x > x^\star$ over x and taking the slope. Then the values of $x^\star$ and λ make the approximation (2.10) very close to the true value of the premium. We can consider them the optimal parameters. This means the difference between the approximation (2.10) and the true premium is smaller than a typical bid-ask spread in practice, and the approximation (2.10) can serve as a good approximation of the premium. This would be further discussed in Section 2.5.3.

Next we decide $\mathcal{E}(x_i; K, \Theta)$. Let $x^\star(K, \Theta)$ and $\lambda(\Theta)$ be the functions of optimal parameters depending on the parameter Θ, which is obtained from the previous step. If we take $\mathcal{E}(x_i; K, \Theta) = g(x_i; K, \lambda(\Theta), x^\star(K, \Theta), \Theta)$, it is obvious that the optimal value of (2.12) is 0 with the optimal solution $\lambda(\Theta)$ and $x^\star(K, \Theta)$.

Now we know how to determine $\mathcal{E}(x_i; K, \Theta)$ if we know prices of American options. However, in the pricing routine, we should know $\mathcal{E}(x_i; K, \Theta)$ before prices of American options. So we need to employ a flexible machine learning technique to learn the value of $\mathcal{E}(x_i; K, \Theta)$.

To elaborate, for many Θ's, we first calculate prices of American options, the optimal parameters $\lambda(\Theta)$ and $x^\star(K, \Theta)$ and then the value of $\mathcal{E}(x_i; K, \Theta) = g(x_i; K, \lambda(\Theta), x^\star(K, \Theta), \Theta)$ for each i. Then we fit the surface of $g(x_i; K, \lambda(\Theta), x^\star(K, \Theta), \Theta)$ over Θ for each i using nonparametric regression. By regression, we assume that $g(x_i; K, \lambda(\Theta), x^\star(K, \Theta), \Theta)$ is close to a continuous function w.r.t. Θ. Let $\hat{g}_i(K, \Theta)$ be the estimate from the regression for each i and let $\mathcal{E}(x_i; K, \Theta) = \hat{g}_i(K, \Theta)$ in the loss function (2.12).

In this way, we use a nearly optimal $\mathcal{E}(x_i; K, \Theta)$ in the loss function (2.12) and the solution λ and $x^\star$ are also nearly optimal. We do not have to calculate $g(x_i; K, \lambda(\Theta), x^\star(K, \Theta), \Theta)$ for each Θ, which speeds up pricing exceedingly.

Moreover, we can use the similar methodology to estimate $\lambda(\Theta)$ and $x^\star(K, \Theta)$ from the pre-calculated quantities and use the estimate as an initial solution in the optimization problem of Equation (2.12) to save time.

2.4.4 Scalability of prices w.r.t. S and K

According to the property of American and European options and the definitions of $w(x, 0; K, \Theta)$ and $g(x; K, \lambda, x^\star, \Theta)$, we have the properties

$$\begin{aligned} p(\alpha\, S, 0; \alpha\, K, \Theta) &= \alpha\, p(S, 0; K, \Theta), \\ P(\alpha\, S, 0; \alpha\, K, \Theta) &= \alpha\, P(S, 0; K, \Theta), \\ w(x + \ln \alpha, 0; \alpha\, K, \Theta) &= \alpha\, w(x, 0; K, \Theta). \end{aligned}$$

Consequently, the exercise boundary $x^\star$ changes along with $\ln(K)$ because $x^\star = \inf\{x : P(e^x, 0; K, \Theta) > K - e^x\}$. If we change K to $\alpha\, K$, then $x^\star$ changes to $x^\star + \ln \alpha$ and $g(x + \ln \alpha; \alpha\, K, \lambda, x^\star + \ln \alpha, \Theta) = \alpha\, g(x; K, \lambda, x^\star, \Theta)$.

λ is the slope of $\ln(P(e^x, 0; K, \Theta) - p(e^x, 0; K, \Theta)), x > x^\star$ against x. It remains the same after changing K to αK.

The definition of x_i makes it shift along with $x^\star$ and $\ln(K)$. If K changes to $\alpha\, K$, x_i will change to

$$x_i' = x^\star + \ln \alpha + \frac{2i}{N}(\ln(\alpha\, K) - (x^\star + \ln \alpha)) = x_i + \ln \alpha$$

and then

$$g(x_i'; \alpha\, K, \lambda, x^\star + \ln \alpha, \Theta) = \alpha\, g(x_i; K, \lambda, x^\star, \Theta).$$

Since $\hat{g}_i(K, \Theta)$ is an estimate of $g(x_i; K, \lambda, x^\star, \Theta)$, we get

$$\hat{g}_i(\alpha K, \Theta) = \alpha\, \hat{g}_i(K, \Theta).$$

In conclusion, we do not have to calculate $g(x_i; K, \lambda(\Theta), x^\star(K, \Theta), \Theta)$ for different K's. We only need a fixed K_0 to estimate $\hat{g}_i(K_0, \Theta)$ and then

$$\hat{g}_i(K, \Theta) = \frac{K}{K_0} \hat{g}_i(K_0, \Theta).$$

2.4.5 Algorithmic overview of the `QAKR` method

The first part is pre-calculation, which is done prior to pricing:

- Choose a set of $\{\Theta_j\}_{j=1}^n$, where $\Theta = (r, q, T, \sigma, \nu, \theta)$ includes all the parameters. Calculate prices of American options $P(S, 0; K_0, \Theta_j)$ by an American option pricing routine, e.g. the CONV method (see [79]), and prices of European options $p(S, 0; K_0, \Theta_j)$ by the FFT technique (see [17]) for $1 \le j \le n$ and $K_0 = 1000$.

- Get the exercise boundary $x^\star(K_0, \Theta_j)$ from the American option pricing routine and regress

$$\ln(P(e^x, 0; K_0, \Theta_j) - p(e^x, 0; K_0, \Theta_j)), x > x^\star$$

  over x to get the slope $\lambda(\Theta_j)$ for each Θ_j.

- Calculate $g(x_i; K_0, \lambda(\Theta_j), x^\star(K_0, \Theta_j), \Theta_j)$ from Equation (2.11) for $1 \le j \le n$ and $0 \le i \le N$.

- Store the data

$$(x^\star(K_0, \Theta_j), \lambda(\Theta_j)), 1 \le j \le n$$

  and

$$g(x_i; K_0, \lambda(\Theta_j), x^\star(K_0, \Theta_j), \Theta_j), 1 \le j \le n, 0 \le i \le N.$$

The second part is the pricing routine: Given the strike K, the stock price $S(0)$, and all the parameters $\Theta = (r, q, T, \sigma, \nu, \theta)$:

- Use a nonparametric regression routine to estimate $\hat{g}_i(K, \Theta)$ from

$$\frac{K}{K_0} g(x_i; K_0, \lambda(\Theta_j), x^\star(K_0, \Theta_j), \Theta_j), 1 \le j \le N.$$

- Use a nonparametric regression routine to estimate the initial solution $(\lambda_0, x_0^\star)$ for the next step from

$$(\lambda(\Theta_j), x^\star(K_0, \Theta_j) + \ln(K/K_0)), 1 \le j \le N.$$

- Minimize the loss function w.r.t. λ and $x^\star$

$$\ell(\lambda, x^\star; \Theta) = \sum_{i=0}^{N} (g(x_i; K, \lambda, x^\star, \Theta) - \hat{g}_i(K, \Theta))^2.$$

- Get the price

$$P(S(0), 0; K, \Theta) \approx \begin{cases} K - S(0) & S(0) \leq \exp(x^\star) \\ p(S(0), 0; K, \Theta) + \exp(\lambda(\ln(S(0)) - x^\star) + b) & S(0) > \exp(x^\star) \end{cases} \tag{2.13}$$

where $b = \ln(K - e^{x^\star} - p(e^{x^\star}, 0; K, \Theta))$.

2.4.6 Insights into the `QAKR` method

Figure 2.1 shows the framework of the `QAKR` method. The circled numbers emphasize the most important parts in the method.

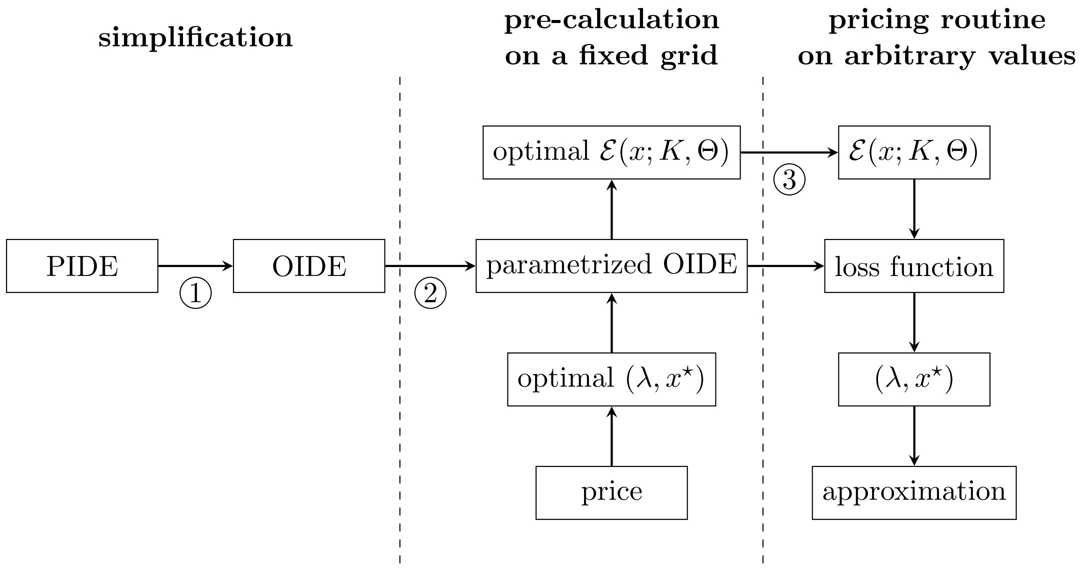

simplification
pre-calculation on a fixed grid
pricing routine on arbitrary values
PIDE
OIDE
①
②
③
optimal $\mathcal{E}(x; K, \Theta)$
parametrized OIDE
optimal $(\lambda, x^\star)$
price
$\mathcal{E}(x; K, \Theta)$
loss function
$(\lambda, x^\star)$
approximation

Figure 2.1: Framework of the `QAKR` method.

First, it transforms the PIDE (2.8) into the OIDE (2.9), which is step ① in Figure 2.1. In this step we get rid of the time steps, which is time-consuming in the finite difference method.

It reduces the calculation time from $O(N_t N_s)$ to $O(N_s)$ where N_t is the number of time steps and N_s the size of the grid with regard to the stock price. Meanwhile, we keep a correction term to improve the accuracy.

Second, it parameterizes the OIDE and turns the problem of solving an equation into an optimization problem, which is step ② in Figure 2.1. This step reduces the unknown function $w(x)$, which is infinite-dimensional, to the correction term $(\mathcal{E}(x_i; K, \Theta))_{i=0}^{N}$, which is only an $(N+1)$-dimensional vector. This is essentially dimension reduction.

Third, as mentioned earlier the method employs nonparametric regression to make use of the information from the pre-calculated data, which is step ③ in Figure 2.1. If we make use of the reduction from the price to the correction term $(\mathcal{E}(x_i; K, \Theta))_{i=0}^{N}$, we can learn the function $\mathcal{E}(x_i; K, \Theta)$ w.r.t Θ.

To summarize the main idea of the `QAKR` method, it should be that the method reduces the solution of the PIDE (2.8) into a low-dimensional correction term vector, uses a nonparametric machine learning technique to fit the surface of the correction term, and then reverses the estimate to an approximated price curve of American options.

2.5 Numerical experiments

2.5.1 Comparison of methods

The range of parameters under consideration is

$$\begin{aligned}\{\Theta = (r, q, T, \sigma, \nu, \theta) : \quad & 0 \le r, q \le 0.1, 0.1 \le T \le 3, \\ & 0.1 \le \sigma \le 0.4, 0.1 \le \nu \le 0.6, -0.5 \le \theta \le -0.1\}.\end{aligned}$$

We pick $S_0 = 2900$ as it is close to the level of S&P 500 Index spot at the time we were writing this chapter.

We compare the following methods in our numerical experiments: the finite difference method in [54], least square Monte Carlo in [78], the CONV method in [79], the simple approach and the

`QAKR` method. The finite difference method and least square Monte Carlo are traditional computational methods. The CONV method performs well both in speed and precision, and has been used for comparison in literature. Its complexity is $O(N_s \log(N_s) N_t)$, where N_s is the number of grid points on the price axis and N_t is the number of time steps. The complexity is the lower bound of the methods which solve the PIDE in time steps. So we choose the CONV method to compare the speed in our numerical experiments. Here are the details:

- The finite difference method using the PIDE in [54]. We use the implicit scheme to solve the prices at each time step and the Bermudan approach to deal with the early exercise of American options.

  Let N_s be the number of grid points of $\ln(S)$ and N_t be the number of time steps from 0 to T. We use two versions of the finite difference method. One is called FDfine, with $N_s = 3000$ and $N_t = 250$. The other one is called FDcoarse, with $N_s = 800$ and $N_t = 80$. When the grid is finer, the finite difference method is very accurate and can be treated as a benchmark to be compared with. However, that can be time-consuming, so we want to use FDcoarse to test the performance of the finite difference method when we accelerate it on a coarser grid.

- The FFT-based method CONV in [79]. Let N_s be the number of grid points of $\ln(S)$ and N_t be the number of time steps from 0 to T. We choose $N_s = 2^{12}$ and $N_t = 30$ in order to compare the running time of the CONV method and that of the `QAKR` method when the errors are similar.

- Simulation with least square Monte Carlo in [78], a generic method to deal with the early exercise of American options. The number of time steps is 250 and the number of samples is 1e5.

- The simple approach.

- The `QAKR` method. In the pre-calculation part, the grid points where we calculate the optimal parameters $\lambda(\Theta)$ and $x^\star(K_0, \Theta)$ and then $g(x_i; K_0, \lambda(\Theta), x^\star(K_0, \Theta), \Theta)$, are the points in the

Cartesian product of the following sets:

$$r, q \in \{0.01, 0.04, 0.07, 0.1\},$$

$$T \in \{0.1\} \cup \{0.25k : 1 \leq k \leq 12, k \in \mathbb{Z}\},$$

$$\sigma \in \{0.1, 0.2, 0.3, 0.4\},$$

$$\nu \in \{0.1, 0.26, 0.43, 0.6\},$$

$$\theta \in \{-0.5, -0.3, -0.1\}.$$

There are $n = 9984$ combinations of the parameters. The grids are chosen to be arithmetic sequences except for T. Since the premium is 0 when $T = 0$ and we cannot perform pre-calculation at $T = 0$, we choose $T = 0.1$ instead. We utilize the CONV method which consumes 0.1s for each combination, and the entire pre-calculation time is less than 20 minutes. In the numerical tests, we take $N = 6$.

In the pricing routine, we use kernel regression as the nonparametric method. The explanatory variable is $\Theta = (r, q, T, \sigma, \nu, \theta)$ and the response variable is $(g(x_i; K_0, \lambda(\Theta), x^\star(K_0, \Theta), \Theta))_{i=0}^N$. The dimensions are 6 and $N + 1$ respectively for the explanatory and response variables. As explained in Appendix A.3, we use the Gaussian kernel and choose the parameter of the kernel by defining a loss function. The training ratio is 70% in the loss function. We repeat the process of finding good kernels for 5 times and take the average of the kernels to be the final kernel parameter to get a robust choice of kernel, which is also covered in Appendix A.3.

In the optimization problem of Equation (2.12), given the initial solution $(\lambda_0, x_0^\star)$ from the second step of the pricing routine, we first optimize $\ell(\lambda_0, x^\star; \Theta)$ w.r.t. $x^\star$ to get the optimal $x_{opt}^\star$ and then optimize $\ell(\lambda, x_{opt}^\star; \Theta)$ w.r.t. λ to get the optimal λ_{opt}. This is because $x^\star$ has a larger influence on the loss function (2.12), and it is enough to get the optimal $x^\star$ given a suboptimal λ. In numerical experiments we use the function `fminbnd` in Matlab for the optimization problem.

The results are shown in Table 2.1-2.5. The parameters Θ in the numerical tests are chosen such that they do not overlap the sample grid in the `QAKR` method. By doing this we avoid in-sample prediction. All the methods are programmed in C and tested in Matlab on an Intel i7-6820HQ, 2.70GHz. The computing time is the average time used to price one option. For the `QAKR` method, it refers to the time in the pricing phase. As we see, the `QAKR` method achieves a good balance between small errors and fast speed among the methods. The simple method is the fastest, but the error is usually larger than the bid-ask spread, which shows the premium of the VG model can hardly be approximated by the premium of the BMS model. That is why we proposed the `QAKR` method. Also, when we tune the parameters of the finite difference method and the CONV method such that they generate a similar size of errors compared with the `QAKR` method, the `QAKR` method is faster than the two methods. In addition, as we see from the tables, when the time to maturity T increases, the errors of all the methods also increase, including the `QAKR` method. In fact, when T increases, the options become less liquid and the bid-ask spread also increases. So whatever T, the error of the `QAKR` method stays within the bid-ask spread.

2.5.2 Convergence of errors

Also, the number of the pre-calculated samples does influence the error. To illustrate that, we randomly remove the pre-calculated samples to keep only $1/2$, $1/4$, $1/8$ and $1/16$ of the original size 9984, redo the calculation in Table 2.4 (where $T = 1$) and plot the curves of the root of mean squared errors (RMSE) and the maximum absolute errors (MAE) versus the sample size in Figure 2.2. As the sample size increases, the errors go down. Thus, errors can be reduced with more pre-calculations. It is reasonable since the sample size plays an important role in kernel regression and also affects the accuracy of the `QAKR` method. On the other hand, the mean computing time only increase from 0.0065 with less than 1000 samples to 0.007 with all the 9984 samples. That is because the kernel regression only takes up a small ratio of time in this routine. More pre-calculated samples will not slow down the method too much.

2.5.3 Intermediate errors

In Section 2.4.3, we claimed that the optimal parameters $x^\star(K,\Theta)$ and $\lambda(\Theta)$ make the approximation (2.10) very close to the true premium, and in fact the difference is smaller than a typical bid-ask spread. In the column “optimal" of Table 2.6, we show the error if we replace $x^\star$ and λ with $x^\star(K,\Theta)$ and $\lambda(\Theta)$ in the approximation formula (2.13) when $T = 0.5$. The error using the optimal parameters is about a half of that of the `QAKR` method. Since $x^\star(K,\Theta)$ and $\lambda(\Theta)$ are the target values of $x^\star$ and λ in the `QAKR` method, the error using the optimal parameters can be seen as a lower bound of that of the `QAKR` method. In Table 2.7, we provide typical bid-ask spreads when $S = 2900.45$. By comparing the bid-ask spreads and the errors in Table 2.1-2.6, we show that the errors caused by the optimal parameters and the errors of the `QAKR` method are smaller than typical bid-ask spreads.

In the first step of the price routine, we use kernel regression to estimate $\hat{g}_i(K,\Theta)$ from

$$\frac{K}{K_0} g(x_i; K_0, \lambda(\Theta_j), x^\star(K_0,\Theta_j), \Theta_j), 1 \le j \le n.$$

Based on the $n = 9984$ samples, when $K = S_0 = 2900$, the leave-one-out cross-validation error for $0 \le i \le N = 6$ are

$$(25.0617, 7.0644, 2.0056, 1.0627, 0.7073, 0.5180, 0.3935)$$

respectively. The details of the leave-one-out cross-validation are included in Appendix A.3. The leave-one-out cross-validation error is a good estimate of the prediction error of $\hat{g}_i(K,\Theta)$. For example, the prediction error of $\hat{g}_0(K,\Theta)$ is around 25.0617 on average and the one of $\hat{g}_1(K,\Theta)$ is about 7.0644 on average.

In the second step of the price routine, we give an initial solution $(\lambda_0, x_0^\star)$. If we use it directly in the approximation formula (2.13), the error is shown in the column “initial” of Table 2.6. While it has some predictability, the error is several times of that of the `QAKR` method. The approximated price is not guaranteed to be always within the bid-ask spread.

In the third step of the price routine, The residual of the loss function $\ell(\lambda, x^{\star}; \Theta)$ after optimization is shown in Table 2.6 in the column “residual”. The residual varies depending on the parameter Θ.

2.6 Conclusion

In this chapter we proposed the `QAKR` method, a fast and practical method for pricing American options under the VG model. There is a twofold aspect to this method. On one hand, it solves an approximated equation with a correction term estimated from the pre-calculated data. On the other hand, the optimization routine provides a mapping from the surface of the premium to the vector of the correction terms, which lies in Euclidean space and is easy to estimate. The mapping converts a pricing problem to an easy machine learning problem.

A high-precision closed-form approximation solution of the American price under the VG model is always attractive, just like the approximation formula in [6] for the BMS model and the one in [72] for the double exponential jump diffusion model. Option prices in many financial models involving diffusion and jumps can be described with a PDE or PIDE. The `QAKR` method can be applied to processes such as NIG, CGMY, and VGSSD.

r	q	K	FDfine	FDcoarse	CONV	QAKR	simulation	simple
0.09	0.02	2600	107.341	107.111	107.146	107.175	105.912	102.134
0.09	0.02	2800	161.662	161.461	161.269	161.679	159.078	155.660
0.09	0.02	3000	237.181	237.047	236.468	237.216	233.853	231.450
0.09	0.02	3200	340.841	340.323	339.597	339.682	338.332	336.219
0.05	0.05	2600	118.457	118.369	118.500	118.698	117.814	117.936
0.05	0.05	2800	177.555	177.604	177.503	177.813	175.413	177.047
0.05	0.05	3000	258.846	259.110	258.617	259.052	259.711	258.642
0.05	0.05	3200	368.363	369.063	367.811	368.315	368.309	368.882
0.02	0.09	2600	138.838	138.740	138.948	138.958	138.937	138.958
0.02	0.09	2800	207.984	208.039	208.012	208.027	207.380	208.027
0.02	0.09	3000	302.990	303.302	302.853	302.874	304.001	302.874
0.02	0.09	3200	430.768	431.470	430.320	430.351	428.909	430.351
		RMSE		0.360	0.487	0.381	1.749	3.141
		MAE		0.703	1.244	1.159	3.328	6.002
		CPU(s)	5.109	0.148	0.015	0.006	7.172	0.002

Table 2.1: American put values under VG. $S_0 = 2900$, $T = 0.5$, $\sigma = 0.15$, $\nu = 0.5$ and $\theta = -0.4$. RMSE is the root of mean squared errors. MAE is the maximum absolute error. CPU is the mean computing time.

σ	ν	θ	K	FDfine	FDcoarse	CONV	QAKR	simulation	simple
0.15	0.20	-0.40	2800	36.370	36.442	36.374	36.427	36.530	36.108
0.35	0.50	-0.40	2800	67.555	67.641	67.522	67.539	68.317	67.901
0.15	0.50	-0.20	2800	26.290	26.356	26.267	26.364	26.092	26.186
0.35	0.20	-0.20	2800	58.394	58.578	58.367	58.364	57.783	58.515
0.15	0.20	-0.40	2900	60.268	60.595	60.242	60.312	59.972	59.918
0.35	0.50	-0.40	2900	88.417	88.350	88.170	88.255	89.483	89.094
0.15	0.50	-0.20	2900	42.637	41.251	42.629	42.792	42.751	42.823
0.35	0.20	-0.20	2900	89.160	89.110	88.995	89.036	89.156	89.365
0.15	0.20	-0.40	3000	105.993	107.356	105.760	105.836	105.702	105.494
0.35	0.50	-0.40	3000	127.373	129.232	126.902	127.042	128.542	128.706
0.15	0.50	-0.20	3000	100.146	100.899	100.427	100.000	100.003	100.466
0.35	0.20	-0.20	3000	147.122	147.551	147.054	147.077	146.534	147.728
			RMSE		0.824	0.194	0.140	0.583	0.530
			MAE		1.859	0.471	0.331	1.169	1.333
			CPU(s)	5.015	0.144	0.014	0.006	6.471	0.002

Table 2.2: American put values under VG. $S_0 = 2900$, $T = 1/12$, $r = 0.05$ and $q = 0.02$. RMSE is the root of mean squared errors. MAE is the maximum absolute error. CPU is the mean computing time.

σ	ν	θ	K	FDfine	FDcoarse	CONV	QAKR	simulation	simple
0.15	0.20	-0.40	2800	80.706	80.736	80.672	80.631	80.546	79.350
0.35	0.50	-0.40	2800	155.690	155.807	155.605	155.828	155.527	156.676
0.15	0.50	-0.20	2800	62.843	62.850	62.802	63.042	62.579	62.030
0.35	0.20	-0.20	2800	133.723	134.050	133.612	133.617	134.090	133.941
0.15	0.20	-0.40	2900	114.750	114.858	114.602	114.615	114.017	112.891
0.35	0.50	-0.40	2900	188.874	189.040	188.713	189.002	187.326	190.665
0.15	0.50	-0.20	2900	90.300	90.561	90.155	90.467	89.757	89.412
0.35	0.20	-0.20	2900	175.617	176.058	175.413	175.435	175.294	175.998
0.15	0.20	-0.40	3000	160.627	160.840	160.398	160.476	159.526	158.207
0.35	0.50	-0.40	3000	229.003	229.410	228.786	229.143	228.484	231.970
0.15	0.50	-0.20	3000	131.089	130.579	130.883	131.046	130.742	130.212
0.35	0.20	-0.20	3000	227.527	228.034	227.305	227.348	227.514	228.255
			RMSE		0.310	0.164	0.144	0.658	1.502
			MAE		0.510	0.229	0.199	1.548	2.966
			CPU(s)	5.068	0.146	0.014	0.006	6.862	0.002

Table 2.3: American put values under VG. $S_0 = 2900$, $T = 1/4$, $r = 0.05$ and $q = 0.02$. RMSE is the root of mean squared errors. MAE is the maximum absolute error. CPU is the mean computing time.

σ	ν	θ	K	FDfine	FDcoarse	CONV	QAKR	simulation	simple
0.15	0.20	-0.40	2700	146.062	145.936	145.844	146.240	144.882	141.430
0.35	0.50	-0.40	2700	319.514	319.713	319.032	320.257	319.010	320.347
0.15	0.50	-0.20	2700	119.519	119.456	119.284	119.095	118.278	115.149
0.35	0.20	-0.20	2700	260.494	261.036	260.070	261.169	259.944	259.863
0.15	0.20	-0.40	2900	224.964	225.106	224.357	224.597	225.570	218.156
0.35	0.50	-0.40	2900	407.472	407.891	406.724	408.271	407.174	410.969
0.15	0.50	-0.20	2900	189.891	189.980	189.345	189.407	188.785	184.015
0.35	0.20	-0.20	2900	354.279	354.997	353.642	354.704	354.371	353.591
0.15	0.20	-0.40	3100	328.756	329.227	327.700	327.834	328.758	319.607
0.35	0.50	-0.40	3100	509.145	509.829	508.110	510.031	506.804	516.749
0.15	0.50	-0.20	3100	288.352	288.672	287.413	288.050	285.557	281.134
0.35	0.20	-0.20	3100	464.743	465.603	463.924	464.859	463.334	464.204
			RMSE		0.466	0.700	0.588	1.306	5.237
			MAE		0.859	1.056	0.922	2.795	9.149
			CPU(s)	5.391	0.155	0.017	0.007	7.979	0.002

Table 2.4: American put values under VG. $S_0 = 2900$, $T = 1$, $r = 0.05$ and $q = 0.02$. RMSE is the root of mean squared errors. MAE is the maximum absolute error. CPU is the mean computing time.

σ	ν	θ	K	FDfine	FDcoarse	CONV	QAKR	simulation	simple
0.15	0.20	-0.40	2600	222.399	221.974	221.481	222.530	220.058	213.470
0.35	0.50	-0.40	2600	512.292	512.638	510.357	513.291	507.555	510.935
0.15	0.50	-0.20	2600	183.357	183.091	182.414	180.836	183.049	173.279
0.35	0.20	-0.20	2600	411.828	412.611	410.387	412.821	412.499	408.966
0.15	0.20	-0.40	2900	347.846	347.951	345.900	346.370	345.961	334.771
0.35	0.50	-0.40	2900	663.230	663.967	660.534	663.767	662.017	668.475
0.15	0.50	-0.20	2900	298.977	299.013	297.141	295.478	300.197	285.549
0.35	0.20	-0.20	2900	561.008	562.074	558.991	560.706	559.766	557.287
0.15	0.20	-0.40	3200	508.103	508.742	504.963	505.204	505.352	491.167
0.35	0.50	-0.40	3200	831.944	833.066	828.439	832.119	827.809	847.781
0.15	0.50	-0.20	3200	454.021	454.400	451.055	451.121	452.341	438.839
0.35	0.20	-0.20	3200	731.759	733.039	729.156	729.910	734.797	727.520
			RMSE		0.714	2.305	1.902	2.470	10.685
			MAE		1.280	3.505	3.499	4.738	16.936
			CPU(s)	5.114	0.149	0.015	0.007	7.280	0.002

Table 2.5: American put values under VG. $S_0 = 2900$, $T = 3$, $r = 0.05$ and $q = 0.02$. RMSE is the root of mean squared errors. MAE is the maximum absolute error. CPU is the mean computing time.

σ	ν	θ	K	FDfine	optimal	QAKR	initial	residual
0.15	0.20	-0.40	2800	124.450	124.476	124.158	124.467	12.302
0.35	0.50	-0.40	2800	243.824	243.807	244.071	244.815	4.527
0.15	0.50	-0.20	2800	100.636	100.804	100.839	102.147	16.722
0.35	0.20	-0.20	2800	207.638	207.466	207.492	207.405	3.513
0.15	0.20	-0.40	2900	163.345	163.268	162.903	163.385	13.196
0.35	0.50	-0.40	2900	283.868	283.824	284.118	285.056	4.857
0.15	0.50	-0.20	2900	134.252	134.417	134.462	136.538	17.938
0.35	0.20	-0.20	2900	253.732	253.462	253.481	253.395	3.768
0.15	0.20	-0.40	3000	210.834	210.715	210.313	211.046	14.122
0.35	0.50	-0.40	3000	328.804	328.755	329.079	330.250	5.197
0.15	0.50	-0.20	3000	177.480	177.601	177.658	180.866	19.196
0.35	0.20	-0.20	3000	306.079	305.759	305.764	305.686	4.033
			RMSE		0.158	0.296	1.409	
			MAE		0.320	0.521	3.386	

Table 2.6: American put values under VG. $S_0 = 2900$, $T = 0.5$, $r = 0.05$ and $q = 0.02$. RMSE is the root of mean squared errors. MAE is the maximum absolute error.

expiration	strike	2800	2850	2900	2950	3000
20190517	bid	10.8	18.1	32.3	58.4	99.2
(one month)	ask	11.1	18.5	32.8	60.7	102.5
	spread	0.3	0.4	0.5	2.3	3.3
20190719	bid	38.6	50.0	65.4	86.7	115.2
(three months)	ask	39.2	50.6	66.1	87.5	117.3
	spread	0.6	0.6	0.7	0.8	2.1
20190920	bid	63.6	76.5	92.3	112.0	136.0
(five months)	ask	64.1	77.1	93.0	112.7	138.2
	spread	0.5	0.6	0.7	0.7	2.2
20200320	bid	115.2	129.5	145.5	163.8	184.7
(11 months)	ask	116.4	130.7	146.9	165.2	186.2
	spread	1.2	1.2	1.4	1.4	1.5
20211217	bid	240.8	258.3	276.8	296.3	318.8
(20 months)	ask	249.4	267.4	286.4	306.6	327.9
	spread	8.6	9.1	9.6	10.3	9.1

Table 2.7: Typical bid-ask spreads of S&P 500 index options on April 17th, 2019. The close of S&P 500 was 2900.45 on April 17th, 2019. Data source: WRDS - OptionMetrics.

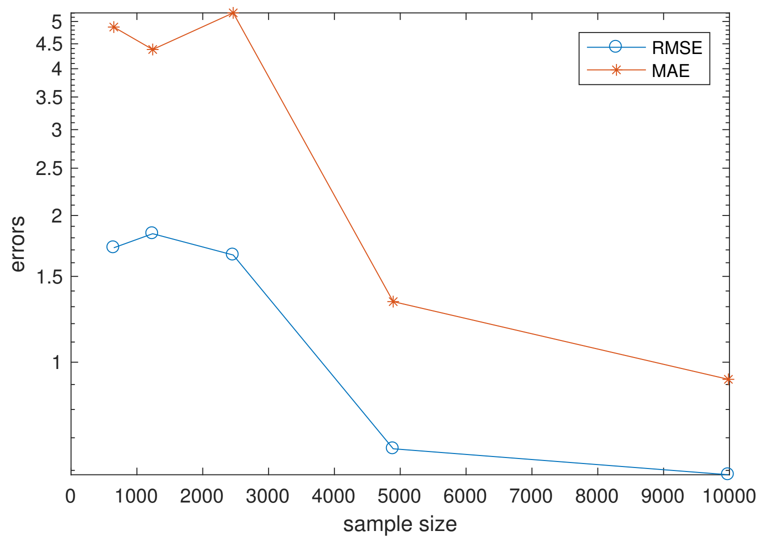


Figure 2.2: Error curves of the root of mean squared error (RMSE) and the maximum absolute error (MAE) of the `QAKR` method versus the sample size when the parameters are the same as in Table 2.4.

Chapter 3: Pricing options under Lévy processes with unsupervised deep learning

3.1 Introduction

In this chapter, we extend the unsupervised deep learning approach to the partial integro-differential equation (PIDE), and propose a pricing method of European/American calls/puts in the models based on Lévy processes by solving the PIDE with a neural network. The neural network is used as the approximated price surface. It only needs to be trained once and is then able to generate prices fast, which is the same as supervised deep learning. The difference from supervised approaches is that this approach is self-contained, which does not need pre-calculated labels of option prices.

The network structure in our method is slightly different from the structures already used in the literature. In the unsupervised deep learning methods to solve equations, they always use a smooth neural network to fit the solutions since the solution is smooth almost everywhere. However, in option pricing, the final payoff (the initial condition) is not a smooth function, which contradicts the smooth neural network. Thus we add additional features called singular terms in the neural network that are smooth almost everywhere but also satisfy the non-smooth property of the initial conditions. The singular terms facilitate fitting short maturity options and make it possible to solve equations with non-smooth initial conditions.

Also, the solution given by the neural network yields not only the option price surface but also the Greeks without any extra effort. In comparison, additional labels of Greeks are needed to fit the Greek surface in the supervised approaches. This is easy to understand, since in the supervised approaches, the neural networks are not required to be smooth and there are no constraints of the Greeks during training. While in the unsupervised approaches, the neural networks are required to be smooth for derivative calculation and the derivatives (Greeks) are involved in the PIDE during training.

The chapter is organized as follows. In Section 3.2, we introduce option pricing under Lévy processes, the PIDE for option pricing and five examples of models under Lévy processes. In Section 3.3, we introduce the structure of multilayer perceptrons used to solve the PIDE, as well as the additional singular terms built to better fit the option price surface and the initial condition. In Section 3.4, we give the boundary conditions and loss functions used for training in all cases of European and American options and summarize the algorithm. In Section 3.5, we explain how to calculate the derivatives and the integral in the PIDE. Some further details are provided in Appendix B.1. In Section 3.6, we test the unsupervised deep learning approach on the five models under Lévy processes and show the results of option prices and Greeks given by the neural network. The code used for the numerical experiments is available at `https://github.com/weilong-columbia/pide`. Section 3.7 summarizes the chapter.

3.2 Option pricing under Lévy processes

3.2.1 Lévy process

A Lévy process $\{X_t\}_{t\geq 0}$ is a stochastically continuous process with stationary independent increments [96]. According to the Lévy-Khinchine theorem, the process $\{X_t\}_{t\geq 0}$ is completely specified by its characteristic component

$$\psi(u) = -\frac{s^2}{2}u^2 + i\gamma u + \int_{\mathbb{R}} \left(e^{iuy} - 1 - iuy1_{|y|\leq 1}\right) m(dy)$$

which satisfies $\mathbb{E}(e^{iuX_t}) = e^{t\psi(u)}$. Here $s \geq 0$ and γ are real constants. The Lévy measure m is a positive measure on $\mathbb{R}$ that satisfies $\int_{\mathbb{R}} \min(1, y^2)m(dy) < \infty$.

If m satisfies $\int_{|y|>1} |y|m(dy) < \infty$, the characteristic component can be written as

$$\psi(u) = -\frac{s^2}{2}u^2 + i\tilde{\gamma}u + \int_{\mathbb{R}} \left(e^{iuy} - 1 - iuy\right) m(dy)$$

where $\tilde{\gamma} = \gamma + \int_{|y|>1} y\, m(dy)$.

3.2.2 Option pricing

We further suppose $\int_{|y|>1} e^y m(dy) < \infty$ and define the risk neutral stock price process for option pricing

$$S_t = S_0 \exp((r-q)t + X_t + \omega t), \tag{3.1}$$

where r is the risk-free interest rate, q is the dividend rate of the stock, and ω is a constant such that $\mathbb{E}(S_t) = S_0 e^{(r-q)t}$, i.e., the discounted price process $e^{-(r-q)t} S_t$ is a martingale. The martingale property of the discounted price is equivalent with the no-arbitrage condition. We get $\omega = -\psi(-i)$ from the definition of ω.

Suppose $\{S_t\}_{t\geq 0}$ is the stock price process, K is the strike price, t is the current time and T is the maturity (expiration) time, the European/American call/put are defined in Table 3.1, where $\bar{t}$ is an arbitrary stopping time between t and T.

Option	Definition
European call	$c(S,t) = e^{-r(T-t)}\mathbb{E}\left((S_T - K)^+ \vert S_t = S\right)$
European put	$p(S,t) = e^{-r(T-t)}\mathbb{E}\left((K - S_T)^+ \vert S_t = S\right)$
American call	$C(S,t) = \sup_{t\leq\bar{t}\leq T} \mathbb{E}\left(e^{-r(\bar{t}-t)}(S_{\bar{t}} - K)^+ \vert S_t = S\right)$
American put	$P(S,t) = \sup_{t\leq\bar{t}\leq T} \mathbb{E}\left(e^{-r(\bar{t}-t)}(K - S_{\bar{t}})^+ \vert S_t = S\right)$

Table 3.1: Definitions of European/American call/put.

3.2.3 PIDE for option pricing

Let $x = \ln(S)$ be the log-price. One can derive the PIDE using the martingale approach [25]:

$$\begin{aligned}\Bigg(\int_{-\infty}^{\infty} \left(\tilde{w}(x+y,t) - \tilde{w}(x,t) - \frac{\partial \tilde{w}}{\partial x}(x,t)(e^y - 1)\right) m(dy) \\ + \frac{\partial \tilde{w}}{\partial t}(x,t) + \frac{s^2}{2}\frac{\partial^2 \tilde{w}}{\partial x^2}(x,t) + \left(r - q - \frac{s^2}{2}\right)\frac{\partial \tilde{w}}{\partial x}(x,t) - r\tilde{w}(x,t)\Bigg) &= 0.\end{aligned}$$

The European options $c(e^x,t)$ and $p(e^x,t)$ satisfy the PIDE, and the American options $C(e^x,t)$ and $P(e^x,t)$ also satisfy the PIDE before the early exercise.

Since the Lévy process is stationary, we look at the time to maturity $\tau = T - t$ instead of the calendar time t. Let $w(x,\tau) = \tilde{w}(x,t)$, and we get the PIDE

$$\begin{aligned} H(w,x,\tau) = \Bigg(\int_{-\infty}^{\infty} \left(w(x+y,\tau) - w(x,\tau) - \frac{\partial w}{\partial x}(x,\tau)(e^y - 1)\right) m(dy) \\ -\frac{\partial w}{\partial \tau}(x,\tau) + \frac{s^2}{2}\frac{\partial^2 w}{\partial x^2}(x,\tau) + \left(r - q - \frac{s^2}{2}\right)\frac{\partial w}{\partial x}(x,\tau) - rw(x,\tau)\Bigg) &= 0. \end{aligned} \tag{3.2}$$

The European/American call/put can be solved by the equations in Table 3.2. The solution should also satisfy $w(x,\tau) \in C^1(\mathbb{R}\times\mathbb{R}_{++})$, where $\mathbb{R}_{++} = \{\tau | \tau > 0\}$ and $w(x,\tau) \in C(\mathbb{R}\times\mathbb{R}_{+})$, where $\mathbb{R}_{+} = \{\tau | \tau \geq 0\}$.

Option	Equations
European call	$\begin{cases} H(w,x,\tau) = 0, & \forall x \in \mathbb{R}, \tau > 0 \\ w(x,0) = (e^x - K)^+, & \forall x \in \mathbb{R} \end{cases}$
European put	$\begin{cases} H(w,x,\tau) = 0, & \forall x \in \mathbb{R}, \tau > 0 \\ w(x,0) = (K - e^x)^+, & \forall x \in \mathbb{R} \end{cases}$
American call	$\begin{cases} \max\left(H(w,x,\tau), (e^x - K)^+ - w(x,\tau)\right) = 0, & \forall x \in \mathbb{R}, \tau > 0 \\ w(x,0) = (e^x - K)^+, & \forall x \in \mathbb{R} \end{cases}$
American put	$\begin{cases} \max\left(H(w,x,\tau), (K - e^x)^+ - w(x,\tau)\right) = 0, & \forall x \in \mathbb{R}, \tau > 0 \\ w(x,0) = (K - e^x)^+, & \forall x \in \mathbb{R} \end{cases}$

Table 3.2: Equations of European/American call/put.

3.2.4 Examples of Lévy processes

In the following examples, the Lévy measure m has a density, i.e., $m(dy) = k(y)dy$, where $k(y)$ is called the Lévy density.

Also, in the risk neutral stock price process (3.1), $X_t + \omega t$ is also a Lévy process. Its characteristic function is

$$\mathbb{E}(e^{iu(X_t+\omega t)}) = e^{t\psi(u)-\psi(-i)uit}$$

and its characteristic component is

$$\begin{aligned}\tilde{\psi}(u) &= \psi(u) - \psi(-i)ui \\ &= -\frac{s^2}{2}u^2 + \int_{\mathbb{R}} \left(e^{iuy} - 1 - iuy\right) m(dy) - \left(\frac{s^2}{2} + \int_{\mathbb{R}} (e^y - 1 - y)\, m(dy)\right) ui\end{aligned}$$

We can see that $\tilde{\gamma}$ in $\psi(u)$ is cancelled in $\tilde{\psi}(u)$ and is not effective in the risk neutral stock price process S_t. So $\tilde{\gamma}$ will be omitted in the examples.

The variance gamma (VG) process

In the VG process [82], there are three model parameters: $\sigma > 0$, $\theta \in \mathbb{R}$, and $\nu > 0$. The Lévy density is given by

$$k(y) = \frac{e^{-My}}{\nu y} 1_{y>0} + \frac{e^{-G|y|}}{\nu |y|} 1_{y<0}$$

where

$$M = \sqrt{\theta^2/\sigma^4 + 2/(\sigma^2\nu)} - \theta/\sigma^2 \tag{3.3}$$

and

$$G = \sqrt{\theta^2/\sigma^4 + 2/(\sigma^2\nu)} + \theta/\sigma^2. \tag{3.4}$$

The characteristic exponent is given by

$$\psi(u) = -\ln\left(1 + \sigma^2 \nu u^2/2 - \mathrm{i}\theta\nu u\right)/\nu$$

where $s = 0$ in the characteristic exponent.

The CGMY process

The CGMY process [15] is a generalization of the VG process. There are four model parameters: $\sigma > 0$, $\theta \in \mathbb{R}$, $\nu > 0$ and $0 \leq Y < 2$. The Lévy density is given by

$$k(y) = \frac{C\,e^{-My}}{y^{1+Y}} 1_{y>0} + \frac{C\,e^{-G|y|}}{|y|^{1+Y}} 1_{y<0}$$

where M and G follow the same definitions in Equation (3.3) and (3.4) and $C = 1/\nu$.

When $Y = 0$, the characteristic exponent is given in the VG model. Otherwise, it is given by

$$\psi(u) = \begin{cases} C\,\Gamma(-Y)\left((G+iu)^Y - G^Y + (M-iu)^Y - M^Y\right), & Y \neq 0, 1 \\ C\left((G+iu)\ln(G+iu) - G\ln G + (M-iu)\ln(M-iu) - M\ln M\right), & Y = 1 \end{cases}$$

where $\Gamma(\cdot)$ is the gamma function and $s = 0$ in the characteristic exponent.

The normal inverse Gaussian (NIG) process

In the NIG process [5], there are three model parameters: $\alpha > 1$, $-\alpha < \beta < \alpha - 1$ and $\delta > 0$.The Lévy density is given by

$$k(y) = \frac{\delta\alpha}{\pi}\frac{e^{\beta x} K_1(\alpha|x|)}{|x|}$$

where $K_1(\cdot)$ is the modified Bessel function of the second kind. The characteristic exponent is given by

$$\psi(u) = -\delta \left(\sqrt{\alpha^2 - (\beta + ui)^2)} - \sqrt{\alpha^2 - \beta^2} \right)$$

Also, $s = 0$ in the characteristic exponent.

