Supervised deep learning in computational finance

Liu, S.

DOI
10.4233/uuid:5966c116-1108-4ecf-8f86-3d8348a3504a

Publication date
2021

Document Version
Final published version

Citation (APA)
Liu, S. (2021). *Supervised deep learning in computational finance*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:5966c116-1108-4ecf-8f86-3d8348a3504a

SUPERVISED DEEP LEARNING IN COMPUTATIONAL FINANCE

Dissertation

for the purpose of obtaining the degree of doctor
at Delft University of Technology
by the authority of the Rector Magnificus prof. dr. ir. T. H. J. J. van der Hagen,
chair of the Board for Doctorates
to be defended publicly on
Monday 1 February 2021 at 10:00 o'clock

by

Shuaiqiang LIU

Master of Science in Applied Mathematics,
Northwestern Polytechnical University, Xi'an, China
born in Handan, Hebei Province, China

This dissertation has been approved by the promotor:
Prof. dr. ir. C. W. Oosterlee

Composition of the doctoral committee:

Rector Magnificus	chairman
Prof. dr. ir. C. W. Oosterlee	Delft University of Technology, promotor
Prof. dr. P. Cirillo	University of Nicosia, Cyprus, co-promotor

Independent members:

Prof. dr. I. Kyriakou	City, University of London (UCL), United Kingdom
Prof. dr. J. Liang	Tongji University, Shanghai, China
Prof. dr. ir. A. W. Heemink	Delft University of Technology, the Netherlands
Prof. dr. ir. C. Vuik	Delft University of Technology, the Netherlands

Other members:

Prof. dr. ir. S. M. Bohté	CWI, Amsterdam, the Netherlands

TU Delft Delft University of Technology

Supervised Deep Learning in Computational Finance.

Dissertation at Delft University of Technology.

This research was supported by the China Scholarship Council (CSC).

ISBN 978-94-6384-191-7
Printed by Ridderprint in the Netherlands.
An electronic version of this dissertation is available at
`https://repository.tudelft.nl/`.

SUMMARY

Mathematical modeling and numerical methods play a key role in the field of quantitative finance, for example, for financial derivative pricing and for risk management purposes. Asset models of increasing complexity, like stochastic volatility models (local stochastic volatility, rough volatility based on fractional Brownian motion) require advanced, efficient numerical techniques to bring them successfully into practice. When computations take too long, an involved asset model is not a feasible option as practical considerations demand a balance between the model's accuracy and the time it takes to compute prices and risk management measures. In the big data era, typical basic computational tasks in the financial industry are often involved and computationally intensive due to the large volumes of financial data that are generated nowadays. Besides the traditional numerical methods in financial derivatives pricing in quantitative finance (like partial differential equation (PDE) discretization and solution methods, Fourier methods, Monte Carlo simulation), recently deep machine learning techniques have emerged as powerful numerical approximation techniques within scientific computing. Following the so-called Universal Approximation Theory, we will employ deep neural networks for financial computations, either to speed up the solution processes or to solve highly complicated, high-dimensional, problems in finance. Particularly, we will employ supervised machine learning techniques, based on intensive learning of so-called labeled information (input-output relations, where sets of parameters form the input to a neural network, and the output to be learned is a solution to a financial problem).

This thesis thus deals with supervised machine learning for different tasks in quantitative finance, and is composed of the following chapters. In Chapter 2, we provide an efficient approximation technique by means of an Artificial Neural Network (ANN), especially for the valuation of options under involved or time-consuming asset price models. This chapter includes a description of ANNs, from the viewpoint of its function approximation capabilities, as well as the general procedure of developing a data-driven numerical solver. The ANN approach is evaluated by using the analytical European option price solution of the Black-Scholes equation as a benchmark, before addressing the computation of implied volatilities and European option pricing under the Heston stochastic volatility asset price model, for which a closed-form solution does not exist. Instead of directly approximating the pricing function itself, we use a gradient-squashing

technique to overcome the issue of reduced accuracy by the ANN when approximating a function with a steep-gradient (i.e. when approximating the inverse function for the implied volatility). By decoupling in an offline (i.e. training) and online (i.e. prediction) stage, the ANN computing time for solving parametric asset models reduces by orders of magnitude in the online prediction stage.

Subsequently, we deal with financial model calibration, which typically gives rise to a non-convex optimization problem, requiring more intensive computations. In Chapter 3, we develop a generic, efficient and robust calibration framework, which we call Calibration Neural Network (CaNN), to estimate model parameters, particularly for high-dimensional models. As both training the ANNs and calibrating the asset models boil down to optimization problems, we integrate three separate ANN phases (training, prediction and calibration) to form the two-staged CaNN as a machine learning platform. In the first stage, an ANN is trained, based on a predefined lablled data set to approximate the valuation function of a given option pricing model. In the second stage, the trained ANN is utilized in a backward manner to estimate the model parameters that need to be calibrated based on observed market option prices. Traditionally, a speed bottleneck occurs when a global optimization technique during calibration is employed to avoid getting stuck in local minima. The CaNN which is based on a group-based global optimization technique (Differential Evolution) overcomes the bottleneck, because the evaluation of the ANN function to price an option is extremely fast. In this thesis, we apply the CaNN to calibrate the Heston stochastic volatility model and the Bates stochastic volatility with jumps model, in which there are five and eight parameters to calibrate, respectively. The numerical results show that even without pre-defined educated initial guesses for these parameters, the CaNN can swiftly find the optimal values of the parameters.

In Chapter 4, based on the methods and results from Chapters 2 and 3, an ANN-based method is developed to extract implied information from American options. As is well-known, early-exercise features of Bermudan and American options cause numerical issues when inverting the pricing model. For example, the inverse function for the (Black-Scholes) implied volatility does not exist because the first derivative of the option price with respect to the volatility, which is named Vega, is equal to zero in the early-exercise region. In this chapter, we also consider some more extreme option pricing situations, that is, we will encounter multiple early-exercise regions which may appear due to negative interest rates. To determine the American option implied volatility, the inverse option pricing function is approximated by means of an artificial neural network on the effective computational domain, which is determined in the off-line stage, thanks to the decoupling of the ANN training and prediction phases. When the implied dividend yield also needs to be determined, the CaNN from the previous chapter

is again employed to estimate simultaneously two different pieces of implied information from the American options (i.e., the implied volatility and the implied dividend yield). Here we generalize the CaNN by using the forward pass to approximate a pair of American option prices (the American call and put prices). The numerical results suggest that the proposed approach gives us an efficient numerical technique to extract implied information from American options.

Finally, from the PDE-based problems (the option pricing PDEs) in the earlier chapters, we move to another class of equations, the Stochastic Differential Equations (SDEs). Chapter 5 is dedicated to an accurate numerical scheme to perform a large time step Monte Carlo simulation, the Seven-League SDE discretization scheme. This SDE discretization is based on a polynomial chaos expansion method, on the basis of accurately determined stochastic collocation (SC) points. The basic idea is to train an ANN to learn these SC points, which is followed by constructing the corresponding conditional transition probability function and generating the required Monte Carlo paths on the basis of large time steps. An error analysis confirms that we can achieve accurate SDE solutions in the sense of the strong convergence properties. With a method variant called the compression-decompression collocation and interpolation technique, we can further reduce the number of neural network functions to be called, so that computational speed of the overall method is enhanced. Numerical experiments show that the novel scheme achieves a high-quality strong convergence error and outperforms some classical numerical methods. We present some applications in financial option valuation. This big time step SDE discretization scheme can be generalized to solving SDEs in other application areas as well.

Summarizing, we develop supervised deep learning techniques as numerical approximation techniques to address some numerical issues in computational finance. Efficient, robust and accurate numerical results are presented.

SAMENVATTING

Wiskundige modellen en numerieke methoden spelen een sleutelrol op het gebied van kwantitatieve financiële wiskunde, bijvoorbeeld voor de prijsstelling van financiële derivaten en voor risicomanagementdoeleinden. Activamodellen met toenemende complexiteit, zoals stochastische volatiliteit aandeelmodellen (lokale stochastische volatiliteit, ruwe volatiliteit gebaseerd op fractionele Brownse beweging), vereisen geavanceerde, efficiënte numerieke technieken om ze met succes in de praktijk te brengen. Wanneer berekeningen te lang duren, is een geavanceerd aandelenmodel geen haalbare optie, aangezien praktische overwegingen een evenwicht vereisen tussen de nauwkeurigheid van het model en de tijd die het kost om prijzen en risicobeheersmaatregelen te berekenen. In het big-data tijdperk worden typische basistaken in de financiële sector vaak rekenintensief vanwege de grote hoeveelheden financiële gegevens die tegenwoordig worden gegenereerd. Naast de traditionele numerieke methoden voor de prijsstelling van financiële derivaten in kwantitatieve financiële wiskunde (zoals partiële differentiaalvergelijking (PDE) discretisatie en oplossingsmethoden, Fourier-methoden, Monte Carlo-simulatie), zijn de laatste tijd technieken voor diepgaand machinaal leren naar voren gekomen als krachtige numerieke benaderingstechnieken. Gebaseerd op de zogenaamde Universele Benaderingstheorie kunnen we diepe neurale netwerken gebruiken voor financiële berekeningen, hetzij om oplossingsprocessen te versnellen ofwel om zeer gecompliceerde, hoogdimensionale problemen in de financiële wereld op te lossen. We zullen in het bijzonder gebruik maken van supervisie-gebaseerde machine leertechnieken, op basis van intensief leren van zogenaamde gelabelde informatie (input-output relaties, waarbij parametersets de input vormen naar een neuraal netwerk, en de output die wordt geleerd is een oplossing voor een financieel probleem).

Dit proefschrift behandelt dus supervised machine learning voor verschillende taken in kwantitatieve financiële wiskunde, en is samengesteld uit de volgende hoofdstukken. In Hoofdstuk 2 bieden we een efficiënte benaderingstechniek aan met behulp van een Artificial Neural Network (ANN), vooral voor de waardering van opties onder betrokken of tijdrovende aandeelprijsmodellen. Dit hoofdstuk bevat een beschrijving van ANNs, vanuit het oogpunt van hun functiebenaderingsmogelijkheden, evenals de algemene procedure voor het ontwikkelen van een datagestuurde numerieke oplosmethode. De ANN-benadering

wordt geëvalueerd door de analytische Europese optieprijsoplossing van de Black-Scholes-vergelijking als benchmark te gebruiken, en ook de berekeningen van de impliciete volatiliteit en de Europese optieprijzen onder het Heston stochastische volatiliteitsmodel, waarvoor een gesloten formulier oplossing niet bestaat, worden uitgevoerd.