The Merton's jump diffusion model

In the Merton's model [85], there are four model parameters: $\sigma > 0$, $\lambda \geq 0$, $\alpha \in \mathbb{R}$, and $\delta > 0$. The Lévy density is given by

$$k(y) = \frac{\lambda}{\sqrt{2\pi}\delta} e^{-(x-\alpha)^2/(2\delta^2)}$$

The characteristic exponent is given by

$$\psi(u) = -\sigma^2 u^2/2 + \lambda \left(\exp(\alpha i u - \delta^2 u^2/2) - 1 \right)$$

and $s = \sigma$ in the characteristic exponent.

The Kou's double exponential jump diffusion model

In the Kou's model [72], there are five model parameters: $\sigma > 0$, $\lambda \geq 0$, $0 \leq p \leq 1$, $\eta_1 > 0$ and $\eta_2 > 0$. The Lévy density is given by

$$k(y) = \lambda p \eta_1 e^{-\eta_1 y} 1_{y>0} + \lambda(1-p)\eta_2 e^{\eta_2 y} 1_{y<0}$$

The characteristic exponent is given by

$$\psi(u) = -\frac{\sigma^2 u^2}{2} + \lambda \left(\frac{p\eta_1}{\eta_1 - iu} + \frac{(1-p)\eta_2}{\eta_2 + iu} - 1 \right)$$

and $s = \sigma$ in the characteristic exponent.

3.2.5 Goal of the chapter

Our goal is to solve the PIDE (3.2) as well as the equations in Table 3.2 utilizing neural networks directly. We treat the solution $w(x, \tau)$ as a function of x and τ as well as other parameters, approximate $w(x, \tau)$ with a multi-layer perceptron with additional features and train the network to satisfy the PIDE. This is an unsupervised method which means there is no need for labels of option prices calculated by other methods. The method will be tested on the five models introduced in Section 3.2.4.

3.3 Neural network as the solution to the PIDE

3.3.1 Traditional multi-layer perceptron

Depending on the task and the goal, there are different neural networks that can be utilized. For example, convolutional neural networks (CNNs, [75]) are suitable for image recognition, while recurrent neural networks (RNNs, [55]) are good at modeling sequential data. For our task, we employ a multi-layer perceptron (MLP).

Here we give a simple description of the MLP to keep the thesis self-contained. The MLP serves as a multi-dimensional function with an input $\boldsymbol{x} \in \mathbb{R}^{n_0}$ and an output $w \in \mathbb{R}$, where n_0 is the length of the input. Suppose the network consists of L hidden layers. Then the MLP can be explained with the equations

$$\begin{aligned} \boldsymbol{x}^{(0)} &= \boldsymbol{x}, \\ \boldsymbol{x}^{(i)} &= g(\boldsymbol{W}^{(i-1)}\boldsymbol{x}^{(i-1)} + \boldsymbol{b}^{(i-1)}), \ \forall 1 \le i \le L, \\ w &= \boldsymbol{W}^{(L)}\boldsymbol{x}^{(L)} + b^{(L)}, \end{aligned}$$

where the ith hidden layer $\boldsymbol{x}^{(i)}$ is a vector of length n_i for $1 \le i \le L$, and n_i is the size of the ith hidden layer. Though it is possible to let the sizes of the layers be different, we let the sizes be the same for simplicity, i.e., $n_i = n, \forall 1 \leq i \leq L$. Thus the dimensions of the parameters are $\boldsymbol{W}^{(0)} \in \mathbb{R}^{n \times n_0}$, $\boldsymbol{W}^{(i)} \in \mathbb{R}^{n \times n}$ for $1 \leq i \leq L-1$, $\boldsymbol{b}^{(i)} \in \mathbb{R}^n$ for $0 \leq i \leq L-1$, $\boldsymbol{W}^{(L)} \in \mathbb{R}^{1 \times n}$ and $b^{(L)} \in \mathbb{R}$. A graph of a traditional MLP with two hidden layers is illustrated in Figure 3.1.

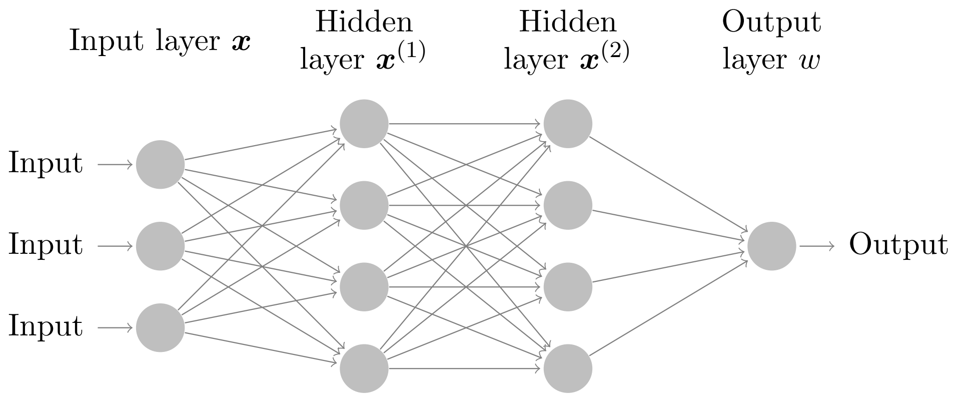

Input layer $\boldsymbol{x}$
Hidden layer $\boldsymbol{x}^{(1)}$
Hidden layer $\boldsymbol{x}^{(2)}$
Output layer w
Input
Input
Input
Output

Figure 3.1: Illustration of the MLP structure.

g is the non-linear activation function which is applied to each coordinate of its input, i.e.,

$$g(\boldsymbol{z}) = (g(z_1), g(z_2), \ldots, g(z_n)),$$

where $\boldsymbol{z} \in \mathbb{R}^n$ and $\{z_i, 1 \leq i \leq n\}$ are the coordinates of $\boldsymbol{z}$. There are some examples of activation functions in Table 3.3. The most commonly used ones are sigmoid, tanh, and ReLU [88]. sigmoid and tanh are smooth functions. However, their derivatives diminish when the input z tends to infinity. In first order optimization algorithms, diminishing derivatives lead to slow convergence. ReLU does not have a vanishing derivative at infinity, but its derivative is not continuous at 0. Also, an MLP composed of ReLU is always locally linear, which contradicts the property of the solution $w(x, \tau)$. So we should use smooth activation functions which would not have the problem of vanishing derivatives. The two activation functions that meet this condition are SiLU (also called swish, [31, 94]) and softplus [30]. In Figure 3.2, we show that SiLU and softplus are two smoothed versions of ReLU. They are different only near $z = 0$.

Function	Definition
sigmoid	$1/(1+e^{-z})$
tanh	$(e^z - e^{-z})/(e^z + e^{-z})$
ReLU	$\max(0, z)$
SiLU	$z/(1+e^{-z})$
softplus	$\ln(1+e^z)$

Table 3.3: Examples of activation functions

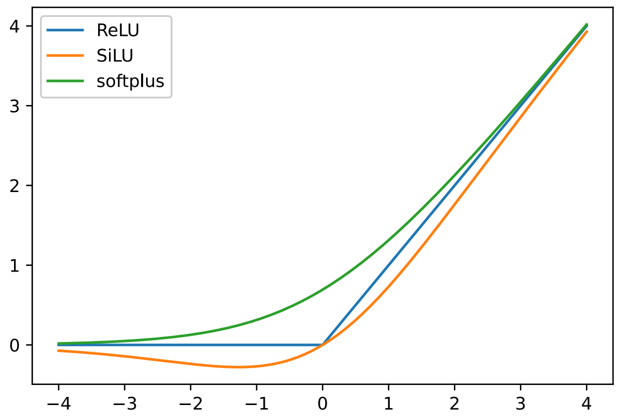


Figure 3.2: Graph of SiLU and softplus compared with ReLU.

3.3.2 Neural network solution

In the PIDE (3.2), the solution $w(x,\tau)$ is not only a function of x and τ, but also dependent on the parameters including r, q and the model parameters of the Lévy process. The neural network need to model the dependence of the solution $w(x,\tau)$ on all the variables and parameters. In the existent papers including [76, 73, 74, 93, 97], the neural network is only a function of the space and time variables. If we need to consider the dependence of the solution on the parameters, the network needs to be either wider or deeper. The input of the neural network is $\boldsymbol{x} = (x, \tau, r, q, \theta_1, \theta_2, \ldots, \theta_d)$ if there are d model parameters. The strike K is kept as a constant throughout the chapter. The output of the neural network is used as the approximation of $w(x,\tau)$. The neural network will be trained to satisfy the equations in Table 3.2. We need to keep in mind that both $w(x,\tau)$ and $k(y)$ depends on r, q and the model parameters $\theta_1, \theta_2, \ldots, \theta_d$. We omit them in the notations for simplicity.

3.3.3 Singular terms in the network

Although the solution $w(x, \tau)$ is a member of $C^1(\mathbb{R}\times\mathbb{R}_{++})$, it is not smooth at $(x, \tau) = (\ln(K), 0)$. In Figure 3.3, we show that the price of European call in the Merton's model is smooth when $\tau \neq 0$, but becomes more like a hockey stick at $S = e^x = K$ when τ converges to 0. Finally, $S = K$ becomes a singular point in the solution at $\tau = 0$. The singular point in the option price surface is universal in all models. However, it causes a problem when we use the smooth neural network as the candidate solution, since the neural network is only able to fit a solution that is also smooth at $(x, \tau) = (\ln(K), 0)$.

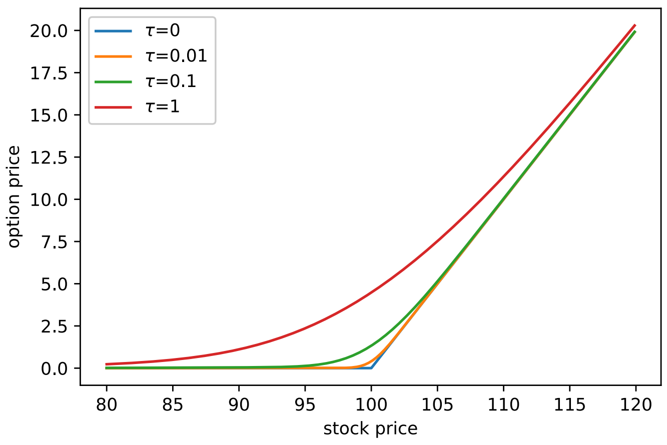


Figure 3.3: European call prices of the Merton's model when $K = 100$, $r = 0$, $q = 0$, $\sigma = 0.1$, $\lambda = 0.1$, $\alpha = 0$ and $\delta = 0.2$.

In order to circumvent this issue, we add a singular term in the neural network to take the singular point into consideration. The singular term is defined as

$$\text{singular}(\boldsymbol{x}) = \text{softplus}\left(\frac{\text{main}(x, \tau, r, q) + \text{bias}(\boldsymbol{x})\tau}{\text{scale}(\boldsymbol{x})\sqrt{\tau}}\right)\text{scale}(\boldsymbol{x})\sqrt{\tau} \tag{3.5}$$

where

$$\text{main}(x,\tau,r,q) = \begin{cases} e^{x-q\tau} - Ke^{-r\tau}, & \text{for European calls} \\ Ke^{-r\tau} - e^{x-q\tau}, & \text{for European puts} \\ e^{x} - K, & \text{for American calls} \\ K - e^{x}, & \text{for American puts} \end{cases} \tag{3.6}$$

and softplus is the activation function in Table 3.3. bias($\boldsymbol{x}$) and scale($\boldsymbol{x}$) are both neural networks with an input of $\boldsymbol{x}$. The singular term is inspired by the Black-Scholes formula:

$$c = N(d_1)Se^{-q\tau} - N(d_2)Ke^{-r\tau}$$

where

$$d_1 = \frac{\ln(S/K) + (r - q + \sigma^2/2)\tau}{\sigma\sqrt{\tau}}$$

and

$$d_2 = \frac{\ln(S/K) + (r - q - \sigma^2/2)\tau}{\sigma\sqrt{\tau}}$$

The part in the softplus function in the singular term is similar to d_1 and d_2 in the Black-Scholes formula. scale($\boldsymbol{x}$) plays the role of σ in the Black-Scholes formula and bias($\boldsymbol{x}$) plays the role of $(r - q \pm \sigma^2/2)$. The singular term satisfies

$$\lim_{\tau\to 0^+} \text{singular}(\boldsymbol{x}) = \begin{cases} (e^x - K)^+, & \text{for calls} \\ (K - e^x)^+, & \text{for puts} \end{cases}$$

If bias($\boldsymbol{x}$) ≈ 0, then

$$\text{singular}(\boldsymbol{x}) \sim \begin{cases} e^{x-q\tau} - Ke^{-r\tau}, & \text{for European calls} \\ e^{x} - K, & \text{for American calls} \\ 0, & \text{for puts} \end{cases}, \text{ when } x \longrightarrow +\infty$$

The singular term replicates the asymptotic behaviors of option price surfaces when $\tau \longrightarrow 0^+$ and $x \longrightarrow +\infty$. If we add the singular term as an additional feature in the neural network, the fitted solution will be able to replicate the singular property around $(x, \tau) = (\ln(K), 0)$. The final neural network will be a function of $\boldsymbol{x}$ as well as singular($\boldsymbol{x}$).

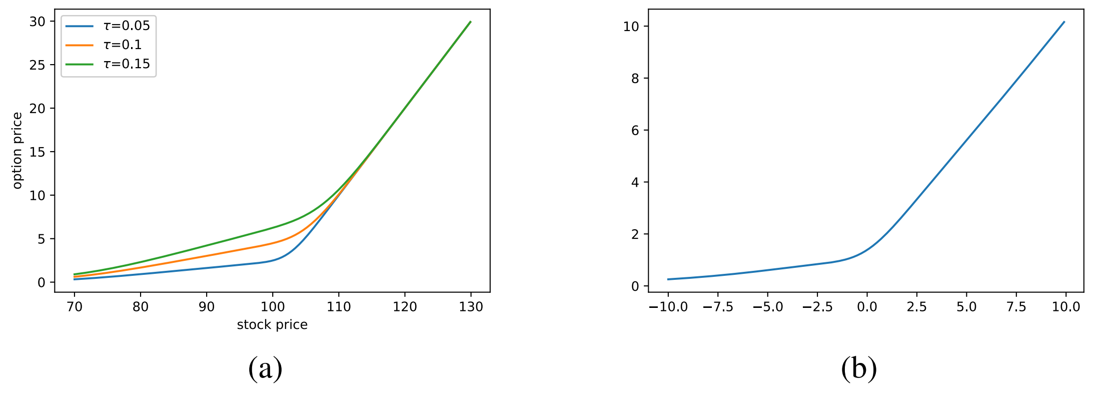


Figure 3.4: (a): European call prices of the Merton's model when $K = 100$, $r = 0$, $q = 0$, $\sigma = 0.1$, $\lambda = 1$, $\alpha = 0.4$ and $\delta = 0.1$. (b): Curve of $0.6\,\text{SiLU}(x) + 2\,\text{softplus}(x/5)$.

The singular term (3.5) is able to replicate the singular property around $(x, \tau) = (\ln(K), 0)$ caused by diffusion and symmetric jumps. However, asymmetric jumps in the Lévy process can generate option price surfaces much different from the shape of the singular term (3.5). For example, if the stock price is driven by large jumps, then the price curves are illustrated in Figure 3.4 (a). In Figure 3.4 (a), the price curves are close to piecewise linear functions, and the linear parts are joined smoothly. For that reason, we add a second singular term in the neural network to better model the curvature in the option surfaces under the models based on Lévy processes. The second singular term is defined similarly to the first one as

$$\text{singular}_2(\boldsymbol{x}) = \text{SiLU}\left(\frac{\text{main}(x, \tau, r, q) + \text{bias}_2(\boldsymbol{x})\tau}{\text{scale}_2(\boldsymbol{x})\sqrt{\tau}}\right)\text{scale}_2(\boldsymbol{x})\sqrt{\tau} \tag{3.7}$$

where main(x, τ, r, q) is defined in Equation (3.6) and bias$_2(\boldsymbol{x})$ and scale$_2(\boldsymbol{x})$ are both neural networks with an input of $\boldsymbol{x}$. The SiLU function has a sharper turn than softplus and is able to replicate the shape of the price curves if combined with softplus (see Figure 3.4 (b)).

3.3.4 Full structure of the neural network

We have introduced the singular terms, as well as the four neural networks bias$(\boldsymbol{x})$, scale$(\boldsymbol{x})$, bias$_2(\boldsymbol{x})$ and scale$_2(\boldsymbol{x})$. Now we discuss how to embed the singular terms into the MLP. Although bias$(\boldsymbol{x})$, scale$(\boldsymbol{x})$, bias$_2(\boldsymbol{x})$ and scale$_2(\boldsymbol{x})$ can be four independent neural networks, we decide to let them share parameters with each other.

The augmented MLP with the singular terms can be explained as follows:

$$\begin{aligned}
\boldsymbol{x}^{(0)} &= \boldsymbol{x}, \\
\boldsymbol{x}^{(j)} &= g\left(\boldsymbol{W}^{(j-1)}\boldsymbol{x}^{(j-1)} + \boldsymbol{b}^{(j-1)}\right), \ \forall 1 \le j \le L_1, \\
\text{bias}(\boldsymbol{x}) &= \boldsymbol{W}^{(\text{bias})}\boldsymbol{x}^{(L_1)} + b^{(\text{bias})}, \\
\text{scale}(\boldsymbol{x}) &= \text{softplus}\left(\boldsymbol{W}^{(\text{scale})}\boldsymbol{x}^{(L_1)} + b^{(\text{scale})}\right), \\
\text{bias}_2(\boldsymbol{x}) &= \boldsymbol{W}^{(\text{bias}_2)}\boldsymbol{x}^{(L_1)} + b^{(\text{bias}_2)}, \\
\text{scale}_2(\boldsymbol{x}) &= \text{softplus}\left(\boldsymbol{W}^{(\text{scale}_2)}\boldsymbol{x}^{(L_1)} + b^{(\text{scale}_2)}\right), \\
\text{singular}(\boldsymbol{x}) &= \text{softplus}\left(\frac{\text{main}(x, \tau, r, q) + \text{bias}(\boldsymbol{x})\tau}{\text{scale}(\boldsymbol{x})\sqrt{\tau}}\right)\text{scale}(\boldsymbol{x})\sqrt{\tau}, \\
\text{singular}_2(\boldsymbol{x}) &= \text{SiLU}\left(\frac{\text{main}(x, \tau, r, q) + \text{bias}_2(\boldsymbol{x})\tau}{\text{scale}_2(\boldsymbol{x})\sqrt{\tau}}\right)\text{scale}_2(\boldsymbol{x})\sqrt{\tau}, \\
\tilde{\boldsymbol{x}}^{(L_1)} &= \text{concatenate}(\boldsymbol{x}^{(L_1)}, \text{singular}(\boldsymbol{x}), \text{singular}_2(\boldsymbol{x})), \\
\boldsymbol{x}^{(L_1+1)} &= g\left(\boldsymbol{W}^{(L_1)}\tilde{\boldsymbol{x}}^{(L_1)} + \boldsymbol{b}^{(L_1)}\right), \\
\boldsymbol{x}^{(j)} &= g\left(\boldsymbol{W}^{(j-1)}\boldsymbol{x}^{(j-1)} + \boldsymbol{b}^{(j-1)}\right), \ \forall L_1 + 1 < j \le L_1 + L_2, \\
w &= \sum_{0 \le j \le L_1 + L_2, j \ne L_1} \boldsymbol{W}^{(j,w)}\boldsymbol{x}^{(j)} + \boldsymbol{W}^{(L_1,w)}\tilde{\boldsymbol{x}}^{(L_1)} + b^{(w)},
\end{aligned} \tag{3.8}$$

where the jth hidden layer $\boldsymbol{x}^{(j)}$ is a vector of length n for $1 \le j \le L = L_1 + L_2$, and the input layer $\boldsymbol{x}$ is of size n_0. scale($\boldsymbol{x}$) and scale$_2$($\boldsymbol{x}$) are past through the softplus function to ensure the positivity. singular($\boldsymbol{x}$) and singular$_2$($\boldsymbol{x}$) are built from the hidden layer $\boldsymbol{x}^{(L_1)}$ and then combined with it as $\tilde{\boldsymbol{x}}^{(L_1)}$. The output w has skip connections from all the previous layers, and the skip connections help stabilize the training process. The dimensions of the neural network parameters are

$$
\begin{aligned}
\boldsymbol{W}^{(0)} &\in \mathbb{R}^{n\times n_0},\\
\boldsymbol{W}^{(j)} &\in \mathbb{R}^{n\times n} \text{ for } 1 \le j \le L_1 - 1 \text{ or } L_1 + 1 \le j \le L_1 + L_2 - 1,\\
\boldsymbol{W}^{(L_1)} &\in \mathbb{R}^{n\times(n+2)},\\
\boldsymbol{W}^{(j)} &\in \mathbb{R}^{1\times n} \text{ for } j = \text{bias, scale, bias}_2, \text{scale}_2,\\
\boldsymbol{W}^{(0,w)} &\in \mathbb{R}^{1\times n_0},\\
\boldsymbol{W}^{(j,w)} &\in \mathbb{R}^{1\times n} \text{ for } 1 \le j \le L_1 - 1 \text{ or } L_1 + 1 \le j \le L_1 + L_2,\\
\boldsymbol{W}^{(L_1,w)} &\in \mathbb{R}^{1\times(n+2)},\\
\boldsymbol{b}^{(j)} &\in \mathbb{R}^{n} \text{ for } 0 \le j \le L_1 + L_2 - 1,\\
b^{(j)} &\in \mathbb{R} \text{ for } j = w, \text{bias, scale, bias}_2, \text{scale}_2.
\end{aligned}
$$

The overall structure is of width n, with L_1 layers before the singular terms and L_2 layers after the singular terms. A graph of the neural network with $(L_1, L_2) = (1, 1)$ is illustrated in Figure 3.5 if we omit the skip connections.

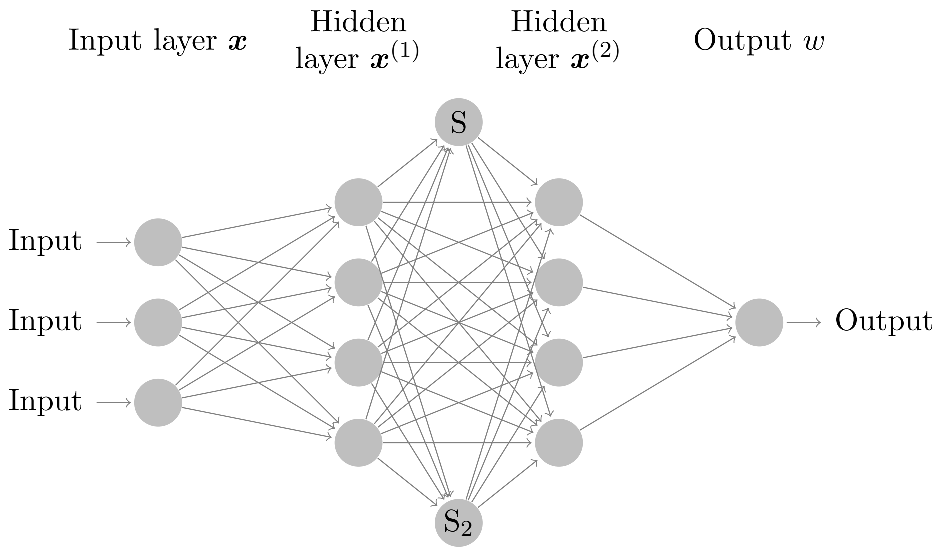

Input layer $\boldsymbol{x}$
Hidden layer $\boldsymbol{x}^{(1)}$
Hidden layer $\boldsymbol{x}^{(2)}$
Output w
S
Input
Input
Input
Output
S_2

Figure 3.5: Illustration of the neural network with the singular terms. ‘S’ and ‘S_2’ stand for the two singular terms.

3.4 Loss function

3.4.1 Initial and boundary conditions

For different kinds of options, we listed their corresponding initial and boundary conditions in Table 3.4, where $x_{\min}$ ($x_{\max}$) is a small (large) enough constant. The neural network will approximate the solution $w(x,\tau)$ within $x_{\min} \le x \le x_{\max}$. Also, in the training process, x will be sampled between $x_{\min}$ and $x_{\max}$.

Option	$\mathrm{IC}(x)$	$\mathrm{BC}_{\min}(\tau, r, q)$	$\mathrm{BC}_{\max}(\tau, r, q)$
European call	$(e^x - K)^+$	0	$e^{x_{\max}-q\tau} - Ke^{-r\tau}$
European put	$(K - e^x)^+$	$Ke^{-r\tau} - e^{x_{\min}-q\tau}$	0
American call	$(e^x - K)^+$	0	$e^{x_{\max}} - K$
American put	$(K - e^x)^+$	$K - e^{x_{\min}}$	0

Table 3.4: Initial and boundary conditions of European/American call/put.

The solution $w(x,\tau)$ should satisfy

$$\begin{aligned} w(x,0) &= \mathrm{IC}(x), \\ w(x_{\min},\tau) &= \mathrm{BC}_{\min}(\tau,r,q), \\ w(x_{\max},\tau) &= \mathrm{BC}_{\max}(\tau,r,q). \end{aligned}$$

3.4.2 Loss function

Denote the neural network parameters as

$$\mathcal{W} = \left\{ \begin{array}{l} \boldsymbol{W}^{(j)}, \boldsymbol{b}^{(j)} \text{ for } 0 \le j \le L_1 + L_2 - 1 \text{ or } j = \text{bias, scale, bias}_2, \text{scale}_2 \\ \boldsymbol{W}^{(j,w)} \text{ for } 0 \le j \le L_1 + L_2 \text{ and } b^{(w)} \end{array} \right\}. \tag{3.9}$$

Given a sample $\boldsymbol{x}$, the loss function for European puts is defined as following:

$$\begin{aligned} L_{\mathrm{EU,P}}(\mathcal{W};\boldsymbol{x}) = &(H(w,x,\tau))^2 + (w(x,0) - \mathrm{IC}(x))^2 \\ &+ (w(x_{\min},\tau) - \mathrm{BC}_{\min}(\tau,r,q))^2 + (w(x_{\max},\tau) - \mathrm{BC}_{\max}(\tau,r,q))^2 \end{aligned} \tag{3.10}$$

where $H(w, x, \tau)$ is defined in Equation (3.2) and $\mathrm{IC}(x)$, $\mathrm{BC}_{\min}(\tau, r, q)$, and $\mathrm{BC}_{\max}(\tau, r, q)$ are defined in Table 3.4. The function $w(x, \tau)$ is now approximated by a neural network with the input of $\boldsymbol{x}$ and the parameters of $\mathcal{W}$.

The loss function for American puts is different from the one for European puts because of the early exercise and it is given as

$$\begin{aligned} L_{\mathrm{AM,P}}(\mathcal{W}; \boldsymbol{x}) = &\left(\max\left(H(w, x, \tau), \mathrm{IC}(x) - w(x, \tau)\right)\right)^2 + (w(x, 0) - \mathrm{IC}(x))^2 \\ &+ (w(x_{\min}, \tau) - \mathrm{BC}_{\min}(\tau, r, q))^2 + (w(x_{\max}, \tau) - \mathrm{BC}_{\max}(\tau, r, q))^2 \end{aligned} \tag{3.11}$$

where $H(w, x, \tau)$ is defined in Equation (3.2) and $\mathrm{IC}(x)$, $\mathrm{BC}_{\min}(\tau, r, q)$, and $\mathrm{BC}_{\max}(\tau, r, q)$ are defined in Table 3.4. The only difference between Equation (3.10) and (3.11) is that the first term of Equation (3.10) is the square of $H(w, x, \tau)$ while the first term of Equation (3.11) is the square of $\max\left(H(w, x, \tau), \mathrm{IC}(x) - w(x, \tau)\right)$. The difference accords with the equations of European and American puts in Table 3.2.

However, for call options, $w(x, \tau) \propto \exp(x)$ when x is large, so the terms in Equation (3.10) and (3.11) grow at the rate of $\exp(2x)$ when x increases. It is necessary to compensate the oversized loss function when x is large, otherwise one sample would dominate in the gradient descent algorithm. We define the weight function as

$$u(x) = \min(1, 4K^2 \exp(-2x)),$$

which only compensates the oversized loss when $S = e^x > 2K$. The loss function for European calls is given as

$$\begin{aligned} L_{\mathrm{EU,C}}(\mathcal{W}; \boldsymbol{x}) = &(H(w, x, \tau))^2\, u(x) + (w(x, 0) - \mathrm{IC}(x))^2\, u(x) \\ &+ (w(x_{\min}, \tau) - \mathrm{BC}_{\min}(\tau, r, q))^2 + (w(x_{\max}, \tau) - \mathrm{BC}_{\max}(\tau, r, q))^2\, u(x_{\max}) \end{aligned}$$

and the one for American calls is given as

$$L_{\mathrm{AM,C}}(\mathcal{W};\boldsymbol{x}) = \left(\max\left(H(w,x,\tau),\mathrm{IC}(x)-w(x,\tau)\right)\right)^2 u(x) + \left(w(x,0)-\mathrm{IC}(x)\right)^2 u(x)$$
$$+\left(w(x_{\min},\tau)-\mathrm{BC}_{\min}(\tau,r,q)\right)^2 + \left(w(x_{\max},\tau)-\mathrm{BC}_{\max}(\tau,r,q)\right)^2 u(x_{\max})$$

The losses $L_{\mathrm{EU,P}}$, $L_{\mathrm{AM,P}}$, $L_{\mathrm{EU,C}}$ and $L_{\mathrm{AM,C}}$ are defined on a single sample $\boldsymbol{x}$. Given a lot of samples $\boldsymbol{x}_1,\boldsymbol{x}_2,\ldots,\boldsymbol{x}_n$, the total loss is an average of the individual losses, e.g.,

$$L_{\mathrm{avg}}\left(\mathcal{W};\{\boldsymbol{x}_j\}_{j=1}^n\right) = \begin{cases} \frac{1}{n}\sum_{j=1}^n L_{\mathrm{EU,P}}(\mathcal{W};\boldsymbol{x}_j), & \text{for European puts,}\\ \frac{1}{n}\sum_{j=1}^n L_{\mathrm{AM,P}}(\mathcal{W};\boldsymbol{x}_j), & \text{for American puts,}\\ \frac{1}{n}\sum_{j=1}^n L_{\mathrm{EU,C}}(\mathcal{W};\boldsymbol{x}_j), & \text{for European calls,}\\ \frac{1}{n}\sum_{j=1}^n L_{\mathrm{AM,C}}(\mathcal{W};\boldsymbol{x}_j), & \text{for American calls.} \end{cases}$$

3.4.3 Summarized algorithms

After we fully define the neural network structure and the loss functions, we summarize the algorithms for both European and American options in Algorithm 1. In all samples of $\boldsymbol{x}_j$, the log-price x needs to satisfy $x_{\min} \le x \le x_{\max}$.

Algorithm 1 Training routine

```
Require: A neural network $w(x,\tau)$ defined in Equation (3.8) with parameters $\mathcal{W}$ defined in Equation (3.9)
  for iter = $1,2,\ldots N$ do
      Generate a batch of random samples $\{\boldsymbol{x}_j\}_{j=1}^{n_b}$, where $n_b$ is the batch size
      Take one gradient descent step w.r.t. $\mathcal{W}$ according to the loss function $L_{\mathrm{avg}}\left(\mathcal{W};\{\boldsymbol{x}_j\}_{j=1}^{n_b}\right)$
  end for
```

3.5 Calculation

3.5.1 Derivatives and integral

In Section 3.3 we introduced how to build the neural network. Since we always use smooth activation functions in the neural network, e.g., SiLU and softplus, the neural network is smooth and the derivatives $\frac{\partial w}{\partial \tau}(x,\tau)$, $\frac{\partial w}{\partial x}(x,\tau)$ and $\frac{\partial^2 w}{\partial x^2}(x,\tau)$ are calculated by back-propagation. For a sample $\boldsymbol{x}$, we calculate the integral

$$\int_{-\infty}^{\infty}\left(w(x+y,\tau)-w(x,\tau)-\frac{\partial w}{\partial x}(x,\tau)(e^y-1)\right)m(dy)$$

in two parts. The inner part

$$\int_{-\epsilon^-\le y\le\epsilon^+}\left(w(x+y,\tau)-w(x,\tau)-\frac{\partial w}{\partial x}(x,\tau)(e^y-1)\right)m(dy)$$

is approximated by

$$\left(\frac{\partial^2 w}{\partial x^2}(x,\tau)-\frac{\partial w}{\partial x}(x,\tau)\right)\int_{-\epsilon^-\le y\le\epsilon^+}\frac{y^2}{2}m(dy)$$

the same way as described in Chapter 5 in [50], where $\epsilon^-,\epsilon^+\ge 0$ are small. For the outer part, we write it as

$$\begin{aligned}&\int_{y<-\epsilon^- \text{ or } y>\epsilon^+}\left(w(x+y,\tau)-w(x,\tau)-\frac{\partial w}{\partial x}(x,\tau)(e^y-1)\right)m(dy)\\ &=\int_{y<-\epsilon^- \text{ or } y>\epsilon^+}\left(w(x+y,\tau)-w(x,\tau)\right)m(dy)-\frac{\partial w}{\partial x}(x,\tau)\int_{y<-\epsilon^- \text{ or } y>\epsilon^+}(e^y-1)m(dy).\end{aligned}$$

The first part is calculated using the Simpson's rule as explained in [108]. Further details of the inner and outer part are included in Appendix B.1. With the help of neural network approximation, we calculate each part on the left hand side of the PIDE (3.2) for a given sample $\boldsymbol{x}$.

3.5.2 Extrapolation of the price function in the integral

During the training process, we need to calculate the integral

$$\int_{y<-\epsilon^- \text{ or } y>\epsilon^+} (w(x+y,\tau) - w(x,\tau))\, m(dy)$$

where $w(x,\tau)$ is approximated by a neural network within the boundary $x_{\min} \le x \le x_{\max}$. However, $x + y$ could be out of the boundary if x is sampled close to the boundary. In this case, $w(x+y,\tau)$ cannot be approximated by the neural network, and thus we need to perform extrapolation to calculate the integral. Also, the extrapolation of $w(x,\tau)$ needs to be continuous at $x = x_{\min}$ or $x = x_{\max}$. Since $\int m(dy)$ might be infinity around zero, the continuity of $w(x,\tau)$ ensures that $w(x+y,\tau) - w(x,\tau)$ is close to 0 and the integral is convergent. The extrapolation is given in Table 3.5.

Option	$w(x,\tau) - w(x_{\min},\tau)$ when $x < x_{\min}$	$w(x,\tau) - w(x_{\max},\tau)$ when $x > x_{\max}$
European call	0	$(\exp(x) - \exp(x_{\max}))\exp(-q\tau)$
European put	$(\exp(x_{\min}) - \exp(x))\exp(-q\tau)$	0
American call	0	$\exp(x) - \exp(x_{\max})$
American put	$\exp(x_{\min}) - \exp(x)$	0

Table 3.5: Extrapolation of $w(x,\tau)$ for European/American call/put.

Take the European call as an example, when $x < x_{\min}$, $w(x,\tau) \approx 0$. So we let

$$w(x,\tau) = w(x_{\min},\tau)$$

such that the continuity is preserved. When $x > x_{\max}$, $w(x,\tau) \approx \exp(x - q\tau) - K\exp(-r\tau)$. Similarly we let

$$w(x,\tau) = w(x_{\max},\tau) + (\exp(x) - \exp(x_{\max}))\exp(-q\tau)$$

such that the continuity is preserved.

3.6 Numerical experiments

3.6.1 Range of parameters and distribution of samples

For all the five models in Section 3.2.4, we consider the option price $w(\cdot)$ with the following parameters:

$$
\begin{aligned}
K &= 100 \\
\ln(K/2) \leq &x \leq \ln(2K) \\
0 < &\tau \leq 3, \\
0 \leq &r, q \leq 0.1.
\end{aligned}
$$

We train the neural network $w(\cdot)$ within the range of the model parameters given in Table 3.6.

Model	Parameters
VG	$0.1 \leq \sigma \leq 0.5, 0.1 \leq \nu \leq 0.6, -0.5 \leq \theta \leq -0.1$
CGMY	$0.1 \leq \sigma \leq 0.5, 0.1 \leq \nu \leq 0.6, -0.5 \leq \theta \leq -0.1, 0 \leq Y \leq 1$
NIG	$5 \leq \alpha \leq 20$, $-2\alpha/3 \leq \beta \leq 2\alpha/3$, $0.1 \leq \delta \leq 3$
Merton's	$0.1 \leq \sigma \leq 0.5$, $0 \leq \lambda \leq 1$, $-0.5 \leq \alpha \leq 0.5$, $0.01 \leq \delta \leq 0.5$
Kou's	$0.1 \leq \sigma \leq 0.5$, $0 \leq \lambda \leq 2$, $0 \leq p \leq 1$, $3 \leq \eta_1 \leq 15$, $3 \leq \eta_2 \leq 15$

Table 3.6: Range of the model parameters in the five models under Lévy process.

All the variables and parameters follow the uniform distribution within the specified ranges in both training and test data, except that

- β in the NIG model follows a conditional uniform distribution over $(-2\alpha/3, 2\alpha/3)$ given α.
- x follows different distributions in the training and test data. In the test data, x follows the uniform distribution on $\ln(K/2) \leq x \leq \ln(2K)$. In the training data, half of the samples follow the uniform distribution on $\ln(K/2) \leq x \leq \ln(2K)$ while the other half follow the uniform distribution on $x_{\min} \leq x \leq x_{\max}$. The distribution of the training data is to make sure the solution is fitted over $x_{\min} \leq x \leq x_{\max}$ while training is focused within $\ln(K/2) \leq x \leq \ln(2K)$.

During the training process, boundaries of x are set to be $x_{\min} = \ln(K/50)$ and $x_{\max} = \ln(50K)$. However, for the CGMY model and the NIG model, we let $x_{\max} = \ln(500K)$ because these two models attach higher probability to large negative jumps such that boundary conditions are satisfied at larger x.

The uniform random samples are given by the Sobol sequence [98], which is a quasi random sequence.

3.6.2 Scope of application of the method

In Appendix B.1.3, we explained how we perform numerical integration to calculate

$$\int_{y<-\epsilon^- \text{ or } y>\epsilon^+} (w(x+y,\tau) - w(x,\tau))\, m(dy)$$

If $k(y) \propto y^{-(2+\delta)}$, where $\delta > 0$, the spike of $k(y)$ is very sharp. While we could perform numerical integration in this case, we need more points in the integral grid and consequently we slow down the method. Thus it is recommended that $k(y)$ satisfies $\lim_{y\to 0} k(y)y^{2+\delta} = 0, \forall\delta > 0$ in this method. The VG, NIG, Merton's and Kou's model all satisfy this condition. The CGMY model with $Y \leq 1$ also satisfies this condition. Hence the scope of application is still wide enough.

3.6.3 Hyper-parameters and training results

We consider the neural network defined in Equation (3.8) consisting of $(L_1, L_2) = (3, 3)$ layers, with $n = 500$ neurons in each layer. The activation function g is SiLU. We use He-normal initialization [47] and Adam algorithm [67] for training. We do not use batch-normalization [60] or dropout [99]. There are 500,000 training samples and 10,000 test samples and the batch size is

200. The learning rate is defined as following:

$$\text{learning rate} = \begin{cases} 10^{-3}, & \text{from epoch 1 to 15,} \\ 10^{-3} \text{ decreasing exponentially to } 10^{-4}, & \text{from epoch 16 to 30,} \\ 10^{-4}, & \text{from epoch 31 to 45.} \end{cases}$$

The results are summarized in Table 3.7 and 3.8. In Table 3.7, we list the root mean square error (RMSE) of each model, i.e.,

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{1 \le j \le n} \left(w(\boldsymbol{x}_j) - \bar{w}(\boldsymbol{x}_j) \right)^2},$$

where $\boldsymbol{x}_j$'s are the samples following the distribution in Section 3.6.1, $w(\boldsymbol{x}_j)$ is the solution given by the neural network and $\bar{w}(\boldsymbol{x}_j)$ is the benchmark. The RMSE is calculated over the 10,000 test samples, which have not been used during the training process. The RMSE is small for each model. To better understand where the large errors happen, we list the top three absolute errors $|w(\boldsymbol{x}) - \bar{w}(\boldsymbol{x})|$ in the test samples for each model in Table 3.8, along with the relative error $(w(\boldsymbol{x}) - \bar{w}(\boldsymbol{x}))/\max(\bar{w}(\boldsymbol{x}), h)$ where $h = 0.25$, the time to maturity τ and the moneyness $S/K = e^x/K$ of the corresponding samples. We can see that the large errors usually happen where $\tau > 1$. The benchmarks are calculated using the FFT method [17] for European options and the CONV method [79] for American options. The average training time of the 20 neural networks in this section is 3 hours 15 minutes on a Tesla P100-PCIE-16GB.

Model	European call	European put	American call	American put
VG	0.05	0.03	0.06	0.03
CGMY	0.07	0.05	0.10	0.07
NIG	0.07	0.05	0.12	0.08
Merton's	0.07	0.04	0.09	0.05
Kou's	0.08	0.07	0.12	0.08

Table 3.7: The root mean square error (RMSE) over the 10,000 test samples under each model.

Model	European call	European put	American call	American put
	0.31 (-0.3%) (2.72, 1.97)	0.18 (1.5%) (2.30, 1.14)	0.59 (-0.6%) (1.43, 1.88)	0.29 (24.3%) (0.01, 0.99)
VG	0.30 (-2.7%) (1.03, 0.96)	0.18 (1.4%) (1.58, 0.95)	0.53 (-0.5%) (2.90, 1.91)	0.28 (11.8%) (0.03, 0.98)
	0.29 (-2.7%) (2.51, 0.95)	0.18 (1.2%) (2.25, 0.88)	0.53 (-0.6%) (2.40, 1.81)	0.28 (-0.8%) (2.78, 0.67)
	1.06 (2.9%) (2.99, 0.70)	0.49 (0.8%) (2.73, 0.62)	0.90 (0.7%) (2.82, 1.88)	0.87 (-1.5%) (2.66, 0.51)
CGMY	0.93 (1.2%) (2.52, 1.61)	0.47 (2.3%) (2.52, 1.61)	0.79 (-1.0%) (1.92, 1.72)	0.73 (-1.3%) (2.59, 0.54)
	0.90 (3.0%) (2.89, 0.50)	0.36 (0.5%) (2.99, 0.70)	0.70 (-1.1%) (2.02, 1.56)	0.68 (-1.3%) (2.98, 0.65)
	1.27 (-1.2%) (1.29, 1.68)	0.70 (-1.7%) (1.29, 1.68)	1.25 (-1.2%) (2.80, 1.92)	0.71 (-2.4%) (2.61, 0.88)
NIG	0.70 (23.0%) (1.32, 1.03)	0.60 (5.2%) (2.31, 0.98)	1.09 (1.1%) (1.79, 1.80)	0.68 (-1.8%) (0.22, 0.72)
	0.69 (15.4%) (2.31, 0.98)	0.48 (8.4%) (2.81, 1.23)	1.05 (-1.7%) (1.47, 1.59)	0.66 (-2.9%) (1.07, 0.79)
	0.65 (61.0%) (0.53, 0.90)	0.60 (-2.4%) (0.32, 0.88)	0.97 (1.8%) (1.41, 1.53)	0.51 (4.6%) (0.53, 0.90)
Merton's	0.57 (-4.3%) (0.32, 0.88)	0.30 (-1.2%) (1.41, 0.80)	0.75 (48.1%) (0.02, 1.01)	0.45 (-2.2%) (2.39, 0.89)
	0.52 (4.2%) (2.44, 0.95)	0.27 (-1.9%) (2.27, 1.05)	0.69 (1.1%) (1.57, 1.59)	0.43 (0.9%) (2.52, 0.52)
	0.97 (4.1%) (2.54, 0.70)	0.67 (12.9%) (2.77, 1.70)	1.10 (1.2%) (2.93, 1.90)	0.72 (21.8%) (2.12, 1.04)
Kou's	0.73 (0.7%) (1.79, 1.80)	0.63 (-7.2%) (0.74, 1.23)	0.96 (0.9%) (2.82, 1.88)	0.62 (22.5%) (2.99, 1.54)
	0.67 (6.1%) (0.34, 1.09)	0.60 (-4.0%) (0.86, 1.10)	0.91 (11.6%) (0.22, 1.03)	0.54 (5.2%) (0.41, 0.94)

Table 3.8: Top three absolute errors over the 10,000 test samples under each model. Each sample is recorded as 'absolute error $|w(\boldsymbol{x}) - \bar{w}(\boldsymbol{x})|$ (relative error $(w(\boldsymbol{x}) - \bar{w}(\boldsymbol{x}))/\max(\bar{w}(\boldsymbol{x}), h)$) (time to maturity τ, moneyness e^x/K)'.

3.6.4 Short maturity fitting

Inclusion of the singular terms in the neural network would help dealing with the initial condition since

$$\lim_{\tau\to 0^+} \text{softplus/SiLU}\left(\frac{\text{main}(x,\tau,r,q)+\text{bias}(\boldsymbol{x})}{\text{scale}(\boldsymbol{x})\sqrt{\tau}}\right)\text{scale}(\boldsymbol{x})\sqrt{\tau} = \begin{cases}(e^x - K)^+, \text{ for calls}\\ \\ (K - e^x)^+, \text{ for puts}\end{cases}$$

Even if we always use smooth activation functions in the neural network, the singular terms enable the neural network to follow the singular property at $(x, \tau) = (\ln(K), 0)$. As a result, the network structure is good at fitting the option price surface at short maturities. In Figure 3.6, we show the fitted solution of the Kou's model at $T = 1/252$ (1 day) and $T = 1/52$ (1 week) as well as the relative error $(w(\boldsymbol{x}) - \bar{w}(\boldsymbol{x}))/\max(\bar{w}(\boldsymbol{x}), h)$ as an example. For this example we use $h = 0.25$. Note that when the benchmark $\bar{w}(\boldsymbol{x})$ is very small, the relative error can be large but the absolute error is still small.

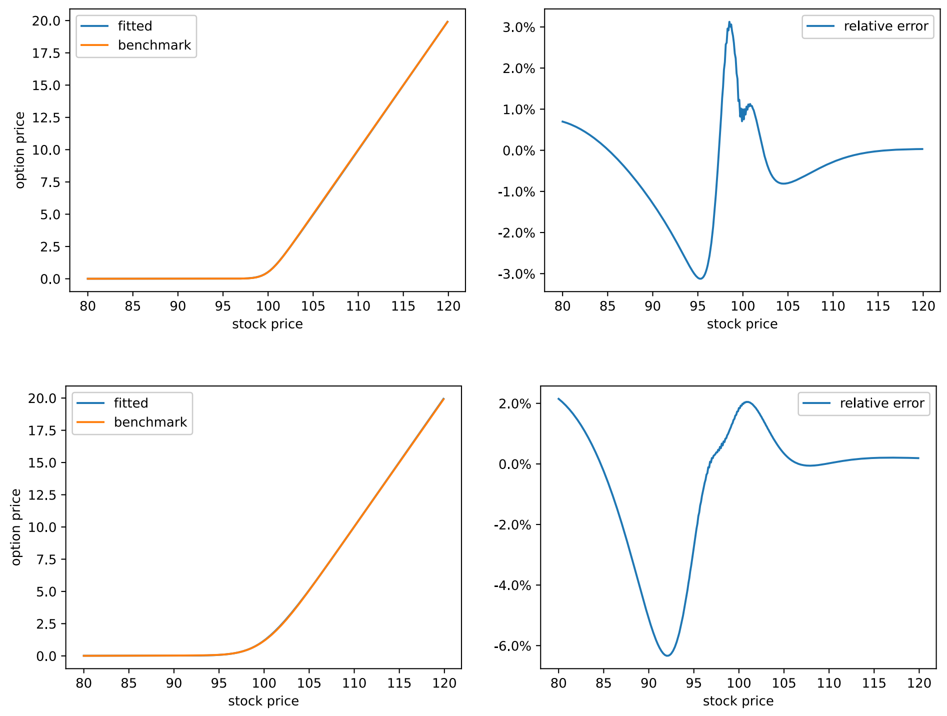


Figure 3.6: American call prices of the Kou's model when $K = 100$, $r = 0.02$, $q = 0.02$, $\sigma = 0.2$, $\lambda = 1$, $p = 0.5$, $\eta_1 = 12$ and $\eta_2 = 12$. Up: $\tau = 1/252$ (1 day). Down: $\tau = 1/52$ (1 week).

3.6.5 Calculation speed

The benefit of the neural network pricing method is that the network can calculate prices of a batch of parameters at the same time. Thus it will be super fast to generate the option prices once trained. The calculation times of different input sizes are summarized in Table 3.9. By means of the GPU acceleration, 1 million prices can be calculated in 0.5 seconds.

Input size	1	3	10	30	100	300	
GPU(s)	0.020	0.020	0.017	0.017	0.017	0.015	
CPU(s)	0.075	0.135	0.027	0.034	0.039	0.059	
Input size	1k	3k	10k	30k	100k	300k	1 million
GPU(s)	0.015	0.015	0.015	0.020	0.057	0.169	0.543
CPU(s)	0.099	0.207	0.580	1.704	6.077	19.059	61.103

Table 3.9: Computation time of the network defined in Equation (3.8) consisting of $(L_1, L_2) = (3, 3)$ layers, with $n = 500$ neurons in each layer. CPU is an Intel Xeon CPU @ 2.20GHz. GPU is a Tesla P100-PCIE-16GB. All times are in seconds.

3.6.6 Greeks

Since the method solves the option price by minimizing a loss function containing the derivatives, we not only get the prices but also the Greeks. The derivatives are obtained by back-propagation. Suppose $V(S,t) = w(\ln(S), T-t)$ is the option price of a certain strike K. Then delta is

$$\Delta = \frac{\partial V}{\partial S} = S^{-1}\frac{\partial w}{\partial x},$$

gamma is

$$\Gamma = \frac{\partial^2 V}{\partial S^2} = S^{-2}\left(\frac{\partial^2 w}{\partial x^2} - \frac{\partial w}{\partial x}\right)$$

and theta is

$$\Theta = \frac{\partial V}{\partial t} = -\frac{\partial w}{\partial \tau}.$$

In Figures 3.7, we use the CGMY as an example to show the curves of the price, delta Δ, gamma Γ, and theta Θ of the fitted solution and the benchmark, and also show the relative error

$$\frac{\text{fitted} - \text{benchmark}}{\text{benchmark}}.$$

We found the consistent patterns for the other models. In some cases when the price, delta and gamma are very close to 0, the relative error can be large. However, the fitted value will also be close to zero and the absolute error is still small.

3.7 Conclusion

In this chapter, we have proposed a pricing approach using unsupervised deep learning. Specifically, we use a neural network with additional singular terms to approximate the solution to the PIDE. The singular terms are designed to meet the non-smooth initial conditions and follow the property of the solution near maturity. The singular term singular($\boldsymbol{x}$) satisfies

$$\lim_{\tau \to 0^+} \text{singular}(\boldsymbol{x}) = \begin{cases} (e^x - K)^+, & \text{for calls} \\ (K - e^x)^+, & \text{for puts} \end{cases}$$

by its definition, and so does the other singular term singular$_2(\boldsymbol{x})$. So the singular point of the singular terms is always fixed to the singular point of the option surface during training. The method has been tested on European/American calls/puts in the five models under Lévy processes and can be applied to other models based on Lévy processes.

The first benefit of this approach is that we only need to train the neural network once for a given model and can then utilize the trained network to calculate option prices super fast as shown in Table 3.9. The second benefit is that we do not need labels for training. The third is that by this approach we obtain not only the option price itself, but also the option Greeks without any extra cost or effort.

In the proposed approach, the singular terms are defined based on the asymptotic behaviors around the singular point. Singular term(s) can be extended to accommodate other types of problems as long as we know the asymptotic behaviors around their singular points. In Chapter 4, we will show how to extend the approach using singular terms to solve barrier options.

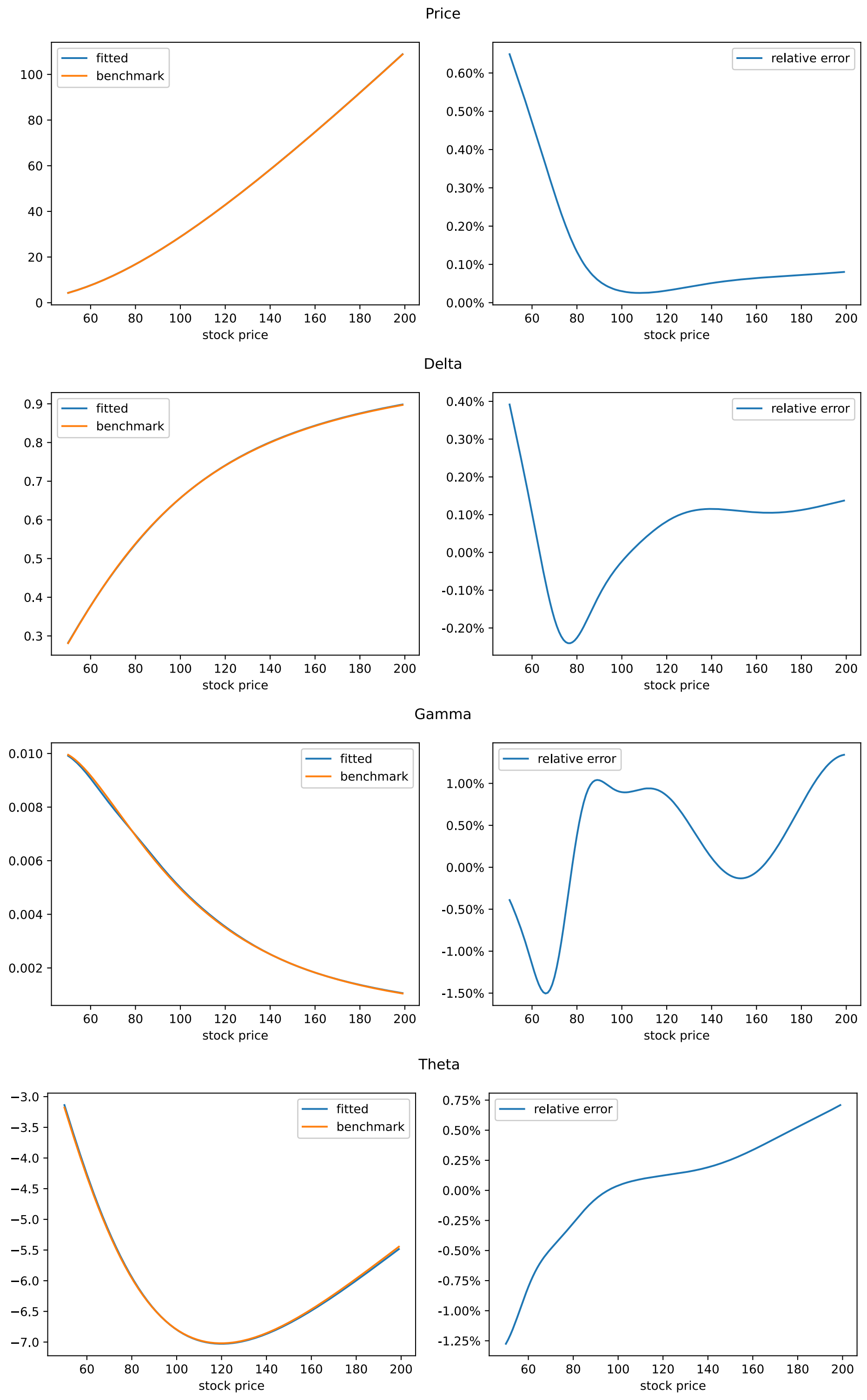


Figure 3.7: American call prices and Greeks of the CGMY model when $K = 100$, $\tau = 2$, $r = 0.05$, $q = 0.02$, $\sigma = 0.3$, $\nu = 0.3$, $\theta = -0.3$ and $Y = 0.5$.

Chapter 4: Pricing barrier options under the Bergomi model with unsupervised deep learning

4.1 Introduction

In this chapter we extend the deep learning approach using PDE to the barrier options, and we propose a pricing method that includes vanilla and barrier options for stochastic volatility models and test it under the Bergomi model [9]. The Bergomi model is a multi-factor stochastic volatility model, which contains more parameters and also a function input. Deep learning is employed to deal with the high dimensionality of the parameter space in the Bergomi model, and is also applicable to the other stochastic models with fewer parameters.

In the proposed method, we fit option price surfaces with neural networks. The biggest challenge for a smooth neural network to fit the barrier options is to fit the discontinuous boundary conditions. Thus we propose two singular terms [37] and embed them into the neural networks such that the neural networks are not smooth at given points, i.e., the strike and barrier levels at maturity, but are smooth anywhere else. In this way, the networks are able to satisfy the boundary conditions and the PDE at the same time. We train the neural networks with different parameters and the neural networks calculate option values fast after being trained. Having fast models for pricing both vanilla and barrier options would be helpful in assessing the effect of model risk [52].

The chapter is organized as follows. In Section 4.2, we introduce the Bergomi model, the definition of the vanilla and barrier options and the equation groups used for option pricing under the Bergomi model. In Section 4.3, we generally introduce the singular terms used for the vanilla and barrier options and the framework of option pricing. In Section 4.4, we give the definition of the singular term and the neural network for the vanilla options. We also discuss the boundary conditions of the volatility factors and the loss functions used to train the networks of vanilla options. In

Section 4.5, we give the definition of the singular term, the neural network and the loss functions for the barrier options. In Section 4.6, we give the details of numerical experiments, including the piecewise constant function input, the range and distribution of samples for training and the hyperparameters of the neural networks. We show the numerical results from the fitted neural network solutions in terms of the root mean squared error, the relative error and the calculation speed. Section 4.7 summarizes the chapter.

4.2 Option pricing under the Bergomi model

In this chapter, we focus on solving the barrier options under the Bergomi model, which is a multi-factor stochastic volatility model. It is a general framework proposed by [9] to capture forward volatility and forward skew risks. The proposed pricing routine is also applicable to other stochastic volatility models after modifications of the boundary conditions of volatility, and of course the basic case of the Black-Merton-Scholes (BMS) model [10]. In the case of the Bergomi model, we will see how deep learning is employed to deal with the high dimensionality embedded in the model.

4.2.1 Bergomi model

The general n-factor Bergomi model is based on the following lognormal dynamics of the forward variances in [9]

$$\xi_t^T = \xi_0^T \exp\left(\omega \sum_{1\le i\le n} w_i e^{-k_i(T-t)} X_t^{(i)} - \frac{\omega^2}{2} \sum_{1\le i,j\le n} w_i w_j e^{-(k_i+k_j)(T-t)} \mathbb{E}\left(X_t^{(i)} X_t^{(j)}\right)\right)$$

where

- $\xi_t^T, 0 \le t \le T$, is the process of forward instantaneous variance for date T observed at t,
- $\xi_0^T, T \ge 0$, is the initial value of forward variances and is also an input of the model,
- $X_t^{(i)}, \forall 1 \le i \le n$, are OU processes that satisfy $\mathrm{d}X_t^{(i)} = -k_i X_t^{(i)}\mathrm{d}t + \mathrm{d}W_t^{(i)}$ and $X_0^{(i)} = 0$,

- $w_i, \forall 1 \le i \le n$, are positive weights and ω is a global scaling factor for the volatility of forward variances,
- $W_t^{(i)}, \forall 1 \le i \le n$, are correlated Brownian motions, where $\mathrm{d}W_t^{(i)}\mathrm{d}W_t^{(j)} = \rho_{i,j}\mathrm{d}t$.

In [9], the author claims that two factors ($n = 2$) afford adequate control on the term structure of volatilities of volatilities, and then the multi-factor model is simplified as the two-factor model. Let $0 \le \theta \le 1$ be a constant and

$$\alpha_\theta = 1/\sqrt{(1-\theta)^2 + \theta^2 + 2\rho_{1,2}\theta(1-\theta)}.$$

The weights in the two-factor model are $w_1 = \alpha_\theta(1-\theta)$ and $w_2 = \alpha_\theta\theta$. By introducing the notation

$$x_t^T = \alpha_\theta \left((1-\theta)e^{-k_1(T-t)}X_t^{(1)} + \theta e^{-k_2(T-t)}X_t^{(2)} \right),$$

the dynamics of the forward variances can be simplified as

$$\xi_t^T = \xi_0^T \exp\left(\omega x_t^T - \frac{\omega^2}{2}\mathrm{var}(x_t^T) \right).$$

The risk neutral stock price S_t is given by

$$\mathrm{d}S_t = (r-q)S_t\mathrm{d}t + S_t\sqrt{\xi_t^t}\mathrm{d}W_t^{(S)}$$

where r is the risk-free interest rate, q is the dividend rate, $\mathrm{d}W_t^{(S)}\mathrm{d}W_t^{(i)} = \rho_i\mathrm{d}t, \forall i = 1, 2$, and ξ_t^t is given by

$$\begin{aligned}
\xi_t^t &= \xi_0^t \exp\left(\omega x_t^t - \frac{\omega^2}{2}\mathrm{var}(x_t^t) \right) \\
x_t^t &= \alpha_\theta \left((1-\theta)X_t^{(1)} + \theta X_t^{(2)} \right) \\
\mathrm{var}(x_t^t) &= \alpha_\theta^2 \left((1-\theta)^2\frac{1-e^{-2k_1t}}{2k_1} + \theta^2\frac{1-e^{-2k_2t}}{2k_2} + 2\theta(1-\theta)\rho_{1,2}\frac{1-e^{-(k_1+k_2)t}}{k_1+k_2} \right).
\end{aligned} \tag{4.1}$$

4.2.2 Option pricing

Suppose $\{S_t\}_{t\geq 0}$ is the stock price process, $s = \ln(S_t)$ is the log-price, K is the strike, B is the barrier level, t is the current time and T is the maturity (expiration) time. Denote the maximum and minimum of the stock price path as

$$m_t^T = \min_{t\leq \bar{t}\leq T} S_{\bar{t}} \text{ and } M_t^T = \max_{t\leq \bar{t}\leq T} S_{\bar{t}}.$$

The vanilla/barrier calls/puts are defined as

$$V(s,t,x_1,x_2) = e^{-r(T-t)}\mathbb{E}\left(\text{payoff}\,|S_t = e^s, X_t^{(1)} = x_1, X_t^{(2)} = x_2\right)$$

where V and payoff are replaced by the corresponding notation and formula in Table 4.1.

Option	V	payoff
vanilla call	C_{v}	$(S_T - K)^+$
vanilla put	P_{v}	$(K - S_T)^+$
up-and-out call	$C_{\text{u-o}}$	$(S_T - K)^+\mathbf{1}_{\{M_t^T<B\}}$
up-and-in call	$C_{\text{u-i}}$	$(S_T - K)^+\mathbf{1}_{\{M_t^T\geq B\}}$
down-and-out call	$C_{\text{d-o}}$	$(S_T - K)^+\mathbf{1}_{\{m_t^T>B\}}$
down-and-in call	$C_{\text{d-i}}$	$(S_T - K)^+\mathbf{1}_{\{m_t^T\leq B\}}$
up-and-out put	$P_{\text{u-o}}$	$(K - S_T)^+\mathbf{1}_{\{M_t^T<B\}}$
up-and-in put	$P_{\text{u-i}}$	$(K - S_T)^+\mathbf{1}_{\{M_t^T\geq B\}}$
down-and-out put	$P_{\text{d-o}}$	$(K - S_T)^+\mathbf{1}_{\{m_t^T>B\}}$
down-and-in put	$P_{\text{d-i}}$	$(K - S_T)^+\mathbf{1}_{\{m_t^T\leq B\}}$

Table 4.1: Payoffs of vanilla/barrier calls/puts.

The barrier options satisfy the following in-out parities according to their definitions.

$$\begin{cases} C_{\text{u-o}}(s,t,x_1,x_2) + C_{\text{u-i}}(s,t,x_1,x_2) = C_{\text{v}}(s,t,x_1,x_2), \forall s,x_1,x_2 \in \mathbb{R}, 0\leq t\leq T \\ C_{\text{d-o}}(s,t,x_1,x_2) + C_{\text{d-i}}(s,t,x_1,x_2) = C_{\text{v}}(s,t,x_1,x_2), \forall s,x_1,x_2 \in \mathbb{R}, 0\leq t\leq T \\ P_{\text{u-o}}(s,t,x_1,x_2) + P_{\text{u-i}}(s,t,x_1,x_2) = P_{\text{v}}(s,t,x_1,x_2), \forall s,x_1,x_2 \in \mathbb{R}, 0\leq t\leq T \\ P_{\text{d-o}}(s,t,x_1,x_2) + P_{\text{d-i}}(s,t,x_1,x_2) = P_{\text{v}}(s,t,x_1,x_2), \forall s,x_1,x_2 \in \mathbb{R}, 0\leq t\leq T \end{cases} \quad (4.2)$$

Our goal is to solve the option values at time $t = 0$, i.e., $V(s, 0, 0, 0)$.

4.2.3 Equations for option pricing

Using the Feynman-Kac formula [66], we can derive the PDE for the two-factor Bergomi model (see Appendix C.1). The option value $V(s, t, x_1, x_2)$ needs to satisfies the following equation in the applicable region for each option:

$$\begin{aligned} H(V, s, t, x_1, x_2) = \Bigg(&\frac{\partial V}{\partial t} - rV + (r - q - \frac{1}{2}\sigma^2(t, x_1, x_2))\frac{\partial V}{\partial s} \\ -k_1 x_1 \frac{\partial V}{\partial x_1} - k_2 x_2 \frac{\partial V}{\partial x_2} &+ \frac{1}{2}\sigma^2(t, x_1, x_2)\frac{\partial^2 V}{\partial s^2} + \frac{1}{2}\frac{\partial^2 V}{\partial x_1^2} + \frac{1}{2}\frac{\partial^2 V}{\partial x_2^2} \\ +\rho_1 \sigma(t, x_1, x_2)\frac{\partial^2 V}{\partial s \partial x_1} &+ \rho_2 \sigma(t, x_1, x_2)\frac{\partial^2 V}{\partial s \partial x_2} + \rho_{1,2}\frac{\partial^2 V}{\partial x_1 \partial x_2}\Bigg) = 0 \end{aligned} \tag{4.3}$$

where $\sigma(t, x_1, x_2)$ satisfies $\sigma^2(t, X_t^{(1)}, X_t^{(2)}) = \xi_t^t$ in Equation (4.1).

The equation groups including the boundary conditions of the vanilla and knock-in options are listed in Table 4.2. The value of the knock-out options can be easily got by the in-out parity in Equation (4.2). Additionally, each option value $V(s, t, x_1, x_2)$ is continuous during $0 \le t < T$.

If we solve the option values for $s, x_1, x_2 \in \mathbb{R}$ and $0 \le t \le T$, we also know $V(s, 0, 0, 0)$ as a result.

4.2.4 Goal of the chapter

Our goal is to solve the equations in Table 4.2 using neural networks directly. The option value $V(\boldsymbol{x})$ is treated as a function of not only the variables s, t, x_1, x_2, but also all the inputs of the model

$$\boldsymbol{x} = (s, t, x_1, x_2, T, B, r, q, \xi_0^t, \omega, k_1, k_2, \theta, \rho_1, \rho_2, \rho_{1,2}).$$

Throughout the paper, the strike K is kept fixed. The function $V(\boldsymbol{x})$ will be approximated by a well-trained neural network. Once the neural network is trained, its output is the option value, and the neural network is able to calculate option values given different parameter sets instantly. Also,

Option	Equations
vanilla call	$\begin{cases} H(C_{\text{v}}, s, t, x_1, x_2) = 0, & \forall s, x_1, x_2 \in \mathbb{R}, 0 < t < T \\ C_{\text{v}}(s, T, x_1, x_2) = (e^s - K)^+, & \forall s, x_1, x_2 \in \mathbb{R} \end{cases}$
vanilla put	$\begin{cases} H(P_{\text{v}}, s, t, x_1, x_2) = 0, & \forall s, x_1, x_2 \in \mathbb{R}, 0 < t < T \\ P_{\text{v}}(s, T, x_1, x_2) = (K - e^s)^+, & \forall s, x_1, x_2 \in \mathbb{R} \end{cases}$
up-and-in call	$\begin{cases} H(C_{\text{u-i}}, s, t, x_1, x_2) = 0, & \forall s < \ln(B), 0 < t < T, x_1, x_2 \in \mathbb{R} \\ C_{\text{u-i}}(s, T, x_1, x_2) = 0, & \forall s < \ln(B), x_1, x_2 \in \mathbb{R} \\ C_{\text{u-i}}(s, t, x_1, x_2) = C_{\text{v}}(s, t, x_1, x_2), & \forall s \geq \ln(B), 0 \leq t \leq T, x_1, x_2 \in \mathbb{R} \end{cases}$
down-and-in call	$\begin{cases} H(C_{\text{d-i}}, s, t, x_1, x_2) = 0, & \forall s > \ln(B), 0 < t < T, x_1, x_2 \in \mathbb{R} \\ C_{\text{d-i}}(s, T, x_1, x_2) = 0, & \forall s > \ln(B), x_1, x_2 \in \mathbb{R} \\ C_{\text{d-i}}(s, t, x_1, x_2) = C_{\text{v}}(s, t, x_1, x_2), & \forall s \leq \ln(B), 0 \leq t \leq T, x_1, x_2 \in \mathbb{R} \end{cases}$
up-and-in put	$\begin{cases} H(P_{\text{u-i}}, s, t, x_1, x_2) = 0, & \forall s < \ln(B), 0 < t < T, x_1, x_2 \in \mathbb{R} \\ P_{\text{u-i}}(s, T, x_1, x_2) = 0, & \forall s < \ln(B), x_1, x_2 \in \mathbb{R} \\ P_{\text{u-i}}(s, t, x_1, x_2) = P_{\text{v}}(s, t, x_1, x_2), & \forall s \geq \ln(B), 0 \leq t \leq T, x_1, x_2 \in \mathbb{R} \end{cases}$
down-and-in put	$\begin{cases} H(P_{\text{d-i}}, s, t, x_1, x_2) = 0, & \forall s > \ln(B), 0 < t < T, x_1, x_2 \in \mathbb{R} \\ P_{\text{d-i}}(s, T, x_1, x_2) = 0, & \forall s > \ln(B), x_1, x_2 \in \mathbb{R} \\ P_{\text{d-i}}(s, t, x_1, x_2) = P_{\text{v}}(s, t, x_1, x_2), & \forall s \leq \ln(B), 0 \leq t \leq T, x_1, x_2 \in \mathbb{R} \end{cases}$

Table 4.2: Equations of vanilla/barrier calls/puts.

no labels of option values from other pricing methods are needed during the training process, so the proposed method is an unsupervised deep learning approach.

4.3 Roadmap

4.3.1 Smooth neural network

The smooth neural networks have been already used to solve PDEs in literature. In [76, 73, 93, 97], the neural network is a function of the space and time variables, while in [37], it is a function of both variables and parameters. The loss of squared residuals of the PDE as well as some boundary conditions is minimized such that the neural network satisfies the equation group.

The building block of the neural networks in this chapter is the multi-layer perceptron (MLP). Here we give a quick introduction of the smooth MLP. An MLP is a multi-dimensional function with an input $\boldsymbol{x} \in \mathbb{R}^{n_0}$ and an output $V(\boldsymbol{x}) \in \mathbb{R}$, where n_0 is the length of the input. An MLP with

L hidden layers can be constructed by the equations

$$\begin{aligned}
\boldsymbol{x}^{(0)} &= \boldsymbol{x}, \\
\boldsymbol{x}^{(j)} &= g(\boldsymbol{W}^{(j-1)}\boldsymbol{x}^{(j-1)} + \boldsymbol{b}^{(j-1)}), \ \forall 1 \le j \le L, \\
V(\boldsymbol{x}) &= \boldsymbol{W}^{(L)}\boldsymbol{x}^{(L)} + b^{(L)},
\end{aligned}$$

where the hidden layers are $\boldsymbol{x}^{(j)} \in \mathbb{R}^n, \forall 1 \le j \le L$ and the parameters are $\boldsymbol{W}^{(0)} \in \mathbb{R}^{n \times n_0}$, $\boldsymbol{W}^{(j)} \in \mathbb{R}^{n \times n}$ for $1 \le j \le L-1$, $\boldsymbol{b}^{(j)} \in \mathbb{R}^n$ for $0 \le j \le L-1$, $\boldsymbol{W}^{(L)} \in \mathbb{R}^{1 \times n}$ and $b^{(L)} \in \mathbb{R}$. g is the non-linear activation function which is applied element-wise. There are some examples of smooth activation functions in Table 4.3. We are going to use SiLU [31] as the activation function in the neural network since it is empirically shown that it outperforms the other smooth activation functions. Nonetheless, the sigmoid function and the softplus [30] function also play important roles in the neural network, which will be covered in the following sections.

Function	Definition
sigmoid	$1/(1+e^{-z})$
SiLU	$z/(1+e^{-z})$
softplus	$\ln(1+e^{z})$

Table 4.3: Examples of smooth activation functions

4.3.2 Singular terms

The largest challenge to apply the smooth neural network approach to the barrier options is that their final payoffs at $(s,t) = (\ln(B), T)$ are not continuous. At first glance, we might be able to use the Heaviside function $g(z) = \mathbf{1}_{\{z>0\}}$ as the activation in the neural network to approximate the discontinuous payoffs. However, the option surface is continuous any time prior to maturity, i.e., for any $t < T$, making the Heaviside function impossible to be used in the neural network. What makes it more challenging is that the solution is discontinuous at one point but continuous anywhere else.

Actually, the discontinuity point $(s,t) = (\ln(B), T)$ is not the only special point. In vanilla options and some barrier options, the point $(s,t) = (\ln(K), T)$ is also a singular point, since their final payoffs are not smooth at this point. A traditional smooth neural network cannot fit well around this point. In [37], a special structure called singular term is used to deal with the non-smoothness around $(s,t) = (\ln(K), T)$.

A singular term is a pre-defined function with specific non-smoothness. It is non-smooth (or discontinuous) at maturity but smooth (or continuous) before maturity and that is exactly what we need. Also, they are able to mimic the asymptotic behaviors around the singular point. The input of the singular term consists of trainable components such that the singular term is able to fit the option surface under different parameters. In this chapter, we are going to follow the idea of singular terms and propose two singular terms for the two singular points on the option surface $(s,t) = (\ln(K), T)$ and $(s,t) = (\ln(B), T)$, such that we extend the smooth neural network approach to the barrier options.

4.3.3 Framework for both vanilla and barrier options

We are going to explain how to solve the eight barrier options in a single framework. Take the up-and-out call as an example. Its payoff and the option values prior to maturity are illustrated in Figure 4.1. The option surface of the up-and-out call contains two singular points. So the neural network solution to the up-and-out call needs to contain two singular terms. We have to admit that training the singular term at the barrier level is more challenging than training the one at the strike, since the option surface is continuous at the strike but discontinuous at the barrier level. Thus it is better not to train the two singular terms at the same time. Fortunately, the option surface of all the knock-in options contains just one singular point. We can solve the knock-in options and then the knock-out options are solved by the in-out parity in Equation (4.2) if we also solve the vanilla options.

In the pricing framework, we use six networks to model two vanilla options and eight barrier options: two networks for the vanilla call and put, and four networks for the four knock-in options. We first train the neural networks for vanilla options and then train the networks for knock-in

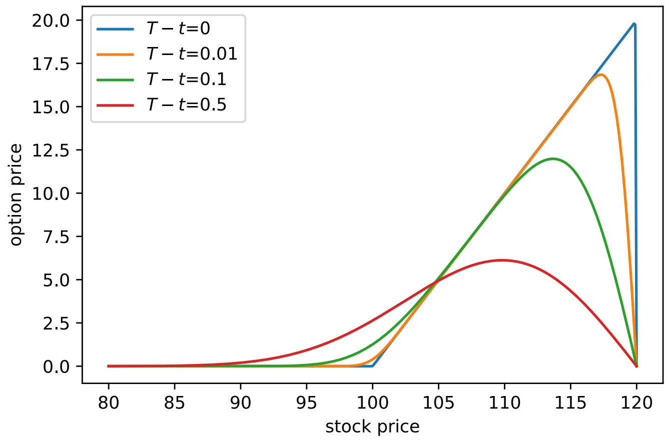


Figure 4.1: Example curves of the up-and-out call when $K = 100, B = 120, T = 0.5, r = q = 0$ and $\xi_0^t = 0.01$.

options with the help of the vanilla options. Then each knock-out option is the difference of the corresponding vanilla option and the corresponding knock-in option. Since the up-and-out call and up-and-in call degenerate to 0 and the vanilla call when $B < K$ and the down-and-out put and down-and-in put degenerate to 0 and the vanilla put when $B > K$, we only solve the barrier options in the region where they are non-degenerate. Although we can even use one network for either the vanilla call or put and use the put-call parity to get the other one, we still train them separately using two neural networks.

4.4 Vanilla options

4.4.1 Singular term for vanilla options

The option surface of vanilla options is smooth when $t < T$, but not at $(s,t) = (\ln(K), T)$. In Figure 4.2, we show the call option curve becomes more like a hockey stick at $S = e^s = K$ when t converges to T. The singular term for vanilla options deals with the singularity around $(s,t) = (\ln(K), T)$. It is modified from the Black-Scholes (BS) formula in Appendix C.2, and it is

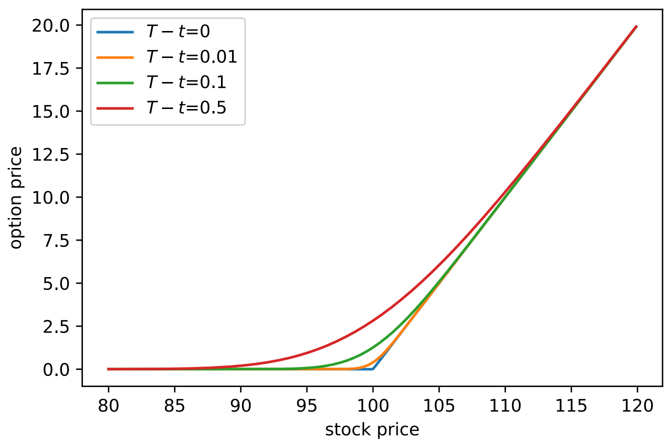


Figure 4.2: Example curves of the vanilla call when $K = 100, T = 0.5, r = q = 0$ and $\xi_0^t = 0.01$.

written as follows:

$$
\begin{aligned}
\alpha_{\mathrm{v}}(\boldsymbol{x}) &= \eta\, e^{s-q(T-t)} N(\eta(h(\boldsymbol{x})/v(\boldsymbol{x}) + v(\boldsymbol{x})/2)) \\
&\qquad - \eta\, K e^{-r(T-t)} N(\eta(h(\boldsymbol{x})/v(\boldsymbol{x}) - v(\boldsymbol{x})/2)), \\
h(\boldsymbol{x}) &= s - \ln(K) + \beta(\boldsymbol{x})(T-t), \\
v(\boldsymbol{x}) &= \gamma(\boldsymbol{x})\sqrt{T-t},
\end{aligned}
\tag{4.4}
$$

where $\beta(\boldsymbol{x})$ and $\gamma(\boldsymbol{x}) > 0$ are both MLPs with an input of $\boldsymbol{x}$. The notation

$$
\eta = \begin{cases} +1, & \text{for vanilla and barrier calls} \\ -1, & \text{for vanilla and barrier puts} \end{cases}
$$

changes the sign according to the option type and will be kept the same hereafter. The function $N(\cdot)$ is the normal CDF and is approximated by

$$
N(z) = \operatorname{sigmoid}\left(2\sqrt{2/\pi}(z + 0.044715z^3)\right) \tag{4.5}
$$

in neural networks according to [91]. Comparing the definition of $\alpha_{\mathrm{v}}(\boldsymbol{x})$ and the BS formula of vanilla options in Appendix C.2, we can find that $r - q$ and σ in the BS formula are replaced with

$\beta(\boldsymbol{x})$ and $\gamma(\boldsymbol{x})$ in $\alpha_{\mathrm{V}}(\boldsymbol{x})$. The singular term $\alpha_{\mathrm{V}}(\boldsymbol{x})$ satisfies the initial condition of vanilla options

$$\lim_{t \to T^-} \alpha_{\mathrm{V}}(\boldsymbol{x}) = (\eta(e^s - K))^+.$$

In [37], a similar singular term is proposed as

$$\tilde{\alpha}_{\mathrm{V}}(\boldsymbol{x}) = \mathrm{softplus}\left(\frac{e^{s-q(T-t)} - Ke^{-r(T-t)} + \beta(\boldsymbol{x})(T-t)}{\eta\,\gamma(\boldsymbol{x})\sqrt{T-t}}\right)\gamma(\boldsymbol{x})\sqrt{T-t}$$

which is also inspired by the BS formula. The argument inside the softplus function is similar to $h(\boldsymbol{x})/v(\boldsymbol{x})$. The singular term $\tilde{\alpha}_{\mathrm{V}}(\boldsymbol{x})$ is simpler and also satisfies the initial condition of vanilla options

$$\lim_{t \to T^-} \tilde{\alpha}_{\mathrm{V}}(\boldsymbol{x}) = (\eta(e^s - K))^+.$$

In the Bergomi model, the instant volatility is an exponential function (see Equation (4.1)) and can be very large. The term $\gamma(\boldsymbol{x})$ plays the role of volatility and tends to infinity in some cases. The singular term $\alpha_{\mathrm{V}}(\boldsymbol{x})$ gives the proper limit

$$\lim_{\gamma(\boldsymbol{x}) \to \infty} \alpha_{\mathrm{V}}(\boldsymbol{x}) = \begin{cases} e^{s-q(T-t)}, & \text{for vanilla calls} \\ Ke^{-r(T-t)}, & \text{for vanilla puts} \end{cases}$$

in this case but the singular term $\tilde{\alpha}_{\mathrm{V}}(\boldsymbol{x})$ does not give a proper limit since

$$\lim_{\gamma(\boldsymbol{x}) \to \infty} \tilde{\alpha}_{\mathrm{V}}(\boldsymbol{x}) = \infty.$$

So the singular term $\alpha_{\mathrm{V}}(\boldsymbol{x})$ is preferred for the Bergomi model.

4.4.2 Dimension reduction

The Bergomi model is a high-dimensional model not only due to the number of parameters, but also because the model input ξ_0^t is a function. Since the input of the neural network needs to be a vector, we need to consider a family of functions that can be parametrized in a finite-dimensional space, such as step functions or linear functions given fixed nodes. However, the dimension could still be so high such that $\gamma(\boldsymbol{x})$ in Equation (4.4) needs to learn a very complex volatility surface. Thus we calculate the average of ξ_0^t to be

$$\bar{\sigma}_t^T = \sqrt{\frac{1}{T-t}\int_t^T \xi_0^{\bar{t}}\,\mathrm{d}\bar{t}}$$

and replace the definition of $v(\boldsymbol{x})$ in Equation (4.4) with

$$v(\boldsymbol{x}) = \gamma(\boldsymbol{x})\bar{\sigma}_t^T\sqrt{T-t}.$$

The average of ξ_0^t lowers the difficulty for $\gamma(\boldsymbol{x})$ to learn the volatility surface.

4.4.3 Network structure

After we introduce the singular term for vanilla options, we give the full expression of the neural network for vanilla options as follows:

$$
\begin{aligned}
\boldsymbol{x}^{(0)} &= \boldsymbol{x}, \\
\boldsymbol{x}^{(j)} &= g(\boldsymbol{W}_{\mathrm{v}}^{(j-1)}\boldsymbol{x}^{(j-1)} + \boldsymbol{b}_{\mathrm{v}}^{(j-1)}),\ \forall 1 \le i \le L, \\
\beta(\boldsymbol{x}) &= \boldsymbol{W}_{\mathrm{v}}^{(\beta)}\boldsymbol{x}^{(L)} + b_{\mathrm{v}}^{(\beta)}, \\
\gamma(\boldsymbol{x}) &= \mathrm{softplus}\left(\boldsymbol{W}_{\mathrm{v}}^{(\gamma)}\boldsymbol{x}^{(L)} + b_{\mathrm{v}}^{(\gamma)}\right), \\
h(\boldsymbol{x}) &= s - \ln(K) + \beta(\boldsymbol{x})(T-t), \\
v(\boldsymbol{x}) &= \gamma(\boldsymbol{x})\bar{\sigma}_t^T\sqrt{T-t}, \\
\alpha_{\mathrm{v}}(\boldsymbol{x}) &= \eta\, e^{s-q(T-t)}N(\eta(h(\boldsymbol{x})/v(\boldsymbol{x}) + v(\boldsymbol{x})/2)) \\
&\quad - \eta\, Ke^{-r(T-t)}N(\eta(h(\boldsymbol{x})/v(\boldsymbol{x}) - v(\boldsymbol{x})/2)), \\
m(\boldsymbol{x}) &= \sum_{j=0}^{L}\boldsymbol{W}_{\mathrm{v}}^{(j,V)}\boldsymbol{x}^{(j)} + b_{\mathrm{v}}^{(V)}, \\
V(\boldsymbol{x}) &= m(\boldsymbol{x}) + \alpha_{\mathrm{v}}(\boldsymbol{x}),
\end{aligned}
\tag{4.6}
$$

where the input layer is $\boldsymbol{x} \in \mathbb{R}^{n_0}$ and the hidden layers are $\boldsymbol{x}^{(j)} \in \mathbb{R}^{n}, \forall 1 \le j \le L$. $\gamma(\boldsymbol{x})$ is passed through the softplus function to ensure the positivity since it describes the volatility. The singular term $\alpha_{\mathrm{v}}(\boldsymbol{x})$ is built from the last hidden layer $\boldsymbol{x}^{(L)}$ and then added to the output. The smooth term $m(\boldsymbol{x})$ has skip connections from all the previous layers $\{\boldsymbol{x}^{(j)}\}_{j=0}^{L}$, which stabilize the training process. The output $V(\boldsymbol{x})$ is a sum of the singular term and the smooth term. The dimensions of the neural network parameters are

$$\begin{aligned}
\boldsymbol{W}_{\mathrm{v}}^{(0)} &\in \mathbb{R}^{n\times n_0},\\
\boldsymbol{W}_{\mathrm{v}}^{(j)} &\in \mathbb{R}^{n\times n}, \forall\, 1 \le j \le L-1,\\
\boldsymbol{W}_{\mathrm{v}}^{(j)} &\in \mathbb{R}^{1\times n}, \forall\, j = \beta, \gamma,\\
\boldsymbol{W}_{\mathrm{v}}^{(0,V)} &\in \mathbb{R}^{1\times n_0},\\
\boldsymbol{W}_{\mathrm{v}}^{(j,V)} &\in \mathbb{R}^{1\times n}, \forall\, 1 \le j \le L,\\
\boldsymbol{b}_{\mathrm{v}}^{(j)} &\in \mathbb{R}^{n}, \forall 0 \le j \le L-1,\\
b_{\mathrm{v}}^{(j)} &\in \mathbb{R}, \forall\, j = \beta, \gamma, V.
\end{aligned}$$

The overall structure is an MLP with L layers of width n, and with a singular term added to the output. A graph of the neural network with $L = 2$ is illustrated in Figure 4.3 if we omit the skip connections.

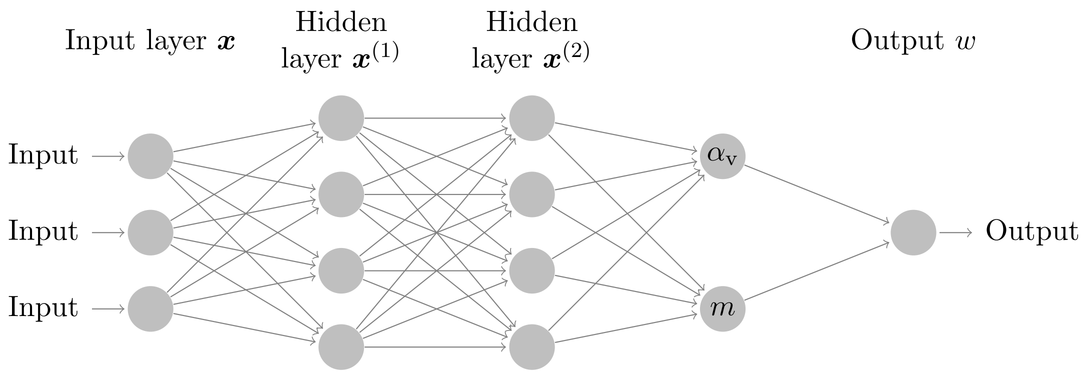

Input layer $\boldsymbol{x}$
Hidden layer $\boldsymbol{x}^{(1)}$
Hidden layer $\boldsymbol{x}^{(2)}$
Output w
Input
Input
Input
α_{v}
m
Output

Figure 4.3: Illustration of the neural network for vanilla options, where α_{v} is the singular term and m is the smooth term.

4.4.4 Boundary conditions of volatility

Since the PDE contains derivatives w.r.t. the volatility factors x_1 and x_2, we need to add boundary conditions for them. We need to anchor the solution on the boundary of x_1 and x_2, otherwise the solution would be far from the true value on the boundary and the solution in the interior would also be inaccurate even if the PDE is satisfied in the interior.

For the Heston model [49], which is also a stochastic volatility model, the dynamics of the stock price and volatility are

$$\begin{aligned}\mathrm{d}S_t &= rS_t\mathrm{d}t + S\sqrt{V_t}\mathrm{d}W_t^{(S)},\\ \mathrm{d}V_t &= k(\theta - V_t)\mathrm{d}t + \sigma\sqrt{V_t}\mathrm{d}W_t^{(V)},\end{aligned}$$

where V_t is the variance process, k, θ and σ are positive constants and $W_t^{(S)}$ and $W_t^{(V)}$ are correlated Brownian motions. Let $C_{\mathrm{H}}(S, v, t)$ be the value of vanilla calls with stock price S and instant volatility $\sqrt{V_t}$ at time t in the Heston model. The theoretical boundary conditions for vanilla calls at $V_t = 0$ and $V_t = \infty$ proposed in [49] are

$$rS\frac{\partial C_{\mathrm{H}}}{\partial S}(S, 0, t) + \kappa\theta\frac{\partial C_{\mathrm{H}}}{\partial v}(S, 0, t) - rC_{\mathrm{H}}(S, 0, t) + \frac{\partial C_{\mathrm{H}}}{\partial t}(S, 0, t) = 0$$

and

$$C_{\mathrm{H}}(S, \infty, t) = S.$$

However, this kind of boundary conditions does not work well in practice for the neural network approach. Although we know a vanilla call with infinity volatility converges to the stock price S, it is hard to know how large could be considered as ‘infinity’ in the numerical routine. In the Bergomi model, we can get similar results for $x_1 = \pm\infty$ and $x_2 = \pm\infty$, but the question remains how large ‘infinity’ is.