In plaats van de prijsfunctie zelf rechtstreeks te benaderen, gebruiken we een gradiënt-reductie-techniek om het probleem van verminderde nauwkeurigheid door de ANN bij het benaderen van een functie met een steile gradiënt (d.w.z. bij het benaderen van de inverse functie voor de impliciete volatiliteit) te ondervangen. Door ontkoppeling in een ANN offline (d.w.z. training) en online (d.w.z. voorspelling) fase, vermindert de ANN-rekentijd voor het oplossen van parametrische aandeelprijsmodellen met ordes van grootte in de online fase. Vervolgens behandelen we de kalibratie van financiële modellen, wat doorgaans aanleiding geeft tot een niet-convex optimalisatieprobleem, dat intensievere berekeningen vereist. In Hoofdstuk 3 ontwikkelen we een generiek, efficiënt en robuust kalibratiekader, dat we Calibration Neural Network (CaNN) noemen, om modelparameters te schatten, met name voor hoog-dimensionale modellen. Omdat zowel het trainen van de ANNs als ook het kalibreren van de aandeelprijsmodellen beschreven wordt door optimalisatieproblemen, integreren we drie afzonderlijke ANN-fasen (training, voorspelling en kalibratie) om het twee-traps CaNN te vormen als een machine-leerplatform. In de eerste fase wordt de ANN getraind op basis van een vooraf gedefiniëerde dataset om de waarderingsfunctie van een bepaald optieprijsmodel te benaderen. In de tweede fase wordt de getrainde ANN achterwaarts gebruikt om de modelparameters te schatten die moeten worden gekalibreerd op basis van waargenomen marktoptieprijzen. Traditioneel treedt een rekentijdprobleem op wanneer een globale optimalisatietechniek tijdens de kalibratie wordt gebruikt om te voorkomen dat het algoritme vastloopt in lokale minima. De CaNN, die is gebaseerd op een op groep gebaseerde globale optimalisatietechniek (Differential Evolution), ondervangt die bottleneck, omdat de evaluatie van de ANN-functie om een optie te prijzen extreem snel is. In dit proefschrift passen we de CaNN toe om het Heston stochastische volatiliteitsmodel en het Bates stochastische volatiliteit met sprongen model te kalibreren, waarin er respectievelijk vijf en acht parameters te kalibreren zijn. De numerieke resultaten laten zien dat zelfs zonder vooraf gedefiniëerde, slimme initiële inschattingen voor deze parameters, de CaNN snel de optimale waarden van de parameters kan vinden.

In Hoofdstuk 4 wordt op basis van de methoden en resultaten uit de Hoofdstukken 2 en 3 een op ANN gebaseerde methode ontwikkeld om impliciete informatie uit Amerikaanse opties te extraheren. Zoals bekend, veroorzaken de vroege uitoefening van Bermudaanse en Amerikaanse opties numerieke proble-

men bij het inverteren van het optieprijsmodel. De inverse functie voor de (Black-Scholes) geïmpliceerde volatiliteit bestaat bijvoorbeeld niet wanneer de eerste afgeleide van de optieprijs naar de volatiliteit, die Vega wordt genoemd, gelijk is aan nul, in het gebied met vroege optieuitoefening. In dit hoofdstuk bekijken we ook enkele extremere prijssituaties voor opties, dat wil zeggen dat we meerdere regio's met vroege uitoefening tegenkomen die op kunnen treden als gevolg van negatieve rentetarieven. Om de impliciete volatiliteit van de Amerikaanse optie te bepalen, wordt de functie van de inverse prijsbepaling van opties benaderd door middel van een kunstmatig neuraal netwerk op het effectieve rekendomein, dat wordt bepaald in de off-line fase, dankzij de ontkoppeling van de ANN trainings- en voorspellingsfasen. Wanneer het impliciete dividendrendement moet worden bepaald, wordt de CaNN uit het vorige hoofdstuk opnieuw gebruikt om gelijktijdig twee verschillende soorten impliciete informatie uit de Amerikaanse opties te bepalen (d.w.z. de impliciete volatiliteit en het impliciete dividendrendement). Hier generaliseren we de CaNN door de eerste fase te gebruiken om twee Amerikaanse optieprijzen (de Amerikaanse call- en putprijzen) tegelijkertijd te benaderen. De numerieke resultaten suggereren dat de voorgestelde benadering ons een efficiënte numerieke techniek geeft om impliciete informatie uit Amerikaanse opties te extraheren.

Ten slotte gaan we van de op PDV gebaseerde problemen (voornamelijk PDVs voor optieprijzen) in de eerdere hoofdstukken naar een andere klasse van vergelijkingen, de stochastische differentiaalvergelijkingen (SDVs). Hoofdstuk 5 is gewijd aan een nauwkeurig numeriek schema voor het uitvoeren van een grote tijdstap Monte Carlo simulatie in de tijd, het zeven-mijls SDV-discretisatieschema. Deze SDV-discretisatie is gebaseerd op een polynoom-chaos-expansiemethode, op basis van nauwkeurig bepaalde stochastische collocatie (SC) punten. Het basisidee is om een neuraal netwerk te trainen om deze SC-punten te leren, waarna de bijbehorende conditionele traditionele kansdichtheidsfunctie wordt geconstrueerd en de vereiste Monte Carlo-paden worden gegenereerd op basis van grote tijdsstappen. Een foutenanalyse bevestigt dat we tot nauwkeurige SDV-oplossingen kunnen komen in de zin van de sterke convergentie-eigenschappen. Met een variant die de compressie-decompressie-collocatie- en interpolatietechniek wordt genoemd, kunnen we het aantal aan te roepen neurale netwerkfuncties verder verminderen, zodat de rekensnelheid van de algehele methode wordt verhoogd. Numerieke experimenten tonen aan dat het nieuwe schema een convergentiefout van hoge kwaliteit behaalt en beter presteert dan sommige klassieke numerieke methoden. We presenteren enkele toepassingen in de waardering van financiële opties. Dit SVE-discretisatieschema met grote stappen kan echter worden gegeneraliseerd naar het oplossen van SDV's in andere toepassingsgebieden.

Samenvattend ontwikkelen we supervised deep learning-technieken als numerieke benaderingstechnieken om numerieke problemen in computationele financiële wiskunde aan te pakken. Er worden efficiënte, robuuste en nauwkeurige numerieke resultaten gepresenteerd.

CONTENTS

1

INTRODUCTION

1.1. MACHINE LEARNING IN FINANCE

With numerous successful applications in computer vision and natural language processing, Artificial Intelligence (AI) has been boosting and reshaping also financial engineering and services [1], with applications including high frequency trading, fraud detection, *robo-advisory* [2] and so on. As an example, during the COVID-19 pandemic situation, the social distancing measure prohibited meeting a financial advisor in person as much as possible. Robo-advisors (not necessarily mechanical robots) for portfolio management, which are composed of efficient algorithms *to calibrate and to compose a financial portfolio* in order to meet the desired reward goals and risk tolerances (e.g., given the retirement age, income, etc) of a customer, are employed, reducing the number of human advisors.

A corner stone of modern AI [3] is found in the Artificial Neural Networks (ANNs), that were put forward already in the early 1940s [4] as a computational model to mimic the human neural network system. There are many reasons why these networks have become so successful this decade, for instance, the rapid increase of computational power, due to the increasing performance of Graphics Processing Units (GPUs) and Tensor Processing Units (TPU), the simple operations within a neuron, the rapid development of variants due to computer science expertise (e.g., Recurrent NN or Convolutional NN to handle complex data structures), the parallel processing of hidden parameters, or their strong expressivity (e.g. by assembling neurons or layers) and ability to approximate highly nonlinear functions.

Computational finance is an important branch of finance, based on numerical mathematics methods and computer science paradigms to address the val-

uation of financial derivatives, risk management computations, and so on. A tremendous number of financial quantities (e.g., financial derivative prices, their sensitivities, etc) have to be calculated in the financial industry every day, thus a central request in computational finance is to develop fast, robust and efficient numerical algorithms. Besides the classical numerical techniques (e.g., finite difference discretizations of partial differential equations, Fourier methods to approximate occurring conditional expectations, Monte Carlo simulation), recently the ANNs are also emerging as powerful numerical tools within scientific computing.

For this reason, deep learning is introduced as a new paradigm, to either speed up solution processes (high speed) or solve highly complicated (e.g., high-dimension) problems. This dissertation is concerned with supervised deep learning in computational finance, in particular with so-called deep neural networks as powerful numerical tools within scientific computing. On the one hand, ANNs can sometimes augment the classical numerical methods, speeding up certain subtasks in a large computation, on the other hand ANNs sometimes replace complete numerical computations. Potential applications do not only occur in financial engineering, but also in financial services. For example, the calibration of financial asset models would require extremely fast model parameter fitting techniques, considering the real-time interaction and the rapidly changing market. In this PhD dissertation, we propose a highly efficient, and robust, neural network-based calibration framework (Calibration Neural Network) in Chapter 3.

1.1.1. FINANCIAL OPTIONS

Financial options are frequently traded in the market. As a financial derivative, the option's holder is given a right, but not an obligation, to trade (i.e., buy or sell) an underlying asset (e.g., equities, bonds, foreign currencies, etc) at a predetermined price over a specified time period, for instance, call or put options. There are various types of options, like European-style, American-style, Asian-style and so on. The well-known Black-Scholes equation gives a fair option value, assuming the underlying asset price follows Geometrical Brownian Motion. The advanced option pricing models include the Heston model, the Bates model and even the rough Heston model (with volatility dynamics being fractal Brownian motion), so on. However, those models require more intensive computation.

1.1.2. IMPLIED INFORMATION

The variance, i.e. volatility, of the asset price movement is an essential factor of determining an option's price. Different from historical volatility (computed

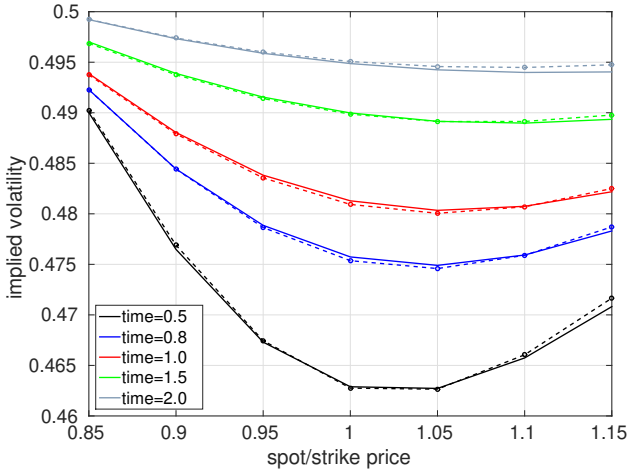


Figure 1.1: Implied volatility patterns.

from past known prices), implied volatility (computed from the observed option prices) reflects the future uncertainty (a forward-looking measure) of the underlying asset price. However, the implied volatility is not constant, either over time to maturity or strike prices. For example, the patterns, like implied volatility smile or skew in Figure 1.1, often present in the financial market, resulting from the heavy tailed distributions of the underlying asset price. In practice, people may prefer implied volatility to the 'real' option price. Thus computing implied volatility is a fundamental task in the financial engineering. Basically, calculating implied volatility can be viewed as a simplified calibration problem, that is, from an observed option price to the corresponding volatility using the Black-Scholes equation. In addition, implied dividend yield may also be an informative quantity for practitioners. When people are inverting option pricing models (including extracting multiple implied information, estimating model parameters) based on the market quotes, a fast calibration procedure is required to quickly capture the market dynamic.