We need a better estimate of $V(\boldsymbol{x})$ when x_1 and x_2 are far from 0. Recall that $X_t^{(1)}$ is defined by

$$\mathrm{d}X_t^{(1)} = -k_1X_t^{(1)}\mathrm{d}t + \mathrm{d}W_t^{(1)}.$$

If $X_t^{(1)} = x_1$ and x_1 is far from 0, the drift term $-k_1 X_t^{(1)} \mathrm{d}t$ dominates in the dynamic. We consider

$$\mathrm{d}\tilde{X}_t^{(1)} = -k_1 \tilde{X}_t^{(1)} \mathrm{d}t$$

and

$$\tilde{X}_u^{(1)} = x_1 e^{-k_1(u-t)}, \forall u \geq t$$

is a deterministic function. We also let

$$\tilde{X}_u^{(2)} = x_2 e^{-k_2(u-t)}, \forall u \geq t.$$

If we replace $X_u^{(1)}$ and $X_u^{(2)}$ with $\tilde{X}_u^{(1)}$ and $\tilde{X}_u^{(2)}$, then $\xi_u^u, \forall u \geq t$ becomes a deterministic function according to its definition in Equation (4.1), and the Bergomi model degenerates to the BMS model, where the volatility rate is

$$\sqrt{\frac{1}{T-t} \int_t^T \xi_u^u \mathrm{d}u}.$$

In this way, we do not require x_1 and x_2 to be infinity in the boundary condition. They are required to be far from 0 such that we can omit the drift terms in the dynamics of the OU processes. A suitable choice of x_1 and x_2 could be the quantiles of the limiting distribution of the OU processes, as we do in Section 4.6.2.

We estimate vanilla options under the Bergomi model when x_1 and x_2 are far from 0 by vanilla options under the BMS model. Although we have to admit that there are still some errors in the estimation since we do not consider the correlation ρ_1, ρ_2 and $\rho_{1,2}$, it is much better than the boundary condition of $C_{\mathrm{H}}(S, \infty, t) = S$ for numerical use.

4.4.5 Loss function

In this part we still write the option value $V(\boldsymbol{x})$ as $V(s, t, x_1, x_2)$ to emphasize the different variables in the boundary conditions. The other parameters are omitted in notations since they will be kept the same in the boundary conditions, but we still need to keep in mind that V is a function of $\boldsymbol{x}$. Let $\tilde{V}(s, t, x_1, x_2)$ be the estimate by the BMS model in Section 4.4.4 when we replace $X_u^{(1)}$ and $X_u^{(2)}$ with $\tilde{X}_u^{(1)}$ and $\tilde{X}_u^{(2)}$. Let s_{m} and s_{M} be the lower and upper boundaries for the variable s. Let $x_{j,\mathrm{m}}(\boldsymbol{x})$ and $x_{j,\mathrm{M}}(\boldsymbol{x})$ be the lower and upper boundaries for the variable x_j, $\forall j = 1, 2$. Note that s_{m} and s_{M} are constants while $x_{j,\mathrm{m}}(\boldsymbol{x})$ and $x_{j,\mathrm{M}}(\boldsymbol{x})$ are functions depending on the other parameters in $\boldsymbol{x}$. In Table 4.4, we list the boundary conditions for vanilla calls and puts. We do not calculate boundary conditions for x_1 and x_2 separately, since we need both x_1 and x_2 to be far from 0 so that we can use the estimate $\tilde{V}(s, t, x_1, x_2)$ as the boundary condition.

Boundary value	Vanilla call	Vanilla put
$V(s, T, x_1, x_2)$	$(e^s - K)^+$	$(K - e^s)^+$
$V(s_{\mathrm{m}}, t, x_1, x_2)$	0	$Ke^{-r(T-t)} - e^{s_{\mathrm{m}}-q(T-t)}$
$V(s_{\mathrm{M}}, t, x_1, x_2)$	$e^{s_{\mathrm{M}}-q(T-t)} - Ke^{-r(T-t)}$	0
$V(s, t, x_{1,\mathrm{m}}(\boldsymbol{x}), x_{2,\mathrm{m}}(\boldsymbol{x}))$	$\tilde{V}(s, t, x_{1,\mathrm{m}}(\boldsymbol{x}), x_{2,\mathrm{m}}(\boldsymbol{x}))$	$\tilde{V}(s, t, x_{1,\mathrm{m}}(\boldsymbol{x}), x_{2,\mathrm{m}}(\boldsymbol{x}))$
$V(s, t, x_{1,\mathrm{M}}(\boldsymbol{x}), x_{2,\mathrm{M}}(\boldsymbol{x}))$	$\tilde{V}(s, t, x_{1,\mathrm{M}}(\boldsymbol{x}), x_{2,\mathrm{M}}(\boldsymbol{x}))$	$\tilde{V}(s, t, x_{1,\mathrm{M}}(\boldsymbol{x}), x_{2,\mathrm{M}}(\boldsymbol{x}))$

Table 4.4: Boundary conditions of vanilla calls and puts.

Let $V(\boldsymbol{x})$ be the neural network defined in Equation (4.6) with parameters

$$\mathcal{W}_{\mathrm{v}} = \left\{ \begin{array}{l} \boldsymbol{W}_{\mathrm{v}}^{(j)}, \boldsymbol{b}_{\mathrm{v}}^{(j)}, \forall 0 \leq j \leq L-1 \text{ or } j = \beta, \gamma \\ \boldsymbol{W}_{\mathrm{v}}^{(j,V)}, \forall 0 \leq j \leq L \text{ and } b_{\mathrm{v}}^{(V)} \end{array} \right\}. \tag{4.7}$$

Given a sample $\boldsymbol{x}$, the loss function for vanilla puts is defined as

$$\begin{aligned} L_{P_{\mathrm{v}}}(\mathcal{W}_{\mathrm{v}}; \boldsymbol{x}) = & (H(V, s, t, x_1, x_2))^2 + \left(V(s, T, x_1, x_2) - (K - e^s)^+\right)^2 \\ & + \left(V(s_{\mathrm{m}}, t, x_1, x_2) - Ke^{-r(T-t)} - e^{s_{\mathrm{m}}-q(T-t)}\right)^2 + (V(s_{\mathrm{M}}, t, x_1, x_2))^2 \\ & + \lambda_1 \left(V(s, t, x_{1,\mathrm{m}}(\boldsymbol{x}), x_{2,\mathrm{m}}(\boldsymbol{x})) - \tilde{V}(s, t, x_{1,\mathrm{m}}(\boldsymbol{x}), x_{2,\mathrm{m}}(\boldsymbol{x}))\right)^2 \\ & + \lambda_1 \left(V(s, t, x_{1,\mathrm{M}}(\boldsymbol{x}), x_{2,\mathrm{M}}(\boldsymbol{x})) - \tilde{V}(s, t, x_{1,\mathrm{M}}(\boldsymbol{x}), x_{2,\mathrm{M}}(\boldsymbol{x}))\right)^2 \end{aligned}$$

where $H(V, s, t, x_1, x_2)$ is defined in Equation (4.3). λ_1 is constant with the default value $\lambda_1 = 0.01$, which means we allow some errors in the boundary conditions for x_1 and x_2.

The loss for vanilla calls is a little different since we need to compensate for the large values and derivatives when s is near the upper boundary s_{M} such that they will not dominate the loss function, which is also used in [37]. The weight is defined as

$$\phi(s) = \min(1, 4K^2 \exp(-2s))$$

since the values and derivatives of vanilla calls grow at the rate of $\exp(s)$. Then the loss function for vanilla calls is defined as

$$\begin{aligned}
L_{C_\mathrm{v}}(\mathcal{W}_\mathrm{v}; \boldsymbol{x}) = {} & \phi(s)\left(H(V, s, t, x_1, x_2)\right)^2 + \phi(s)\left(V(s, T, x_1, x_2) - (e^s - K)^+\right)^2 \\
& + \left(V(s_\mathrm{m}, t, x_1, x_2)\right)^2 + \phi(s_\mathrm{M})\left(V(s_\mathrm{M}, t, x_1, x_2) - e^{s_\mathrm{M} - q(T-t)} - Ke^{-r(T-t)}\right)^2 \\
& + \lambda_1 \phi(s)\left(V(s, t, x_{1,\mathrm{m}}(\boldsymbol{x}), x_{2,\mathrm{m}}(\boldsymbol{x})) - \tilde{V}(s, t, x_{1,\mathrm{m}}(\boldsymbol{x}), x_{2,\mathrm{m}}(\boldsymbol{x}))\right)^2 \\
& + \lambda_1 \phi(s)\left(V(s, t, x_{1,\mathrm{M}}(\boldsymbol{x}), x_{2,\mathrm{M}}(\boldsymbol{x})) - \tilde{V}(s, t, x_{1,\mathrm{M}}(\boldsymbol{x}), x_{2,\mathrm{M}}(\boldsymbol{x}))\right)^2 .
\end{aligned}$$

The losses L_{C_v} and L_{P_v} are defined on a single sample $\boldsymbol{x}$. Given multiple samples $\boldsymbol{x}_{(1)}, \boldsymbol{x}_{(2)}, \ldots, \boldsymbol{x}_{(n)}$, the total loss is an average of the individual losses, i.e.,

$$L_{\mathrm{v,avg}}\left(\mathcal{W}_\mathrm{v}; \{\boldsymbol{x}_{(j)}\}_{j=1}^n\right) = \begin{cases} \frac{1}{n}\sum_{j=1}^n L_{P_\mathrm{v}}(\mathcal{W}_\mathrm{v}; \boldsymbol{x}_{(j)}), & \text{for vanilla puts,} \\ \frac{1}{n}\sum_{j=1}^n L_{C_\mathrm{v}}(\mathcal{W}_\mathrm{v}; \boldsymbol{x}_{(j)}), & \text{for vanilla calls.} \end{cases}$$

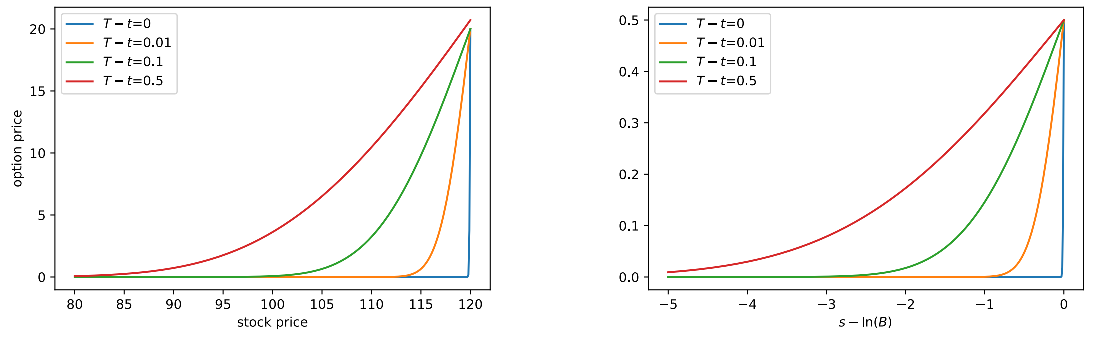
We minimize the loss function w.r.t. $\mathcal{W}_\mathrm{v}$ such that the network $V(\boldsymbol{x})$ approximates the true value of vanilla options.

4.5 Barrier options

4.5.1 Singular term for barrier options

The singular term for vanilla options is to deal with the singularity around $(s, t) = (\ln(K), T)$. Although the option surface is not smooth around $(s, t) = (\ln(K), T)$, it is continuous. A smooth neural network without the singular term is still able to fit the entire vanilla option surface with small errors, except that it cannot completely meet the initial condition. The singular term is an improvement of the neural network but not a requirement.

However, the case is different for the barrier options. In Figure 4.4 (a), we show the curves of the up-and-in call. As t approaches T, the curve becomes more and more vertical near $S = e^s = B$. The option surface is not continuous at $(s,t) = (\ln(B), T)$ and cannot be fitted by a continuous smooth neural network. The optimization routine would fail since the boundary conditions cannot be fitted. Thus it is necessary to add the singular term for barrier options to the smooth neural



(a) (b)

Figure 4.4: (a) Example curves of the up-and-in call when $K = 100, B = 120, T = 0.5, r = q = 0$ and $\xi_0^t = 0.04$. (b) Example curves of the singular term $F_1(\beta(\boldsymbol{x}), \gamma(\boldsymbol{x}), \boldsymbol{x})$ when $r - q + \beta(\boldsymbol{x}) = 0$ and $\gamma(\boldsymbol{x})\bar{\sigma}_t^T = 3$.

network to overcome this problem:

$$\begin{aligned}
\alpha_{\mathrm{b}}(\boldsymbol{x}) &= F_1(\beta(\boldsymbol{x}), \gamma(\boldsymbol{x}), \boldsymbol{x}) \\
&= N(\zeta\, h_B(\boldsymbol{x})/v(\boldsymbol{x})), \\
h_B(\boldsymbol{x}) &= s - \ln(B) + (r - q + \beta(\boldsymbol{x}))(T - t), \\
v(\boldsymbol{x}) &= \gamma(\boldsymbol{x})\bar{\sigma}_t^T \sqrt{T - t},
\end{aligned}$$

where $\beta(\boldsymbol{x})$ and $\gamma(\boldsymbol{x}) > 0$ are MLPs with an input of $\boldsymbol{x}$. The notation ζ is defined as

$$\zeta = \begin{cases} +1, & \text{for up-and-in options,} \\ -1, & \text{for down-and-in options.} \end{cases}$$

The normal CDF is approximated by Equation (4.5). The singular term is designed such that

$$\lim_{t\to T^-} F_1(\beta(\boldsymbol{x}), \gamma(\boldsymbol{x}), \boldsymbol{x}) = \mathbf{1}_{\{\zeta(s-\ln(B))>0\}}.$$

It is a Heaviside function at maturity but is smooth before maturity. In Figure 4.4 (b), the singular term is similar to the option curves in the region $s < \ln(B)$ when t converges to T.

The singular term $F_1(\beta(\boldsymbol{x}), \gamma(\boldsymbol{x}), \boldsymbol{x})$ is able to replicate the discontinuity around $(s, t) = (\ln(B), T)$. However, we need to pay special attention to the cases of up-and-in puts and down-and-in calls since their curves are not necessarily monotone w.r.t. the stock price. This is more obvious when volatility is small and the difference between r and q is large. In Figure 4.5, we show the curves of the up-and-in put. The curves of up-and-in puts are increasing when $r \leq q$, while the curve is not monotone when r is much larger than q. This phenomenon increases the difficulty of fitting at longer maturities since the singular term $F_1(\beta(\boldsymbol{x}), \gamma(\boldsymbol{x}), \boldsymbol{x})$ is always monotone. Once again, we

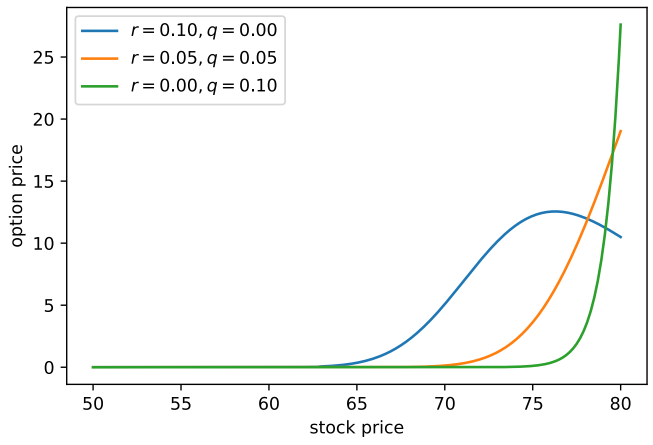


Figure 4.5: Example curves of the up-and-in put when $K = 100, B = 80, T = 1, t = 0$ and $\xi_0^t = 0.0025$.

think of the BS formula for the barrier options, which is summarized in Appendix C.2, and propose the following singular term for up-and-in puts and down-and-in calls:

$$\begin{aligned}
\alpha_{\mathrm{b}}(\boldsymbol{x}) &= F_2(\beta(\boldsymbol{x}), \gamma(\boldsymbol{x}), \boldsymbol{x}) \\
&= F_{2,1}(\beta(\boldsymbol{x}), \gamma(\boldsymbol{x}), \boldsymbol{x}) \\
&\quad + F_{2,2}(\beta(\boldsymbol{x}), \gamma(\boldsymbol{x}), \boldsymbol{x}) \exp((s - \ln(B))(1 - 2(r - q)/(\bar{\sigma}_t^T)^2))
\end{aligned}$$

where the two components are

$$F_{2,1}(\beta(\boldsymbol{x}), \gamma(\boldsymbol{x}), \boldsymbol{x}) = \begin{cases} \eta\, e^{s-q(T-t)} N(\eta(h_K(\boldsymbol{x})/v(\boldsymbol{x}) + v(\boldsymbol{x})/2)) & \\ -\eta\, K e^{-r(T-t)} N(\eta(h_K(\boldsymbol{x})/v(\boldsymbol{x}) - v(\boldsymbol{x})/2)) & \\ -\eta\, e^{s-q(T-t)} N(\eta(h_B(\boldsymbol{x})/v(\boldsymbol{x}) + v(\boldsymbol{x})/2)) & \text{if } \eta(K - B) < 0, \\ +\eta\, K e^{-r(T-t)} N(\eta(h_B(\boldsymbol{x})/v(\boldsymbol{x}) - v(\boldsymbol{x})/2)), & \\ 0, & \text{else,} \end{cases}$$

and

$$\begin{aligned}
F_{2,2}(\beta(\boldsymbol{x}), \gamma(\boldsymbol{x}), \boldsymbol{x}) = \eta\, B^2 e^{-s-q(T-t)} N(\eta(\tilde{h}(\boldsymbol{x})/v(\boldsymbol{x}) + v(\boldsymbol{x})/2)) \\
- \eta\, K e^{-r(T-t)} N(\eta(\tilde{h}(\boldsymbol{x})/v(\boldsymbol{x}) - v(\boldsymbol{x})/2)),
\end{aligned}$$

and the elements in the normal CDF are

$$\begin{aligned}
h_B(\boldsymbol{x}) &= s - \ln(B) + (r - q + \beta(\boldsymbol{x}))(T - t), \\
h_K(\boldsymbol{x}) &= s - \ln(K) + (r - q + \beta(\boldsymbol{x}))(T - t), \\
\tilde{h}(\boldsymbol{x}) &= \begin{cases} 2\ln(B) - s - \ln(K) + (r - q + \beta(\boldsymbol{x}))(T - t), & \text{if } \eta(K - B) \geq 0, \\ \ln(B) - s + (r - q + \beta(\boldsymbol{x}))(T - t), & \text{else,} \end{cases} \\
v(\boldsymbol{x}) &= \gamma(\boldsymbol{x}) \bar{\sigma}_t^T \sqrt{T - t}.
\end{aligned}$$

$\beta(x)$ and $\gamma(\boldsymbol{x}) > 0$ are still two MLPs. The singular term is actually modified from the BS formula for up-and-in puts and down-and-in calls. It is easy to see the singular term replaces $r - q$ and σ in the normal CDF with $r - q + \beta(x)$ and $\gamma(\boldsymbol{x})\bar{\sigma}_t^T$ respectively. While we should be able to modify the BS formula for up-and-in calls and down-and-in puts to get a singular term, $F_1(\beta(\boldsymbol{x}), \gamma(\boldsymbol{x}), \boldsymbol{x})$ is capable of this job and is beneficial for its simplicity and numerical stability.

4.5.2 Network structure

After introducing the singular term for barrier options, we define the neural networks for knock-in options as follows:

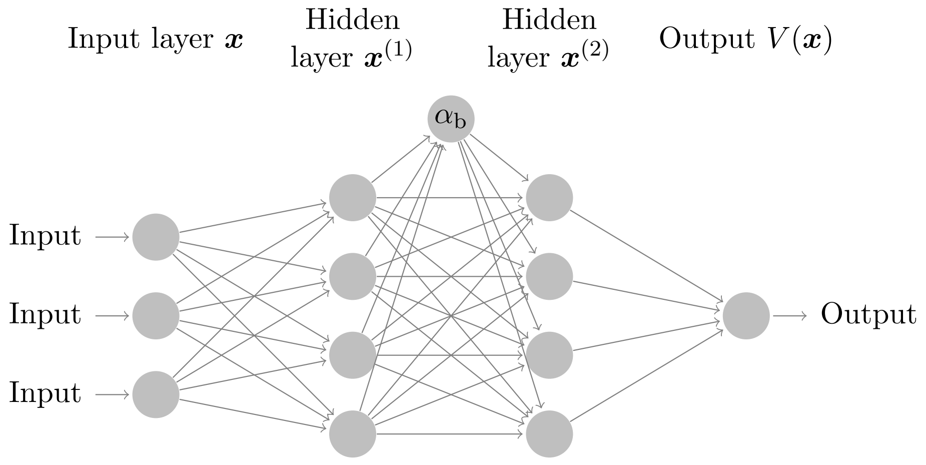
$$
\begin{aligned}
\boldsymbol{x}^{(0)} &= \boldsymbol{x}, \\
\boldsymbol{x}^{(j)} &= g(\boldsymbol{W}_{\mathrm{b}}^{(j-1)}\boldsymbol{x}^{(j-1)} + \boldsymbol{b}_{\mathrm{b}}^{(j-1)}),\ \forall 1 \le i \le L_1, \\
\beta(\boldsymbol{x}) &= \boldsymbol{W}_{\mathrm{b}}^{\beta}\boldsymbol{x}^{(L_1)} + b_{\mathrm{b}}^{\beta}, \\
\gamma(\boldsymbol{x}) &= \mathrm{softplus}\left(\boldsymbol{W}_{\mathrm{b}}^{\gamma}\boldsymbol{x}^{(L_1)} + b_{\mathrm{b}}^{\gamma}\right), \\
\alpha_{\mathrm{b}}(\boldsymbol{x}) &= \begin{cases} F_1(\beta(\boldsymbol{x}), \gamma(\boldsymbol{x}), \boldsymbol{x}), & \text{for up-and-in calls and down-and-in puts,} \\ F_2(\beta(\boldsymbol{x}), \gamma(\boldsymbol{x}), \boldsymbol{x}), & \text{for up-and-in puts and down-and-in calls,} \end{cases} \\
\tilde{\boldsymbol{x}}^{(L_1)} &= \mathrm{concatenate}(\boldsymbol{x}^{(L_1)}, \alpha_{\mathrm{b}}(\boldsymbol{x})), \\
\boldsymbol{x}^{(L_1+1)} &= g\left(\boldsymbol{W}_{\mathrm{b}}^{(L_1)}\tilde{\boldsymbol{x}}^{(L_1)} + \boldsymbol{b}_{\mathrm{b}}^{(L_1)}\right), \\
\boldsymbol{x}^{(j)} &= g\left(\boldsymbol{W}_{\mathrm{b}}^{(j-1)}\boldsymbol{x}^{(j-1)} + \boldsymbol{b}_{\mathrm{b}}^{(j-1)}\right),\ \forall L_1 + 1 < j \le L_1 + L_2, \\
V(\boldsymbol{x}) &= \boldsymbol{W}_{\mathrm{b}}^{(L_1+L_2)}\boldsymbol{x}^{(L_1+L_2)} + b_{\mathrm{b}}^{(L_1+L_2)},
\end{aligned}
\tag{4.8}
$$

where the input layer $\boldsymbol{x}$ is of size n_0 and the hidden layers are $\boldsymbol{x}^{(j)} \in \mathbb{R}^n, \forall 1 \le j \le L_1 + L_2$. The singular term $\alpha_{\mathrm{b}}(\boldsymbol{x})$ is built from the middle hidden layer $\boldsymbol{x}^{(L_1)}$ and then combined with $\boldsymbol{x}^{(L_1)}$ to be fed to the next hidden layer. The singular term has to be embedded in the middle since we need the neural network to figure out how to combine the singular term and the continuous part by itself.

The dimensions of the neural network parameters are

$$
\begin{aligned}
\boldsymbol{W}_{\mathrm{b}}^{(0)} &\in \mathbb{R}^{n\times n_0},\\
\boldsymbol{W}_{\mathrm{b}}^{(j)} &\in \mathbb{R}^{n\times n}, \forall\, 1 \le j \le L_1-1, L_1+1 \le j \le L_1+L_2-1,\\
\boldsymbol{W}_{\mathrm{b}}^{(j)} &\in \mathbb{R}^{1\times n}, \forall\, j = \beta, \gamma, L_1+L_2,\\
\boldsymbol{W}_{\mathrm{b}}^{(L_1)} &\in \mathbb{R}^{n\times(n+1)}\\
\boldsymbol{b}_{\mathrm{b}}^{(j)} &\in \mathbb{R}^{n}, \forall 0 \le j \le L_1+L_2-1,\\
b_{\mathrm{b}}^{(j)} &\in \mathbb{R}, \forall\, j = \beta, \gamma, L_1+L_2.
\end{aligned}
$$

The overall structure is an MLP with $L_1 + L_2$ layers of width n, and with a singular term embedded in the middle. A graph of the neural network with $L_1 = L_2 = 1$ is illustrated in Figure 4.6.



Input layer $\boldsymbol{x}$
Hidden layer $\boldsymbol{x}^{(1)}$
Hidden layer $\boldsymbol{x}^{(2)}$
Output $V(\boldsymbol{x})$
α_{b}
Input
Input
Input
Output

Figure 4.6: Illustration of the neural network for knock-in options, where α_{b} is the singular term.

4.5.3 Loss function

We now go over the boundary conditions and loss functions for barrier options. We only apply the boundary conditions for s at s_{m}, $\ln(B)$ or s_{M} and do not apply the boundary conditions for x_1 and x_2 for the following two reasons. First, the BMS model cannot serve as an estimate since the barrier options are path-dependent and the instant volatility in the Bergomi model changes fast. Second, the vanilla option value in the boundary conditions for s is already a reference of the barrier options when x_1 and x_2 are far from 0. The boundary conditions for knock-in options are listed in Table 4.5. In this part we still only use the arguments s, t, x_1, x_2 and omit the other parameters in notations.

Boundary condition	Initial	Lower	Middle	Upper
Options	$(t = T)$	$(s = s_{\mathrm{m}})$	$(s = \ln(B))$	$(s = s_{\mathrm{M}})$
up-and-in call	0	0	$C_{\mathrm{v}}(\ln(B), t, x_1, x_2)$	N/A
up-and-in put	0	0	$P_{\mathrm{v}}(\ln(B), t, x_1, x_2)$	N/A
down-and-in call	0	N/A	$C_{\mathrm{v}}(\ln(B), t, x_1, x_2)$	0
down-and-in put	0	N/A	$P_{\mathrm{v}}(\ln(B), t, x_1, x_2)$	0

Table 4.5: Boundary conditions of knock-in options. 'N/A' means the boundary condition is not applicable for certain barrier options since the boundary is on the other side of the barrier level compared with the stock price.

Let $V(\boldsymbol{x})$ be the neural network defined in Equation (4.8) with parameters

$$\mathcal{W}_{\mathrm{b}} = \left\{ \boldsymbol{W}_{\mathrm{b}}^{(j)}, \boldsymbol{b}_{\mathrm{b}}^{(j)}, \forall 0 \le j \le L_1 + L_2 \text{ or } j = \beta, \gamma \right\}.$$

Given a sample $\boldsymbol{x}$, the loss functions for up-and-in options are defined as

$$\begin{aligned} L_{C_{\text{u-i}}}(\mathcal{W}_{\mathrm{b}}; \boldsymbol{x}) &= (H(V, s, t, x_1, x_2))^2 + \lambda_2 \left(V(s, T, x_1, x_2)\right)^2 + \left(V(s_{\mathrm{m}}, t, x_1, x_2)\right)^2 \\ &\quad + \left(V(\ln(B), t, x_1, x_2) - C_{\mathrm{v}}(\ln(B), t, x_1, x_2)\right)^2, \\ L_{P_{\text{u-i}}}(\mathcal{W}_{\mathrm{b}}; \boldsymbol{x}) &= (H(V, s, t, x_1, x_2))^2 + \lambda_2 \left(V(s, T, x_1, x_2)\right)^2 + \left(V(s_{\mathrm{m}}, t, x_1, x_2)\right)^2 \\ &\quad + \left(V(\ln(B), t, x_1, x_2) - P_{\mathrm{v}}(\ln(B), t, x_1, x_2)\right)^2, \end{aligned}$$

where $H(V, s, t, x_1, x_2)$ is defined in Equation (4.3). λ_2 is a constant with the default value $\lambda_2 = 25$, which strengthens the initial boundary condition.

The loss functions for down-and-in options are defined as

$$\begin{aligned}
L_{C_{\text{d-i}}}(\mathcal{W}_{\text{b}};\boldsymbol{x}) &= (H(V,s,t,x_1,x_2))^2 + \lambda_2\,(V(s,T,x_1,x_2))^2 + (V(s_{\text{M}},t,x_1,x_2))^2 \\
&\quad + (V(\ln(B),t,x_1,x_2) - C_{\text{v}}(\ln(B),t,x_1,x_2))^2\,, \\
L_{P_{\text{d-i}}}(\mathcal{W}_{\text{b}};\boldsymbol{x}) &= (H(V,s,t,x_1,x_2))^2 + \lambda_2\,(V(s,T,x_1,x_2))^2 + (V(s_{\text{M}},t,x_1,x_2))^2 \\
&\quad + (V(\ln(B),t,x_1,x_2) - P_{\text{v}}(\ln(B),t,x_1,x_2))^2\,.
\end{aligned}$$

The losses are defined on a single sample $\boldsymbol{x}$. Given multiple samples $\boldsymbol{x}_{(1)},\boldsymbol{x}_{(2)},\ldots,\boldsymbol{x}_{(n)}$, the total loss is an average of the individual losses, i.e.,

$$L_{\text{b,avg}}\left(\mathcal{W}_{\text{b}};\{\boldsymbol{x}_{(j)}\}_{j=1}^{n}\right) = \begin{cases} \frac{1}{n}\sum_{j=1}^{n} L_{C_{\text{u-i}}}(\mathcal{W}_{\text{b}};\boldsymbol{x}_{(j)}), & \text{for up-and-in call,} \\ \frac{1}{n}\sum_{j=1}^{n} L_{P_{\text{u-i}}}(\mathcal{W}_{\text{b}};\boldsymbol{x}_{(j)}), & \text{for up-and-in put,} \\ \frac{1}{n}\sum_{j=1}^{n} L_{C_{\text{d-i}}}(\mathcal{W}_{\text{b}};\boldsymbol{x}_{(j)}), & \text{for down-and-in call,} \\ \frac{1}{n}\sum_{j=1}^{n} L_{P_{\text{d-i}}}(\mathcal{W}_{\text{b}};\boldsymbol{x}_{(j)}), & \text{for down-and-in put.} \end{cases}$$

We train the neural networks of vanilla options, get $C_{\text{v}}(\boldsymbol{x})$ and $P_{\text{v}}(\boldsymbol{x})$ and fix them before we train the networks of barrier options. After that, the loss function $L_{\text{b,avg}}$ is minimized only w.r.t. $\mathcal{W}_{\text{b}}$ and only the neural network of barrier options is trained.

4.6 Numerical experiments

4.6.1 Piecewise constant ξ_0^t

In the Bergomi model, the model input ξ_0^t is a function over $[0,T]$ and we consider the family of step functions

$$\xi_0^t = \xi_j,\ \text{if}\ \ t_{j-1} \le t \le t_j$$

given the nodes $0 = t_0 < t_1 < \cdots < t_m$, where $\{\xi_j\}_{j=1}^m$ are parameters. In the numerical experiments, we test the following two cases:

- The constant case $\xi_0^t = \xi, \forall 0 \le t \le 3$ as a baseline. In this case the network input is

$$\boldsymbol{x} = (s, t, x_1, x_2, T, B, r, q, \xi, \omega, k_1, k_2, \theta, \rho_1, \rho_2, \rho_{1,2}).$$

- The nine-segment case where $m = 9$ and

$$(t_j)_{j=1}^9 = (1/52, 1/26, 1/12, 1/6, 1/4, 1/2, 1, 2, 3).$$

  The nodes permit enough flexibility for options with both short and long time to maturities. In this case the network input is

$$\boldsymbol{x} = (s, t, x_1, x_2, T, B, r, q, \xi_1, \ldots, \xi_9, \omega, k_1, k_2, \theta, \rho_1, \rho_2, \rho_{1,2}).$$

  The input dimension is 24 and the neural network is employed to deal with the high-dimensional case.

4.6.2 Parameter range and sampling

Although the proposed method is unsupervised and does not need labels of prices generated from other pricing methods for training, we still need random samples for training. We also need option prices calculated from a benchmark method that are used to evaluate the results of neural networks after training. Here are the ranges of the parameters following the same constraints in both training and test samples.

$$\begin{aligned} K = 100, \quad & 0 \le T \le 3, \quad & 0.05^2 \le \xi_0^t \le 0.5^2, \\ 0 \le r, q \le 0.1, \quad & 0 \le \omega \le 3, \quad & 0 \le \theta \le 1, \\ 0.1 \le k_1 \le 4, \quad & 2 \le k_2 \le 12, \quad & -0.9 \le \rho_1, \rho_2 \le 0.2. \end{aligned}$$

We choose a feasible range for each parameter. For example, k_1 and k_2 are chosen such that $X_t^{(1)}$ and $X_t^{(2)}$ are long-time and short-time volatility factors. ρ_1 and ρ_2 are mostly negative since returns and volatilities are usually negatively correlated. ξ_0^t is similar to σ^2 in the BMS model and its range is chosen based on the scale of volatility. The parameter $\rho_{1,2}$ needs to satisfy the following constraints to ensure the positive semidefinite property of the covariance matrix of the correlated Brownian motions

$$\rho_1\rho_2 - \sqrt{(1-\rho_1^2)(1-\rho_2^2)} \leq \rho_{1,2} \leq \rho_1\rho_2 + \sqrt{(1-\rho_1^2)(1-\rho_2^2)}.$$

The variables t, x_1 and x_2 follow different constraints in training and test samples:

	Training range	Test range
t	$0 \leq t \leq T$	$t = 0$
x_1	$x_{1,\mathrm{m}}(\boldsymbol{x}) \leq x_1 \leq x_{1,\mathrm{M}}(\boldsymbol{x})$	$x_1 = 0$
x_2	$x_{2,\mathrm{m}}(\boldsymbol{x}) \leq x_2 \leq x_{2,\mathrm{M}}(\boldsymbol{x})$	$x_2 = 0$

where

$$x_{j,\mathrm{M}}(\boldsymbol{x}) = -x_{j,\mathrm{m}}(\boldsymbol{x}) = 3\sqrt{1/(2k_j) + 0.01}$$

for $j = 1, 2$. The bound for x_1 and x_2 is built according to the variance of the limiting distribution of the OU process $1/(2k_j)$. These variables are equal to 0 in the test samples since we only need the option price at time 0, i.e., $V(s, 0, 0, 0)$. The ranges of $\ln(B)$ and s are trickier since they are dependent on the option type. We sample $\ln(B)$ instead of B based on the following rules

	Range for both training and test samples
vanilla options	(not applicable)
up-and-in/out call	$\ln(K) \leq \ln(B) \leq \ln(1.5K)$
down-and-in/out put	$\ln(K/1.5) \leq \ln(B) \leq \ln(K)$
others	$\ln(K/1.5) \leq \ln(B) \leq \ln(1.5K)$

The range of $\ln(B)$ is halved for up-and-in/out calls and down-and-in/out puts since we only calculate the non-degenerate case and the degenerate case falls into vanilla options. Finally, the range of s is listed for each case as follows:

	Training range	Test range
vanilla options	$\ln(K/20) \leq s \leq \ln(20K)$	$\ln(K/2) \leq s \leq \ln(2K)$
up-and-in/out option	$\ln(K/20) \leq s \leq \ln(B)$	$\ln(K/2) \leq s \leq \ln(B)$
down-and-in/out option	$\ln(B) \leq s \leq \ln(20K)$	$\ln(B) \leq s \leq \ln(2K)$

The range of s is narrower in the test samples since we would like to focus more on the liquid options.

After choosing the range of each argument in $\boldsymbol{x}$, we introduce how to sample them within the given range. All variables and parameters are sampled from the uniform distribution over the given intervals. If the lower and upper boundaries depend on other parameters, we use the conditional uniform distribution given the parameters in their boundaries. For example, $\rho_{1,2}$ follows the conditional uniform distribution over

$$\left[\rho_1\rho_2 - \sqrt{(1-\rho_1^2)(1-\rho_2^2)}, \rho_1\rho_2 + \sqrt{(1-\rho_1^2)(1-\rho_2^2)}\right]$$

given ρ_1 and ρ_2. The only exceptions are x_1 and x_2. The variables (x_1, x_2) are sampled from their marginal distribution at time t, which is the two-dimensional normal distribution

$$N\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \frac{1-e^{-2k_1t}}{2k_1} + 0.01 & \rho_{1,2}\frac{1-e^{-(k_1+k_2)t}}{k_1+k_2} \\ \rho_{1,2}\frac{1-e^{-(k_1+k_2)t}}{k_1+k_2} & \frac{1-e^{-2k_2t}}{2k_2} + 0.01 \end{pmatrix}\right)$$

and is then clipped within their range $x_{j,\mathrm{m}}(\boldsymbol{x}) \leq x_j \leq x_{j,\mathrm{M}}(\boldsymbol{x}), \forall j = 1, 2$. 0.01 is added to their variances to prevent the degenerate distribution at time $t = 0$.

4.6.3 Training and results

We consider the neural network defined in Equation (4.6) consisting of $L = 5$ layers for vanilla options and the neural network defined in Equation (4.8) consisting of $(L_1, L_2) = (3, 2)$ layers for barrier options. Each hidden layer contains $n = 500$ neurons. The same network is used for the constant ξ_0^t case and nine-segment ξ_0^t case. The activation function g is SiLU. The training batch size is 1000 and the training size is determined as follows:

Training size	Vanilla options	Barrier options
constant ξ_0^t	10, 000, 000	20, 000, 000
9-segment ξ_0^t	100, 000, 000	200, 000, 000

There are 10,000 test samples in each case. We use the Adam algorithm [67] for training. The network is trained for 45 epochs in the constant ξ_0^t case and 9 epochs in the nine-segment ξ_0^t case. The learning rate decreases exponentially from 10^{-3} to 10^{-5}.

We use simulation to calculate the benchmark, which is introduced in Appendix C.3 and C.4. The results are summarized in Table 4.6. In Table 4.6, we list the root mean square error (RMSE) of the neural network solution for each option, i.e.,

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{1 \le j \le n} \left(V(\boldsymbol{x}_{(j)}) - \bar{V}(\boldsymbol{x}_{(j)})\right)^2},$$

where $\boldsymbol{x}_{(j)}$ are the test samples following the ranges and distributions in Section 4.6.2, $V(\boldsymbol{x}_{(j)})$ is the solution given by the neural network and $\bar{V}(\boldsymbol{x}_{(j)})$ is the benchmark. The RMSE is calculated over the 10,000 test samples, which have not been used during the training process. We also list the RMSE of the benchmark

$$\sqrt{\frac{1}{n} \sum_{1 \le j \le n} \left(\text{se}(\bar{V}(\boldsymbol{x}_{(j)}))\right)^2},$$

where $\text{se}(\bar{V}(\boldsymbol{x}_{(j)}))$ is the standard error of the estimate $\bar{V}(\boldsymbol{x}_{(j)})$ in the simulation benchmark. Since the benchmarks are noisy, the RMSE of the neural network cannot be much smaller than the RMSE of the benchmark.

	RMSE of network solutions		RMSE of simulation	
Option	constant ξ_0^t	9-segment ξ_0^t	constant ξ_0^t	9-segment ξ_0^t
vanilla call	0.0686	0.0685	0.1422	0.1415
vanilla put	0.1066	0.1039	0.0926	0.0940
up-and-out call	0.0772	0.1031	0.0622	0.0640
up-and-in call	0.1117	0.1309	0.1225	0.1146
down-and-out call	0.1479	0.1676	0.2060	0.2094
down-and-in call	0.1329	0.1576	0.1336	0.1427
up-and-out put	0.1133	0.1372	0.1063	0.1067
up-and-in put	0.1069	0.1336	0.0963	0.1000
down-and-out put	0.0736	0.0923	0.0600	0.0617
down-and-in put	0.1171	0.1271	0.0992	0.0999

Table 4.6: The root mean square error (RMSE) of the neural network solution over the 10,000 test samples for each option, compared with the RMSE of the simulation benchmark.

4.6.4 Fitted curves

The singular terms are included in the neural networks such that the non-smooth and discontinuous boundary conditions can be fitted. Thus the neural networks keep the singular properties around $(s,t) = (\ln(K), T)$ and $(s,t) = (\ln(B), T)$ and are good at fitting option price curves of short maturities. Consequently, they are also able to replicate the prices of longer maturity given they are fitted to satisfy the PDE. In Figures 4.7 and 4.8, we show the fitted neural network solution and the simulation benchmark of the barrier calls at $T = 1/252$ (1 day), $T = 1/52$ (1 week) and $T = 1/2$ (half a year) as well as the relative error $(V(\boldsymbol{x}) - \bar{V}(\boldsymbol{x}))/\max(\bar{V}(\boldsymbol{x}), h)$ as an example. For these examples we use $h = 0.25$. The barrier calls are taken as the examples since the barrier puts are bounded and are usually fitted with smaller errors.

4.6.5 Calculation speed

The neural network can calculate prices of a batch of parameter sets at the same time. Thus it will be super fast to generate the option prices once trained. The calculation times of the neural networks used in the numerical experiments are summarized in Table 4.7. Note that the vanilla and knock-in options only use one neural network, while the knock-out options are calculated as a difference of the vanilla and knock-in options and make use of two networks. By means of the GPU acceleration, 200,000 prices can be calculated in 0.133 seconds as most.

	Input size	1	10	100	1k	10k	100k	200k
vanilla &	GPU(s)	0.017	0.027	0.021	0.021	0.025	0.055	0.070
knock-in	CPU(s)	0.032	0.033	0.034	0.092	0.533	4.943	9.296
knock-out	GPU(s)	0.070	0.062	0.067	0.062	0.070	0.083	0.133
	CPU(s)	0.078	0.073	0.093	0.194	1.347	10.717	18.585

Table 4.7: Computation time of the neural network solutions in the numerical experiments consisting of 5 layers, with 500 neurons in each layer. CPU is an Intel Xeon CPU @ 2.20GHz. GPU is a Tesla V100-SXM2-16GB. All times are in seconds.

4.7 Conclusion

In this chapter, we have developed an unsupervised deep learning method to solve the barrier options under the two-factor Bergomi model. The neural networks serve as the approximate option surfaces and are trained to satisfy the PDE as well as the boundary conditions. A trained neural network can calculate option values extremely fast. Having fast algorithms for pricing both vanilla and exotic options, e.g. barrier options, would facilitate to assess the effect of model risk [52].

Here we summarize the main innovations based on the unsupervised deep learning method:

- We propose two singular terms to deal with the non-smoothness at the strike level and the discontinuity at the barrier level so that the neural network can fit the boundary conditions of the barrier options. Since we already know the singular points of vanilla and barrier option surfaces are at the strike and barrier levels, the singular points of the singular terms are defined to be also the strike and barrier levels, and will not change during training.

- We use six networks to express the eight barrier options in one framework. We do not train the eight options separately, but make use of the in-out parity. We build networks for knock-in options given they contain only one singularity and are easier to be fitted.

- The neural network is employed to deal with the high dimensionality coming with the large number of parameters and the function input in the multi-factor forward variances in the Bergomi model.

- Boundary conditions of the volatility factors are estimated by the BMS model, which increases the accuracy of the method.

The proposed method can also deal with the other stochastic volatility models, as long as we find the suitable boundary conditions of volatility and the suitable estimate. The other stochastic volatility models should not be more complex than the Bergomi model given there are fewer parameters and there is no function input in the model. So the method for the Bergomi model serves as a good example of the applications to the stochastic volatility models.

The two proposed singular terms are good examples for the case that we need to solve heat equations with non-smooth or discontinuous initial conditions. We can incorporate multiple singular terms into one neural network for more complex initial conditions, which would facilitate fitting the asymptotic behaviors of the solution near the initial condition. Moreover, the idea of singular terms can be extended to deal with other types of problems as long as we know the overall shape and approximate position of the non-smoothness or discontinuity in their solutions.