In this book, we will take advantage of deep neural networks in a supervised-learning fashion to accelerate the computation of financial models and take a step forward to bring into production those models which were too slow to implement in practice.

1.2. SUPERVISED LEARNING

In general, present-day machine learning techniques fall essentially into one of three categories, i.e., supervised learning, unsupervised learning or reinforce-

ment learning. The different types of machine learning paradigms are used for different purposes. For example, supervised learning is typically employed for classification or regression tasks, unsupervised learning is used for clustering or for generative models, while reinforcement learning is able to perform a series of actions to maximize a certain target variable. In reality, a combination of these methods may even be needed to solve practical problems.

Supervised learning aims to find the mapping function given a ground truth of pairs of input and output quantities. Regarding ANNs, the universal approximation theorem (UAT) gives evidence that any continuous function can be approximated to any desired precision with the proper choice of ANN components, numbers of neurons and number of layers within ANNs.

The UAT can be presented in two ways, by means of an arbitrary width (an unlimited number of artificial neurons), in a shallow structure [5], or by an arbitrary depth (meaning an unlimited number of hidden layers, each consisting of a limited number of neurons) [6]. The above two UATs guarantee the construction of ANNs to accurately approximate a wide range of linear and nonlinear functions, however, they do not provide details about the way to achieve such a robust ANN construction. The training of ANNs used to suffer from all sorts of convergence problems, hampering the quality of function approximation. Issues like vanishing gradients in certain hidden layers, getting stuck in local optima, high-dimensional optimization landscape, over-fitting due to too many ANN parameters, lack of computation power had an enormous effect. Recent advances in the development of the new generation of ANNs (e.g. with a dropout functionality, back-propagation methodology to optimize the parameters, convolutions to avoid over-fitting, residual-based ANNs) enabled a robust and efficient training of present-day deep neural networks, possibly with very many hidden layers [7]. Furthermore, in many occasions, deep neural networks have given convincing evidence of their ability to approximate complex and highly nonlinear functions [8].

1.3. OUTLINE OF THIS DISSERTATION

We start with computing option prices using neural networks, then develop an ANN-based calibration framework to swiftly calibrate various option pricing models. Furthermore, we also address large time step Monte Carlo simulations of stochastic differential equations, which are widely used to describe the uncertainty in finance.

More specifically, this dissertation is subdivided into six chapters. The first chapter introduces generally machine learning in finance. The second chapter presents a data-driven neural network approach to price option prices, including multi-dimensional models. The third chapter is related to financial model cali-

bration. This means solving an inverse problem, given financial option values, to determine the corresponding asset model parameters, which is addressed by a newly developed ANN methodology, which we call the Calibration Neural Network (CaNN). The fourth chapter deals with extracting implied information from so-called American-style options, with early-exercise features, using the CaNN. The fifth chapter develops a novel numerical scheme to discretize stochastic differential equations (SDEs) on the basis of a large time step, and still obtain a highly accurate solution. It is well-known that SDEs are widely used to describe asset models in finance.

2

PRICING OPTIONS AND COMPUTING IMPLIED VOLATILITIES

This chapter introduces a data-driven approach, by means of an Artificial Neural Network (ANN), to value financial options and to calculate implied volatilities with the aim of accelerating the corresponding numerical methods. With ANNs being universal function approximators, this method trains an optimized ANN on a data set generated by a sophisticated financial model, and runs the trained ANN as an agent of the original solver in a fast and efficient way. We test this approach on three different types of solvers, including the analytic solution for the Black-Scholes equation, the COS method for the Heston stochastic volatility model and Brent's iterative root-finding method for the calculation of implied volatilities. The numerical results show that the ANN solver can reduce the computing time significantly.

2.1. INTRODUCTION

In computational finance, numerical methods are commonly used for the valuation of financial derivatives and also in modern risk management. Generally speaking, advanced financial asset models are able to capture nonlinear features that are observed in the financial markets. However, these asset price models are often multi-dimensional, and, as a consequence, do not give rise to closed-form solutions for option values.

Different numerical methods have therefore been developed to solve the corresponding option pricing partial differential equation (PDE) problems, e.g. fi-

This chapter is based on the article 'Pricing options and computing implied volatilities using Neural Networks', published in *Risks*, 2019, 7(1):16.

nite differences, Fourier methods and Monte Carlo simulation. In the context of financial derivative pricing, there is a stage in which the asset model needs to be calibrated to market data. In other words, the open parameters in the asset price model need to be fitted. This is typically *not* done by historical asset prices, but by means of *option prices*, i.e., by matching the market prices of heavily traded options to the option prices from the mathematical model, under the so-called risk-neutral probability measure. In the case of model calibration, thousands of option prices need to be determined in order to fit these asset parameters. However, due to the requirement of a highly efficient computation, certain high quality asset models are discarded. Efficient numerical computation is also increasingly important in financial risk management, especially when we deal with real-time risk management (e.g., high frequency trading) or counterparty credit risk issues, where a trade-off between efficiency and accuracy seems often inevitable.

Artificial neural networks (ANNs) with multiple hidden layers have become successful machine learning methods to extract features and detect patterns from large data sets. There are different neural network variants for particular tasks, for example, convolutional neural networks for image recognition [9] and recurrent neural networks for time series analysis [10]. It is well-known that ANNs can approximate nonlinear functions [11], [12], [5], and can thus be used to approximate solutions to PDEs [13], [14]. Recent advances in data science have shown that by using deep learning techniques even highly nonlinear multi-dimensional functions can be accurately represented [3]. Essentially, ANNs can be used as powerful universal function approximators without assuming any mathematical form for the functional relationship between the input variables and the output. Moreover, ANNs easily allow for parallel processing to speed up evaluations, especially on Graphics Processing Units (GPUs) or even Tensor Processing Units (TPUs) [15].

We aim to take advantage of a classical ANN to speed up option valuation by learning the results of an option pricing method. From a computational point of view, the ANN does not suffer much from the dimensionality of a PDE. An "ANN solver" is typically decomposed into two separate phases, a training phase and a test (or prediction) phase. During the training phase, the ANN "learns" the PDE solver, by means of the data set generated by the sophisticated models and corresponding numerical solvers. This stage is usually time consuming, however, it can be done off-line. During the test phase, the trained model can be employed to approximate the solution on-line. The ANN solution can typically be computed as a set of matrix multiplications, which can be implemented in parallel and highly efficiently. As a result, the trained ANN delivers financial derivative prices, or other quantities, efficiently, and the on-line time for accurate option

pricing may be reduced, especially for involved asset price models. We will show in this chapter that this data-driven approach is highly promising.

The proposed approach in this chapter attempts to accelerate the pricing of European options under a unified data-driven ANN framework. ANNs have been used in option pricing for some decades already. There are basically two directions. One is that based on observed market option prices and the underlying asset values, ANN-based regression techniques have been applied to fit a model-free, non-parametric pricing function, see, for example, [16–19]. Furthermore, the authors of [20, 21] designed special kernel functions to incorporate prior financial knowledge into the neural network while forecasting option prices.

Another direction is to improve the performance of model-based pricing by means of ANNs. The interest in accelerating classical PDE solvers via ANNs is rapidly growing. The papers [22–24] take advantage of reinforcement learning to speed up solving high-dimensional stochastic differential equations. The author of [25] proposes an optimization algorithm, the so-called stochastic gradient descent in continuous time, combined with a deep neural network to price high-dimensional American options. In [26] the pricing performance of financial models is enhanced by non-parametric learning approaches that deal with a systematic bias of pricing errors. Of course, this trend takes place not only in computational finance, but also in other engineering fields where PDEs play a key role, like computational fluid dynamics, see [14, 27–29]. The work in this chapter belongs to this latter direction. Here, we use traditional solvers to generate artificial data, then we train the ANN to learn the solution for different problem parameters. Compared to [13] or [14], our data-driven approach finds, next to the solutions of the option pricing PDEs, the implicit relation between variables and a specific parameter (i.e., the implied volatility).

This chapter is organized as follows. In Section 2.2, two fundamental option pricing models, the Black-Scholes and the Heston stochastic volatility PDEs, are briefly introduced. In addition to European option pricing, we also analyze robustness issues of root-finding methods to compute the so-called implied volatility. In Section 2.3, the employed ANN is presented with suitable hyperparameters. After training the ANN to learn the results of the financial models for different problem parameters, numerical ANN results with the corresponding errors are presented in Section 2.4.

2.2. OPTION PRICING AND ASSET MODELS

In this section, two asset models are briefly presented, the geometric Brownian motion (GBM) asset model, which gives rise to the Black-Scholes option pricing PDE, and the Heston stochastic volatility asset model, leading to the Heston PDE. We also discuss the concept of implied volatility. We will use European op-

tion contracts as the examples, however, other types of options can be taken into consideration in a similar way.

2.2.1. THE BLACK-SCHOLES PDE

A first model for asset prices is GBM,

$$dS(t) = \mu S(t)dt + \sqrt{\nu}S(t)dW_s(t), S(t_0) = S_0 > 0, \tag{2.1}$$

where S is the price of an non-dividend paying asset, and W^s is a Wiener process, with t being the time, μ the drift parameter, and ν the variance parameter. The volatility parameter is $\sigma = \sqrt{\nu}$. Under a risk-neutral measure (i.e. $r := \mu$, where r is the risk-free interest rate), a European option contract on the underlying stock price can be valued via the Black-Scholes PDE, which can be derived from Itô's Lemma under a replicating portfolio approach or via the martingale approach. Denoting the option price by $V(t,S)$, the Black-Scholes equation reads,

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0, \tag{2.2}$$

with time t until to maturity T, and r the risk-free interest rate. The PDE is accompanied by a final condition representing the specific payoff, for example, the European call option payoff at time T,

$$V(t = T, S) = \max(S(T) - K, 0), \tag{2.3}$$

where K is the option's strike price. See standard textbooks for more information about the basics in financial mathematics.

An analytic solution to (2.2), (4.19) exists for European plain vanilla options, i.e.,

$$V_{eu}^C(t,S) = SN(d_1) - Ke^{-r\tau}N(d_2), \tag{2.4a}$$

$$d_1 = \frac{\log(S/K) + (r - 0.5\sigma^2)\tau}{\sigma\sqrt{\tau}}, \; d_2 = d_1 - \sigma\sqrt{\tau}, \tag{2.4b}$$

where $\tau := T - t$ represents time to maturity, $V_{eu}^C(t,S)$ is the European call option value at time t for stock value S, with $N(\cdot)$ being the cumulative function of the standard normal distribution. This solution procedure (2.4) is denoted by $V(\cdot) = BS(\cdot)$.