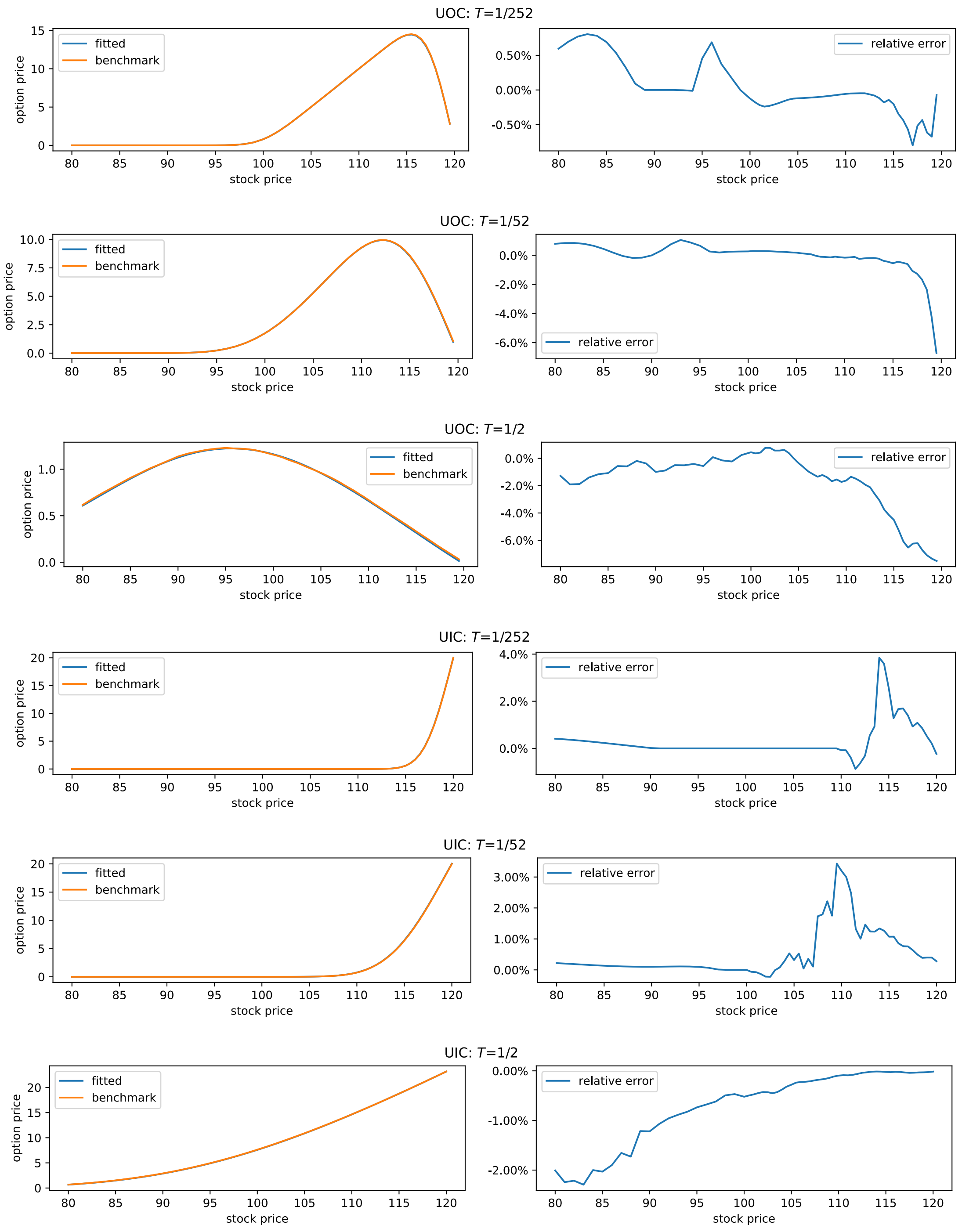


Figure 4.7: Comparison of the fitted neural network solution and the simulation benchmark of the up-and-out/in call when $K = 100, B = 120, r = q = 0, \xi_0^t = 0.1, \omega = 1, k_1 = 1, k_2 = 10, \theta = 0.5, \rho_1 = \rho_2 = -0.5, \rho_{1,2} = 0$ and $t = x_1 = x_2 = 0$. UOC and UIC stand for the up-and-out and up-and-in call respectively.

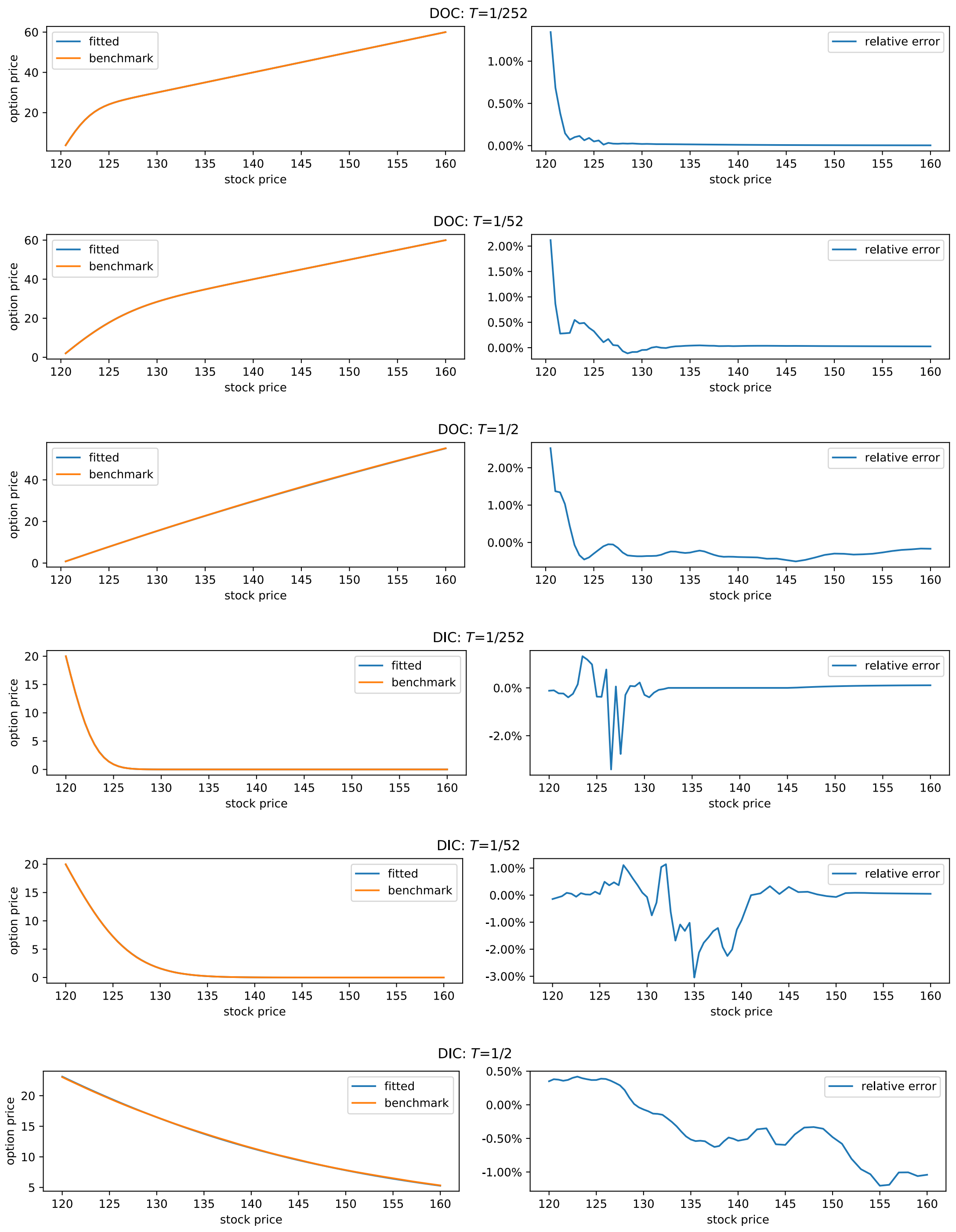


Figure 4.8: Comparison of the fitted neural network solution and the simulation benchmark of the down-and-out/in call when $K = 100, B = 120, r = q = 0, \xi_0^t = 0.1, \omega = 1, k_1 = 1, k_2 = 10, \theta = 0.5, \rho_1 = \rho_2 = -0.5, \rho_{1,2} = 0$ and $t = x_1 = x_2 = 0$. DOC and DIC stand for the down-and-out and down-and-in call respectively.

Chapter 5: Simulation of financial time series using generative adversarial networks with attention

5.1 Introduction

Parametric models for financial time series are robust and simple, but they have strong assumptions, which limit their further applications. For example, in the GARCH model, the returns have no impact on the future volatilities, which is not consistent with the empirical stock return data. So it is difficult for a parametric model to replicate all the major statistical properties of real data. The generative adversarial networks (GANs) [41] are purely data-driven models with weaker assumptions. The GANs have been proved to be successful in many applications, e.g. simulation of realistic human portraits, landscape paintings and musical compositions.

It is shown in [106] that long-range dependency is a major challenge in financial time series simulation. The attention mechanism [4, 80] is perfectly suitable for modeling the stylized facts of long-range dependency due to the large receptive field size of the attention layer. Thus we are motivated to use the attention-based GANs to simulate the financial time series. It is important to note the difference between time series and fixed-dimensional variables, since a time series can have an arbitrary length. Thus we need to modify the attention-based GANs to make it agnostic to the length of the time series. Our findings based on numerical results show that the attention-based GANs perform as well as the temporal convolutional network (TCN or WaveNet [101]) in replication of the major stylized facts, including heavy tails, autocorrelation and cross-correlation, and are better at simulating smooth autocorrelation of returns and satisfying the no-arbitrage condition of option surfaces. It is well known in the literature that both GANs and transformers [102], which contain multiple attention layers, are hard to train, and it is a challenging problem to combine them together. Millions or billions of samples are usually used to train the transformer GANs in text generation and image simulation. Authors in [64] pointed it out that transformers are data-hungry, and thus they made used of data augmentation techniques to improve the transformer GANs. In this chapter, we propose a new transformer GAN using sparse attention [22, 27] and train it using a small amount of financial time series data (around 3000 samples).

Besides the attention mechanism, we also tried a structure like the regularized GAN (RegGAN) [29], which employs a pre-trained network as a second discriminator, to emphasize selected stylized facts, e.g. high kurtosis. We train a discriminator on the high kurtosis data, which is able to replicate the high kurtosis data. Then we use the pre-trained discriminator as the second discriminator when we train the GANs on the low kurtosis data to increase the prior of the high kurtosis data. However, this approach does not work as we expected, and it is shown that the discriminator trained on one dataset may not be suitable to distinguish the real data in another dataset.

The rest of this chapter is organized as follows. In Section 5.2, we define the generative model and emphasize the difference between the generative models of time series and fixed-dimensional distributions. We also introduce the GANs, which employ a discriminator to train the generator. In Section 5.3, we introduce and compare the regular and causal convolutional layers and attention layers, which are building blocks of the proposed GANs. In Section 5.4, we propose to employ attention as the tool to model the long-range dependencies, and give the detailed structure of the proposed temporal attention GAN (`TAGAN`) and temporal transformer GAN (`TTGAN`). In Sections 5.5 and 5.6, we show the numerical results of the proposed GANs for the S&P 500 index simulation and its index option surface simulation respectively. Section 5.7 summarizes the chapter.

5.2 Generative model of financial time series

5.2.1 Problem formulation

Suppose we have a financial time series $\{\boldsymbol{x}_t \in \mathbb{R}^d\}_{t\in\mathbb{Z}}$, e.g., the historical of prices or volatilities, and would like to generate a time series $\{\boldsymbol{y}_t \in \mathbb{R}^d\}_{t\in\mathbb{Z}}$ that has the same statistical properties given a series of i.i.d. random noise $\{z_t \in \mathbb{R}^{d_n}\}_{t\in\mathbb{Z}}$ via deep learning. Let $z_{i:j}$ denote the sequence $\{z_i, z_{i+1}, \ldots, z_j\}$ and the same notation is used for all time series hereafter.

Here we follow the definition of the *neural process* in [106]. We would like to develop a generator $G(\cdot; \theta_G)$, a neural network with the parameter $\theta_G \in \Theta_G$, which takes random noise from $\{z_t\}_{t\in\mathbb{Z}}$ as its input and outputs the time series $\{\boldsymbol{y}_t\}_{t\in\mathbb{Z}}$, i.e.,

$$\boldsymbol{y}_t = G(z_{t-f+1:t}; \theta_G), \forall t \in \mathbb{Z}$$

where $f > 0$ is called the receptive field size (RFS) and means the length of noise variables of which $\boldsymbol{y}_t$ is composed. By this definition, $\boldsymbol{y}_t$ and $\boldsymbol{y}_{t+\tau}$ would be independent if $\tau \geq f$. Also, since $\boldsymbol{y}_t$ is computed from $\{z_\tau\}_{\tau\leq t}$, we know $\{\boldsymbol{y}_t\}_{t\in\mathbb{Z}}$ is adapted to $\{z_t\}_{t\in\mathbb{Z}}$.

The generator of time series has to satisfy the following conditions that make it different from a generator of a fixed-dimensional distribution.

(a) The generator should be able to take in random noise of length $l + f - 1$ to output the aimed time series of an arbitrary length l, i.e.,

$$\boldsymbol{y}_{t-l+1:t} = G(z_{t-l-f+2:t}; \theta_G), \forall t \in \mathbb{Z}.$$

(b) The generated time series can be prolonged in a consistent manner. For arbitrary $t_1, t_2, t_3, t_4 \in \mathbb{Z}$ such that $[t_1, t_2] \cap [t_3, t_4] \neq \emptyset$,

$$\boldsymbol{y}_{t_1:t_2} = G(z_{t_1-f+1:t_2}; \theta_G)$$

and

$$\tilde{\boldsymbol{y}}_{t_3:t_4} = G(z_{t_3-f+1:t_4}; \theta_G),$$

the overlapping part of the generated time series must be equal, i.e.,

$$\boldsymbol{y}_{\max\{t_1,t_3\}:\min\{t_2,t_4\}} = \tilde{\boldsymbol{y}}_{\max\{t_1,t_3\}:\min\{t_2,t_4\}}.$$

This means the generated time series $\{\boldsymbol{y}_t\}_{t\in\mathbb{Z}}$ is uniquely determined by the random noise series $\{z_t\}_{t\in\mathbb{Z}}$.

Given the two conditions are satisfied, we can always let the generator $G(\cdot;\theta_G)$ compute sequences of length l and then combine the sequences to make up a longer sequence $\boldsymbol{y}_{1:T}$, where $T > l$. To be more specific, we first calculate the pieces

$$\boldsymbol{y}_{(i-1)l+1:il} = G(\boldsymbol{z}_{(i-1)l-f+2:il};\theta_G), \forall 1 \leq i \leq \lceil T/l \rceil,$$

and then $\boldsymbol{y}_{1:T}$ is a subsequence of $\boldsymbol{y}_{1:l\lceil T/l\rceil}$, where $\lceil\cdot\rceil$ is the ceiling function.

5.2.2 Training through generative adversarial network

We give a quick introduction of GANs as well as the loss functions for training GANs. The GAN introduces a discriminator

$$D(\cdot;\theta_D) : \mathbb{R}^{l\times d} \to \mathbb{R},$$

where $\theta_D \in \Theta_D$, to evaluate the similarity between the real historical data $\{\boldsymbol{x}_t\}_{t\in\mathbb{Z}}$ and the simulated data $\{\boldsymbol{y}_t\}_{t\in\mathbb{Z}}$. A higher output value from $D(\cdot;\theta_D)$ means the discriminator holds a stronger belief that the input sample comes from the real data.

Suppose we have a sequence of real data $\boldsymbol{x}_{1:T}$. Let $\mathbb{P}_{\boldsymbol{X}}$ be the uniform distribution over the window data of length l, $\{\boldsymbol{x}_{i:i+l-1}, \forall 1 \leq i \leq T-l+1\}$. Also, let $\mathbb{P}_{\boldsymbol{Z}}$ be distribution of $z_{1:l+f-1}$. Then we draw $\boldsymbol{X} \sim \mathbb{P}_{\boldsymbol{X}}$ to be a piece of real data of length l and $\boldsymbol{Z} \sim \mathbb{P}_{\boldsymbol{Z}}$ the random noise of length $l+f-1$ and get the simulated sequence $\boldsymbol{Y} = G(\boldsymbol{Z};\theta_G) \in \mathbb{R}^{l\times d}$. The GAN trains the generator and the discriminator by minimizing the following loss functions

$$\min_{\theta_G} \mathbb{E}_{\boldsymbol{Z}} \mathcal{L}_G(D(G(\boldsymbol{Z};\theta_G);\theta_D))$$

and

$$\min_{\theta_D} \mathbb{E}_{\boldsymbol{X},\boldsymbol{Z},\tilde{\boldsymbol{X}}} \mathcal{L}_D(D(\boldsymbol{X};\theta_D), D(G(\boldsymbol{Z};\theta_G);\theta_D), \nabla_{\tilde{\boldsymbol{X}}} D(\tilde{\boldsymbol{X}};\theta_D)),$$

where $\mathcal{L}_G(\cdot)$ and $\mathcal{L}_D(\cdot,\cdot,\cdot)$ are the loss functions of the discriminator and the generator. The third argument in $\mathcal{L}_D(\cdot,\cdot,\cdot)$ is not included in the original GAN but related with gradient penalty, where $\tilde{\boldsymbol{X}} = (1-U)\boldsymbol{X} + U\boldsymbol{Y}$ is a linear interpolation between $\boldsymbol{X}$ and $\boldsymbol{Y}$ requiring U follows the uniform distribution over $(0, 1)$.

The loss functions of the original GAN [41] are

$$\begin{aligned} \mathcal{L}_G(d_f) &= -\ln(\sigma(d_f)) \\ \mathcal{L}_D(d_r, d_f, \boldsymbol{g}) &= -\ln(\sigma(d_r)) - \ln(1-\sigma(d_f)) \end{aligned} \tag{5.1}$$

where $\sigma(d) = 1/(1+e^{-d})$ is the sigmoid function and $\sigma(D(\boldsymbol{X};\theta_D))$ means the probability that the discriminator considers $\boldsymbol{X}$ belongs to the real data. A quick derivation of the losses is included in Appendix D.2.

Besides the original losses, the loss functions of the Wasserstein GAN [2] with gradient penalty (WGAN-GP) [43] are also widely used, where the gradient norm penalty is used to achieve Lipschitz continuity:

$$\begin{aligned} \mathcal{L}_G(d_f) &= -d_f \\ \mathcal{L}_D(d_r, d_f, \boldsymbol{g}) &= -d_r + d_f + \lambda(\|\boldsymbol{g}\| - 1)^2 \end{aligned} \tag{5.2}$$

where λ is a constant and $\lambda = 10$ by default. $\|\cdot\|$ is the Frobenius norm. In Appendix D.2, we introduce how the losses are derived.

5.3 Network layers

In this section, we going to list all the layers that will be used in the proposed network structure. Some of the layers are already introduced in literature, but we still give a short introduction for each to make the thesis self-contained. The layers are classified into regular layers and causal layers. In the causal layers, each output node only depends on the input nodes with equal or smaller time indices. Suppose the input of the causal layer is $\boldsymbol{I} \in \mathbb{R}^{n_l \times n_i}$ with rows $\{\boldsymbol{I}_{t,\cdot}\}_{t=1}^{n_l}$ and the output is $\boldsymbol{O} \in \mathbb{R}^{(n_l-f+1)\times n_o}$ with rows $\{\boldsymbol{O}_{t,\cdot}\}_{t=f}^{n_l}$, then each row of the output $\boldsymbol{O}_{t,\cdot}$ only depends on

$\{\boldsymbol{I}_{\tau,\cdot}\}_{\tau=t-f+1}^{t}$. However, the regular layers are not subject to this restriction. Since the output of the generator needs to be adapted to the input noise, the causal layers are used in the generator. While the regular layers admit more flexibility and are used in the discriminator.

5.3.1 Regular convolutional layer

In [75], the authors proposed the convolutional layer, which is good at extracting local information. The two-dimensional case is widely used in computer vision and the one-dimensional case is used in sequence models. Although the convolutional layer is widely used and well-known, we reiterate the definition to show the difference between the different layers. Suppose the input is $\boldsymbol{I} \in \mathbb{R}^{n_l \times n_i}$ and it passes through a one-dimensional regular convolutional layer with kernel size n_k, output channel n_o and stride s. The kernel size n_k is an odd number by default. The parameters are the weight $\boldsymbol{W} \in \mathbb{R}^{n_k \times n_i \times n_o}$ and the intercept $\boldsymbol{b} \in \mathbb{R}^{n_o}$. The output of the regular convolutional layer is $\boldsymbol{O} \in \mathbb{R}^{\lfloor n_l/s \rfloor \times n_o}$ given by

$$O_{i_l,i_o} = \sum_{i=1}^{n_i} \sum_{i_k=1}^{n_k} W_{i_k,i,i_o} I_{s(i_l-1)+1-(n_k+1)/2+i_k,i} + b_{i_o}, \forall 1 \le i_l \le \lfloor n_l/s \rfloor, 1 \le i_o \le n_o,$$

where $\lfloor \cdot \rfloor$ is the floor function. The 'same' padding rule is applied to the input, i.e., $I_{i_l,i} = I_{1,i}, \forall i_l < 1$ and $I_{i_l,i} = I_{n_l,i}, \forall i_l > n_l$. The regular convolutional layer is illustrated in Figure 5.1. It is denoted as $\texttt{conv}_{\texttt{r}}^{(n_k,n_o,s)}(\cdot)$, where n_k, n_o and s are the kernel size, output channel and stride respectively.

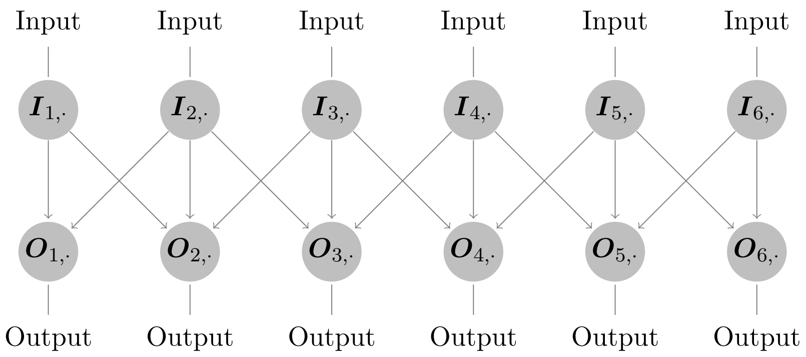

Input
Input
Input
Input
Input
Input
$\boldsymbol{I}_{1,\cdot}$
$\boldsymbol{I}_{2,\cdot}$
$\boldsymbol{I}_{3,\cdot}$
$\boldsymbol{I}_{4,\cdot}$
$\boldsymbol{I}_{5,\cdot}$
$\boldsymbol{I}_{6,\cdot}$
$\boldsymbol{O}_{1,\cdot}$
$\boldsymbol{O}_{2,\cdot}$
$\boldsymbol{O}_{3,\cdot}$
$\boldsymbol{O}_{4,\cdot}$
$\boldsymbol{O}_{5,\cdot}$
$\boldsymbol{O}_{6,\cdot}$
Output
Output
Output
Output
Output
Output

Figure 5.1: Illustration of the regular convolutional layer (length $n_l = 6$, kernel size $n_k = 3$ and stride $s = 1$).

5.3.2 Causal convolutional layer

In [101], the authors proposed the causal convolutional layer to model the audio data. Suppose the input is $\boldsymbol{I} \in \mathbb{R}^{n_l \times n_i}$ and it passes through a causal convolutional layer with kernel size n_k and output channel n_o. The parameters are the weight $\boldsymbol{W} \in \mathbb{R}^{n_k \times n_i \times n_o}$ and the intercept $\boldsymbol{b} \in \mathbb{R}^{n_o}$. The output of the causal convolutional layer is $\boldsymbol{O} \in \mathbb{R}^{(n_l - n_k + 1) \times n_o}$. In the causal layer, the time index of the output is taken from $\{n_k, n_k + 1, \ldots, n_l\}$. The output is given by

$$O_{t,i_o} = \sum_{i=1}^{n_i} \sum_{i_k=1}^{n_k} W_{i_k,i,i_o} I_{t-n_k+i_k,i} + b_{i_o}, \forall n_k \leq t \leq n_l, 1 \leq i_o \leq n_o.$$

The RFS of the causal convolutional layer is equal to n_k. The causal convolutional layer is illustrated in Figure 5.2. It is denoted as $\text{conv}_{\text{C}}^{(n_k,n_o)}(\cdot)$, where n_k and n_o are the kernel size and output channel.

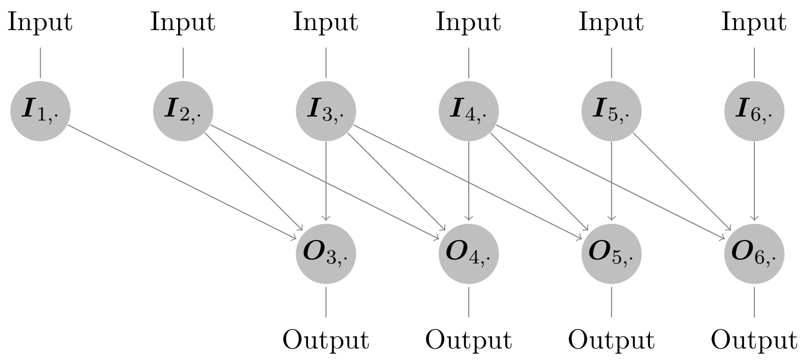

Input
Input
Input
Input
Input
Input
$\boldsymbol{I}_{1,\cdot}$
$\boldsymbol{I}_{2,\cdot}$
$\boldsymbol{I}_{3,\cdot}$
$\boldsymbol{I}_{4,\cdot}$
$\boldsymbol{I}_{5,\cdot}$
$\boldsymbol{I}_{6,\cdot}$
$\boldsymbol{O}_{3,\cdot}$
$\boldsymbol{O}_{4,\cdot}$
$\boldsymbol{O}_{5,\cdot}$
$\boldsymbol{O}_{6,\cdot}$
Output
Output
Output
Output

Figure 5.2: Illustration of the causal convolutional layer (length $n_l = 6$ and kernel size $n_k = 3$).

5.3.3 Regular attention layer

The attention layer was introduced by [4, 80]. It is designed to receive and process global information to improve the performance of convolutional or recurrent networks. The attention layer was first used in text translation, where input words and output words may follow different orders because of different grammar. So, the attention layer needs to search from the entire inputs to decide which input word is corresponding to a specific output word. The best match in the input becomes the 'attention' of the layer. Authors in [102] proposed the transformer network, which consists of only attention layers and multi-layer perceptrons, and proved attention layers are capable of modeling sequences without help from convolutional or recurrent layers. Suppose the input of the regular multi-head attention layer is $\boldsymbol{I} \in \mathbb{R}^{n_l \times n_i}$, the hidden size is n_a and the number of heads is n_h, which satisfy $\mathrm{mod}(n_a, n_h) = 0$. The parameters are the weights $\boldsymbol{W}^Q, \boldsymbol{W}^K, \boldsymbol{W}^V \in \mathbb{R}^{n_i \times n_a}$, $\boldsymbol{W}^O \in \mathbb{R}^{n_a \times n_i}$ and intercepts $\boldsymbol{b}^Q, \boldsymbol{b}^K, \boldsymbol{b}^V \in \mathbb{R}^{n_a}$, $\boldsymbol{b}^O \in \mathbb{R}^{n_i}$. Let $\mathbf{1}$ be the vector of length n_l with all elements of 1. The formulae in the regular attention layer are

$$\begin{aligned}
\boldsymbol{Q} &= \boldsymbol{I}\boldsymbol{W}^Q + \mathbf{1}\boldsymbol{b}^{Q\top} \\
\boldsymbol{K} &= \boldsymbol{I}\boldsymbol{W}^K + \mathbf{1}\boldsymbol{b}^{K\top} \\
\boldsymbol{V} &= \boldsymbol{I}\boldsymbol{W}^V + \mathbf{1}\boldsymbol{b}^{V\top} \\
\boldsymbol{A}_{(i_h)} &= \mathrm{softmax}\left(\boldsymbol{Q}_{(i_h)}\boldsymbol{K}_{(i_h)}^\top\right)\boldsymbol{V}_{(i_h)}, \forall 1 \le i_h \le n_h \\
\boldsymbol{O} &= \boldsymbol{A}\boldsymbol{W}^O + \mathbf{1}\boldsymbol{b}^{O\top}
\end{aligned} \tag{5.3}$$

where $\boldsymbol{A}_{(i_h)}$ means the submatrix from column $(i_h - 1)n_h + 1$ to column $i_h n_h$ and $\boldsymbol{O} \in \mathbb{R}^{n_l \times n_i}$ is the output of the attention layer. The softmax function is evaluated along each row of the input matrix. While the fourth equation in Equation (5.3) is often replaced with

$$\boldsymbol{A}_{(i_h)} = \mathrm{softmax}\left(\boldsymbol{Q}_{(i_h)}\boldsymbol{K}_{(i_h)}^\top / \sqrt{n_a/n_h}\right)\boldsymbol{V}_{(i_h)}$$

when the size of a single attention head n_a/n_h is large, we stick with the equation in Equation (5.3) since it shows better empirical results when n_a/n_h is small. The dependence relationship of the output on the input in the regular attention layer is illustrated in Figure 5.3. The regular attention layer is denoted as $\mathrm{attn}_{\mathtt{r}}^{(n_a,n_h)}(\cdot)$, where n_a and n_h are the hidden size and number of heads.

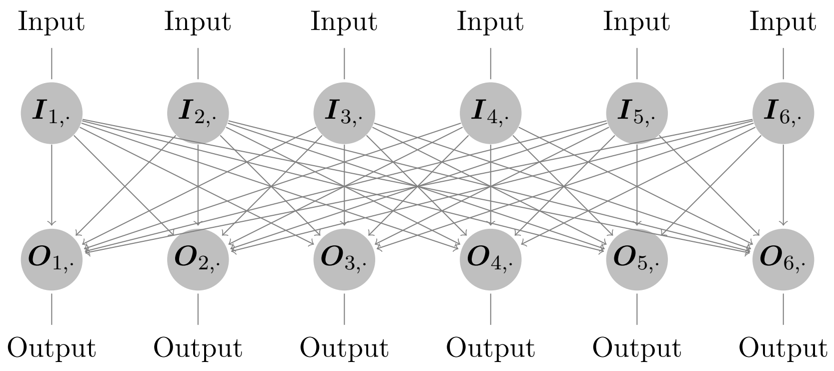


Figure 5.3: Dependence relationship in the regular attention layer (length $n_l = 6$).

5.3.4 Sparse attention layer

The sparse attention layer is introduced by [22] to accelerate computation and improve the focus of the attention layer. The sparse attention introduces the sparse masks before the softmax function, i.e. replacing the fourth equation in Equation (5.3) with

$$\boldsymbol{A}_{(i_h)} = \text{softmax}\left(\boldsymbol{Q}_{(i_h)}\boldsymbol{K}_{(i_h)}^{\top} + \boldsymbol{M}^{(i_h)}\right)\boldsymbol{V}_{(i_h)} \tag{5.4}$$

where $\boldsymbol{M}^{(i_h)} \in \mathbb{R}^{n_i \times n_i}, \forall 1 \leq i_h \leq n_h$ are the sparse mask matrices. In the mask $\boldsymbol{M}^{(i_h)}$, the elements in the masked positions are assigned a large negative value $-L$, while the elements outside the mask are assigned 0. Since they are sparse masks, the 0 elements are sparse and the large negative value elements are dense. The masked positions are mapped to 0 by the softmax function since $e^{-L} \approx 0$, where we usually let $L = 10^3$. In this way, the masked positions in $\boldsymbol{Q}_{(i_h)}\boldsymbol{K}_{(i_h)}^{\top}$ are not involved in the result of the softmax function and the attention is limited within the sparse mask.

In this chapter we use the sparse masks proposed in [27]. They are generated in the following way. Let $s = \lfloor\sqrt{n_l}\rfloor$ be the stride, where $\lfloor\cdot\rfloor$ is the floor function. We then create the index sets for the masks:

- Left floor mask: $S_1 = \{(i, j) : \lfloor(i-1)/s\rfloor = \lfloor(j-1)/s\rfloor \text{ and } i \geq j\}$
- Right floor mask: $S_2 = \{(i, j) : \lfloor(i-1)/s\rfloor = \lfloor(j-1)/s\rfloor \text{ and } i \leq j\}$

- Left repetitive mask: $S_3 = \{(i, j) : \text{mod}(j, s) = 0 \,\text{or}\, i = j\}$
- Right repetitive mask: $S_4 = \{(i, j) : \text{mod}(j, s) = 1 \,\text{or}\, i = j\}$

The corresponding masks are defined as

$$M_{i,j}^{(i_h)} = \begin{cases} 0, & \text{if}\,(i, j) \in S_{i_s}, \\ -L, & \text{if}\,(i, j) \notin S_{i_s}, \end{cases} \quad \text{when}\, i_h \equiv i_s (\text{mod}\, 4).$$

As a result, the number of heads n_h needs to be a multiple of 4 when we use the sparse attention. The sparse attention limits the attention of each node to a specific region such that the network would converge faster. In Figure 5.4, we show an example of the sparse masks. As shown in Figure 5.4, the left and right floor masks focus on local features and the left and right repetitive masks focus on periodic features. The sparse attention layer is denoted as $\text{attn}_{\text{S}}^{(n_a, n_h)}(\cdot)$, where n_a and n_h are the hidden size and number of heads.

5.3.5 Causal attention layer

To make use of the attention layers in the generator of time series, we need to make sure the output only depends on the input elements in the past. This is also achieved by using masks. We suppose the same input $\boldsymbol{I} \in \mathbb{R}^{n_l \times n_i}$ as before, and the parameters are defined the same as in the regular attention. Let the RFS of the layer be n_f. The definition of the attention layer in Equation (5.3) needs to be replaced with

$$\begin{aligned} \boldsymbol{Q} &= \boldsymbol{I}\boldsymbol{W}^Q + \mathbf{1}\boldsymbol{b}^{Q^\top} \\ \boldsymbol{K} &= \boldsymbol{I}\boldsymbol{W}^K + \mathbf{1}\boldsymbol{b}^{K^\top} \\ \boldsymbol{V} &= \boldsymbol{I}\boldsymbol{W}^V + \mathbf{1}\boldsymbol{b}^{V^\top} \\ \boldsymbol{A}_{(i_h)} &= \text{softmax}\left(\boldsymbol{Q}_{(i_h)}\boldsymbol{K}_{(i_h)}^\top + \boldsymbol{M}\right)\boldsymbol{V}_{(i_h)}, \forall 1 \le i_h \le n_h \\ \boldsymbol{O} &= \left(\boldsymbol{A}\boldsymbol{W}^O + \mathbf{1}\boldsymbol{b}^{O^\top}\right)_{n_f:n_l,\cdot} \end{aligned}$$

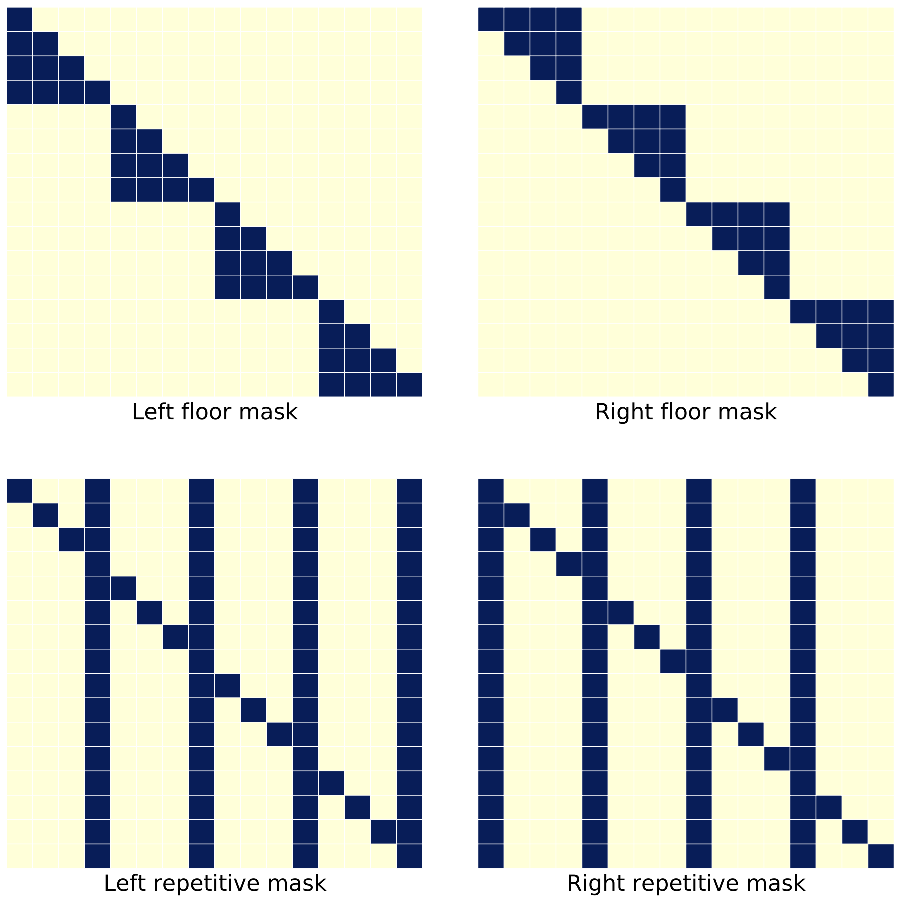


Figure 5.4: Example of the sparse masks when $n_l = 16$. Light elements are masked while dark elements are not.

where $(\cdot)_{n_f:n_l,\cdot}$ means the submatrix from row n_f to row n_l, and $\boldsymbol{M}$ is a mask matrix with the elements

$$M_{i,j} = \begin{cases} 0, & \text{if } 0 \leq i - j \leq n_f - 1, \\ -L, & \text{else.} \end{cases}$$

In this way, each output only depends on the current input and $n_f - 1$ past inputs. The benefit of the causal attention layer is that it can increase the RFS n_f arbitrarily without introducing additional parameters. The size of the parameters does not depend on the input length n_l or the RFS n_f. The dependence relationship of the output on the input in the causal attention layer is illustrated in Figure 5.5. The causal attention layer is denoted as $\text{attn}_{\text{C}}^{(n_a,n_h,n_f)}(\cdot)$, where n_a, n_h and n_f are the hidden size, number of heads and RFS respectively.

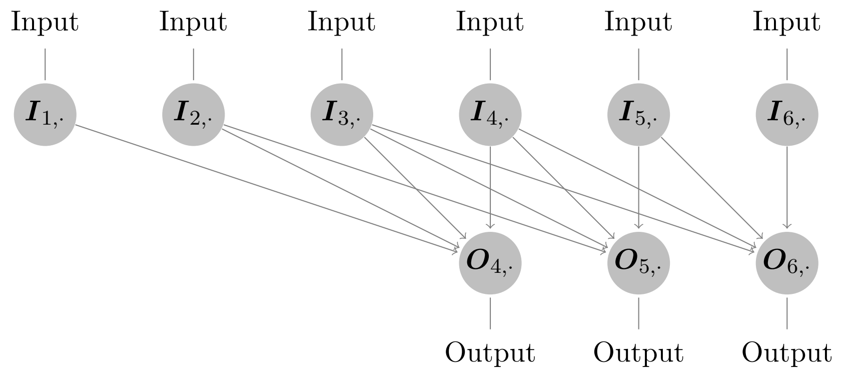


Figure 5.5: Dependence relationship in the causal attention layer (length $n_l = 6$ and RFS $n_f = 4$).

5.3.6 Multi-layer perceptron block

The multi-layer perceptron (MLP) is a key component of the transformer architecture. Suppose the input is $\boldsymbol{I} \in \mathbb{R}^{n_l \times n_i}$, the hidden size is n_m and the activation function is $h(\cdot)$. The parameters are the weights $\boldsymbol{W}^{(1)} \in \mathbb{R}^{n_i \times n_m}$, $\boldsymbol{W}^{(2)} \in \mathbb{R}^{n_m \times n_i}$ and intercepts $\boldsymbol{b}^{(1)} \in \mathbb{R}^{n_m}$, $\boldsymbol{b}^{(2)} \in \mathbb{R}^{n_i}$. Let $\mathbf{1}$ be the vector of length n_l with all elements of 1. The formulae in the MLP block are

$$\begin{aligned} \boldsymbol{H} &= \boldsymbol{I}\boldsymbol{W}^{(1)} + \mathbf{1}\boldsymbol{b}^{(1)\top} \\ \boldsymbol{O} &= h(\boldsymbol{H})\boldsymbol{W}^{(2)} + \mathbf{1}\boldsymbol{b}^{(2)\top} \end{aligned}$$

where the activation function $h(\cdot)$ is applied element-wise and $\boldsymbol{O} \in \mathbb{R}^{n_l \times n_i}$ is the output. All the activation functions will be applied element-wise in the chapter. The MLP block is denoted as $\mathrm{mlp}^{(n_m,h)}(\cdot)$, where n_m and $h(\cdot)$ are the hidden size and the activation function respectively.

5.4 Network structures

5.4.1 Need for a large receptive field size

The difficulty of financial time series simulation is to model the long-range dependencies. Figure 5.6 shows the autocorrelation of absolute values of the returns of the S&P 500 index from May 2010 to November 2018. The autocorrelation is positive, inferring the asset returns admit phases of high activity and low activity in terms of price changes. This stylized fact is called volatility clustering. The positive correlation decays to almost 0 when the lag is greater than 100. This means we need a generator of RFS larger than 100 to model the positive correlation in this data. Authors in [101, 106] make use of the temporal convolutional network to increase the RFS, of which the structure is summarized in Appendix D.3. While in this chapter we use the attention layer to build a generator of a large RFS.

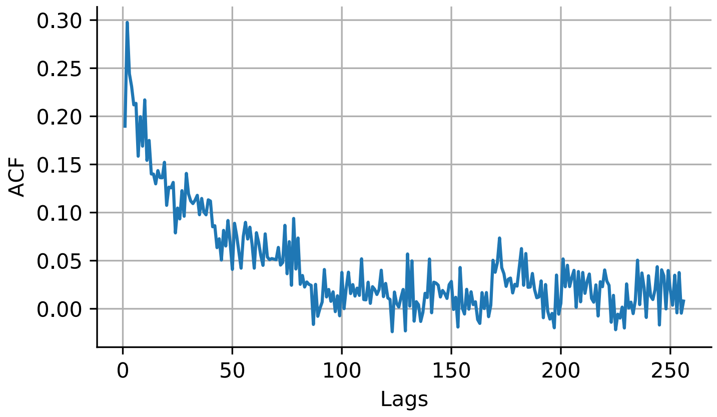


Figure 5.6: ACF of absolute values of the S&P 500 index returns.

5.4.2 Temporal attention GAN

The temporal attention GAN (TAGAN) is composed of a generator and a discriminator which are both convolutional networks with one self-attention layer. In the generator, we always use the causal layers while in the discriminator we only use the regular layers. This proposed model is the modification to the self-attention GAN [111] which has shown good performance in image generation. The hyper-parameters of `TAGAN` are listed in Table 5.1.

hyper-parameter	meaning
l	data length
f	receptive field size
d_n	noise channel
d	data channel
d_h	hidden channel in the generator
d_s	start hidden channel in the discriminator
d_m	max hidden channel in the discriminator
n_k	kernel size in convolutions
L_1^G/L_2^G	number of convolutional blocks before/after attention in the generator
L_1^D/L_2^D	number of convolutional blocks before/after attention in the discriminator
n_h	number of heads in attention
$n_{a,G}/n_{a,D}$	attention hidden size in the generator/discriminator
$h_G(\cdot)/h_D(\cdot)$	activation function in the generator/discriminator

Table 5.1: Hyper-parameters in `TAGAN`.

Suppose the input noise of the generator is $\boldsymbol{Z} \in \mathbb{R}^{(l+f-1)\times d_n}$ and the output sample is $\boldsymbol{Y} \in \mathbb{R}^{l\times d}$. With the notations of the network layers introduced in Section 5.3, the generator can be written as follows

$$\begin{aligned}
\boldsymbol{H}^{(0)} &= \mathrm{conv}_{\mathrm{C}}^{(1,d_h)}(\boldsymbol{Z}) \\
\boldsymbol{H}^{(j)} &= \mathrm{conv}_{\mathrm{C}}^{(n_k,d_h)} \circ h_G \circ \mathrm{conv}_{\mathrm{C}}^{(n_k,d_h)} \circ h_G \left(\boldsymbol{H}^{(j-1)}\right), \forall 1 \le j \le L_1^G \\
\boldsymbol{H}^{(L_1^G+1)} &= \mathrm{attn}_{\mathrm{C}}^{\left(n_{a,G},n_h,f-2(L_1+L_2)(n_k-1)\right)}\left(\boldsymbol{H}^{(L_1^G)}\right) \\
\boldsymbol{H}^{(j)} &= \mathrm{conv}_{\mathrm{C}}^{(n_k,d_h)} \circ h_G \circ \mathrm{conv}_{\mathrm{C}}^{(n_k,d_h)} \circ h_G \left(\boldsymbol{H}^{(j-1)}\right), \forall L_1^G + 2 \le j \le L_2^G + 1 \\
\boldsymbol{Y} &= \mathrm{conv}_{\mathrm{C}}^{(1,d)} \circ h_G \left(\boldsymbol{H}^{(L_2^G+1)}\right)
\end{aligned}$$

where $\circ$ means composition of the layers. The attention layer is embedded in the middle of the network to extend the receptive field and the convolutional layers are responsible for learning the local characteristics. The network structure of the generator of `TAGAN` is illustrated in Figure 5.7.