2.2.2. IMPLIED VOLATILITY

Implied volatility is considered an important quantity in finance. Given an observed market option price V^{mkt}, the Black-Scholes implied volatility σ^* can be

determined by solving $BS(\sigma^*; S, K, \tau, r) = V^{mkt}$. The monotonicity of the Black-Scholes equation with respect to the volatility guarantees the existence of $\sigma^* \in [0, +\infty]$. We can write the implied volatility as an implicit formula,

$$\sigma^*(K, T) = BS^{-1}(V^{mkt}; S, K, \tau, r), \tag{2.5}$$

where BS^{-1} denotes the inverse Black-Scholes function. Moreover, by adopting moneyness, $m = \frac{S(t)}{K}$, and time to maturity, $\tau = T - t$, one can express the implied volatility as $\sigma^*(m, \tau)$, see [30].

For simplicity, we denote here $\sigma^*(m, \tau)$ by σ^*. An analytic solution for Equation (2.5) does not exist. The value of σ^* is determined by means of a numerical iterative technique, since Equation (2.5) can be converted into a root-finding problem,

$$g(\sigma^*) = BS(S, \tau, K, r, \sigma^*) - V^{mkt}(S, \tau; K, r) = 0. \tag{2.6}$$

2.2.3. THE HESTON MODEL

One of the limitations of using the Black-Scholes model is the assumption of a constant volatility σ in (2.2), (2.4). A major modeling step away from the assumption of constant volatility in asset pricing, was made by modeling the volatility/variance as a diffusion process. The resulting models are the stochastic volatility (SV) models. The idea to model volatility as a random variable is confirmed by practical financial data which indicates the variable and unpredictable nature of the stock price's volatility. The most significant argument to consider the volatility to be stochastic is the implied volatility smile/skew, which is present in the financial market data, and can be accurately recovered by SV models, especially for options with a medium to long time to the maturity date T. With an additional stochastic process, which is correlated with the asset price process $S(t)$, we deal with a *system of SDEs*, for which option valuation is more computationally expensive than for a scalar asset price process.

The most popular SV model is the Heston model [31], for which the system of stochastic equations under the risk-neural measure reads,

$$dS(t) = rS(t)dt + \sqrt{v(t)}S(t)dW_s(t), \;\; S(t_0) = S_0 > 0, \tag{2.7a}$$

$$dv(t) = \kappa(\bar{v} - v(t))dt + \gamma\sqrt{v(t)}dW_v(t), \;\; v(t_0) = v_0 > 0, \tag{2.7b}$$

$$dW_s(t)dW_v(t) = \rho dt, \tag{2.7c}$$

with $v(t)$ the instantaneous variance, and $W_s(t), W_v(t)$ are two Wiener processes with correlation coefficient ρ. The second equation in (2.7) models a mean reversion process for the variance, with the parameters, r the risk-free interest rate,

$\bar{v}$ the long term variance, κ the reversion speed; γ is the volatility of the variance, determining the volatility of $v(t)$. There is an additional parameter v_0, the t_0-value of the variance.

By the martingale approach, we arrive at the following multi-dimensional Heston option pricing PDE,

$$\begin{aligned} \frac{\partial V}{\partial t} &+ rS\frac{\partial V}{\partial S} + \kappa(\bar{v}-v)\frac{\partial V}{\partial v} + \frac{1}{2}vS^2\frac{\partial^2 V}{\partial S^2} \\ &+ \rho\gamma Sv\frac{\partial^2 V}{\partial S\partial v} + \frac{1}{2}\gamma^2 v\frac{\partial^2 V}{\partial v^2} - rV = 0, \end{aligned} \tag{2.8}$$

with the given terminal condition $V(T,S,v;T,K)$, where $V = V(t,S,v;T,K)$ is the option price at time t.

The typically observed implied volatility shapes in the market, e.g. smile or skew, can be reproduced by varying the above parameters $\{\kappa,\rho,\gamma,v_0,\bar{v}\}$. In general, the parameter γ impacts the kurtosis of the asset return distribution, and the coefficient ρ controls its asymmetry. The Heston model does not have analytic solutions, and is thus solved numerically.

Numerical methods in option pricing generally fall into three categories, finite differences (FD), Monte Carlo (MC) simulation and numerical integration methods. Finite differences for the PDE problem are often used for free boundary problems, as they occur when valuing American options, or for certain exotic options like barrier options. Meanwhile, the derivatives of the option prices (the so-called option Greeks) are accurately computed with finite differences.

Monte Carlo simulation and numerical integration rely on the Feyman-Kac Theorem, which essentially states that (European) option values can be written as discounted expected values of the option's payoff function at the terminal time T, under the risk-neutral measure. Monte Carlo methods are often employed in this context for the valuation of path-dependent high-dimensional options, and also for the computation of all sorts of valuation adjustments in modern risk management. However, Monte Carlo methods are typically somewhat slow to converge, and particularly in the context of model calibration this can be an issue.

The numerical integration methods are also based on the Feyman-Kac Theorem. The preferred way to employ them is to first transform to the Fourier domain. The availability of the asset price's characteristic function is a pre-requisite to using Fourier techniques. One of the efficient techniques in this context is the COS method [32], which utilizes Fourier-cosine series expansions to approximate the asset price's probability density function, but is based on the characteristic function. The COS method can be used to compute European option values under the Heston model highly efficiently. However, for many different, modern

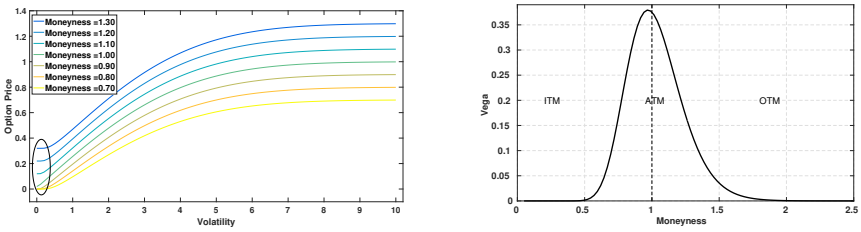
asset models the characteristic function is typically not available. We will use the Heston model with the COS method here during the training of the Heston-ANN, so that training time is still relatively small.

2.2.4. NUMERICAL METHODS FOR IMPLIED VOLATILITY

Focussing on the implied volatility σ^*, there are several iterative numerical techniques to solve (2.6), for example, the Newton-Raphson method, the bisection method or the Brent method. The Newton-Raphson iteration reads,

$$\sigma^*_{k+1} = \sigma^*_k - \frac{V(\sigma^*_k) - V^{mkt}}{g'(\sigma^*_k)}, \; k = 0, \ldots. \tag{2.9}$$

Starting with an initial guess, σ^*_0, the approximate solutions, σ^*_{k+1}, $k = 0, \ldots$, iteratively improve, until a certain criterion is satisfied. The first derivative of Black-Scholes option value with respect to the volatility, named the option's Vega, in the denominator of (2.9) can be obtained analytically for European options.



(a) Option price vs. volatility

(b) Vega vs. Moneyness.

Figure 2.1: Vega tends to be zero in certain regions of deep ITM or OTM options. (**a**) Option price vs. volatility; (**b**) Vega vs. Moneyness.

However, the Newton-Raphson method may fail to converge, either when the Vega is extremely small or when the convergence stalls. The Black-Scholes equation monotonically maps an unbounded interval $\sigma \in [0, +\infty)$ to a finite range $V(t,S) \in [0, S(t) - Ke^{-r\tau}]$, as illustrated in Figure 2.1a . As a result, near-flat function forms appear in certain σ-regions, especially when the option is either deep in-the-money (ITM) or deep out-the-money (OTM). For example, Figure 2.1b shows that the option's Vega can be very close to zero in the regions with small or large volatilities of deep ITM or OTM options, although its value is relatively large in the at-the money (ATM) region.

There exist some techniques to remedy this problem in the iterative process. A possible robust root-finding algorithm for solving this problem is to employ a hybrid of the Newton-Raphson and the bisection methods. In addition, the

author of [33] proposed to select a suitable initial value at the beginning of the iteration to avoid divergence.

Alternatively, a closed-form expression can be derived to approximate the implied volatility of a financial option in certain parameter ranges, see [34–36]. Such methods are based on a Taylor series expansion and the analytical solution of the European-style option-pricing model. One of the drawbacks, however, is that the derived formulas perform well only near at-the-money (ATM), but give rise to inaccurate implied volatility for deep ITM/OTM options.

In the next subsection, we will introduce a derivative-free, robust and efficient algorithm to find the implied volatility.

BRENT'S METHOD FOR IMPLIED VOLATILITY

As a derivative-free, robust and efficient algorithm, Brent's method [37] combines bisection, inverse quadratic interpolation and the secant method. In order to determine the next iterant, an inverse quadratic interpolation employs three prior points (i.e., iterants) to fit an inverse quadratic function, which resembles the gradient of Newton's method, i.e.

$$\begin{aligned}\sigma_{k+1} = &\frac{\sigma_k g(\sigma_{k-1}) g(\sigma_{k-2})}{(g(\sigma_{(k)}) - g(\sigma_{k-1}))(g(\sigma_k) - g(\sigma_{k-2}))} \\ &+ \frac{\sigma_{k-1} g(\sigma_{k-2}) g(\sigma_k)}{(g(\sigma_{k-1}) - g(\sigma_{k-2}))(g(\sigma_{k-1}) - g(\sigma_k))} \\ &+ \frac{\sigma_{k-2} g(\sigma_{k-1}) g(\sigma_k)}{(g(\sigma_{k-2}) - g(\sigma_{k-1}))(g(\sigma_{k-2}) - g(\sigma_k))}.\end{aligned} \tag{2.10}$$

When two consecutive approximations are identical, for example, $\sigma_k = \sigma_{k-1}$, the quadratic interpolation is replaced by an approximation based on the secant method,

$$\sigma_{k+1} = \sigma_{k-1} - g(\sigma_{k-1}) \frac{\sigma_{k-1} - \sigma_{k-2}}{g(\sigma_{k-1}) - g(\sigma_{k-2})}. \tag{2.11}$$

Here, Brent's method is used to compute the BS implied volatility related to the Heston option prices in Section 2.4.4. We will develop an ANN to approximate the implicit function relating the volatility to the option price.

2.2.5. COS METHOD FOR PRICING OPTIONS

In this section, a brief description of the COS method to compute European-style option prices is presented, with the aim of generating the simulation data sets needed in Chapters 2 and 3. An advanced variant for calculating American-style option prices will be introduced in Chapter 4.

Based on the Feynman-Kac Theorem, the solution of the governing option valuation PDEs (for example, for the Heston stochastic volatility PDE) is given by the risk-neutral valuation formula,

$$V(t_0,x,v) = e^{-r\Delta t}\int_{-\infty}^{\infty} V(T,y,v)f(y|x)dy,$$

where $V(t,x,v)$ is the option value, and x,y are increasing functions of the underlying at t_0 and T, respectively, and v is the asset's variance. To arrive at the COS formula for European option valuation, we need to truncate the integration range, so that

$$V(t_0,x,v) \approx e^{-r\Delta t}\int_{a}^{b} V(T,y,v)f(y|x)dy, \tag{2.12}$$

with $|\int_{\mathbb{R}} f(y|x)dy - \int_a^b f(y|x)dy| < TOL$.

The probability density function of the underlying price, under the risk-neutral pricing measure, is then approximated by means of the corresponding characteristic function, however, with a truncated Fourier cosine expansion, as follows,

$$f(y|x) \approx \frac{2}{b-a}\sum_{k=0}^{N_{COS}-1}{}' Re\left\{\hat{f}\left(\frac{k\pi}{b-a};x\right)\exp\left(-i\frac{ak\pi}{b-a}\right)\right\}\cos\left(k\pi\frac{y-a}{b-a}\right), \tag{2.13}$$

where N_{COS} represents the number of cosine terms, and $Re\{\cdot\}$ stands for taking the real part of the expression in the brackets. The function $\hat{f}(\omega;x)$ is the characteristic function of $f(y|x)$, which is defined as

$$\hat{f}(\omega;x) = \mathbb{E}(e^{i\omega y}|x). \tag{2.14}$$

The prime at the sum symbol in (2.13) indicates that the first term in the cosine expansion should be multiplied by one-half. Replacing $f(y|x)$ by its approximation (2.13) in (2.12) and interchanging the integration and summation operations, based on Fubini's Theorem, gives us the COS method to approximate the value of a European option:

$$V(t_0,x,v) = e^{-r\Delta t}\sum_{k=0}^{N_{COS}-1}{}' Re\left\{\hat{f}\left(\frac{k\pi}{b-a};x\right)e^{-ik\pi\frac{a}{b-a}}\right\}H_k, \tag{2.15}$$

where

$$H_k = \frac{2}{b-a}\int_a^b V(T,y,v)\cos\left(k\pi\frac{y-a}{b-a}\right)dy, \tag{2.16}$$

represent the Fourier cosine coefficients of $H(t,y) = V(T,y,v)$, which are available in closed-form for several European-style option payoff functions. The size of the integration interval $[a,b]$ can be determined by a rule of thumb, as follows,

$$[a,b] := \left[\xi_1 - L_{COS}\sqrt{\xi_2+\sqrt{\xi_4}}, \xi_1 + L_{COS}\sqrt{\xi_2+\sqrt{\xi_4}}\right], \tag{2.17}$$

where ξ_n are the n-th cumulants, and $L_{COS} > 0$ is a user-defined parameter for the interval size. The COS method exhibits an exponential convergence rate for those processes whose transitional probability density function, $f(y|x) \in C^{\infty}$, and $(a,b) \subset \mathbb{R}$. More details can be found in the paper [38].

Equation (2.15) can now be directly applied to calculate the value of European options. It also forms the basis for the pricing of Bermudan options, as explained in Chapter 4.

2.3. METHODOLOGY

In this section, we present a neural network to approximate a function for financial models. The procedure comprises two main components, the generator to create the financial data for training the model and the predictor (the ANN) to approximate the option prices based on the trained model. The data-driven framework consists of the following steps,

Algorithm 1 Model framework

– Generate the sample data points for input parameters,

– Calculate the corresponding output (option price or implied volatility) to form a complete data set with inputs and outputs,

– Split the above data set into a training and a test part,

– Train the ANN on the training data set,

– Evaluate the ANN on the test data set,

– Replace the original solver by the trained ANN in applications.

2.3.1. ARTIFICIAL NEURAL NETWORKS

ANNs generally constitute three levels of components, i.e., neurons, layers and the architecture, from bottom to top. The architecture is determined by a combination of different layers, that are made up of numerous artificial neurons. A neuron, which involves learnable weights and biases, is the fundamental unit of ANNs. By connecting the neurons of adjacent layers, output signals of a previous layer enter a next layer as input signal. By stacking layers on top of each other, signals travel from the input layer through the hidden layers to the output layer potentially through cyclic or recurrent connections, and the ANN builds a mapping among input-output pairs.

As shown in Figure 2.2a, an artificial neuron basically consists of the following three consecutive operations:

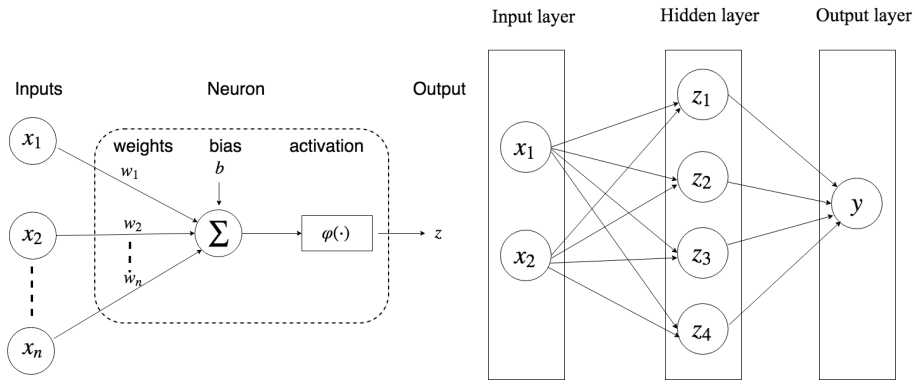
1. Calculation of a summation of weighted inputs,
2. Addition of a bias to the summation,
3. Computation of the output by means of a transfer function.

The basic ANN is the multi-layer perceptron (MLP), which can be written mathematically as a composite function,

$$\hat{H}(\acute{\mathbf{x}}|\hat{\boldsymbol{\Theta}}) = \hat{h}_{L_A}(...\hat{h}_2(\hat{h}_1(\acute{\mathbf{x}};\hat{\boldsymbol{\theta}}_1);\hat{\boldsymbol{\theta}}_2);...\hat{\boldsymbol{\theta}}_{L_A}), \tag{2.18}$$

where $\acute{\mathbf{x}} = (x_1, x_2, \ldots, x_n)$ stands for the input variables, and L_A is the number of hidden layers, and $\hat{\boldsymbol{\theta}}_i = (\mathbf{w}_i, \mathbf{b}_i)$, with $\mathbf{w}_i$ being a weight matrix and $\mathbf{b}_i$ being a bias vector. Meanwhile, $\hat{h}(\cdot)$ represents the function corresponding to a hidden layer. For simplicity, in this book, the mapping function may be expressed as follows,

$$y(\acute{\mathbf{x}}) = \hat{H}(\acute{\mathbf{x}}|\hat{\boldsymbol{\Theta}}). \tag{2.19}$$



(a) A neuron (b) An example of MLP

Figure 2.2: Illustration of an MLP configuration. (**a**) A neuron; (**b**) An example of MLP.

Let $z_j^{(\ell)}$ denote the value of the j-th neuron in the ℓ-th layer, then the corresponding layer function reads,

$$z_j^{(\ell)} = \varphi^{(\ell)}\left(\sum_i w_{ij}^{(\ell)} z_i^{(\ell-1)} + b_j^{(\ell)}\right), \tag{2.20}$$

where $z_i^{(\ell-1)}$ is the output value of the i-th neuron in the $(\ell-1)$-th layer and $\varphi(\cdot)$ is an activation function, with $w_{ij}^{(\ell)} \in \mathbf{w}_\ell$, $b_j^{(\ell)} \in \mathbf{b}_\ell$. In other words, the hidden layer function $\hat{h}_\ell$ applies the weights and biases to the inputs and directs the

summation through an activation function $\varphi^{(\ell)}(\cdot)$ as the output of $\hat{h}_\ell$. When $\ell = 0$, $z^{(0)} = x$ is the input layer; When $\ell = L_A$, $z^{(L_A)} = y$ is the output layer; Otherwise, $z^{(\ell)}$ represents an intermediate variable. The activation function $\varphi(\cdot)$ adds nonlinearity to the system, for example, the following activation functions may be employed,

- Relu, $\varphi(x) = \max(x, 0)$,
- Sigmoid, $\varphi(x) = \dfrac{1}{1+e^{-x}}$,
- Leaky ReLu, $\varphi(x) = \max(x, ax)$, $0 < a < 1$;

see [3] for more activation functions. An MLP with "one-hidden-layer" is shown in Figure 2.2b, and Equation (2.21) presents its mathematical formula,

$$\begin{cases} y = \varphi^{(2)}\left(\sum_j w_j^{(2)} z_j^{(1)} + b^{(2)}\right) \\ z_j^{(1)} = \varphi^{(1)}\left(\sum_i w_{ij}^{(1)} x_i + b_j^{(1)}\right). \end{cases} \tag{2.21}$$

According to the Universal Approximation Theorem [11], a single-hidden-layer ANN with a sufficient number of neurons can approximate any continuous function. The distance between two functions is measured by the norm of a function $||\cdot||$,

$$D(\hat{F}(\acute{\mathbf{x}}), \hat{H}(\acute{\mathbf{x}})) = ||\hat{F}(\acute{\mathbf{x}}) - \hat{H}(\acute{\mathbf{x}})||, \quad \acute{\mathbf{x}} \in \acute{\Omega} \tag{2.22}$$

where $\acute{\Omega}$ is the definition domain, and $\hat{F}(\acute{\mathbf{x}})$ stands for the objective function, and $\hat{H}(\acute{\mathbf{x}})$ for the neural network approximated function. For example, the p-norm in the domain $\acute{\Omega}$ reads,

$$D(\hat{F}(\acute{\mathbf{x}}), \hat{H}(\acute{\mathbf{x}})) = ||\hat{F}(\acute{\mathbf{x}}) - \hat{H}(\acute{\mathbf{x}}|\hat{\mathbf{\Theta}})||_p = \sqrt[p]{\int_{\acute{\Omega}} |\hat{F}(\acute{\mathbf{x}}) - \hat{H}(\acute{\mathbf{x}}|\hat{\mathbf{\Theta}})|^p d\acute{\mathbf{x}}},$$

where $1 \le p < \infty$. We choose p=2 to evaluate the averaged accuracy, which corresponds to the root mean squared error (RMSE) and can be easily converted to the popular error MSE (mean squared error). Within supervised learning, the loss function $L(\cdot)$ is equivalent to the above distance,

$$L(\hat{\mathbf{\Theta}}) := D(\hat{F}(\acute{\mathbf{x}}), \hat{H}(\acute{\mathbf{x}}|\hat{\mathbf{\Theta}})). \tag{2.23}$$