Let $\boldsymbol{Y} \in \mathbb{R}^{l\times d}$ denote either the real data or the fake data from the generator and $D(\boldsymbol{Y};\theta_D)$ be

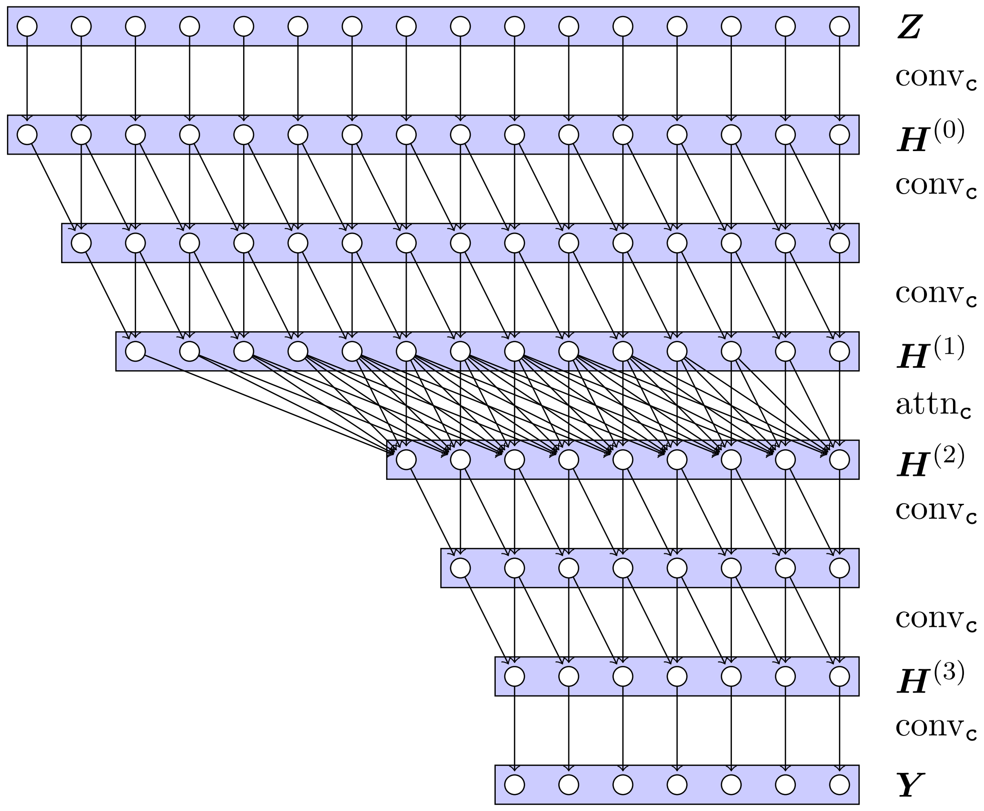

$\boldsymbol{Z}$
$\mathrm{conv}_{\mathtt{c}}$
$\boldsymbol{H}^{(0)}$
$\mathrm{conv}_{\mathtt{c}}$
$\mathrm{conv}_{\mathtt{c}}$
$\boldsymbol{H}^{(1)}$
$\mathrm{attn}_{\mathtt{c}}$
$\boldsymbol{H}^{(2)}$
$\mathrm{conv}_{\mathtt{c}}$
$\mathrm{conv}_{\mathtt{c}}$
$\boldsymbol{H}^{(3)}$
$\mathrm{conv}_{\mathtt{c}}$
$\boldsymbol{Y}$

Figure 5.7: Illustration of the generator of `TAGAN` with $n_k = 2$ and $L_2^G = L_2^G = 1$.

the output of the discriminator. For the notation simplicity, we let

$$\zeta_1^{(j)} = \mathrm{conv}_{\mathtt{r}}^{\left(n_k, \min(2^{j-1} d_s, d_m), 1\right)}$$
$$\zeta_2^{(j)} = \mathrm{conv}_{\mathtt{r}}^{\left(n_k, \min(2^{j-1} d_s, d_m), 2\right)}.$$

Then the discriminator can be written as follows:

$$\begin{aligned}
\boldsymbol{U}^{(0)} &= \boldsymbol{Y} \\
\boldsymbol{U}^{(j)} &= \zeta_2^{(j)} \circ h_D \circ \zeta_1^{(j)} \circ h_D \left(\boldsymbol{U}^{(j-1)}\right), \forall 1 \leq j \leq L_1^D \\
\boldsymbol{U}^{(L_1^D+1)} &= \operatorname{attn}_{\mathtt{r}}^{(n_{a,D},n_h)} \left(\boldsymbol{U}^{(L_1^D)}\right) \\
\boldsymbol{U}^{(j)} &= \zeta_2^{(j-1)} \circ h_D \circ \zeta_1^{(j-1)} \circ h_D \left(\boldsymbol{U}^{(j-1)}\right), \forall L_1^D + 2 \leq j \leq L_2^D + 1 \\
D(\boldsymbol{Y}; \theta_D) &= \sum_{i_1=1}^{n_l} \sum_{i_2=1}^{\min(2^{L_1^D+L_2^D-1} d_s, d_m)} h_D \left(U_{i_1,i_2}^{(L_2^D+1)}\right) w_{i_2}
\end{aligned}$$

where $\boldsymbol{w} \in \mathbb{R}^{\min(2^{L_1^D+L_2^D-1} d_s, d_m)}$ is a trainable weight used before the output in the discriminator. The discriminator is the classical architecture of the convolutional network which shrinks the length but increases the channel of hidden layers.

We apply batch normalization [60] before activation functions, spectrum normalization [86] to the layer weights, and residual connections [48] and skip connections [106] to the generator and discriminator in `TAGAN` during training to stabilize the statistics of generated samples and improve the performance. Batch normalization normalizes the output of the layers, and the spectrum normalization limits the spectral norm of the weight parameters, both of which reduce extreme values in the network. Residual connections and skip connections accelerate training when the networks become deep.

5.4.3 Temporal transformer GAN

The temporal transformer GAN (`TTGAN`) is composed of two transformer networks as its generator and discriminator. Similar models can be found in [64, 59]. A transformer consists of several attention layers with each layer followed by a two-layer MLP. We use the causal attention layers in the generator and the sparse attention layer in the discriminator. Each causal attention layer has a flexible RFS. The hyper-parameters of `TTGAN` are listed in Table 5.2.

Suppose the input noise of the generator is $\boldsymbol{Z} \in \mathbb{R}^{(l+f-1)\times d_n}$ and the output sample is $\boldsymbol{Y} \in \mathbb{R}^{l\times d}$.

hyper-parameter	meaning
l	data length
f	receptive field size
d_n	noise channel
d	data channel
d_h	hidden channel
n_h	number of heads in attention
n_a	attention hidden size
L	number of attention layers
$\{f_j\}_{j=1}^L$	the RFS of attentions in the generator
n_m	hidden size in the multi-layer perceptron
$h(\cdot)$	activation function

Table 5.2: Hyper-parameters in `TTGAN`.

With the notations of the network layers introduced in Section 5.3, the generator can be written as follows

$$\begin{aligned}
\boldsymbol{H}^{(0)} &= \mathrm{conv}_{\mathrm{C}}^{(1,d_h)}(\boldsymbol{Z}) \\
\boldsymbol{H}^{(j)} &= \mathrm{mlp}^{(n_m,h)} \circ \mathrm{attn}_{\mathrm{C}}^{(n_a,n_h,f_j)}\left(\boldsymbol{H}^{(j-1)}\right), \forall 1 \le j \le L \\
\boldsymbol{Y} &= \mathrm{conv}_{\mathrm{C}}^{(1,d)}\left(\boldsymbol{H}^{(L)}\right)
\end{aligned}$$

where $\circ$ means composition of the layers. The RFS of each attention layer needs to satisfy the equation $f - 1 = \sum_{j=1}^{L}(f_j - 1)$. This is because the length shrinkage of each attention layer is equal to RFS-1 and the total length shrinkage is equal to the sum of the shrinkage of all attention layers. The network structure of the generator of `TTGAN` is illustrated in Figure 5.8. At a first glance at Figures 5.7 and 5.8, it seems as if there is no difference between attention layers and convolutional layers. However, one should note that the formula of attention layers is different from that of convolutional layers and the RFS of attention layers can be much larger.

Let $\boldsymbol{Y} \in \mathbb{R}^{l\times d}$ denote either the real data or the fake data from the generator and $D(\boldsymbol{Y};\theta_D)$ be

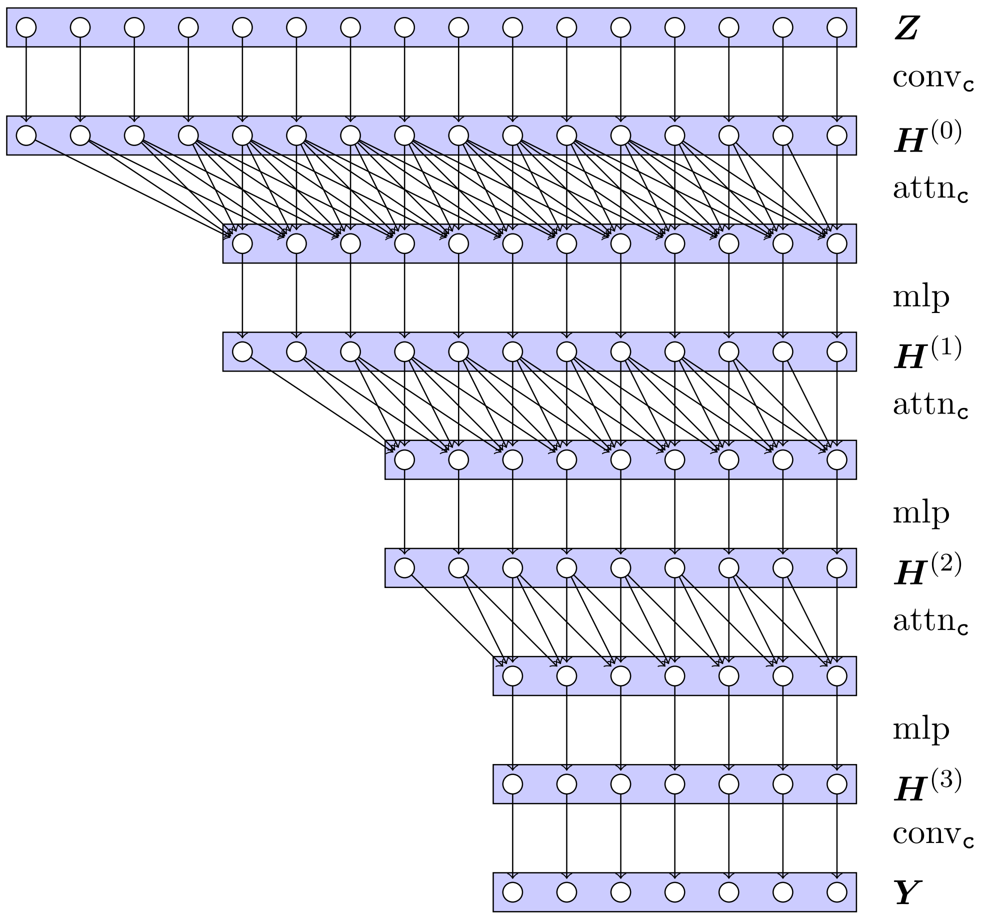


Figure 5.8: Illustration of the generator of `TTGAN` with $L = 3$ and $(f_1, f_2, f_3) = (5, 4, 3)$.

the output of the discriminator. The discriminator can be written as follows.

$$\begin{aligned}
\boldsymbol{U}^{(0)} &= \mathrm{conv}_{\mathtt{r}}^{(1,d_h,1)}(\boldsymbol{Y}) \\
\boldsymbol{U}^{(j)} &= \mathrm{mlp}^{(n_m,h)} \circ \mathrm{attn}_{\mathtt{s}}^{(n_a,n_h)}\left(\boldsymbol{U}^{(j-1)}\right), \forall 1 \le j \le L \\
D(\boldsymbol{Y};\theta_D) &= \sum_{i_1=1}^{l}\sum_{i_2=1}^{n_h} U^{(L)}_{i_1,i_2} W_{i_1,i_2}
\end{aligned}$$

where $\boldsymbol{W} \in \mathbb{R}^{l \times n_h}$ is a trainable weight used before the output in the discriminator. The discriminator is the classical architecture of the transformer encoder but with sparse attention.

We apply batch normalization [60] before each attention layer and MLP block, spectrum normalization [86] to the layer weights, residual connections [48] and skip connections [106] to the generator and discriminator in `TTGAN` during training to stabilize the statistics of generated samples and improve the performance.

5.5 Simulation of the S&P 500 index

In this section, we show the numerical results of the proposed networks for the S&P 500 index simulation.

5.5.1 Stylized facts and metrics

It has been summarized in [24] that the returns of the S&P 500 index and also the equities admit the following characteristics called *stylized facts*:

- Asset returns shows heavier tails than the normal distribution.
- Escalator up and elevator down: large drawdowns but not equally large upward movements are observed.
- Autocorrelations of (daily) asset returns are often insignificant.
- Volatility clustering: volatility displays a positive autocorrelation.
- The autocorrelation function (ACF) of absolute returns decays slowly as a function of the time lag.
- Leverage effect: volatility of an asset is negatively correlated with the returns of that asset.

Suppose we have a sequence of historical prices $p_{0:T_x} = \{p_t\}_{t=0}^{T_x}$. We take the log returns to be the real data, i.e. $x_{1:T_x} = \{x_t\}_{t=1}^{T_x}$ where $x_t = \ln(p_t/p_{t-1})$. Then we sample N sequences of returns from the GAN and denote them as $\{y_{1:T}^{(i)}\}_{i=1}^{N}$. The evaluation metrics are listed as follows:

- The Wasserstein-1 distance of daily and multi-day returns. Let $F_\tau^h(x)$ denote the empirical

CDF of the historical τ-day returns

$$\left\{\sum_{j=0}^{\tau-1} x_{t+j} : 1 \leq t \leq T_x - \tau + 1\right\}$$

and $F_\tau^g(x)$ the empirical CDF of the generated τ-day returns

$$\left\{\sum_{j=0}^{\tau-1} y_{t+j}^{(i)} : 1 \leq i \leq N, 1 \leq t \leq T_x - \tau + 1\right\}.$$

The Wasserstein-1 distance is given by

$$W_1^{(\tau)}(x_{1:T_x}, \{y_{1:T}^{(i)}\}_{i=1}^N) = \int_{\mathbb{R}} |F_\tau^g(x) - F_\tau^h(x)| dx. \tag{5.5}$$

We will calculate the Wasserstein-1 distance of 1-, 5-, 20-, 100- and 200-day returns.

- High order moment scores: skewness and kurtosis. We calculate

$$\left|\text{skew}(x_{1:T_x}) - \frac{1}{N} \sum_{1\leq i\leq N} \text{skew}\left(y_{1:T}^{(i)}\right)\right|$$

and

$$\left|\text{kurt}(x_{1:T_x}) - \frac{1}{N} \sum_{1\leq i\leq N} \text{kurt}\left(y_{1:T}^{(i)}\right)\right|$$

as the scores of skewness and kurtosis.

- Correlation scores. We look at the following four scores:
  - Autocorrelation of returns: $\text{ACF}_\tau(x_{1:T_x}) = \text{corr}(x_t, x_{t+\tau})$
  - Autocorrelation of absolute returns: $\text{ACF}_\tau^{(\text{abs})}(x_{1:T_x}) = \text{corr}(|x_t|, |x_{t+\tau}|)$
  - Autocorrelation of squared returns: $\text{ACF}_\tau^{(\text{sq})}(x_{1:T_x}) = \text{corr}(x_t^2, x_{t+\tau}^2)$
  - Leverage effect: $\text{Lev}_\tau(x_{1:T_x}) = \text{corr}(x_t, x_{t+\tau}^2)$

Each score is calculated for lag $1 \leq \tau \leq \delta$. Then we calculate

$$\sqrt{\sum_{1\leq\tau\leq\delta}\left(\mathrm{score}_\tau(x_{1:T_x}) - \frac{1}{N}\sum_{1\leq i\leq N}\mathrm{score}_\tau\left(y_{1:T}^{(i)}\right)\right)^2}$$

where score stands for ACF, $\mathrm{ACF}^{(\mathrm{abs})}$, $\mathrm{ACF}^{(\mathrm{sq})}$ and Lev.

Those metrics do not participate in the loss functions of the GANs during training, so they are suitable to evaluate the samples generated from the GANs.

5.5.2 Training

After we have the real data $x_{1:T_x}$, we apply a rolling window of length l to get the real dataset $\{x_{t:t+l-1}\}_{t=1}^{T_x-l+1}$ for training.

One important stylized fact of the asset returns is that tails of their distributions are heavier than that of the normal distribution. However, we usually use the normal distribution as the random noise input of GANs. Thus it is a question whether GANs are able to generate heavy-tailed distributions given normal noise. In [106], the authors use the inverse Lambert transform to make the returns closer to the normal distribution such that GANs do not need to generate heavy tails. But in our experiments, we still use the original returns as the training data. We would like to show that the proposed GANs can learn to generate heavy tails by themselves even if they are given normal noise input.

We also test the case of adding the cumulative sum of the returns as additional channels to the input of the discriminator. The output from the generator is a sequence $y_{1:l}$. Then the augmented input to the discriminator is $\tilde{\boldsymbol{Y}} \in \mathbb{R}^{l\times 2}$ where

$$\tilde{Y}_{t,j} = \begin{cases} y_{t,j}, & \text{if } j = 1 \\ \sum_{i_1=1}^{t} y_{i_1,j}, & \text{if } j = 2. \end{cases}$$

In this way, the input of the discriminator includes not only the returns, but also the log-prices starting from 0. The additional cumulative sum feature is added so that the discriminator can observe the returns over large intervals by taking the difference of the log-prices at two time points instead of taking the cumulative sum of the returns over long intervals.

In the rest of this section, we compare the performance of the proposed `TAGAN` and `TTGAN` with `QuantGAN` [106], which has shown good results for the S&P 500 index simulation using the temporal convolutional network [101]. The structure of `QuantGAN` is summarized in Appendix D.3. The data channel is $d = 1$. The data length is $l = 128$ for `TAGAN` and `TTGAN` while $l = 127$ for `QuantGAN`. The RFS is $f = 127$ for each GAN. The number of hidden channels is 64 in `TAGAN` and `TTGAN` and is 80 in `QuantGAN`. We calculate $M = 512$ simulated paths of length $T = 2560$ for evaluation. In the correlation scores, we let $\delta = 250$ in accordance with [106]. The loss for training `QuantGAN` is the loss of the original GAN in Equation (5.1), as used in their paper. We use the loss functions of the WGAN-GP in Equation (5.2) to train `TAGAN` and `TTGAN`.

5.5.3 Simulation of the medium kurtosis data

To make a fair comparison with `QuantGAN`, we use the same data in the paper of `QuantGAN` [106], which is the S&P 500 index daily data from May 1, 2009 to Nov 30, 2018 with $T_x = 2414$. The skewness of the data is -0.4667 and the kurtosis is 4.0648.

We test `TAGAN`, `TTGAN` and `QuantGAN` with and without the additional cumulative sum feature. We only present the cases of good performance since not every case works. The selected results of the three GANs without the additional cumulative sum feature, as well as `TTGAN` with the additional cumulative sum feature, are shown in Table 5.3. Here is the summary of results:

- The performance of the four candidates in Table 5.3 are close to each other and the difference is not significant.
- The cumulative sum feature only improves the performance of `TTGAN`. This means the transformer is more suitable to process features of different scales than the convolutional network. With the help of the cumulative sum feature, `TTGAN` reduces the Wasserstein-1 distance

score of 200-day returns, which agrees with the purpose of the additional cumulative sum feature.

scores	TAGAN	TTGAN (w/o cumsum)	TTGAN (w/ cumsum)	QuantGAN
$W_1^{(1)}$	4.569e-04	2.143e-04	3.319e-04	2.940e-04
$W_1^{(5)}$	9.764e-04	4.803e-04	7.367e-04	6.999e-04
$W_1^{(20)}$	2.677e-03	1.574e-03	2.234e-03	1.800e-03
$W_1^{(100)}$	3.363e-03	4.338e-03	3.311e-03	4.952e-03
$W_1^{(200)}$	1.016e-02	1.128e-02	7.281e-03	1.377e-02
skewness	5.284e-02	1.110e-01	1.752e-01	2.014e-01
kurtosis	5.248e-01	3.363e-01	1.237e-01	3.096e-01
ACF	3.450e-01	3.609e-01	3.628e-01	3.420e-01
ACF$^{(abs)}$	3.799e-01	3.727e-01	3.552e-01	3.742e-01
ACF$^{(sq)}$	3.300e-01	3.274e-01	3.238e-01	3.301e-01
Lev	3.248e-01	3.368e-01	3.376e-01	3.305e-01

Table 5.3: Scores of the S&P 500 index simulation given the medium kurtosis data from May 1, 2009 to Nov 30, 2018.

5.5.4 Simulation of the high kurtosis data

To further test the ability of the GANs to generate data with high (negative) skewness and high kurtosis, we also use the S&P 500 index daily data from May 1, 2009 to Dec 31, 2020 as the training data, which includes the drawdowns in 2020. The size of the dataset is $T_x = 2938$, the skewness is -0.8132 and the kurtosis is 15.1333.

We test `TAGAN`, `TTGAN` and `QuantGAN` for the dataset. For `TAGAN` and `QuantGAN`, no cumulative sum is used, while for `TTGAN`, we always use the cumulative sum feature. We also test `TTGAN` using batch normalization by default and its variant where we replace batch normalization with layer normalization [3]. Layer normalization normalizes the input values across the features, while batch normalization normalizes the input values across the batch dimension. The results are summarized in Table 5.4. The results of `TAGAN`, `TTGAN` with batch normalization and `QuantGAN` are further illustrated in Figure 5.11, 5.12 and 5.13. Here are the summary of the results:

- All the GANs perform well in fitting the distribution.

- Although layer normalization is more often used in the transformer architecture, we found that the layer normalization transformer fails to generate samples with high kurtosis in our tests.

- The convolution-based GANs, `TAGAN` and `QuantGAN`, are very sensitive to the autocorrelation curves, while `TTGAN` tends to smooth the autocorrelation curves. The fluctuations in the autocorrelation curves are likely to be caused by randomness in the market. The convolution-based GANs are preferred if we need to replicate the realization of randomness, while the transformer-based `TTGAN` is more suitable if we would like to filter out the randomness.

scores	`TAGAN`	`TTGAN` (BN)	`TTGAN` (LN)	`QuantGAN`
$W_1^{(1)}$	4.823e-04	4.907e-04	2.431e-04	2.605e-04
$W_1^{(5)}$	1.097e-03	1.525e-03	7.800e-04	9.530e-04
$W_1^{(20)}$	2.844e-03	4.963e-03	1.804e-03	2.840e-03
$W_1^{(100)}$	5.542e-03	8.432e-03	1.265e-02	6.347e-03
$W_1^{(200)}$	2.050e-02	1.774e-02	3.033e-02	1.797e-02
skewness	2.539e-01	4.883e-02	1.663e-01	3.870e-02
kurtosis	2.173e-01	2.121e-01	4.591e+00	5.674e-01
ACF	3.323e-01	4.067e-01	4.273e-01	3.437e-01
ACF$^{(abs)}$	3.792e-01	3.465e-01	3.740e-01	3.647e-01
ACF$^{(sq)}$	2.409e-01	2.496e-01	3.175e-01	2.415e-01
Lev	2.300e-01	2.957e-01	2.945e-01	2.319e-01

Table 5.4: Scores of the S&P 500 index simulation given the high kurtosis data from May 1, 2009 to Dec 31, 2020.

5.6 Simulation of the option surface

In this section, we show the numerical results of the proposed networks for the option surface simulation.

5.6.1 Formulation

Suppose we have N_K relative strikes

$$\mathcal{K} = \{K_1, K_1 + \Delta K, \ldots, K_1 + (N_K - 1)\Delta K\}$$

and N_M maturities

$$\mathcal{M} = \{M_1, M_2, \ldots, M_{N_M}\}.$$

Let $d = N_M \times N_K$. The real data is $\{\boldsymbol{x}_t \in \mathbb{R}^d\}_{t=1}^{T_x}$ with the elements $\boldsymbol{x}_t = (x_{t,j})_{j=1}^{d}$, where $x_{t,(j_1-1)N_M+j_2} = \ln \sigma_{t,(j_1-1)N_M+j_2}$ is the log-volatility at time t with the relative strike $K_{j_2} = K_1 + (j_2 - 1)\Delta K$ and the maturity M_{j_1}.

5.6.2 Training

Having the real data $\boldsymbol{x}_{1:T_x} = \{\boldsymbol{x}_t\}_{t=1}^{T_x}$, we apply a rolling window of length l to get the dataset $\{\boldsymbol{x}_{t:t+l-1}\}_{t=1}^{T_x-l+1}$ for training. The output $\{\hat{\boldsymbol{y}}_{1:T}^{(i)}\}_{i=1}^{N}$ from the GAN generator is not guaranteed to be arbitrage-free. We apply the method presented in Appendix D.1 to detect and remove arbitrage to obtain the arbitrage-free surface $\{\boldsymbol{y}_{1:T}^{(i)}\}_{i=1}^{N}$. In [105], the authors use the discrete local volatilities [13] to replace the implied volatilities when generating arbitrage-free option surfaces. The proposed networks are compatible with discrete local volatilities, but we still expect them to generate the implied volatilities and examine to what extent the outputs from the GANs violate the no-arbitrage condition.

The option volatility data is a high-dimensional data with high cross-correlation, so we could use principal component analysis (PCA) to reduce dimensionality. We perform PCA on the original data $\boldsymbol{x}_{1:T_x}$ and get the first $\tilde{d}$ principal components $\{\tilde{\boldsymbol{x}}_t \in \mathbb{R}^{\tilde{d}}\}_{t=1}^{T_x}$. To be more specific, suppose the real data matrix $\boldsymbol{X} \in \mathbb{R}^{T_x \times d}$ is $\boldsymbol{X} = \left(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{T_x}\right)^\top$. We get its singular value decomposition (SVD) as $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^\top$, and then take the first $\tilde{d}$ columns of $\boldsymbol{U}$ to be the principal components, i.e. $\boldsymbol{U}_{\cdot,1:\tilde{d}} = \left(\tilde{\boldsymbol{x}}_1, \tilde{\boldsymbol{x}}_2, \ldots, \tilde{\boldsymbol{x}}_{T_x}\right)^\top$. Next, we apply a rolling window of length l to get the real dataset $\{\tilde{\boldsymbol{x}}_{t:t+l-1}\}_{t=1}^{T_x-l+1}$ for training. The GAN generator is responsible for generating the first $\tilde{d}$ principal components $\{\tilde{\boldsymbol{y}}_{1:T}^{(i)}\}_{i=1}^{N}$, and they are used to recover the log-volatility surfaces $\{\hat{\boldsymbol{y}}_{1:T}^{(i)}\}_{i=1}^{N}$ through reverse PCA, where $\hat{\boldsymbol{y}}_t^{(i)} = \boldsymbol{V}_{\cdot,1:\tilde{d}}\boldsymbol{D}_{1:\tilde{d},1:\tilde{d}}\tilde{\boldsymbol{y}}_t^{(i)}$. Finally we apply the method in Appendix D.1 to get the arbitrage-free surfaces $\{\boldsymbol{y}_{1:T}^{(i)}\}_{i=1}^{N}$. This process is summarized in Figure 5.9.

Since we find it is helpful to include both returns and log-prices in the S&P 500 index simulation, we think it could also be helpful when the differences of the log-volatilities (called log-

$$\begin{tikzcd}
\boldsymbol{x}_{1:T_x} \arrow[r, "\text{PCA}"] & \tilde{\boldsymbol{x}}_{1:T_x} \arrow[r, "\text{rolling}", "\text{window}"'] & \{\tilde{\boldsymbol{x}}_{t:t+l-1}\}_{t=1}^{T_x-l+1} \arrow[d, "\text{simulate}"'] \\
\{\boldsymbol{y}_{1:T}^{(i)}\}_{i=1}^{N} & \{\hat{\boldsymbol{y}}_{1:T}^{(i)}\}_{i=1}^{N} \arrow[l, "\text{remove}"', "\text{arbitrage}"] & \{\tilde{\boldsymbol{y}}_{1:T}^{(i)}\}_{i=1}^{N} \arrow[l, "\text{reverse}"', "\text{PCA}"]
\end{tikzcd}$$

Figure 5.9: Pipeline of GANs using PCA.

volatility returns hereafter) are used as the additional feature. The output from the generator is a sequence $\hat{\boldsymbol{y}}_{1:l}$. Then the augmented input to the discriminator is $\bar{\boldsymbol{Y}} \in \mathbb{R}^{l \times 2d}$ where

$$\bar{Y}_{t,j} = \begin{cases} \hat{y}_{t,j}, & \text{if } 1 \le j \le d \\ \hat{y}_{t,j-d} - \hat{y}_{t-1,j-d}, & \text{if } 2 \le t \le l \text{ and } d+1 \le j \le 2d \\ 0, & \text{if } t = 1 \text{ and } d+1 \le j \le 2d. \end{cases}$$

In this way, the input of the discriminator includes both the log-volatilities and the log-volatility returns.

To summarize, we have three choices to train the GANs:

- Use the log-volatility surfaces as the real data. The generators simulate the log-volatility surfaces.
- Use the principal components of log-volatility surfaces as the real data. The generators simulate the principal components.
- Use the log-volatility surfaces and their returns as the real data. The generators simulate the log-volatility surfaces.

5.6.3 Stylized facts and metrics

Here are some stylized facts of the option surface summarized in [107].

- The volatility smile. Deep in-the-money and out-of-the-money volatility are generally higher than at-the-money volatility.

- Volatilities have high serial autocorrelation.

- Volatilities show high cross-correlation. The correlation matrix of the log-volatilities of different relative strikes and maturities in the S&P 500 index option data in Section 5.6.4 is shown in Figure 5.10. Higher cross-correlation is observed for proximate relative strikes and maturities. Volatilities of longer maturities have higher cross-correlation than volatilities of shorter maturities.

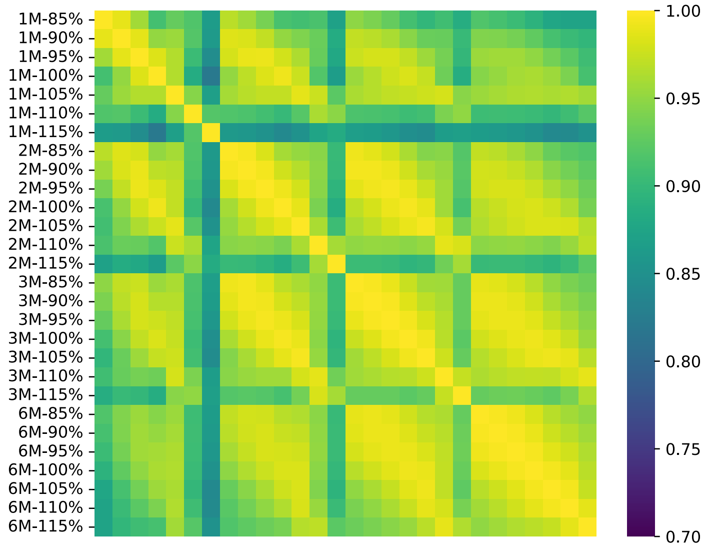


Figure 5.10: Cross-correlation matrix of the log-volatilities of the S&P 500 index options. The label means ‘maturity - relative strike’.

Based on the stylized facts, the evaluation metrics are listed as follows:

- The Wasserstein-1 distance of distribution of volatilities

$$\frac{1}{d}\sum_{j=1}^{d} W_1^{(1)}\left(\{x_{t,j}\}_{t=1}^{T_x}, \{\{y_{t,j}^{(i)}\}_{t=1}^{T}\}_{i=1}^{N}\right)$$

  where the Wasserstein-1 distance is already defined in Equation (5.5).

- High order moment scores of skewness

$$\frac{1}{d}\sum_{j=1}^{d}\left|\text{skew}(x_{1:T_x,j}) - \frac{1}{N}\sum_{1\le i\le N}\text{skew}\left(y_{1:T,j}^{(i)}\right)\right|$$

  and kurtosis

$$\frac{1}{d}\sum_{j=1}^{d}\left|\text{kurt}(x_{1:T_x,j}) - \frac{1}{N}\sum_{1\le i\le N}\text{kurt}\left(y_{1:T,j}^{(i)}\right)\right|.$$

- Autocorrelation score of the series and returns. Define

$$\text{ACF}_\tau^{(\text{r})}(x_{1:T_x}) = \text{ACF}_\tau(x_{2:T_x} - x_{1:T_x-1}).$$

  We calculate

$$\frac{1}{d}\sum_{j=1}^{d}\sqrt{\sum_{1\le\tau\le\delta}\left(\text{score}_\tau(x_{1:T_x,j}) - \frac{1}{N}\sum_{1\le i\le N}\text{score}_\tau\left(y_{1:T,j}^{(i)}\right)\right)^2}$$

  where score stands for ACF and ACF$^{(\text{r})}$.

- Cross-correlation score. Let $\Sigma_{\boldsymbol{x}} \in \mathbb{R}^{d\times d}$ be the cross-correlation matrix of $\{\boldsymbol{x}_t, 1 \le t \le T_x\}$ and $\Sigma_{\boldsymbol{y}} \in \mathbb{R}^{d\times d}$ be the cross-correlation matrix of $\{\boldsymbol{y}_t^{(i)}, 1 \le t \le T_x, 1 \le i \le N\}$. Then the score is defined to the Frobenius norm of the difference $\|\Sigma_{\boldsymbol{x}} - \Sigma_{\boldsymbol{y}}\|_F$.

- Arbitrage rate. The score is calculated as the percentage of the outputs from the GANs $\hat{\boldsymbol{y}}_{1:T}^{(i)}$ that violate the no-arbitrage condition

$$\#\{(i,t) : \hat{\boldsymbol{y}}_t^{(i)} \text{ that violates the no-arbitrage condition}, 1 \le i \le N, 1 \le t \le T\}/(NT).$$

  This is a score that shows how well the GAN can learn the no-arbitrage condition.

5.6.4 Data and results

We use the daily data of the S&P 500 index options from Jan 02, 2009 to Oct 30, 2020 as the real data. The maturities are

$$\mathcal{M} = \{\text{1-month, 2-month, 3-month, 6-month}\}$$

and the relative strikes are

$$\mathcal{K} = \{85\%, 90\%, 95\%, 100\%, 105\%, 110\%, 115\%\}.$$

The data channel is $d = 28$ and the sequence length is $T_x = 2979$.

We compare the performance of the proposed `TAGAN` and `TTGAN` with `QuantGAN` [106] and try both PCA and the additional log-volatility return feature. The data length is $l = 128$ for `TAGAN` and `TTGAN` while $l = 127$ for `QuantGAN`. The RFS is $f = 383$ for each GAN. We let $\tilde{d} = 10$ for PCA. The number of hidden channels is 64 in `TAGAN` and `TTGAN` and is 80 in `QuantGAN`. We calculate $M = 512$ simulated paths of length $T = 2560$ for evaluation. In the correlation scores, we let $\delta = 64$. The loss for training `QuantGAN` is the loss of the original GAN in Equation (5.1). We use the loss functions of the WGAN-GP in Equation (5.2) to train `TAGAN` and `TTGAN`.

The results of `TAGAN` with and without PCA, `TTGAN` with and without the return feature, and `QuantGAN`, are summarized in Table 5.5. The good candidates, which are `TAGAN` with PCA, `TTGAN` with the return feature, and `QuantGAN` are further illustrated in Figure 5.14, 5.15 and 5.16. Here are some key points of results:

- Only `TTGAN` is improved by the additional return feature. It is not a surprise to see `TTGAN` can accept the additional return feature, since it accepts both log-prices and log returns for the S&P 500 index simulation. The additional return feature improves the score of autocorrelation and cross-correlation, and facilitates the GAN to learn the no-arbitrage condition.

- Only `TAGAN` is improved by PCA. If a GAN is able to generate option surfaces by means of principle components, that will significantly reduce the score of cross-correlation and reduce the rate that the output needs to be modified by the no-arbitrage condition.

- In Figure 5.17, we show examples of autocorrelation of log-volatility returns from the three GANs. There are huge fluctuations in the autocorrelation of `QuantGAN`. Also, some fluctuations are observed at the small time lags in the autocorrelation of `TAGAN`. In contrast, the autocorrelation of `TTGAN` is flat. It means the attention layer is better at generating sequences with smooth autocorrelation, which matches the results of the S&P 500 index simulation.

scores	`TAGAN` (w/o PCA)	`TAGAN` (w/ PCA)	`TTGAN` (w/o returns)	`TTGAN` (w/ returns)	`QuantGAN`
$W_1^{(1)}$	1.788e-02	1.651e-02	1.239e-02	1.512e-02	1.355e-02
skewness	2.434e-01	2.450e-01	2.077e-01	8.204e-02	8.560e-02
kurtosis	6.052e-01	2.710e-01	5.212e-01	5.607e-01	4.065e-01
ACF	3.065e-01	3.444e-01	4.359e-01	1.845e-01	1.754e-01
ACF$^{(r)}$	3.601e-01	2.580e-01	3.727e-01	2.667e-01	8.683e-01
cross-corr	4.883e-01	1.016e-01	6.027e-01	2.618e-01	2.284e-01
arbitrage rate	30.10%	1.55%	21.16%	8.88%	12.86%

Table 5.5: Scores of the S&P 500 index option surface simulation.

5.7 Conclusion

In this chapter, we first define the generative model of time series, distinguish it from the generator of fixed dimension distributions. We then propose two GANs, the temporal attention GAN and the temporal transformer GAN, based on the causal attention layer, which is able to increase the receptive field size without introducing more parameters. We have successfully trained the temporal transformer GAN using around 3000 samples of financial time series with the help of sparse attention, despite the fact that both GANs and transformers are notoriously known for being difficult to train.

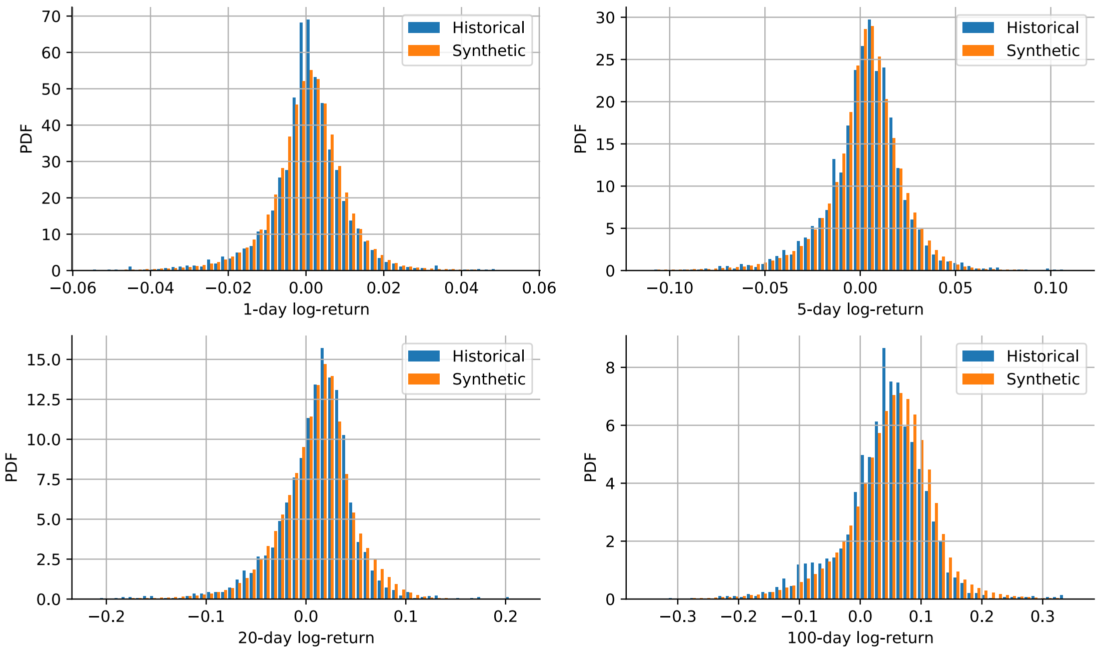
In the numerical experiments, we compare the two proposed GANs with `QuantGAN` for the stock index and option surface simulation and evaluate the results with the scores based on the stylized facts. The proposed GANs are able to replicate the distribution, the heavy tails and the long-range dependencies, as well as the cross-correlation in the multivariate case. Specifically, the attention-based GANs show the following advantages:

- The `TTGAN` tends to generate smoother autocorrelation of returns. However, the convolution-based `QuantGAN` tends to overfit autocorrelation curves. For option surface simulation, the `QuantGAN` even fails to replicate the autocorrelation of volatility returns. The `TAGAN`, as a mixture of convolutions and attention, lies between `TTGAN` and `QuantGAN`.
- The transformer discriminator in `TTGAN` is more flexible in a way that can accept both level and return features. We can make use of its ability to process features of different scales to improve the performance of GANs.
- The `TAGAN` is able to learn and generate samples in the space of principal components, which makes it possible to simulate time series in higher-dimensional spaces utilizing PCA.
- The receptive field size of the attention-based GANs is not bounded by the number of parameters or the network depth. This is useful especially when the size of real data is limited and a large number of parameters would lead to overfitting.

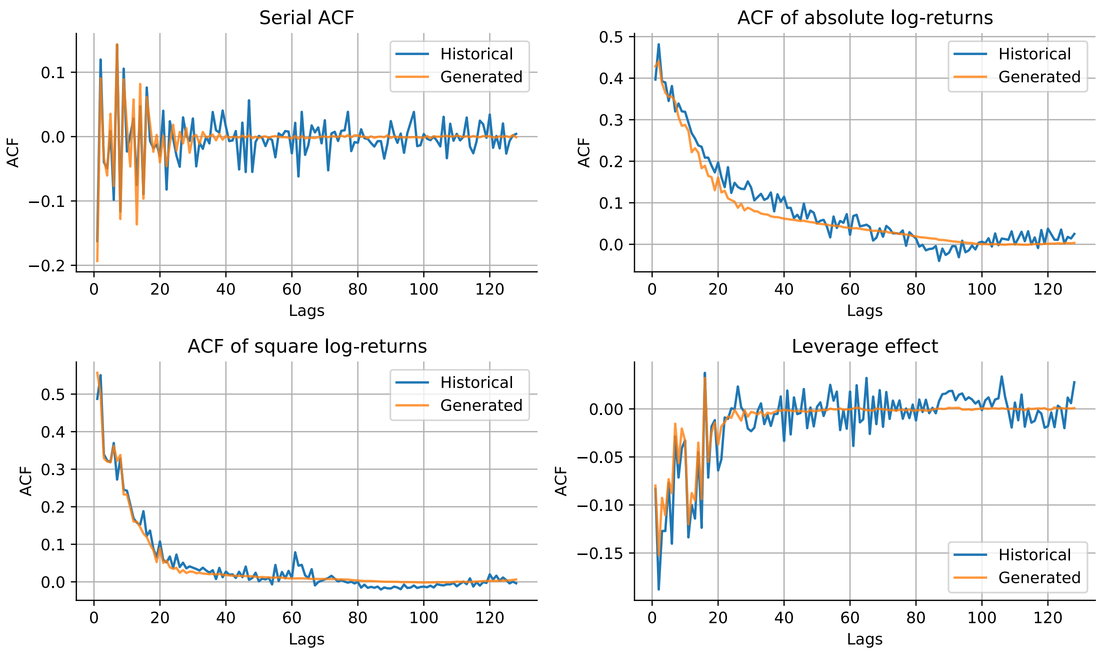
The generative models discussed in the chapter are all unconditional models, which generate time series given noise series. In the future, it would be interesting to compare the performance of unconditional models and conditional models, which generate future time series given noise as well as historical time series. The conditional models are trickier for the following reasons:

- The conditional models need to learn the conditional distribution given history time series, which is generally more complex than the unconditional distribution.
- The input of the unconditional model is random noise generated during training. Thus, the unconditional model would not memorize the input. However, the conditional model can easily remember real data and perform extremely well when real data is used as the condition input. When the conditional model uses the time series generated by itself as the condition

input and tries to prolong the generated time series, its performance could be much worse. For that reason, we need additional techniques to deal with overfitting.