The discrete form of the loss function (2.23) is widely used in practice. Suppose, in the domain $\acute{\Omega}$, there is a collection of data points $\{\acute{\mathbf{x}}_k\}$, $k = 1, \ldots, M_D$, and their corresponding function value $\{\acute{\mathbf{y}}_k := \hat{F}(\acute{\mathbf{x}}_k)\}$, which form a vector of input-output pairs $(\acute{\mathbf{X}}, \acute{\mathbf{Y}}) = \{(\acute{\mathbf{x}}_k, \acute{\mathbf{y}}_k)\}_{k=1,\ldots,M_D}$. The training process aims to learn the optimal weights and biases in Equation (2.19) to make the loss function as small

as possible. For example, when $p = 2$, the above process can be formulated as an optimization problem,

$$\underset{\hat{\boldsymbol{\Theta}}}{\operatorname{argmin}} L(\hat{\boldsymbol{\Theta}}|(\acute{\mathbf{X}}, \acute{\mathbf{Y}})) \approx \underset{\hat{\boldsymbol{\Theta}}}{\operatorname{argmin}} \sqrt{\frac{1}{M_D} \sum_{k=1}^{M_D} \left(\acute{\mathbf{y}}_k - \hat{H}(\acute{\mathbf{x}}_k|\hat{\boldsymbol{\Theta}})\right)^2}, \tag{2.24}$$

given the known input-output pairs $(\acute{\mathbf{X}}, \acute{\mathbf{Y}})$ and a loss function $L(\hat{\boldsymbol{\Theta}})$. When the training data set $(\acute{\mathbf{X}}, \acute{\mathbf{Y}})$ can define the true function on the domain $\acute{\Omega}$, ANNs with sufficiently many neurons can approximate this function in a certain norm, e.g., the l_2-norm.

Quantitative theoretical error bounds for deep ANNs to approximate any function are not yet available. For continuous functions, in the case of a single hidden layer, the number of neurons should grow exponentially with the input dimensionality [39]. In the case of two hidden layers, the number of neurons should grow polynomially. The authors in [40] proved that any continuous function defined on the unit hypercube $C[0,1]^d$ can be uniformly approximated to arbitrary precision by a two hidden layer MLP, with $3d$ and $6d+3$ neurons in the first and second hidden layer, respectively. In [41] the error bounds for approximating smooth functions by ANNs with adaptive depth architectures are presented. The theory gets complicated when the ANN structure goes deeper, however, these deep neural networks have recently significantly increased the power of ANNs, see, for example the Residual Neural Networks [6].

Several back-propagation gradient descent methods have been successfully applied to optimize the system (2.24), for instance, Stochastic Gradient Descent (SGD) [42]. These optimization algorithms start with initial values and move in the direction in which the loss function decreases significantly. The formulas for updating the parameters read,

$$\begin{cases} \mathbf{w} \leftarrow \mathbf{w} - \eta(i) \dfrac{\partial L}{\partial \mathbf{w}}, \\ \mathbf{b} \leftarrow \mathbf{b} - \eta(i) \dfrac{\partial L}{\partial \mathbf{b}}, \\ i = 0, 1, 2, ..., \end{cases} \tag{2.25}$$

where η is a learning rate, which may vary during the iterations. The learning rate plays an important role during the training, as a "large" learning rate value causes the ANN's convergence to oscillate, whereas a small one results in ANNs learning slowly, and even getting trapped in local optima regions. An adaptive learning rate is often preferred, and more details will be given in Section 2.3.3.

In practice, the gradients are computed over mini-batches because of computer memory limitations. Instead of all input samples, a portion is randomly

selected within each iteration to calculate an approximation of the gradient of the objective function. The size of the mini-batch is used to determine the portion. Due to the architecture of the GPUs, batch sizes of powers of two can be efficiently implemented. Several variants of SGD have been developed in the past decades, e.g., RMSprop and Adam [43], where the latter method handles an optimization problem adaptively by adjusting the involved parameters over time.

2.3.2. HYPER-PARAMETERS OPTIMIZATION

Training deep neural networks involves numerous choices for the commonly called "ANN hyper-parameters". These include the number of layers, neurons, and the specific activation function. Determining the depth (the number of hidden layers) and the width (the number of neurons) of the ANN is a challenging problem.

We experimentally find that an MLP architecture with four hidden layers has an optimal capacity of approximating option pricing formulas of our current interest. Built on a four hidden layer architecture, the other hyper-parameters are optimized also by using machine learning [44]. There are different techniques to implement the automatic search. In a grid search technique, all candidate parameters are systematically parameterized on a pre-defined grid, and all possible candidates are explored in a brute-force way. The authors of [45] concluded that random search is more efficient for hyper-parameters optimization. Recently, Bayesian hyper-parameter optimization [46] has been developed to efficiently reduce the computational cost by navigating through the hyper-parameters space. However, it is difficult to outperform random search in combination with certain expert knowledge.

Neural networks may not necessarily converge to a global minimum. However, using a proper *random initialization* may help the model with suitable initial values. *Batch normalization* scales the output of a layer by subtracting the batch mean and dividing by the batch standard deviation. This can speed up the training of the neural network. The batch size indicates the number of samples that enter the model to update the learnable parameters within one iteration. A *dropout operation* selects a random set of neurons and deactivates them, which forces the network to learn more robust features. The dropout rate refers to the proportion of the deactivated neurons in a layer.

There are two stages to complete the hyper-parameter optimization. During the model selection process, over-fitting can be reduced by adopting the k-fold cross validation as follows.

In the first stage, we employ random search combined with a 3-fold cross validation to find initial hyper-parameter configurations for the neural network.

Algorithm 2 k-fold cross validation

–Split the training data set into k different subsets,

–Select one set as the validation data set,

–Train the model on the remaining k-1 subsets,

–Calculate the metric by evaluating the trained model on the validation part,

–Continue the above steps by exploring all subsets,

–Calculate the final metric which is averaged over k cases,

–Explore the next set of hyper-parameters,

–Rank the candidates according to their averaged metric.

Table 2.1: The setting of random search for hyper-parameters optimization

Parameters	Options or Range
Activation	ReLu, tanh, sigmoid
Dropout rate	[0.0, 0.2]
Neurons	[200, 600]
Initialization	uniform, glorot_uniform, he_uniform
Batch normalization	yes, no
Optimizer	SGD, RMSprop, Adam
Batch size	[256, 3000]

As shown in Table 2.1, each model is trained 200 epochs using MSE as the loss metric. An epoch is the moment when the model has processed the whole training data set. It is found that the prediction accuracy increases with the training data set size (more related details will be discussed in Section 2.4.1). Therefore, the random search is implemented on a small data set, which is then followed by training the selected ANN on larger data sets in the next stage.

In the second stage, we further enhance the top 5 network configurations by averaging the different values, to yield the final ANN model, as listed in Table 3.2. As Table 3.2 shows, the optimal parameter values, neurons and batch size, do not lie at the boundaries of the search space (except for the drop out rate). Compared to the Sigmoid activation function, ReLu is more likely to give rise to a better convergence (e.g. overcome the vanishing gradient in a deep neural network). As an extension to SGD, the Adam optimizer can handle an optimization problem in a more robust way. However, batch normalization and drop-out did not improve the model accuracy in this regression problem, and one possible reason for this is

Table 2.2: The selected model after the random search

Parameters	Options
Hidden layers	4
Neurons(each layer)	400
Activation	ReLu
Dropout rate	0.0
Batch-normalization	No
Initialization	Glorot_uniform
Optimizer	Adam
Batch size	1024

that the output value is sensitive to the input parameters, which is different from sparse features in an image (where these operations usually work very well). Subsequently, we train the selected network on the whole (training and validation) data set, to obtain the final weights. This procedure resulted in an ANN with sufficient accuracy to approximate the financial option values.

2.3.3. LEARNING RATES

The learning rate, one of the key hyper-parameters, represents the rate at which the weights are updated each iteration. A large learning rate leads to fluctuations around a local minimum, and sometimes even to divergence. Small learning rates may cause an inefficiently slow training stage. It is common practice to start with a large learning rate and then gradually decrease it until a well-trained model has resulted. There are different ways to vary the learning rate during training, e.g. by step-wise annealing, exponential decay, cosine annealing, see [47] for a cyclical learning rate (CLR) and [48] for the stochastic descent gradient restart (SDGR). The basic idea of CLR and SDGR is that at certain points of the training stage, a relatively large learning rate may move the weights from their current values, by which ANNs may leave a local optimum and converge to a better one.

We employ the method proposed in [47] to determine the learning rate. The method is based on the insight of how the averaged training loss varies over different learning rates, by starting with a small learning rate and increasing it progressively in the first few iterations. By monitoring the loss function against the learning rate, it is shown in Figure 2.3 that the loss stabilizes when the learning rate is small, then drops rapidly and finally oscillates and diverges when the learning rate is too large. The optimal learning rate lies here between 10^{-5} and

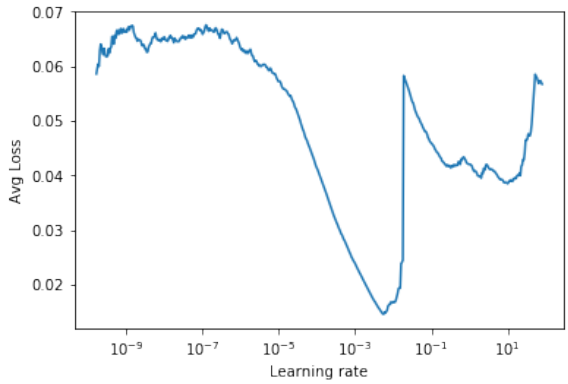


Figure 2.3: Average training loss against varying learning rates.

10^{-3}, where the slope is the steepest and the training loss reduces quickly. Therefore, the learning rate in CLR is reduced from 10^{-3} to 10^{-5} in our experiments.

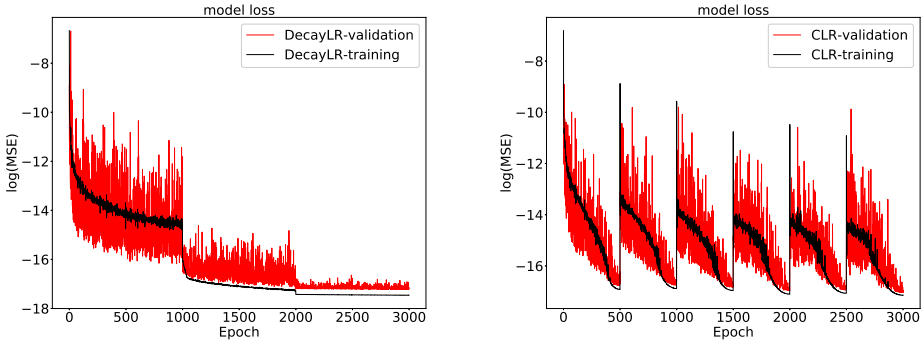
We present, as an example, the results of the training stage of the ANN solver for the Heston model option prices to compare three different learning rate schedules. Figure 2.4 demonstrates that the training error and the validation error agree well and that over-fitting does not occur when using these schedules. As shown in Figure 2.5, in this case a decay rate-based schedule outperforms the CLR with the same learning rate bounds, although with the CLR the differences between training and validation losses are smaller. This is contrary to the conclusion in [47], but their network included batch normalization and L2 regularization. For the tests in this chapter, we will employ the CLR to find the optimal range of learning rates, which is then applied in the DecayLR schedule to train the ANNs.