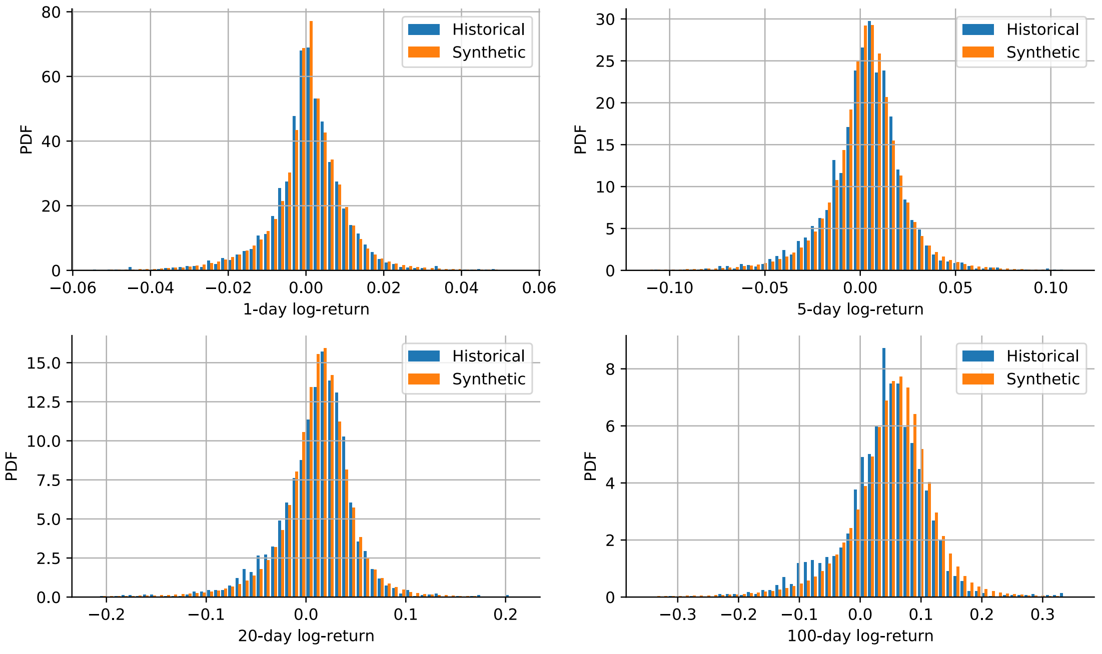


(a)

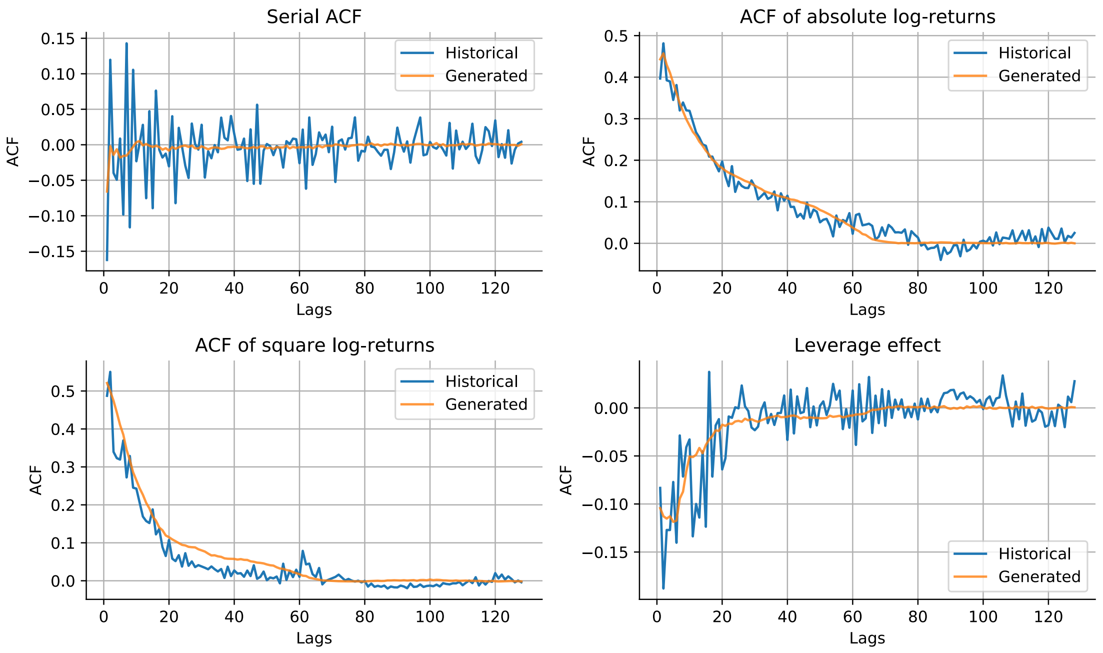


(b)

Figure 5.11: TAGAN simulation results of the S&P 500 index. (a) Comparison of the generated and historical densities of log returns. (b) Comparison of the generated mean autocorrelation and historical autocorrelation of daily log returns.

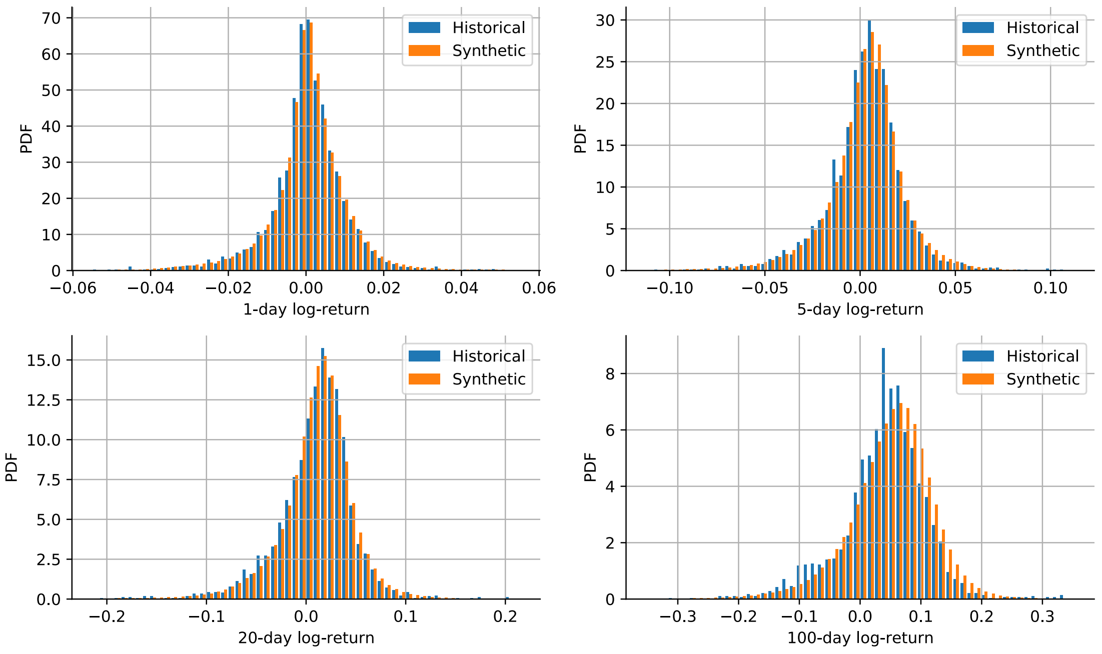


(a)

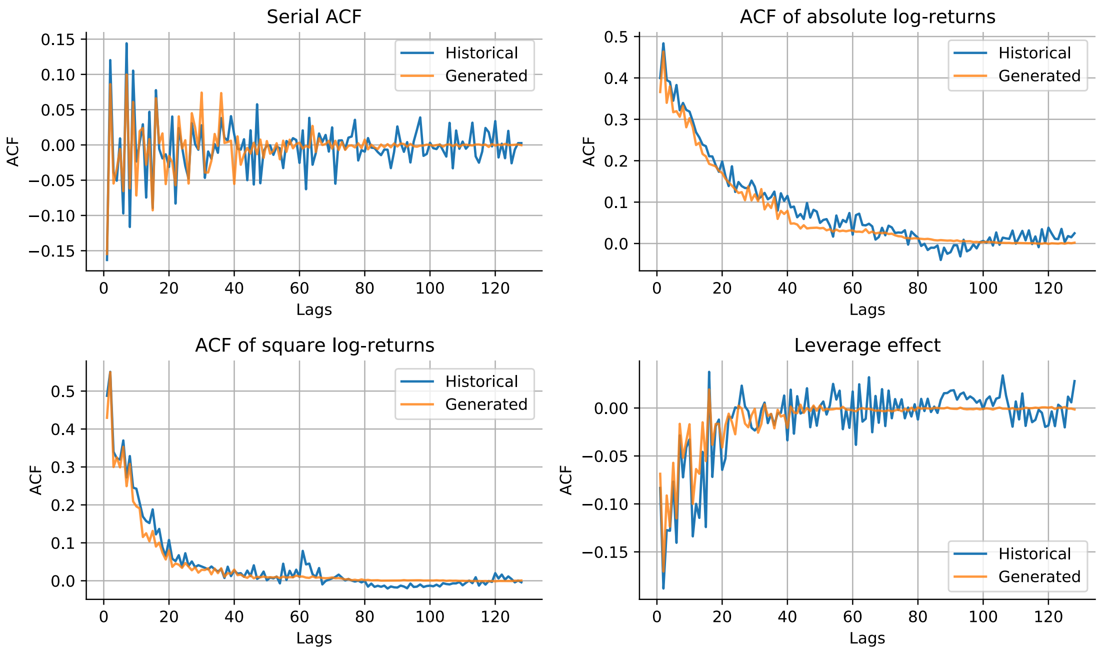


(b)

Figure 5.12: TTGAN simulation results of the S&P 500 index. (a) Comparison of the generated and historical densities of log returns. (b) Comparison of the generated mean autocorrelation and historical autocorrelation of daily log returns.

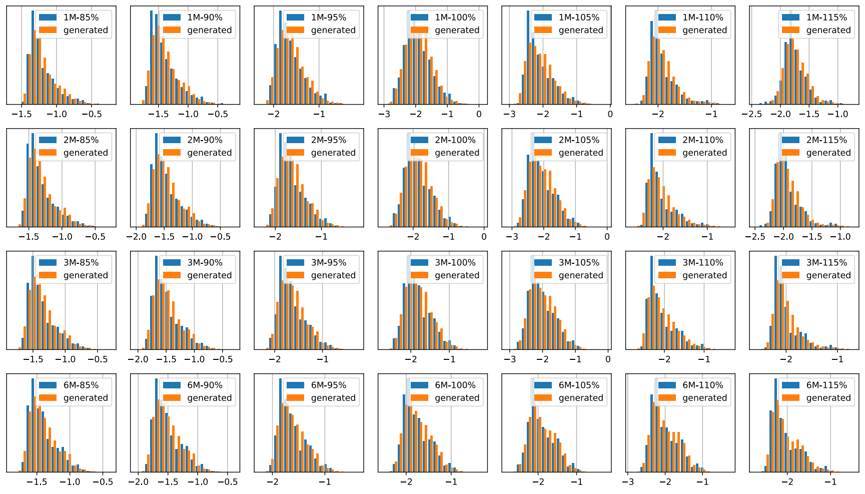


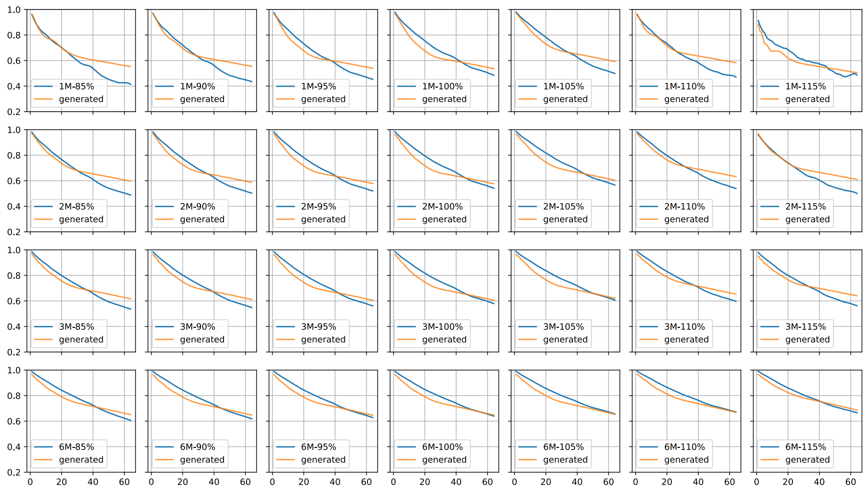
(a)



(b)

Figure 5.13: `QuantGAN` simulation results of the S&P 500 index. (a) Comparison of the generated and historical densities of log returns. (b) Comparison of the generated mean autocorrelation and historical autocorrelation of daily log returns.


(a)


(b)

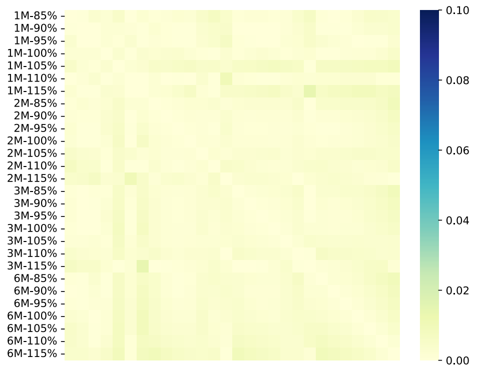
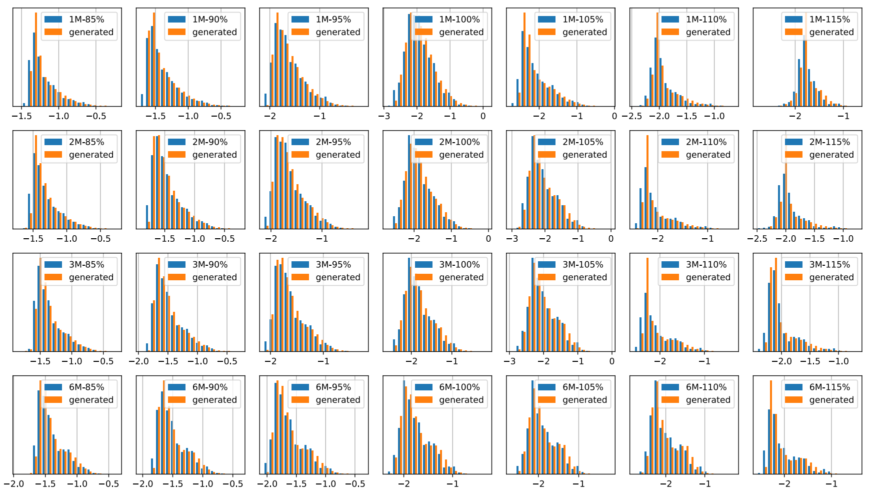
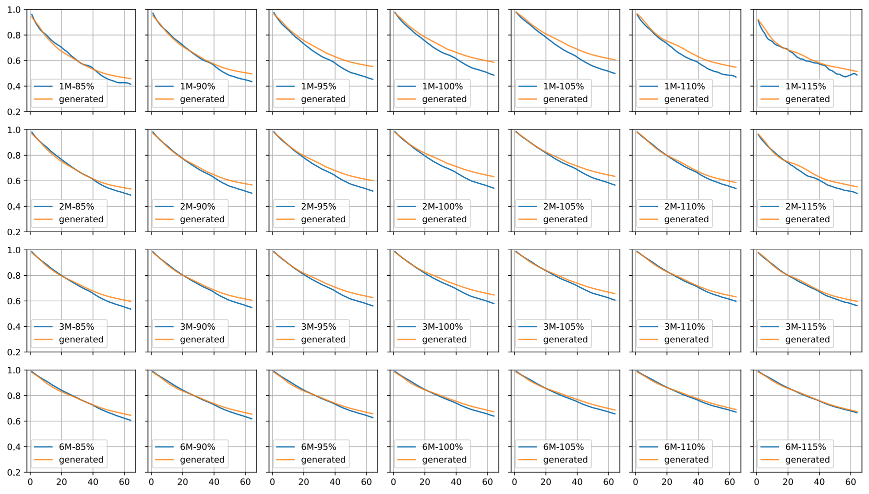

(c)

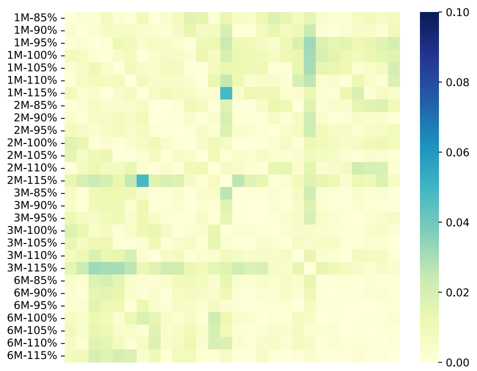
Figure 5.14: TAGAN simulation results of the S&P 500 index options. (a) Comparison of the generated and historical densities of log-volatilities. (b) Comparison of the generated mean autocorrelation and historical autocorrelation of log-volatilities. (c) Difference of the generated and historical cross-correlation of log-volatilities $|\Sigma_x - \Sigma_y|$.



(a)

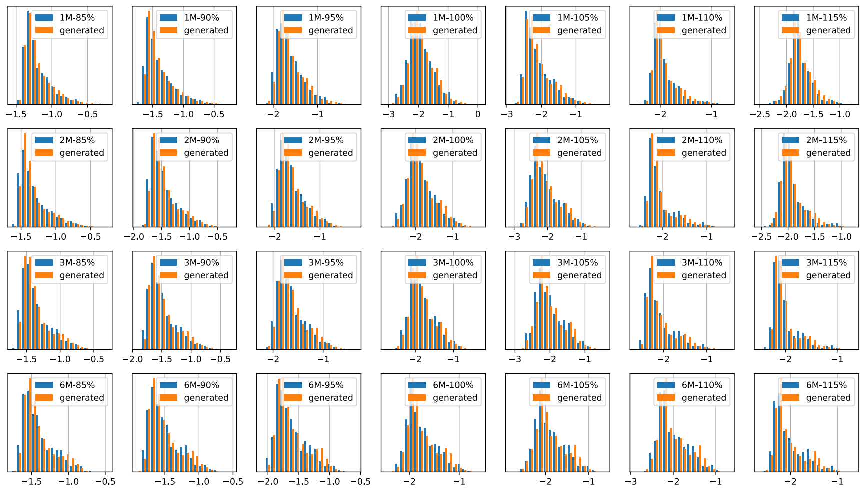


(b)

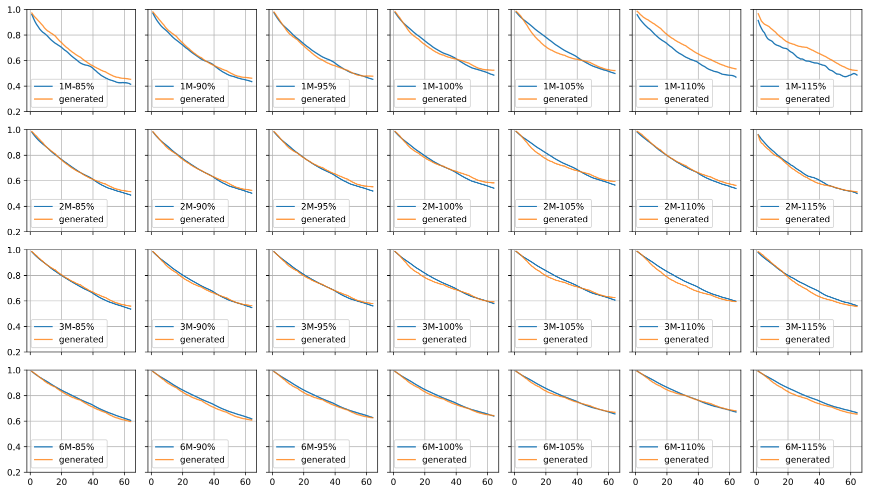


(c)

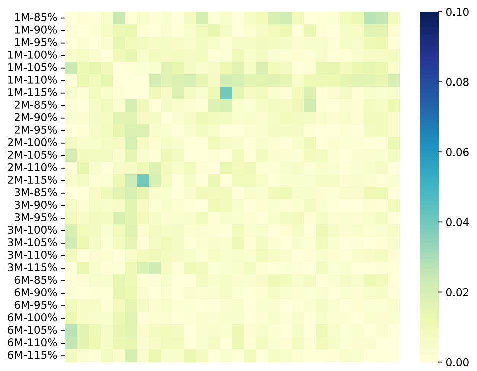
Figure 5.15: `TTGAN` simulation results of the S&P 500 index options. (a) Comparison of the generated and historical densities of log-volatilities. (b) Comparison of the generated mean autocorrelation and historical autocorrelation of log-volatilities. (c) Difference of the generated and historical cross-correlation of log-volatilities $|\Sigma_x - \Sigma_y|$.



(a)

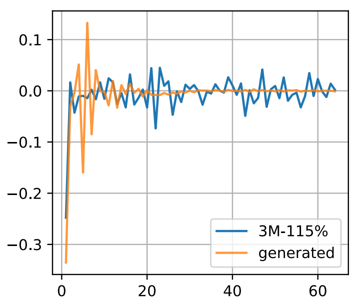


(b)

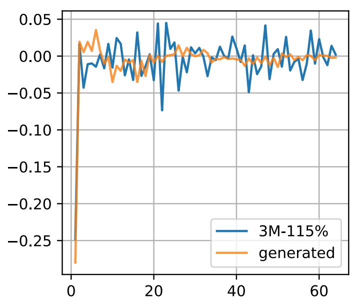


(c)

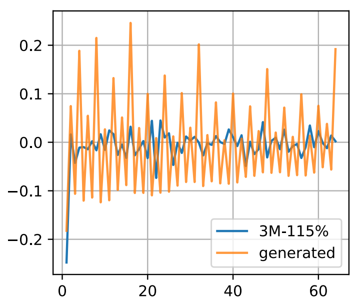
Figure 5.16: `QuantGAN` simulation results of the S&P 500 index options. (a) Comparison of the generated and historical densities of log-volatilities. (b) Comparison of the generated mean autocorrelation and historical autocorrelation of log-volatilities. (c) Difference of the generated and historical cross-correlation of log-volatilities $|\Sigma_x - \Sigma_y|$.



(a)



(b)



(c)

Figure 5.17: Example of the generated mean autocorrelation and historical autocorrelation of log-volatility returns of the S&P 500 index options. (a) `TAGAN`. (b) `TTGAN`. (c) `QuantGAN`.

References

[1] N. Achtsis, R. Cools, and D. Nuyens, “Conditional sampling for barrier option pricing under the heston model,” in *Monte Carlo and Quasi-Monte Carlo Methods 2012*, Springer, 2013, pp. 253–269.

[2] M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein GAN,” *arXiv:1701.07875*, Dec. 2017, arXiv: 1701.07875.

[3] J. L. Ba, J. R. Kiros, and G. E. Hinton, “Layer normalization,” *arXiv preprint arXiv:1607.06450*, 2016.

[4] D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” *arXiv preprint arXiv:1409.0473*, 2014.

[5] O. E. Barndorff-Nielsen, “Processes of normal inverse Gaussian type,” *Finance and stochastics*, vol. 2, no. 1, pp. 41–68, 1997.

[6] G. Barone-Adesi and R. E. Whaley, “Efficient analytic approximation of American option values,” *The Journal of Finance*, vol. 42, no. 2, pp. 301–320, 1987.

[7] C. Bayer, B. Horvath, A. Muguruza, B. Stemper, and M. Tomas, “On deep calibration of (rough) stochastic volatility models,” *arXiv preprint arXiv:1908.08806*, 2019.

[8] C. Beck, S. Becker, P. Grohs, N. Jaafari, and A. Jentzen, “Solving stochastic differential equations and kolmogorov equations by means of deep learning,” *arXiv preprint arXiv:1806.00421*, 2018.

[9] L. Bergomi, “Smile dynamics III,” *Available at SSRN 1493308*, 2008.

[10] F. Black and M. Scholes, “The pricing of options and corporate liabilities,” *Journal of Political Economy*, vol. 81, no. 3, pp. 637–654, 1973. eprint: `https://doi.org/10.1086/260062`.

[11] T. Bollerslev, “Generalized autoregressive conditional heteroskedasticity,” *Journal of Econometrics*, vol. 31, no. 3, pp. 307–327, Apr. 1986.

[12] A. Brock, J. Donahue, and K. Simonyan, “Large scale GAN training for high fidelity natural image synthesis,” *arXiv preprint arXiv:1809.11096*, 2018.

[13] H. Buehler and E. Ryskin, “Discrete local volatility for large time steps (extended version),” *Available at SSRN 2642630*, 2017.

[14] N. Cai and S. G. Kou, "Option pricing under a mixed-exponential jump diffusion model," *Management Science*, vol. 57, no. 11, pp. 2067–2081, Nov. 2011.

[15] P. Carr, H. Geman, D. B. Madan, and M. Yor, "The fine structure of asset returns: An empirical investigation," *The Journal of Business*, vol. 75, no. 2, pp. 305–333, Apr. 2002.

[16] P. Carr and A. Hirsa, "Forward evolution equations for knock-out options," in *Advances in Mathematical Finance*, M. C. Fu, R. A. Jarrow, J.-Y. J. Yen, and R. J. Elliott, Eds. Boston, MA: Birkhäuser Boston, 2007, pp. 195–217, ISBN: 978-0-8176-4545-8.

[17] P. Carr and D. Madan, "Option valuation using the fast Fourier transform," *The Journal of Computational Finance*, vol. 2, no. 4, pp. 61–73, 1999.

[18] P. Carr and A. Mayo, "On the numerical evaluation of option prices in jump diffusion processes," *The European Journal of Finance*, vol. 13, no. 4, pp. 353–372, Jun. 2007.

[19] A. Cartea and D. del Castillo-Negrete, "Fractional diffusion models of option prices in markets with jumps," *Physica A: Statistical Mechanics and its Applications*, vol. 374, no. 2, pp. 749–763, 2007.

[20] Z. Che, S. Purushotham, G. Li, B. Jiang, and Y. Liu, "Hierarchical deep generative models for multi-rate multivariate time series," in *International Conference on Machine Learning*, 2018, pp. 784–793.

[21] C. Chiarella, B. Kang, and G. H. Meyer, "The evaluation of barrier option prices under stochastic volatility," *Computers & Mathematics with Applications*, vol. 64, no. 6, pp. 2034–2048, 2012.

[22] R. Child, S. Gray, A. Radford, and I. Sutskever, "Generating long sequences with sparse transformers," *arXiv preprint arXiv:1904.10509*, 2019.

[23] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[24] R. Cont, "Empirical properties of asset returns: Stylized facts and statistical issues," *Quantitative Finance*, vol. 1, no. 2, pp. 223–236, 2001.

[25] R. Cont and E. Voltchkova, "A finite difference scheme for option pricing in jump diffusion and exponential Lévy models," *SIAM Journal on Numerical Analysis*, vol. 43, no. 4, pp. 1596–1626, Jan. 2005.

[26] S. Cuomo, V. Di Somma, E. di Lorenzo, and G. Toraldo, "A sequential monte carlo approach for the pricing of barrier option in a stochastic volatility model," *Electronic Journal of Applied Statistical Analysis*, vol. 13, no. 1, pp. 128–145, 2020.

[27] G. Daras, A. Odena, H. Zhang, and A. G. Dimakis, "Your local GAN: Designing two dimensional local attention mechanisms for generative models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[28] F. De Meer Pardo, "Enriching financial datasets with generative adversarial networks," PhD thesis, Master's thesis, Delft University of Technology, the Netherlands, 2019.

[29] G. Di Cerbo, A. Hirsa, and A. Shayaan, "Regularized generative adversarial network," *arXiv preprint arXiv:2102.04593*, 2021.

[30] C. Dugas, Y. Bengio, F. Bélisle, C. Nadeau, and R. Garcia, "Incorporating second-order functional knowledge for better option pricing," *Advances in neural information processing systems*, vol. 13, pp. 472–478, 2000.

[31] S. Elfwing, E. Uchibe, and K. Doya, "Sigmoid-weighted linear units for neural network function approximation in reinforcement learning," 2017. arXiv: `1702.03118 [cs.LG]`.

[32] C. Esteban, S. L. Hyland, and G. Rätsch, "Real-valued (medical) time series generation with recurrent conditional gans," *arXiv preprint arXiv:1706.02633*, 2017.

[33] F. Fang and C. W. Oosterlee, "Pricing early-exercise and discrete barrier options by fourier-cosine series expansions," *Numerische Mathematik*, vol. 114, no. 1, pp. 27–62, Nov. 2009.

[34] R. Ferguson and A. Green, "Deeply learning derivatives," *arXiv preprint arXiv:1809.02233*, 2018.

[35] R. Fu, J. Chen, S. Zeng, Y. Zhuang, and A. Sudjianto, "Time series simulation by conditional generative adversarial net," *arXiv preprint arXiv:1904.11419*, 2019.

[36] W. Fu and A. Hirsa, "Fast pricing of american options under variance gamma," *Journal of Computational Finance*, vol. 25, no. 1, pp. 29–49, 2021.

[37] W. Fu and A. Hirsa, "An unsupervised deep learning approach to solving partial integro-differential equations," *Quantitative Finance*, 2022. eprint: `https://doi.org/10.1080/14697688.2022.2057870`.

[38] H. Funahashi and T. Higuchi, "An analytical approximation for single barrier options under stochastic volatility models," *Annals of Operations Research*, vol. 266, no. 1, pp. 129–157, 2018.

[39] N. Ganesan, Y. Yu, and B. Hientzsch, "Pricing barrier options with deepbsdes," *arXiv preprint arXiv:2005.10966*, 2020.

[40] I. V. Girsanov, "On transforming a certain class of stochastic processes by absolutely continuous substitution of measures," *Theory of Probability & Its Applications*, vol. 5, no. 3, pp. 285–301, 1960. eprint: `https://doi.org/10.1137/1105027`.

[41] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, vol. 27, Curran Associates, Inc., 2014.

[42] C. Guardasoni and S. Sanfelici, "Fast numerical pricing of barrier options under stochastic volatility and jumps," *SIAM Journal on Applied Mathematics*, vol. 76, no. 1, pp. 27–57, 2016.

[43] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved Training of Wasserstein GANs," in *Advances in Neural Information Processing Systems*, vol. 30, Curran Associates, Inc., 2017.

[44] X. Guo and Y. Li, "Valuation of American options under the CGMY model," *Quantitative Finance*, vol. 16, no. 10, pp. 1529–1539, Oct. 2016.

[45] P. S. Hagan, D. Kumar, A. S. Lesniewski, and D. E. Woodward, "Managing smile risk," *The Best of Wilmott*, vol. 1, pp. 249–296, 2002.

[46] J. Han, A. Jentzen, and E Weinan, "Solving high-dimensional partial differential equations using deep learning," *Proceedings of the National Academy of Sciences*, vol. 115, no. 34, pp. 8505–8510, 2018.

[47] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," 2015. arXiv: `1502.01852 [cs.CV]`.

[48] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[49] S. L. Heston, "A closed-form solution for options with stochastic volatility with applications to bond and currency options," *The review of financial studies*, vol. 6, no. 2, pp. 327–343, 1993.

[50] A. Hirsa, *Computational Methods in Finance*. CRC Press, 2016.

[51] A. Hirsa and P. Carr, "Why be backward? Forward equations for American options," *Risk*, vol. 16, no. 1, pp. 103–107, 2003.

[52] A. Hirsa, G. Courtadon, and D. B. Madan, "The effect of model risk on the valuation of barrier options," *The Journal of Risk Finance*, 2003.

[53] A. Hirsa, T. Karatas, and O. Amir, “Supervised deep neural networks (DNNs) for pricing/calibration of vanilla/exotic options under various different processes,” 2019.

[54] A. Hirsa and D. B. Madan, “Pricing American options under variance gamma,” *Journal of Computational Finance*, vol. 7, no. 2, pp. 63–80, 2003.

[55] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[56] M. Hofert, A. Prasad, and M. Zhu, “Multivariate time-series modeling with generative neural networks,” *Econometrics and Statistics*, vol. 23, pp. 147–164, 2022.

[57] B. Horvath, A. Muguruza, and M. Tomas, “Deep learning volatility: A deep neural network perspective on pricing and calibration in (rough) volatility models,” *Quantitative Finance*, pp. 1–17, 2020.

[58] S. Howison, *Barrier options*, https://people.maths.ox.ac.uk/howison/barriers.pdf, (accessed: Apr 2022).

[59] D. A. Hudson and L. Zitnick, “Generative adversarial transformers,” in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 139, PMLR, 2021, pp. 4487–4499.

[60] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *Proceedings of the 32nd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 37, Lille, France: PMLR, 2015, pp. 448–456.

[61] A. Itkin, “Deep learning calibration of option pricing models: Some pitfalls and solutions,” 2019.

[62] A. Itkin and P. Carr, “Using pseudo-parabolic and fractional equations for option pricing in jump diffusion models,” *Computational Economics*, vol. 40, no. 1, pp. 63–104, Jun. 2012.

[63] K. R. Jackson, S. Jaimungal, and V. Surkov, “Fourier space time-stepping for option pricing with Lévy models,” *Journal of Computational Finance*, vol. 12, no. 2, pp. 1–29, 2008.

[64] Y. Jiang, S. Chang, and Z. Wang, “TransGAN: Two Pure Transformers Can Make One Strong GAN, and That Can Scale Up,” in *Advances in Neural Information Processing Systems*, vol. 34, Curran Associates, Inc., 2021, pp. 14 745–14 758.

[65] N. Ju and R. Zhong, “An approximate formula for pricing American options,” *Journal of Derivatives*, vol. 7, no. 2, pp. 31–40, 1999.

[66] M. Kac, “On distributions of certain wiener functionals,” *Transactions of the American Mathematical Society*, vol. 65, no. 1, pp. 1–13, 1949.

[67] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

[68] J. L. Kirkby, D. Nguyen, and Z. Cui, “A unified approach to bermudan and barrier options under stochastic volatility models with jumps,” *Journal of Economic Dynamics and Control*, vol. 80, pp. 75–100, 2017.

[69] A. Kondratyev and C. Schwarz, “The market generator,” *Available at SSRN 3384948*, 2019.

[70] A. Koochali, P. Schichtel, A. Dengel, and S. Ahmed, “Probabilistic Forecasting of Sensory Data With Generative Adversarial Networks – ForGAN,” *IEEE Access*, vol. 7, pp. 63 868–63 880, 2019.

[71] A. Koshiyama, N. Firoozye, and P. Treleaven, “Generative adversarial networks for financial trading strategies fine-tuning and combination,” *Quantitative Finance*, pp. 1–17, Sep. 2020.

[72] S. G. Kou and H. Wang, “Option pricing under a double exponential jump diffusion model,” *Management Science*, vol. 50, no. 9, pp. 1178–1192, Sep. 2004.

[73] I. E. Lagaris, A. Likas, and D. I. Fotiadis, “Artificial neural networks for solving ordinary and partial differential equations,” *IEEE transactions on neural networks*, vol. 9, no. 5, pp. 987–1000, 1998.

[74] I. E. Lagaris, A. C. Likas, and D. G. Papageorgiou, “Neural-network methods for boundary value problems with irregular boundaries,” *IEEE Transactions on Neural Networks*, vol. 11, no. 5, pp. 1041–1049, 2000.

[75] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, “Backpropagation Applied to Handwritten Zip Code Recognition,” *Neural Computation*, vol. 1, no. 4, pp. 541–551, Dec. 1989.

[76] H. Lee and I. S. Kang, “Neural algorithm for solving differential equations,” *Journal of Computational Physics*, vol. 91, no. 1, pp. 110–131, 1990.

[77] S. Liu, A. Borovykh, L. A. Grzelak, and C. W. Oosterlee, “A neural network-based framework for financial model calibration,” *Journal of Mathematics in Industry*, vol. 9, no. 1, p. 9, Dec. 2019.

[78] F. A. Longstaff and E. S. Schwartz, “Valuing American options by simulation: A simple least-squares approach,” *The Review of Financial Studies*, vol. 14, no. 1, pp. 113–147, 2001.

[79] R. Lord, F. Fang, F. Bervoets, and C. W. Oosterlee, "A fast and accurate FFT-based method for pricing early-exercise options under Lévy processes," *SSRN Electronic Journal*, 2007.

[80] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *arXiv preprint arXiv:1508.04025*, 2015.

[81] D. B. Madan, P. P. Carr, and E. C. Chang, "The variance gamma process and option pricing," *Review of Finance*, vol. 2, no. 1, pp. 79–105, Apr. 1998.

[82] D. B. Madan and E. Seneta, "The variance gamma (V.G.) model for share market returns," *The Journal of Business*, vol. 63, no. 4, p. 511, Jan. 1990.

[83] S. C. Maree, L. Ortiz-Gracia, and C. W. Oosterlee, "Pricing early-exercise and discrete barrier options by Shannon wavelet expansions," *Numerische Mathematik*, vol. 136, no. 4, pp. 1035–1070, Aug. 2017.

[84] O. Marom and E. Momoniat, "A comparison of numerical solutions of fractional diffusion models in finance," *Nonlinear Analysis: Real World Applications*, vol. 10, no. 6, pp. 3435–3442, Dec. 2009.

[85] R. C. Merton, "Option pricing when underlying stock returns are discontinuous," *Journal of Financial Economics*, vol. 3, no. 1, pp. 125–144, 1976.

[86] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, "Spectral normalization for generative adversarial networks," *arXiv preprint arXiv:1802.05957*, 2018.

[87] O. Mogren, "C-rnn-gan: Continuous recurrent neural networks with adversarial training," *arXiv preprint arXiv:1611.09904*, 2016.

[88] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *ICML*, 2010.

[89] H. Ni, L. Szpruch, M. Wiese, S. Liao, and B. Xiao, "Conditional Sig-Wasserstein GANs for Time Series Generation," *arXiv:2006.05421*, Jun. 2020, arXiv: 2006.05421.

[90] C. O'Sullivan, "Path dependent option pricing under Lévy processes," *EFA 2005 Moscow Meetings Paper*, p. 24, Feb. 2005.

[91] E. Page, "Approximations to the cumulative normal function and its inverse for use on a pocket calculator," *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, vol. 26, no. 1, pp. 75–76, 1977. eprint: `https://rss.onlinelibrary.wiley.com/doi/pdf/10.2307/2346872`.

[92] H. Pham, *Continuous-time stochastic control and optimization with financial applications*. Springer Science & Business Media, 2009, vol. 61.

[93] M. Raissi, “Deep hidden physics models: Deep learning of nonlinear partial differential equations,” *The Journal of Machine Learning Research*, vol. 19, no. 1, pp. 932–955, 2018.

[94] P. Ramachandran, B. Zoph, and Q. V. Le, “Searching for activation functions,” 2017. arXiv: `1710.05941 [cs.NE]`.

[95] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[96] K.-I. Sato, *Lévy processes and infinitely divisible distributions*. Cambridge university press, 1999.

[97] J. Sirignano and K. Spiliopoulos, “Dgm: A deep learning algorithm for solving partial differential equations,” *Journal of Computational Physics*, vol. 375, pp. 1339–1364, 2018.

[98] I. M. Sobol’, “On the distribution of points in a cube and the approximate evaluation of integrals,” *Zhurnal Vychislitel’noi Matematiki i Matematicheskoi Fiziki*, vol. 7, no. 4, pp. 784–802, 1967.

[99] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[100] S. Takahashi, Y. Chen, and K. Tanaka-Ishii, “Modeling financial time-series with generative adversarial networks,” *Physica A: Statistical Mechanics and its Applications*, vol. 527, p. 121 261, Aug. 2019.

[101] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, “WaveNet: A Generative Model for Raw Audio,” in *Proc. 9th ISCA Workshop on Speech Synthesis Workshop (SSW 9)*, 2016, p. 125.

[102] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, “Attention is all you need,” in *Advances in Neural Information Processing Systems*, vol. 30, Curran Associates, Inc., 2017.

[103] C. Villani, *Optimal transport: old and new*. Springer, 2009, vol. 338.

[104] I. Wang, J. Wan, and P. Forsyth, “Robust numerical valuation of European and American options under the CGMY process,” *The Journal of Computational Finance*, vol. 10, no. 4, pp. 31–69, Jun. 2007.

[105] M. Wiese, L. Bai, B. Wood, and H. Buehler, “Deep hedging: Learning to simulate equity option markets,” *arXiv preprint arXiv:1911.01700*, 2019.

[106] M. Wiese, R. Knobloch, R. Korn, and P. Kretschmer, "Quant GANs: Deep generation of financial time series," *Quantitative Finance*, vol. 20, no. 9, pp. 1419–1440, Sep. 2020.

[107] M. Wiese, B. Wood, A. Pachoud, R. Korn, H. Buehler, P. Murray, and L. Bai, "Multi-asset spot and option market simulation," *arXiv preprint arXiv:2112.06823*, 2021.

[108] Wikipedia, *Simpson's rule*, https://en.wikipedia.org/wiki/Simpson%27s_rule, (accessed: Oct 2021).

[109] J. Yoon, D. Jarrett, and M. van der Schaar, "Time-series generative adversarial networks," in *Advances in Neural Information Processing Systems*, vol. 32, Curran Associates, Inc., 2019.

[110] B. Yu, X. Xing, and A. Sudjianto, "Deep-learning based numerical bsde method for barrier options," *arXiv preprint arXiv:1904.05921*, 2019.

[111] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, "Self-attention generative adversarial networks," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 97, PMLR, 2019, pp. 7354–7363.

[112] K. Zhang, G. Zhong, J. Dong, S. Wang, and Y. Wang, "Stock Market Prediction Based on Generative Adversarial Network," *Procedia Computer Science*, vol. 147, pp. 400–406, 2019.

[113] X. Zhou, Z. Pan, G. Hu, S. Tang, and C. Zhao, "Stock Market Prediction on High-Frequency Data Using Generative Adversarial Nets," *Mathematical Problems in Engineering*, vol. 2018, pp. 1–11, 2018.

Appendix A: Appendices for Chapter 2

A.1 Development of the simple approach

This part follows [50]. We can split the integral term in Equation (2.7) into two terms, the integrals on $|y| \leq \epsilon$ and $|y| > \epsilon$ respectively.

In the region $|y| \leq \epsilon$,

$$w(x+y,\tau) = w(x,\tau) + y\frac{\partial w}{\partial x}(x,\tau) + \frac{y^2}{2}\frac{\partial^2 w}{\partial x^2}(x,\tau) + O(y^3)$$

and

$$e^y = 1 + y + \frac{y^2}{2} + O(y^3).$$

Using those two approximations, we get

$$\begin{aligned} & \int_{|y|\leq\epsilon} \left[w(x+y,\tau) - w(x,\tau) - \frac{\partial w}{\partial x}(x,\tau)(e^y - 1) \right] k(y)dy \\ = & \int_{|y|\leq\epsilon} \left[\frac{y^2}{2}\frac{\partial^2 w}{\partial x^2}(x,\tau) - \frac{y^2}{2}\frac{\partial w}{\partial x}(x,\tau) + O(y^3) \right] k(y)dy \\ \approx & \int_{|y|\leq\epsilon} \left[\frac{y^2}{2}\frac{\partial^2 w}{\partial x^2}(x,\tau) - \frac{y^2}{2}\frac{\partial w}{\partial x}(x,\tau) \right] k(y)dy. \end{aligned}$$

Define $\sigma^2(\epsilon) = \int_{|y|\leq\epsilon} y^2 k(y)dy$ and we get

$$\int_{|y|\leq\epsilon} \left[w(x+y,\tau) - w(x,\tau) - \frac{\partial w}{\partial x}(x,\tau)(e^y - 1) \right] k(y)dy \approx \frac{1}{2}\sigma^2(\epsilon)\left(\frac{\partial^2 w}{\partial x^2}(x,\tau) - \frac{\partial w}{\partial x}(x,\tau) \right).$$

In the region $|y| > \epsilon$,

$$\begin{aligned}
&\int_{|y|>\epsilon} \left[w(x+y,\tau) - w(x,\tau) - \frac{\partial w}{\partial x}(x,\tau)(e^y - 1) \right] k(y)dy \\
= &\int_{|y|>\epsilon} \left[w(x+y,\tau) - w(x,\tau) \right] k(y)dy + \frac{\partial w}{\partial x}(x,\tau)\omega(\epsilon),
\end{aligned}$$

where $w(\epsilon) = \int_{|y|>\epsilon}(1-e^y)k(y)dy$.

Combine the two parts of integrals and put them back to Equation (2.7), and we get

$$\begin{aligned}
&\frac{1}{2}\sigma^2(\epsilon)\frac{\partial^2 w}{\partial x^2}(x,\tau) + \int_{|y|>\epsilon} \left[w(x+y,\tau) - w(x,\tau) \right] k(y)dy \\
&-\frac{\partial w}{\partial \tau}(x,\tau) + (r - q + \omega(\epsilon) - \frac{1}{2}\sigma^2(\epsilon))\frac{\partial w}{\partial x}(x,\tau) - rw(x,\tau) = 0. \qquad (A.1)
\end{aligned}$$

If we omit the integral term in Equation (A.1), we can get a BMS equation

$$-\frac{\partial w}{\partial \tau}(x,\tau) + \frac{1}{2}\sigma^2(\epsilon)\frac{\partial^2 w}{\partial x^2}(x,\tau) + (r - q + \omega(\epsilon) - \frac{1}{2}\sigma^2(\epsilon))\frac{\partial w}{\partial x}(x,\tau) - rw(x,\tau) = 0.$$

It describes the option price of a stock with volatility $\sqrt{\sigma^2(\epsilon)}$ and dividend $q - \omega(\epsilon)$. So we decide to use the premium of this BMS model to approximate the premium in the VG model.

A.2 A short summary of Ju-Zhong method

Here we give a short summary of the method for pricing American options under the BMS model in [65].

Suppose the prices of American and European put options under the BMS model are denoted by $P(S)$ and $p(S)$, where S is the current stock price and $p(S)$ is calculated by the B-S formula. Also, we are given the strike K, the interest rate r, the dividend rate q, the volatility rate σ and the time to maturity τ.