2.4. NUMERICAL RESULTS

We show the performance of the ANNs for solving the financial models, based on the following accuracy metrics (which forms the basis for the training),

$$\text{MSE} = \frac{1}{n}\sum(y_i - \hat{y}_i)^2, \tag{2.26}$$

where n is the number of involved data points, and y_i stands for the actual value, and $\hat{y}_i$ for the ANN predicted value. The MSE is used as the loss function in (2.23) to update the weights, and different metrics are employed to evaluate the se-



(a) (b)

Figure 2.4: The history of training and validation losses for the Heston model. (**a**) Losses with decaying learning rates; (**b**) Losses with cyclical learning rates.

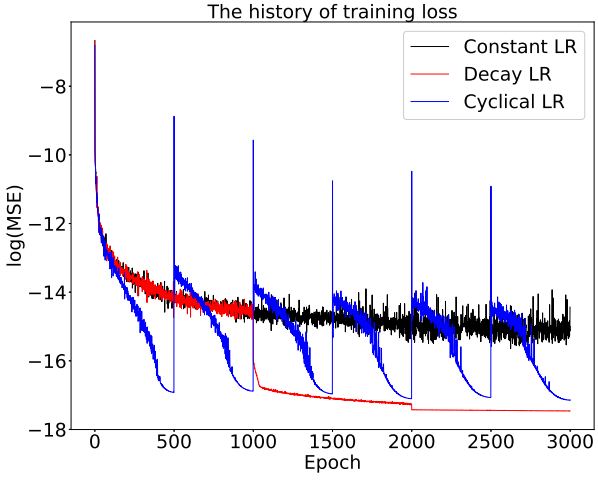


Figure 2.5: Different learning rate schedules for training ANNs on Heston model.

lected ANN. For completeness, we also report the other well-known metrics,

$$\text{RMSE} = \sqrt{\text{MSE}}, \tag{2.27a}$$

$$\text{MAE} = \frac{1}{n}\sum |y_i - \hat{y}_i|, \tag{2.27b}$$

$$\text{MAPE} = \frac{1}{n}\sum \frac{|y_i - \hat{y}_i|}{y_i}. \tag{2.27c}$$

We start with the Black-Scholes model which gives us closed-form option prices, that are learned by the ANN. We also train the ANN to learn the implied volatility, based on the iterative root-finding Brent method. Finally, the ANN learns the results obtained by the COS method to solve the Heston model with several different parameters.

2.4.1. DETAILS OF THE DATA SET

Like with any data-driven approach, the quality of a data set has a significant impact on the performance of the resulting model. Theoretically, an arbitrary number of samples can be generated since the mathematical model is known. In reality, a sampling technique with good space-filling properties should be preferable. Latin hypercube sampling (LHS) [49] is able to generate random samples of the parameter values from a multidimensional distribution, resulting in a better representation of the parameter space. When the sample data set for the input parameters is available, we select the appropriate numerical methods to generate the training results. For the Black-Scholes model, the option prices are obtained by the closed-form formula. For the Heston model, the prices are calculated by the COS method with a robust COS method version. With the Heston prices determined, Brent's method will be used to find the corresponding implied volatility. The whole data set is randomly divided into two groups, 90% will be the training and 10% the test set.

Table 2.3: The different sizes of training data set when training the ANN

Case	0	1	2	3	4	5	6
Training size (×24300)	1/8	1/4	1/2	1	2	4	8

In order to investigate the relation between the prediction accuracy and the size of the training set, we increase the number of training samples from $\frac{1}{8}$ to 8 times the baseline set, as shown in Table 2.3. Meanwhile, the test data is kept unchanged. The example here is learning the implied volatility. We first train the ANN for each data set by using a decaying learning rate, as described in Section 2.3.3, and repeat the training stage for each case 5 times with different random seeds and average the model performance. As shown in Figure 2.6, with an increasing data size, the prediction accuracy increases and the corresponding variance, indicated by the error bar, decreases. So, we employ random search for the hyper-parameters on a small-sized data set, and train the selected ANN on a large data set. The schedule of decaying learning rates is as discussed in Section 2.3.3. The training and validation losses remain close, which indicates that there

is no over-fitting.

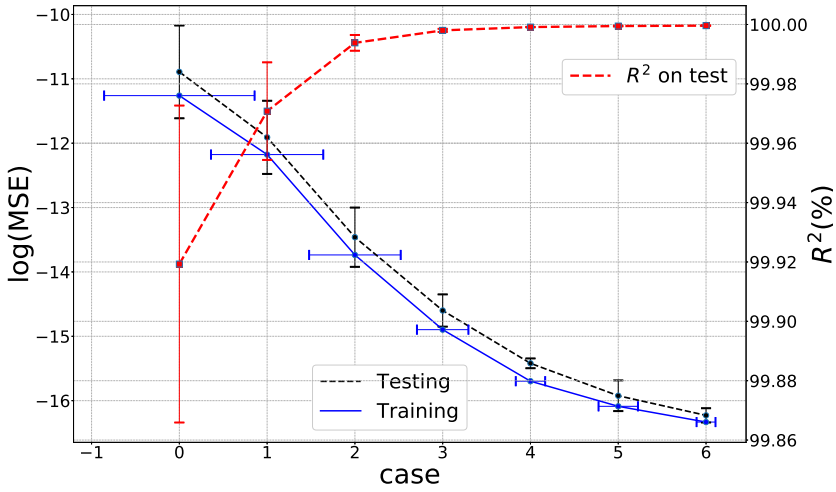


Figure 2.6: R^2 and MSE vs. size of the training set. The figure shows improvement in the loss, which is an indicator of the model performance.

2.4.2. BLACK-SCHOLES MODEL

Focusing on European call options, we generate 1,000,000 random samples for the input parameters, see Table 2.4. We calculate the corresponding European option prices $V(t,S)$ of Equation (2.2) with the solution in (2.4). As a result, each sample contains five variables $\{S_0/K, \tau, r, \sigma, V/K\}$. The training samples are fed into the ANN, where the input includes $\{S_0/K, \tau, r, \sigma\}$, and the output is the scaled option price V/K.

Table 2.4: Wide and narrow Black-Scholes parameter ranges

BS-ANN	Parameters	Wide Range	Narrow Range	Unit
Input	Stock price(S_0/K)	[0.4, 1.6]	[0.5, 1.5]	-
	Time to Maturity(τ)	[0.2, 1.1]	[0.3, 0.95]	year
	Risk free rate(r)	[0.02, 0.1]	[0.03, 0.08]	-
	Volatility(σ)	[0.01, 1.0]	[0.02, 0.9]	-
Output	Call price(V/K)	(0.0, 0.9)	(0.0, 0.73)	-

We distinguish, during the evaluation of the ANN, two different *test data sets*, i.e., a wide test set and a slightly more narrow test set. The reason is that we observe that often the ANN approximations in the areas very close to parameter domain boundaries may give rise to somewhat larger approximation errors, and the predicted values in the middle part are of higher accuracy. We wish to alleviate this boundary-related issue.

The wide test data set is based on the same parameter ranges as the training data set. As shown in Table 2.5, the root averaged mean-squared error (RMSE) is around 9×10^{-5}, which is an indication that the average pricing error is 0.009% of the strike price. Figure 2.7a shows the histogram of prediction errors, where it can be seen that the error approximately exhibits a normal distribution, and the maximum absolute error is around 0.06%.

Table 2.5: BS-ANN performance on the test data set.

BS-ANN	MSE	RMSE	MAE	MAPE
Training-wide	8.04×10^{-9}	8.97×10^{-5}	6.73×10^{-5}	3.75×10^{-4}
Testing-wide	8.21×10^{-9}	9.06×10^{-5}	6.79×10^{-5}	3.79×10^{-4}
Testing-narrow	7.00×10^{-9}	8.37×10^{-5}	6.49×10^{-5}	3.75×10^{-4}

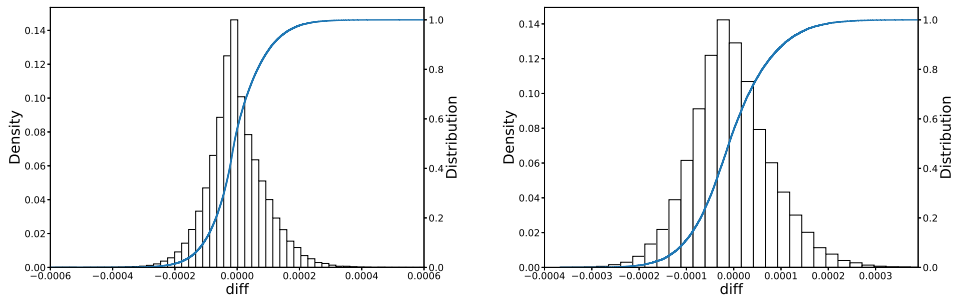
The narrow test set is based on a somewhat more narrow parameter range than the training data set. As Table 2.5 shows, when the range of parameters in the test set is smaller than the training data set, ANN's test performance slightly improves. Figure 2.7 shows that the largest deviation becomes smaller, being less than 0.04%. The goodness of fit R^2-criterion measures the distance between the actual values and the predicted ones. There is no significant difference in R^2 in both cases.

Overall, it seems a good practice to train the ANN on a (slightly too) wide data set, when the parameter range of interest is somewhat smaller.

2.4.3. IMPLIED VOLATILITY

The aim here is to learn the implicit relationship between implied volatilities and option prices, which is guided by Equation (2.5). The option Vega can become arbitrarily small, which may give rise to a steep gradient problem in the ANN context. It is well-known that an ANN may generate significant prediction errors in regions with large gradients. We therefore propose a gradient-squash approach to handle this issue.

First of all, each option price can be split into a so-called *intrinsic value* and a *time value*, and we subtract the intrinsic value, as follows,



(a) (b)

Figure 2.7: BS-ANN performance. (**a**) Error distribution on the wide test data set; (**b**) Error distribution on the narrow test data set.

$$\tilde{V} := V(t,S) - \max(S(t) - Ke^{-r\tau}, 0),$$

where $\tilde{V}$ is the option time value. Please note that this change only applies to ITM options, since the OTM intrinsic option value is equal to zero. The proposed approach to overcome approximation issues is to reduce the gradient's steepness by furthermore working under a log-transformation of the option value. The resulting input is then given by $\{\log(\tilde{V}/K), S_0/K, r, \tau\}$. The adapted gradient approach increases the prediction accuracy significantly, as we will see below.