Then the American put options can be priced by the formula

$$P(S) = \begin{cases} p(S) + \frac{hA(h)e^{\lambda(h)y}}{1-by^2-cy} & \text{if } \phi(S^* - S) > 0 \\ \phi(S - K) & \text{if } \phi(S^* - S) \leq 0 \end{cases}$$

where $\phi = -1$ ($\phi = 1$ for call options), $y = \ln(S/S^*)$, $hA(h) = \phi(S^* - K) - p(S^*)$ and S^* solves the equation

$$\phi = \phi e^{-q\tau} N(\phi d_1(S^*)) + \frac{\lambda(h)(\phi(S^* - K) - p(S^*))}{S^*}$$

and b and c are given by

$$\begin{aligned} b &= \frac{(1-h)\alpha\lambda'(h)}{2(2\lambda(h) + \beta - 1)} \\ c &= -\frac{(1-h)\alpha}{2\lambda(h) + \beta - 1}\left(\frac{1}{hA(h)}\frac{\partial p(S^*)}{\partial h} + \frac{1}{h} + \frac{\lambda'(h)}{2\lambda(h) + \beta - 1}\right) \end{aligned}$$

where

$$\begin{aligned} \alpha &= \frac{2r}{\sigma^2}, \quad \beta = \frac{2(r-q)}{\sigma^2}, \quad h(\tau) = 1 - e^{-r\tau} \\ \lambda(h) &= \frac{-(\beta - 1) + \phi\sqrt{(\beta-1)^2 + \frac{4\alpha}{h}}}{2}, \quad \lambda'(h) = -\frac{\phi\alpha}{h^2\sqrt{(\beta-1)^2 + \frac{4\alpha}{h}}} \\ d_1(S^*) &= \frac{\ln(S^*/K) + (r - q + \sigma^2/2)\tau}{\sigma\sqrt{\tau}}, \quad d_2(S^*) = d_1(S^*) - \sigma\sqrt{\tau} \end{aligned}$$

and

$$\begin{aligned} \frac{\partial p(S^*)}{\partial h} &= \frac{\partial p(S^*)}{\partial \tau} \bigg/ \frac{\partial h}{\partial \tau} \\ &= \frac{S^* n(d_1(S^*))\sigma e^{(r-q)\tau}}{2r\sqrt{\tau}} - \phi q S^* N(\phi d_1(S^*))e^{(r-q)\tau}/r + \phi K N(\phi d_2(S^*)). \end{aligned}$$

Here $N(\cdot)$ and $n(\cdot)$ stands for the cumulative distribution function and the probability density function of the normal distribution.

A.3 Kernel regression

Kernel regression is a nonparametric machine learning technique that is used to find a non-linear relationship between a pair of variables x and y. Both x and y can be vectors. Let d_x and d_y be the dimensions of x and y. Suppose we collect data $x_1, x_2, \dots, x_n$ and $y_1, y_2, \dots, y_n$ and want to find a suitable estimate of y given x.

First, we need a kernel function $\kappa(x', x'')$, where x' and x'' are two points in the space of x. Then the estimate $\hat{y} = f(x)$ given x is a weighted average

$$f(x) = \frac{\sum_{i=1}^{n} \kappa(x, x_i) y_i}{\sum_{i=1}^{n} \kappa(x, x_i)}. \tag{A.2}$$

Second we need to choose a suitable kernel function $\kappa(x', x'')$ to get a good estimation. In this paper we choose the Gaussian kernel, i.e.,

$$\kappa_a(x', x'') = \exp\left(- \sum_{j=1}^{d_x} a_j (x'_j - x''_j)^2 \right),$$

where $\{a_j, 1 \le j \le d_x\}$ are positive numbers and x'_j and x''_j are the jth component of the vectors x' and x''.

Here we define a loss function to choose the kernel parameter a. A reasonable loss function can be defined as

$$\ell(a) = \sum_{i=1}^{n} \|y_i - \hat{y}_i^S(a)\|^2,$$

where $\|\cdot\|$ is the Euclidean norm and

$$\hat{y}_i^S(a) = \frac{\sum_{l \in S} \kappa_a(x_i, x_l) y_l}{\sum_{l \in S} \kappa_a(x_i, x_l)}$$

is the estimate of y_i given x_i and is also a function of a. S is a subset of $\{1, 2, \dots, n\}$ chosen randomly and is served as the training set. This step aims to avoid overfitting. By minimizing $\ell(a)$ w.r.t. a, we can get a suitable kernel function $\kappa_a(x', x'')$ for prediction using Equation (A.2).

Next, we can repeat the second step for several times and take the average of a for robustness. Suppose we have M random and independent training sets S_j, where $1 \le j \le M$. $|S_j| = nR$ for $1 \le j \le M$, where R is the training ratio. Let $a^{(j)}$ be the minimizer of the loss function $\ell(a)$ when the training set is S_j, i.e.,

$$a^{(j)} = \text{argmin}_a \sum_{i=1}^{n} \|y_i - \hat{y}_i^{S_j}(a)\|^2.$$

Then for prediction we could use $\bar{a} = \frac{1}{M}\sum_{j=1}^{M} a^{(j)}$ and the predicted value at x is

$$\hat{y} = f(x) = \frac{\sum_{i=1}^{n} \kappa_{\bar{a}}(x, x_i) y_i}{\sum_{i=1}^{n} \kappa_{\bar{a}}(x, x_i)}. \tag{A.3}$$

Finally we calculate the leave-one-out cross-validation error to estimate the prediction error. Let $S_{-i} = \{1, 2, \ldots, n\} \backslash \{i\}$. Then we predict y_i with the training set S_{-i} and get the estimator

$$\hat{y}_i^{S_{-i}}(\bar{a}) = \frac{\sum_{1 \le l \le n, l \ne i} \kappa_{\bar{a}}(x_i, x_l) y_l}{\sum_{1 \le l \le n, l \ne i} \kappa_{\bar{a}}(x_i, x_l)}.$$

Then the leave-one-out cross-validation error of the jth dimension is

$$\text{err}_j = \sqrt{\frac{1}{n}\sum_{i=1}^{n}((y_i)_j - (\hat{y}_i^{S_{-i}}(\bar{a}))_j)^2}$$

where $(y_i)_j$ and $(\hat{y}_i^{S_{-i}}(\bar{a}))_j$ are the jth components of y_i and $\hat{y}_i^{S_{-i}}(\bar{a})$ respectively. Then

$$(\text{err}_1, \text{err}_2, \ldots, \text{err}_{d_y})$$

is approximately the averaged prediction error in all dimensions of y.

Appendix B: Appendices for Chapter 3

B.1 Evaluation of the PIDE

B.1.1 Split of the integral in the PIDE

This part follows Chapter 5 in [50] mostly. We split the integral term in Equation (3.2) into two parts, the integrals on $\{-\epsilon^- \leq y \leq \epsilon^+\}$ and $\{y < -\epsilon^- \text{ or } y > \epsilon^+\}$ respectively.

In the region $\{-\epsilon^- \leq y \leq \epsilon^+\}$,

$$w(x+y,\tau) = w(x,\tau) + y\frac{\partial w}{\partial x}(x,\tau) + \frac{y^2}{2}\frac{\partial^2 w}{\partial x^2}(x,\tau) + O(y^3)$$

and

$$e^y = 1 + y + \frac{y^2}{2} + O(y^3).$$

Using those two approximations, we get

$$\begin{aligned}
& \int_{-\epsilon^- \leq y \leq \epsilon^+} \left(w(x+y,\tau) - w(x,\tau) - \frac{\partial w}{\partial x}(x,\tau)(e^y - 1) \right) m(dy) \\
= & \int_{-\epsilon^- \leq y \leq \epsilon^+} \left(\frac{y^2}{2}\frac{\partial^2 w}{\partial x^2}(x,\tau) - \frac{y^2}{2}\frac{\partial w}{\partial x}(x,\tau) + O(y^3) \right) m(dy) \\
\approx & \int_{-\epsilon^- \leq y \leq \epsilon^+} \left(\frac{y^2}{2}\frac{\partial^2 w}{\partial x^2}(x,\tau) - \frac{y^2}{2}\frac{\partial w}{\partial x}(x,\tau) \right) m(dy).
\end{aligned}$$

Define $\sigma^2(\epsilon^-, \epsilon^+) = \int_{-\epsilon^- \le y \le \epsilon^+} y^2 m(dy)$ and we get

$$\begin{aligned}
&\int_{-\epsilon^- \le y \le \epsilon^+} \left(w(x+y,\tau) - w(x,\tau) - \frac{\partial w}{\partial x}(x,\tau)(e^y - 1) \right) m(dy) \\
\approx &\frac{1}{2}\sigma^2(\epsilon^-, \epsilon^+) \left(\frac{\partial^2 w}{\partial x^2}(x,\tau) - \frac{\partial w}{\partial x}(x,\tau) \right).
\end{aligned}$$

In the region $\{y < -\epsilon^- \text{ or } y > \epsilon^+\}$,

$$\begin{aligned}
&\int_{y<-\epsilon^- \text{ or } y>\epsilon^+} \left(w(x+y,\tau) - w(x,\tau) - \frac{\partial w}{\partial x}(x,\tau)(e^y - 1) \right) m(dy) \\
= &\int_{y<-\epsilon^- \text{ or } y>\epsilon^+} \left(w(x+y,\tau) - w(x,\tau) \right) m(dy) + \frac{\partial w}{\partial x}(x,\tau)\omega(\epsilon^-, \epsilon^+),
\end{aligned}$$

where $\omega(\epsilon^-, \epsilon^+) = \int_{y<-\epsilon^- \text{ or } y>\epsilon^+} (1 - e^y) m(dy)$.

Combining the two parts of integrals and putting them back to Equation (3.2), we get

$$\begin{aligned}
&\left(\int_{y<-\epsilon^- \text{ or } y>\epsilon^+} \left(w(x+y,\tau) - w(x,\tau) \right) m(dy) + \frac{s^2 + \sigma^2(\epsilon^-, \epsilon^+)}{2} \frac{\partial^2 w}{\partial x^2}(x,\tau) \right. \\
&\left. - \frac{\partial w}{\partial \tau}(x,\tau) + \left(r - q - \frac{s^2 + \sigma^2(\epsilon^-, \epsilon^+)}{2} + \omega(\epsilon^-, \epsilon^+) \right) \frac{\partial w}{\partial x}(x,\tau) - r w(x,\tau) \right) = 0.
\end{aligned} \tag{B.1}$$

In Equation (B.1), the function $w(x,\tau)$ as well as its derivatives $\frac{\partial w}{\partial x}(x,\tau)$, $\frac{\partial^2 w}{\partial x^2}(x,\tau)$ and $\frac{\partial w}{\partial \tau}(x,\tau)$ can be calculated by the neural network itself or the back-propagation of the neural network. Then the terms remaining to be calculated are $\int_{y<-\epsilon^- \text{ or } y>\epsilon^+} \left(w(x+y,\tau) - w(x,\tau) \right) m(dy)$, $\sigma^2(\epsilon^-, \epsilon^+)$ and $\omega(\epsilon^-, \epsilon^+)$.

B.1.2 Pre-calculations

In Table B.1, we list the choice of ϵ^-, ϵ^+ and the expressions of $\sigma^2(\epsilon^-, \epsilon^+)$ and $\omega(\epsilon^-, \epsilon^+)$ for each model. $\sigma^2(\epsilon^-, \epsilon^+)$ and $\omega(\epsilon^-, \epsilon^+)$ are calculated before training.

Model	ϵ^-	ϵ^+	$\sigma^2(\epsilon^-,\epsilon^+)$	$\omega(\epsilon^-,\epsilon^+)$
VG	$0.02/G$	$0.02/M$	$\sigma^2_{CGMY}(C,G,M,0,\epsilon^-,\epsilon^+)$	$\omega_{CGMY}(C,G,M,0,\epsilon^-,\epsilon^+)$
CGMY	$0.01/G$	$0.01/M$	$\sigma^2_{CGMY}(C,G,M,Y,\epsilon^-,\epsilon^+)$	$\omega_{CGMY}(C,G,M,Y,\epsilon^-,\epsilon^+)$
NIG	$0.05/\alpha$	$0.05/\alpha$	-	-
Merton's	0	0	0	$-\lambda(\exp(\alpha+\delta^2/2)-1)$
Kou's	0	0	0	$-\lambda\left(\frac{p\eta_1}{\eta_1-1}+\frac{(1-p)\eta_2}{\eta_2+1}-1\right)$

Table B.1: ϵ^-, ϵ^+, $\sigma^2(\epsilon^-,\epsilon^+)$ and $\omega(\epsilon^-,\epsilon^+)$ for each model.

where

$$\begin{aligned}\sigma^2_{CGMY}(C,G,M,Y,\epsilon^-,\epsilon^+) &= C\,G^{Y-2}(\Gamma(2-Y)-\Gamma(2-Y,G\epsilon^-))\\ &\quad + C\,M^{Y-2}(\Gamma(2-Y)-\Gamma(2-Y,M\epsilon^+)),\end{aligned}$$

and

$$\begin{aligned}\omega_{CGMY}(C,G,M,Y,\epsilon^-,\epsilon^+) &= C\left(M^Y\Gamma(-Y,M\epsilon^+)-(M-1)^Y\Gamma(-Y,(M-1)\epsilon^+)\right)\\ &\quad + C\left(G^Y\Gamma(-Y,G\epsilon^-)-(G+1)^Y\Gamma(-Y,(G+1)\epsilon^-)\right).\end{aligned}$$

M and G follow the definitions in Equation (3.3) and (3.4) and $C = 1/\nu$. $\Gamma(\cdot)$ is the gamma function and $\Gamma(\cdot,\cdot)$ is the incomplete gamma function following the definition

$$\Gamma(s,y) = \int_y^\infty u^{s-1}e^{-u}du.$$

ϵ^+ and ϵ^- could be dependent on the model parameters. For the NIG model, we do not have the closed-form expressions of $\sigma^2(\epsilon^-,\epsilon^+)$ and $\omega(\epsilon^-,\epsilon^+)$ and they are calculated using numerical integration in `scipy`. $\sigma^2(\epsilon^-,\epsilon^+) = 0$ in the Merton's model and the Kou's model since $\epsilon^+ = \epsilon^- = 0$.

B.1.3 Numerical integral

The integral $\int_{y<-\epsilon^- \text{ or } y>\epsilon^+} (w(x+y,\tau) - w(x,\tau))\, m(dy)$ is calculated using the Simpson's rule [108]. If there are $2N+1$ grid points $y_0, y_1, \ldots, y_{2N}$, which satisfy $2y_{2j+1} = y_{2j} + y_{2j+2}, \forall 0 \le j \le N-1$ and $y_0 = \epsilon^+$, the numerical integral is

$$\begin{aligned}
&\int_{y>\epsilon^+} (w(x+y,\tau) - w(x,\tau))\, m(dy) \\
=&\int_{y>\epsilon^+} (w(x+y,\tau) - w(x,\tau))\, k(y)dy \\
\approx&\sum_{j=0}^{N-1} \big(w(x+y_{2j},\tau) - w(x,\tau)\big)\, k(y_{2j})\frac{y_{2j+2}-y_{2j}}{6} + \\
&+\sum_{j=0}^{N-1} \big(w(x+y_{2j+1},\tau) - w(x,\tau)\big)\, k(y_{2j+1})\frac{2(y_{2j+2}-y_{2j})}{3} \\
&+\sum_{j=0}^{N-1} \big(w(x+y_{2j+2},\tau) - w(x,\tau)\big)\, k(y_{2j+2})\frac{y_{2j+2}-y_{2j}}{6} \\
&,
\end{aligned}$$

which is a linear combination of the values of $w(\cdot,\tau)$. The integral on $\{y < -\epsilon^-\}$ is calculated in the same way.

Since the shape of $k(y)$ depends on the model parameters, it is not efficient if we use the same integral grid for different sample points. Take the Merton's model as an example, where

$$k(y) = \frac{\lambda}{\sqrt{2\pi}\delta} e^{-(x-\alpha)^2/(2\delta^2)}$$

The density function has a center parameter α and a scale parameter δ. If we define the integral grid points to be $y_j = \alpha + \delta z_j, \forall 0 \le j \le 2N$, where z_j's are fixed, we will assign enough grid points around the peak of $k(y)$ whatever the model parameters and the integral will be more accurate. For other models, the density $k(y)$ is centered around 0, but it decreases at different rates when the model parameters are different. So we always use a scaled integral grid. In Table B.2, we list the relationship between the scaled grid points y_j and fixed gird points $0 \leq z_0 \leq z_1 \leq \cdots \leq z_{2N}$ used for integral. The values of z_j can be found at `https://github.com/weilong-columbia/pide`.

Model	y_j in the negative part	y_j in the positive part
VG	$-z_j/G$	z_j/M
CGMY	$-z_j/G$	z_j/M
NIG	$-z_j/\alpha$	z_j/α
Merton's	$\alpha - \delta z_j$	$\alpha + \delta z_j$
Kou's	$-z_j/\eta_2$	z_j/η_1

Table B.2: Relationship between the scaled grid points y_j and fixed grid points z_j.

Appendix C: Appendices for Chapter 4

C.1 PDE for the Bergomi model

The PDE (4.3) for the two-factor Bergomi model is derived according to the multidimensional version of the Feynman-Kac formula (see Theorem 1.3.17 in [92]). In Section 4.2.1, we introduce the dynamics of the Bergomi model, which are summarized as

$$\begin{aligned}
\mathrm{d}s_t &= (r - q - \sigma^2(t, X_t^{(1)}, X_t^{(2)})/2)\mathrm{d}t + \sigma(t, X_t^{(1)}, X_t^{(2)})\mathrm{d}W_t^{(S)},\\
\mathrm{d}X_t^{(1)} &= -k_1 X_t^{(1)}\mathrm{d}t + \mathrm{d}W_t^{(1)},\\
\mathrm{d}X_t^{(2)} &= -k_2 X_t^{(2)}\mathrm{d}t + \mathrm{d}W_t^{(2)},
\end{aligned}$$

where $s_t = \ln(S_t)$ is the log-price process and $\sigma(t, x_1, x_2)$ satisfies $\sigma^2(t, X_t^{(1)}, X_t^{(2)}) = \xi_t^t$. The correlations of the Brownian motions are $\mathrm{d}W_t^{(S)}\mathrm{d}W_t^{(i)} = \rho_i \mathrm{d}t, \forall i = 1, 2$ and $\mathrm{d}W_t^{(1)}\mathrm{d}W_t^{(2)} = \rho_{1,2}\mathrm{d}t$. For an applicable function $V(s, t, x_1, x_2)$, the infinitesimal generator $\mathcal{L}_t$ is defined as

$$\begin{aligned}
\mathcal{L}_t V =&(r - q - \sigma^2(t, x_1, x_2)/2)\frac{\partial V}{\partial s} - k_1 x_1 \frac{\partial V}{\partial x_1} - k_2 x_2 \frac{\partial V}{\partial x_2}\\
&+ \frac{1}{2}\sigma^2(t, x_1, x_2)\frac{\partial^2 V}{\partial s^2} + \frac{1}{2}\frac{\partial^2 V}{\partial x_1^2} + \frac{1}{2}\frac{\partial^2 V}{\partial x_2^2}\\
&+ \rho_1 \sigma(t, x_1, x_2)\frac{\partial^2 V}{\partial s \partial x_1} + \rho_2 \sigma(t, x_1, x_2)\frac{\partial^2 V}{\partial s \partial x_2} + \rho_{1,2}\frac{\partial^2 V}{\partial x_1 \partial x_2}.
\end{aligned}$$

According to the Feynman-Kac formula, the vanilla options

$$V(s, t, x_1, x_2) = \mathbb{E}\left(e^{-r(T-t)}(\eta(e^{s_T} - K))^+ \,|S_t = e^s, X_t^{(1)} = x_1, X_t^{(2)} = x_2\right)$$

satisfy the equation

$$\frac{\partial V}{\partial t} + \mathcal{L}_t V - rV = 0.$$

The barrier options are path-dependent and cannot be fully explained by the Feynman-Kac formula. However, they also satisfy the equation and this can be explained by the no-arbitrage property of the option values: the discounted option value should be a martingale. The increment of the discounted option value is

$$\begin{aligned} \mathrm{d}(e^{-rt}V(s_t, t, X_t^{(1)}, X_t^{(2)})) = &- re^{-rt}V\mathrm{d}t + e^{-rt}\frac{\partial V}{\partial t}\mathrm{d}t + e^{-rt}\mathcal{L}_t V\mathrm{d}t \\ &+ e^{-rt}\frac{\partial V}{\partial s}\sigma\mathrm{d}W_t^{(S)} + e^{-rt}\frac{\partial V}{\partial x_1}\mathrm{d}W_t^{(1)} + e^{-rt}\frac{\partial V}{\partial x_2}\mathrm{d}W_t^{(2)}. \end{aligned}$$

The drift term of the increment of a martingale should be 0, which leads to the same equation.

C.2 Black-Scholes formula of vanilla and barrier options

The BS formula of vanilla options was proposed in [10]. We use the variant with the dividend rate. Suppose s is the log-price, K is the strike, t is the current time, T is the maturity (expiration) time, r is the risk-free interest rate and q is the dividend rate. The vanilla call and put are priced using

$$\begin{aligned} C_{\mathrm{v}}(s; K) &= e^{s-q(T-t)}N(h/v + v/2) - Ke^{-r(T-t)}N(h/v - v/2), \\ P_{\mathrm{v}}(s; K) &= -e^{s-q(T-t)}N(-(h/v + v/2)) + Ke^{-r(T-t)}N(-(h/v - v/2)), \end{aligned}$$

where

$$\begin{aligned} h &= s - \ln(K) + (r - q)(T - t), \\ v &= \sigma\sqrt{T - t}. \end{aligned}$$

The formula of barrier options needs the digital call and put of which the payoffs are $\mathbf{1}_{\{S_T > K\}}$ and $\mathbf{1}_{\{S_T < K\}}$ and the prices are

$$
\begin{aligned}
C_{\mathrm{d}}(s;K) &= e^{-r(T-t)}N(h/v - v/2),\\
P_{\mathrm{d}}(s;K) &= e^{-r(T-t)}N(-h/v + v/2).
\end{aligned}
$$

Suppose B is the barrier level, we summarize the pricing formulae in [58] as follows:

$$
\begin{aligned}
C_{\text{u-i}}(s;K,B) =& C_{\mathrm{v}}(s;B) + (B-K)C_{\mathrm{d}}(s;B)\\
&+ \delta(C_{\mathrm{v}}(\tilde{s};K) - C_{\mathrm{v}}(\tilde{s};B) + (K-B)C_{\mathrm{d}}(\tilde{s};B)),\\
C_{\text{u-o}}(s;K,B) =& C_{\mathrm{v}}(s;K) - C_{\text{u-i}}(s;K,B),\\
C_{\text{d-i}}(s;K,B) =& C_{\mathrm{v}}(s;K) - C_{\mathrm{v}}(s;\max(B,K)) - \max(0,B-K)C_{\mathrm{d}}(s;B)\\
&+ \delta(C_{\mathrm{v}}(\tilde{s};\max(B,K)) + \max(0,B-K)C_{\mathrm{d}}(\tilde{s};B)),\\
C_{\text{d-o}}(s;K,B) =& C_{\mathrm{v}}(s;K) - C_{\text{d-i}}(s;K,B),\\
P_{\text{u-i}}(s;K,B) =& P_{\mathrm{v}}(s;K) - P_{\mathrm{v}}(s;\min(B,K)) - \max(0,K-B)P_{\mathrm{d}}(s;B)\\
&+ \delta(P_{\mathrm{v}}(\tilde{s};\min(B,K)) + \max(0,K-B)P_{\mathrm{d}}(\tilde{s};B)),\\
P_{\text{u-o}}(s;K,B) =& P_{\mathrm{v}}(s;K) - P_{\text{u-i}}(s;K,B),\\
P_{\text{d-i}}(s;K,B) =& P_{\mathrm{v}}(s;B) - (B-K)P_{\mathrm{d}}(s;B)\\
&+ \delta(P_{\mathrm{v}}(\tilde{s};K) - P_{\mathrm{v}}(\tilde{s};B) + (B-K)P_{\mathrm{d}}(\tilde{s};B)),\\
P_{\text{d-o}}(s;K,B) =& P_{\mathrm{v}}(s;K) - P_{\text{d-i}}(s;K,B),\\
\delta =& (e^{s}/B)^{1+(2(q-r))/\sigma^2},\\
\tilde{s} =& 2\ln(B) - s,
\end{aligned}
$$

where the up-and-in/out formulae are applicable where $s \le \ln(B)$ and the down-and-in/out formulae are applicable where $s \ge \ln(B)$. Furthermore, the up-and-in/out calls are applicable when $B \ge K$ and the down-and-in/out puts are applicable when $B \le K$.

C.3 Benchmark of vanilla options

The dynamics of the Bergomi model in Section 4.2.1 are summarized as

$$\begin{aligned}
\mathrm{d}s_t &= (r - q - \xi_t^t/2)\mathrm{d}t + \sqrt{\xi_t^t}\mathrm{d}W_t^{(S)},\\
\xi_t^t &= \xi_0^t \exp\left(\omega x_t^t - \frac{\omega^2}{2}\mathrm{var}(x_t^t)\right),\\
x_t^t &= \alpha_\theta\left((1-\theta)X_t^{(1)} + \theta X_t^{(2)}\right),\\
\mathrm{d}X_t^{(1)} &= -k_1 X_t^{(1)}\mathrm{d}t + \mathrm{d}W_t^{(1)},\\
\mathrm{d}X_t^{(2)} &= -k_2 X_t^{(2)}\mathrm{d}t + \mathrm{d}W_t^{(2)},
\end{aligned} \tag{C.1}$$

where $s_t = \ln(S_t)$. The correlated Brownian motions can be expressed using independent Brownian motions $Z_t^{(j)}, j = 1, 2, 3$ as

$$\begin{aligned}
W_t^{(1)} &= Z_t^{(1)},\\
W_t^{(2)} &= \mu_{21}Z_t^{(1)} + \mu_{22}Z_t^{(2)},\\
W_t^{(S)} &= \mu_{31}Z_t^{(1)} + \mu_{32}Z_t^{(2)} + \mu_{33}Z_t^{(3)},
\end{aligned}$$

where $\mu_{21} = \rho_{1,2}$, $\mu_{22} = \sqrt{1-\rho_{1,2}^2}$, $\mu_{31} = \rho_1$, $\mu_{32} = \frac{\rho_2 - \rho_1\rho_{1,2}}{\sqrt{1-\rho_{1,2}^2}}$ and

$$\mu_{33} = \sqrt{\frac{1-\rho_1^2-\rho_2^2-\rho_{1,2}^2+2\rho_1\rho_2\rho_{1,2}}{1-\rho_{1,2}^2}}.$$

Clearly s_t is dependent on $X_t^{(1)}$ and $X_t^{(2)}$ but not conversely. Thus we can determine the volatility process first and then the stock price process. This means the vanilla option prices can be evaluated given the condition of volatility. For example, the call option is

$$\mathbb{E}((S_T-K)^+|S_0) = \mathbb{E}\left(\mathbb{E}\left((S_T-K)^+ \,\middle|\, \{X_t^{(1)}\}_{t=0}^T, \{X_t^{(2)}\}_{t=0}^T, S_0\right) \,\middle|\, S_0\right).$$

Given $\{X_t^{(1)}\}_{t=0}^T$ and $\{X_t^{(2)}\}_{t=0}^T$, we also know the paths of $\{Z_t^{(1)}\}_{t=0}^T$, $\{Z_t^{(2)}\}_{t=0}^T$ and $\{\xi_t^t\}_{t=0}^T$. The SDE of s_t becomes

$$\begin{aligned}
\mathrm{d}s_t =&(r - q - \xi_t^t/2)\mathrm{d}t + \sqrt{\xi_t^t}(\mu_{31}\mathrm{d}Z_t^{(1)} + \mu_{32}\mathrm{d}Z_t^{(2)} + \mu_{33}\mathrm{d}Z_t^{(3)}) \\
=&(r - q - (\mu_{31}^2 + \mu_{32}^2)\xi_t^t/2)\mathrm{d}t + \sqrt{\xi_t^t}(\mu_{31}\mathrm{d}Z_t^{(1)} + \mu_{32}\mathrm{d}Z_t^{(2)}) \\
&- \mu_{33}^2\xi_t^t/2\,\mathrm{d}t + \sqrt{\xi_t^t}\mu_{33}\mathrm{d}Z_t^{(3)}
\end{aligned}$$

where only $Z_t^{(3)}$ is random and the volatility function is fixed. The equivalent spot is

$$\widetilde{S_0} = S_0 \exp\left(\int_0^T \sqrt{\xi_t^t}(\mu_{31}\mathrm{d}Z_t^{(1)} + \mu_{32}\mathrm{d}Z_t^{(2)}) - (\mu_{31}^2 + \mu_{32}^2)\xi_t^t/2\, dt\right)$$

and the equivalent volatility rate during $[0, T]$ is

$$\tilde{\sigma}_0^T = \sqrt{\frac{\mu_{33}^2}{T}\int_0^T \xi_t^t dt}$$

The conditional expectation can be calculated by the Black-Scholes formula:

$$\mathbb{E}((S_T - K)^+ | \{X_t^{(1)}\}_{t=0}^T, \{X_t^{(2)}\}_{t=0}^T, S_0) = \text{BS-}C_\text{v}(\widetilde{S_0}, K, T, \tilde{\sigma}_t^T, r, q)$$

Then we just need to sample paths of $\{X_t^{(1)}\}_{t=0}^T$ and $\{X_t^{(2)}\}_{t=0}^T$ and take the average of the conditional expectation to get the vanilla option price. The same applies to the vanilla put. The variance of the conditional expectation is far less than the variance of trivial simulation. However, this approach does not work for the barrier options. Since the payoff of barrier options are path-dependent and we cannot get the equivalent spot and volatility rate.

C.4 Benchmark of barrier options

Since we cannot use conditional expectation for barrier options as in Appendix C.3, we need to sample the log-price $\{s_t\}_{t=0}^T$ for simulation. Note that ξ_t^t contains an exponential function and could be very large. Under this case, s_t converges to $-\infty$ quickly, and S_t converges to 0 quickly. When we evaluate the barrier puts, this not a problem. However, this is a problem for barrier calls. There will be very few or no samples of positive values, and the barrier calls will be underestimate. As a result, the Euler scheme is directly applied to Equation (C.1) to price barrier puts, while we use importance sampling to price barrier calls for variance reduction.

The importance sampling is implemented according to Girsanov theorem [40]. First, let

$$\mathrm{d}Z_t^{(3)} = \mathrm{d}\tilde{Z}_t^{(3)} + \mu_{33}\sqrt{\xi_t^t}\mathrm{d}t$$

where $\tilde{Z}_t^{(3)}$ is a Brownian motion under the measure $\mathbb{Q}$ while $Z_t^{(3)}$ is a Brownian motion under the measure $\mathbb{P}$ with the Radon-Nikodym derivative

$$\frac{\mathrm{d}\mathbb{P}}{\mathrm{d}\mathbb{Q}} = \exp\left(-\int_0^t \mu_{33}^2\xi_u^u/2\,du - \int_0^t \mu_{33}\sqrt{\xi_u^u}\mathrm{d}\tilde{Z}_u^{(3)}\right).$$

After that we replace $\mathrm{d}Z_t^{(3)}$ using $\mathrm{d}\tilde{Z}_t^{(3)}$ in the SDE of s_t such that

$$\begin{aligned}\mathrm{d}s_t =&(r-q-(\mu_{31}^2+\mu_{32}^2)\xi_t^t/2)\mathrm{d}t + \sqrt{\xi_t^t}(\mu_{31}\mathrm{d}Z_t^{(1)} + \mu_{32}\mathrm{d}Z_t^{(2)})\\ &- \mu_{33}^2\xi_t^t/2\,\mathrm{d}t + \sqrt{\xi_t^t}\mu_{33}\mathrm{d}Z_t^{(3)}\\ =&(r-q-(\mu_{31}^2+\mu_{32}^2)\xi_t^t/2)\mathrm{d}t + \sqrt{\xi_t^t}(\mu_{31}\mathrm{d}Z_t^{(1)} + \mu_{32}\mathrm{d}Z_t^{(2)})\\ &+ \mu_{33}^2\xi_t^t/2\,\mathrm{d}t + \sqrt{\xi_t^t}\mu_{33}\mathrm{d}\tilde{Z}_t^{(3)}.\end{aligned}$$

We sample $\{s_t\}_{t=0}^T$ under $\mathbb{Q}$, i.e.,

$$s_t = s_0 + \int_0^t (r-q-(1-2\mu_{33}^2)\xi_u^u/2)\mathrm{d}u + \sqrt{\xi_u^u}(\mu_{31}\mathrm{d}Z_u^{(1)} + \mu_{32}\mathrm{d}Z_u^{(2)} + \mathrm{d}\tilde{Z}_u^{(3)}).$$

Each sample path $\{s_t\}_{t=t_0}^T$ is attached the following weight

$$\exp\left(-\int_0^T \mu_{33}^2\xi_u^u/2\,du - \int_0^t \mu_{33}\sqrt{\xi_u^u}\mathrm{d}\tilde{Z}_u^{(3)}\right).$$

Since the drift term $-\mu_{33}^2 \xi_t^t / 2 \, \mathrm{d}t$ in the original SDE is changed to $\mu_{33}^2 \xi_t^t / 2 \, \mathrm{d}t$ in the SDE under $\mathbb{Q}$, there will be enough large samples of s_T and the barrier call options will not be underestimated. After we collect enough sample paths, we use the definition of barrier options in Table 4.1 to evaluate them.

Appendix D: Appendices for Chapter 5

D.1 Arbitrage-free option surface

In [13], the authors give the condition that a call option surface is arbitrage-free. Suppose we have the set of relative strikes

$$\mathcal{K} = \{K_0, K_1, K_2, \ldots, K_{N_K}, K_{N_K+1}\}$$

where $K_i < K_{i+1}, 0 \leq i \leq N_K$ and the set of maturities

$$\mathcal{M} = \{M_1, M_2, \ldots, M_{N_M}\}$$

where $M_j < M_{j+1}, 1 \leq j \leq N_M - 1$. Let $C_{i,j}$ be the call price at strike K_i and maturity M_j. K_0 is sufficiently small and K_{N_K+1} is sufficiently large, so that we have $C_{0,j} = 1 - K_0$ and $C_{N_K+1,j} = 0$. Then the variables $\{C_{i,j}, 1 \leq i \leq N_K, 1 \leq j \leq N_M\}$ need to satisfy the following conditions:

$$\begin{cases} C_{1,j} \geq 1 - K_1, & \forall 1 \leq j \leq N_M \\ C_{N_K,j} \geq 0, & \forall 1 \leq j \leq N_M \\ C_{i,j} \geq C_{i,j-1}, & \forall 1 \leq i \leq N_K, 2 \leq j \leq N_M \\ \frac{C_{i,j} - C_{i-1,j}}{K_i - K_{i-1}} \leq \frac{C_{i+1,j} - C_{i,j}}{K_{i+1} - K_i}, & \forall 1 \leq i \leq N_K, 1 \leq j \leq N_M \end{cases} \tag{D.1}$$

Suppose $\{\hat{C}_{i,j}, 1 \leq i \leq N_K, 1 \leq j \leq N_M\}$ is the call option surface that does not satisfy the conditions, we use the following linear programming to solve the closest arbitrage-free surface.

$$\min_{C_{i,j}, 1 \leq i \leq N_K, 1 \leq j \leq N_M} \sum_{i=1}^{N_K} \sum_{j=1}^{N_M} |\hat{C}_{i,j} - C_{i,j}|$$

such that the constraints in (D.1) are satisfied.

If we start from the volatility surface, we calculate the call options and detect any violations of the constraints in (D.1). If so, we perform the linear programming to remove the arbitrages and calculate the corrected implied volatilities.

D.2 Losses of GANs

In [41], the authors proposed the original loss of GANs:

$$\min_{\theta_G} \max_{\theta_D} \mathbb{E}_{\boldsymbol{X}} \ln(D(\boldsymbol{X}; \theta_D)) + \mathbb{E}_{\boldsymbol{Z}} \ln(1 - D(G(\boldsymbol{Z}; \theta_G); \theta_D))$$

where the discriminator is $D(\cdot; \theta_D) : \mathbb{R}^{l \times d} \to (0, 1)$ and the output of the discriminator stands for the probability that its input is considered real data. Thus the losses for the generator and the discriminator are

$$\min_{\theta_G} \mathbb{E}_{\boldsymbol{Z}} \ln(1 - D(G(\boldsymbol{Z}; \theta_G); \theta_D))$$
$$\min_{\theta_D} -\mathbb{E}_{\boldsymbol{X}} \ln(D(\boldsymbol{X}; \theta_D)) - \mathbb{E}_{\boldsymbol{Z}} \ln(1 - D(G(\boldsymbol{Z}; \theta_G); \theta_D))$$

respectively. In practice, $D(G(\boldsymbol{Z}; \theta_G); \theta_D)$ is close to 0 in the beginning since the generator has not learned anything. The gradient of $\ln(1 - D(G(\boldsymbol{Z}; \theta_G); \theta_D))$ is small and convergency would be slow. So the loss for the generator is replaced with

$$\min_{\theta_G} -\mathbb{E}_{\boldsymbol{Z}} \ln(D(G(\boldsymbol{Z}; \theta_G); \theta_D)).$$

In this paper, we use the discriminator $D(\cdot;\theta_D) : \mathbb{R}^{l\times d} \to \mathbb{R}$ to include the case of WGAN-GP. So the sigmoid function $\sigma(D(\cdot;\theta_D))$ stands for the probability. Thus the original losses of GANs are written as

$$\min_{\theta_G} \mathbb{E}_{\boldsymbol{Z}} - \ln(\sigma(D(G(\boldsymbol{Z};\theta_G);\theta_D)))$$
$$\min_{\theta_D} \mathbb{E}_{\boldsymbol{X},\boldsymbol{Z}} - \ln(\sigma(D(\boldsymbol{X};\theta_D))) - \ln(1-\sigma(D(G(\boldsymbol{Z};\theta_G);\theta_D))).$$

The Wasserstein GAN in [2] approaches the loss of GANs in a different way. It tries to minimize the Wasserstein-1 distance between the real distribution and the generated distribution:

$$W(\mathbb{P}_r,\mathbb{P}_g) = \inf_{\gamma\in\Pi(\mathbb{P}_r,\mathbb{P}_g)} \mathbb{E}_{(\boldsymbol{X},\boldsymbol{Y})\sim\gamma} \|\boldsymbol{X}-\boldsymbol{Y}\|$$

where $\mathbb{P}_r$ is the real distribution of $\boldsymbol{X}$ and $\mathbb{P}_g$ is the generated distribution of $G(\boldsymbol{Z};\theta_G)$, and $\Pi(\mathbb{P}_r,\mathbb{P}_g)$ denotes the set of all joint distributions γ whose marginals are respectively $\mathbb{P}_r$ and $\mathbb{P}_g$. $\|\cdot\|$ is the Frobenius norm. Then they make use of the Kantorovich-Rubinstein duality [103] to get

$$W(\mathbb{P}_r,\mathbb{P}_g) = \sup_{\|f\|_L\leq 1} \mathbb{E}_{\boldsymbol{X}} f(\boldsymbol{X}) - \mathbb{E}_{\boldsymbol{Z}} f(G(\boldsymbol{Z};\theta_G))$$

where $\|f\|_L \leq 1$ means the 1-Lipschitz function f. For a differentiable f, this means $\|\nabla f\| \leq 1$. The discriminator $D(\cdot;\theta_D) : \mathbb{R}^{l\times d} \to \mathbb{R}$ is used to approximate the function f that reaches the supremum and its loss is

$$\min_{\theta_D} -\mathbb{E}_{\boldsymbol{X}} D(\boldsymbol{X};\theta_D) + \mathbb{E}_{\boldsymbol{Z}} D(G(\boldsymbol{Z};\theta_G);\theta_D).$$

The generator tries to minimize the Wasserstein-1 distance, which means

$$\min_{\theta_G} \mathbb{E}_{\boldsymbol{Z}} - D(G(\boldsymbol{Z};\theta_G);\theta_D).$$

Note that the discriminator needs to be 1-Lipschitz continuous such that the Kantorovich-Rubinstein duality holds. Thus the authors in [43] proposed to add a gradient penalty to keep the Lipschitz continuity, and the loss for the discriminator becomes

$$\min_{\theta_D} \mathbb{E}_{\boldsymbol{X},\boldsymbol{Z},\tilde{\boldsymbol{X}}} - D(\boldsymbol{X};\theta_D) + D(G(\boldsymbol{Z};\theta_G);\theta_D) + \lambda(\|\nabla_{\tilde{\boldsymbol{X}}} D(\tilde{\boldsymbol{X}};\theta_D)\| - 1)^2$$

where λ is a constant, $\tilde{\boldsymbol{X}} = (1-U)\boldsymbol{X} + U\,G(\boldsymbol{Z};\theta_G)$ is a linear interpolation between $\boldsymbol{X}$ and $G(\boldsymbol{Z};\theta_G)$, and U follows the uniform distribution over $(0,1)$.

D.3 A short summary of `QuantGAN`

The `QuantGAN` was defined in [106]. It makes use of two temporal convolutional networks as the generator and discriminator. The temporal convolutional networks are further composed of dilated causal convolutional layers.

Here is the definition of the dilated causal convolutional layer. Suppose the input is $\boldsymbol{I} \in \mathbb{R}^{n_l \times n_i}$ and it passes through a causal convolutional layer with kernel size n_k, output channel n_o and dilation n_d. The parameters are the weight $\boldsymbol{W} \in \mathbb{R}^{n_k \times n_i \times n_o}$ and the intercept $\boldsymbol{b} \in \mathbb{R}^{n_o}$. The output of the causal convolutional layer is $\boldsymbol{O} \in \mathbb{R}^{(n_l - n_d(n_k-1)) \times n_o}$. In the causal layer, the time index of the output is taken from $\{n_d(n_k-1)+1, n_d(n_k-1)+2, \ldots, n_l\}$. The output is given by

$$O_{t,i_o} = \sum_{i=1}^{n_i} \sum_{i_k=1}^{n_k} W_{i_k,i,i_o} I_{t-n_d(n_k-i_k),i} + b_{i_o}, \forall n_d(n_k-1)+1 \le t \le n_l, 1 \le i_o \le n_o.$$

The RFS of the layer is $n_d(n_k-1)+1$. Denote the dilated causal convolutional layer as $\text{conv}_d^{(n_k,n_o,n_d)}(\cdot)$, where n_k, n_o and n_d are the kernel size, output channel and dilation.

The hyper-parameters of the `QuantGAN` are summarized in Table D.1. The data length l and the RFS f are not independent parameters but are calculated using $l = f = 1 + 2(n_k-1)(d_f^L - 1)/(d_f - 1)$. Suppose the input noise of the generator is $\boldsymbol{Z} \in \mathbb{R}^{(l+f-1)\times d_n}$ and the output sample is

hyper-parameter	meaning
d_n	noise channel
d	data channel
d_h	hidden channel
$d_f \geq 2$	dilation factor
L	number of layers
$h(\cdot)$	activation function

Table D.1: Hyper-parameters in `QuantGAN`.

$\boldsymbol{Y} \in \mathbb{R}^{l \times d}$. The generator can be written as follows

$$\begin{aligned}
\boldsymbol{H}^{(0)} &= h \circ \mathrm{conv}_{\mathtt{d}}^{(1,d_h,1)} \circ h \circ \mathrm{conv}_{\mathtt{d}}^{(1,d_h,1)}(\boldsymbol{Z}) \\
\boldsymbol{H}^{(j)} &= h \circ \mathrm{conv}_{\mathtt{d}}^{\left(n_k,d_h,d_f^{j-1}\right)} \circ h \circ \mathrm{conv}_{\mathtt{d}}^{\left(n_k,d_h,d_f^{j-1}\right)}\left(\boldsymbol{H}^{(j-1)}\right), \forall 1 \leq j \leq L \\
\boldsymbol{Y} &= \mathrm{conv}_{\mathtt{d}}^{(1,d,1)}\left(\boldsymbol{H}^{(L)}\right)
\end{aligned}$$

where $\circ$ means composition of layers. A simplified structure of the generator is illustrated in Figure D.1.

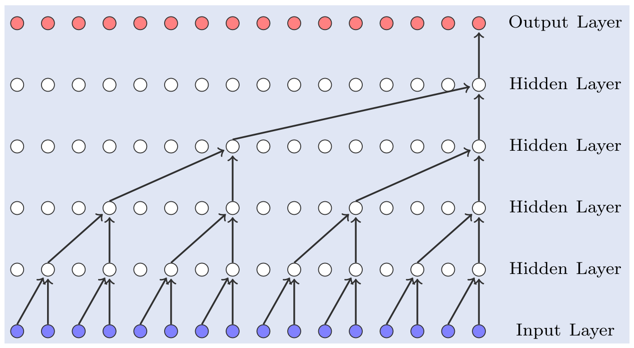


Figure D.1: Illustration of the simplified structure of the temporal convolutional network in `QuantGAN` with $n_k = 2$ and $d_f = 2$.

Let $\boldsymbol{Y} \in \mathbb{R}^{l \times d}$ denote either the real data or the fake data from the generator and $D(\boldsymbol{Y}; \theta_D)$ be the output of the discriminator. Then the discriminator can be written as follows:

$$\begin{aligned}
\boldsymbol{U}^{(0)} &= h \circ \mathrm{conv}_{\mathtt{d}}^{(1,d_h,1)} \circ h \circ \mathrm{conv}_{\mathtt{d}}^{(1,d_h,1)}\,(\boldsymbol{Y}) \\
\boldsymbol{U}^{(j)} &= h \circ \mathrm{conv}_{\mathtt{d}}^{\left(n_k,d_h,d_f^{j-1}\right)} \circ h \circ \mathrm{conv}_{\mathtt{d}}^{\left(n_k,d_h,d_f^{j-1}\right)} \left(\boldsymbol{U}^{(j-1)}\right), \forall 1 \le j \le L \\
D(\boldsymbol{Y};\theta_D) &= \mathrm{conv}_{\mathtt{d}}^{(1,1,1)} \left(\boldsymbol{U}^{(L)}\right)
\end{aligned}$$

The network structure of the generator and discriminator are the same except the input and output channels.