MODEL PERFORMANCE

In this case, the data samples can be created in a forward stage, i.e., we will work with the Black-Scholes solution (instead of the root-finding method) to generate the training data set. Given σ, τ, K, r and S, the generator, i.e., the Black-Scholes formula, gives us the option price $V(t_0, S_0) = BS(S_0, K, \tau, r, \sigma)$. For the data collection $\{V, S_0, K, \tau, r, \sigma\}$, we then take the input σ as the implied volatility $\sigma^* \equiv \sigma$ and place it as the output of the ANN. Meanwhile, the other variables $\{V, S_0, K, \tau, r\}$ will become the input of the ANN, followed by the log-transformation $\log(\tilde{V}/K)$. In addition, we do not take into consideration the samples whose time values are extremely small, like those for which $\tilde{V} < 10^{-7}$.

Two implied volatility ANN (IV-ANN) solvers are trained based on the dataset of Table 2.6. After that, Table 2.7 compares the performance of the ANN with the scaled and original (unscaled) input, where it is clear that scaling improves the ANN performance significantly. Figure 2.8 shows the out-of-sample performance of the trained ANN on the scaled inputs. The error distribution also

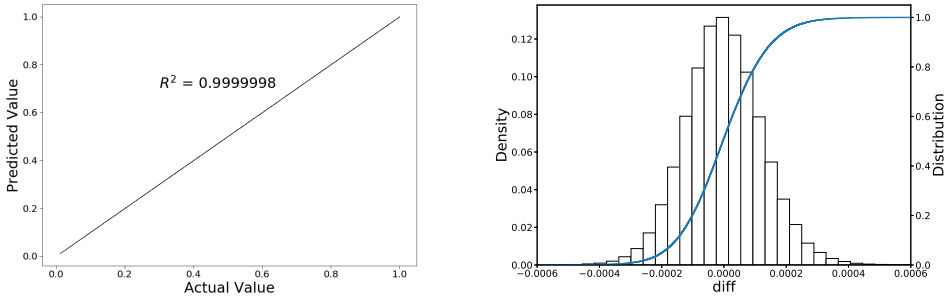
Table 2.6: Parameter range of data set.

IV-ANN	Parameters	Range	Unit
Input	Stock price (S_0/K)	[0.5, 1.4]	-
	Time to maturity (τ)	[0.05, 1.0]	year
	Risk-free rate (r)	[0.0, 0.1]	-
	Scaled time value ($\log(\tilde{V}/K)$)	[-16.12, -0.94]	-
Output	Volatility (σ)	(0.05, 1.0)	-

approximately follows a normal distribution, where the maximum deviation is around $6 \cdot 10^{-4}$, and most of implied volatilities equal their true values.

Table 2.7: Out-of-Sample ANN performance comparison.

IV-ANN	MSE	MAE	MAPE	R^2
Input: $m, \tau, r, V/K$ Output: σ^*	6.36×10^{-4}	1.24×10^{-2}	7.67×10^{-2}	0.97510
Input: $m, \tau, r, log(\tilde{V}/K)$ Output: σ^*	1.55×10^{-8}	9.73×10^{-5}	2.11×10^{-3}	0.9999998



(a) (b)

Figure 2.8: Out-of-Sample IV-ANN performance on the scaled input. (**a**) Comparison of implied volatilities; (**b**) The error distribution.

COMPARISON OF ROOT-FINDING METHODS

We compare the performance of five different implied-volatility-finding methods, including IV-ANN, Newton-Raphson, Brent, the secant and the bisection methods, in terms of run-time on a CPU and on a GPU. For this purpose, we compute 20,000 European call options for which all numerical methods can find the implied volatility. The σ-value range for bisection and for Brent's method is set to $[0, 1.1]$, and the initial guesses for the Newton-Raphson and secant method are $\sigma_0^* = 0.5$. The true volatility varies in the range $[0.01, 0.99]$, with the parameters, $r = 0$, $T = 0.5$, $K = 1.0$, $S_0 = 1.0$.

Table 2.8 shows that Brent's method is the fastest *among the robust* iterative methods (without requiring domain knowledge to select a suitable initial value). From a statistical point-of-view, the ANN solver gives rise to an acceptable averaged error MAE $\approx 10^{-4}$, and, importantly, its computation is faster by a factor 100 on a GPU and 10 on a CPU, as compared to the Newton-Raphson iteration. By the GPU architecture, the ANN processes the input 'in batch mode', calculating several implied volatilities simultaneously, which is the reason for the much higher speed. Besides, the acceleration on the CPU is also obvious, as only matrix multiplications or inner products are required.

Table 2.8: Performance comparison: CPU (Intel i5, 3.33GHz with cache size 4MB) and GPU(NVIDIA Tesla P100).

Method	**GPU (seconds)**	**CPU (seconds)**	**Robustness**
Newton-Raphson	19.68	23.06	No
Brent	52.08	60.67	Yes
Secant	88.73	103.76	No
Bi-section	337.94	390.91	Yes
IV-ANN	0.20	1.90	Yes

2.4.4. HESTON MODEL FOR OPTION PRICES

The Heston option prices are computed by means of the COS method in this section. The solution to the Heston model also can be obtained by other numerical techniques, like PDEs discretization or Monte Carlo methods. The COS method has been proved to guarantee a high accuracy with less computational expense.

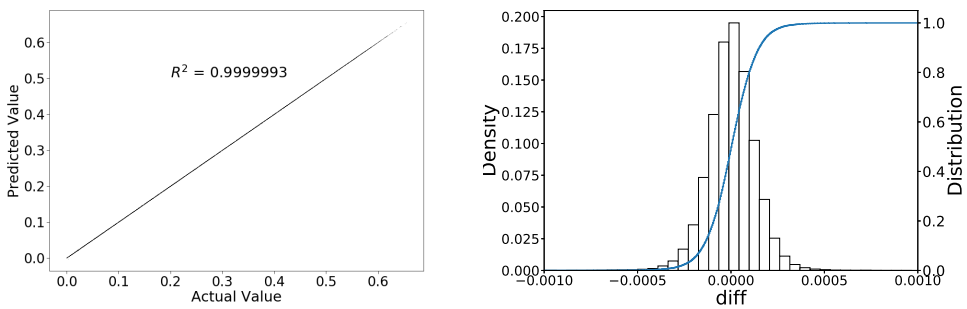
According to the given ranges of Heston parameters in Table 2.9, for the COS method, the integration interval is based on $L_{COS} = 50$, with the number of Fourier cosine terms in the expansion being $N_{COS} = 1500$. The prices of deep OTM European call options are calculated using the put-call parity, as the COS method call

Table 2.9: The Heston parameter ranges for traing the ANN

ANN	Parameters	Range	Method
Input	Moneyness, $m = S_0/K$	(0.6, 1.4)	LHS
	Time to maturity, τ	(0.1, 1.4)(year)	LHS
	Risk free rate, r	(0.0%, 10%)	LHS
	Correlation, ρ	$(-0.95, 0.0)$	LHS
	Reversion speed, κ	(0.0, 2.0)	LHS
	Long average variance, $\bar{\nu}$	(0.0, 0.5)	LHS
	Volatility of volatility, γ	(0.0, 0.5)	LHS
	Initial variance, ν_0	(0.05, 0.5)	LHS
Output	European call price, V	(0, 0.67)	COS

Table 2.10: The trained Heston-ANN performance

Heston-ANN	MSE	MAE	MAPE	R^2
Training	1.34×10^{-8}	8.92×10^{-5}	5.66×10^{-4}	0.9999994
Testing	1.65×10^{-8}	9.51×10^{-5}	6.27×10^{-4}	0.9999993



(a) (b)

Figure 2.9: Out-of-sample Heston-ANN performance. (**a**) COS vs. Heston-ANN prices; (**b**) The error distribution.

prices that are close to zero may be inaccurate due to truncation errors. In Table 2.9, we list the range of the six Heston input parameters ($r, \rho, \kappa, \bar{\nu}, \gamma, \nu_0$) as well as the two option contract-related parameters (τ, m), with a fixed strike price, K =

1. We generate around one million data points by means of the Latin hypercube sampling, using 10% as testing, 10% as validation and 80% as the training data set. After 3000 epochs with a decaying learning rate schedule, as shown in Table 2.10, the Heston-ANN solver has been well trained, avoiding over-fitting and approximating the prices accurately. Although the number of input parameters is doubled as compared to the Black-Scholes model, the Heston-ANN accuracy is also highly satisfactory and the error pattern is similar to that of the BS-ANN solver, see Figure 2.9.

HESTON MODEL AND IMPLIED VOLATILITY

We design two experiments to illustrate the ANN's ability of computing the implied volatility based on the Heston option prices. In the first experiment, the ground truth for the implied volatility is generated by means of two steps. Given the Heston input parameters, we first use the COS method to compute the option prices, after which we use Brent's method to compute the Black-Scholes implied volatility σ^*. The machine learning approach is also based on two steps, as shown in Figure 2.10. First of all, the Heston-ANN is used to compute the option prices, and, subsequently, we use IV-ANN to compute the corresponding implied volatilities. We compare these two approaches in Figure 2.10.

Please note that the ANN solver performs best in the middle of all parameter ranges, and tends to become worse at the domain boundaries. We therefore first choose the range of moneyness $m \in [0.7, 1.3]$ and the time to maturity $\tau \in [0.3, 1.1]$. Table 2.11 shows the overall performance of the ANN. As the IV-ANN takes the output of the Heston-ANN as the input, the accumulated error reduces the overall accuracy slightly. However, the root averaged mean error is still small, RMSE$\approx 7 \cdot 10^{-4}$. Then we reduce the range of the parameters, as listed in the third row of Table 2.11, and find that the prediction accuracy increases with the parameter ranges shrinking. Comparing the results in Figure 2.11 and Table 2.11, the goodness of fit as well as the error distribution improve with the slightly smaller parameter range, which is similar to our findings for the BS-ANN solver.

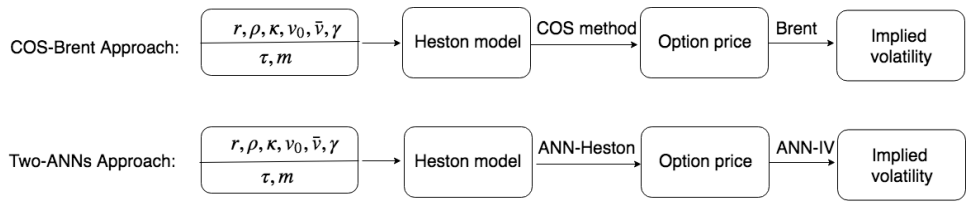
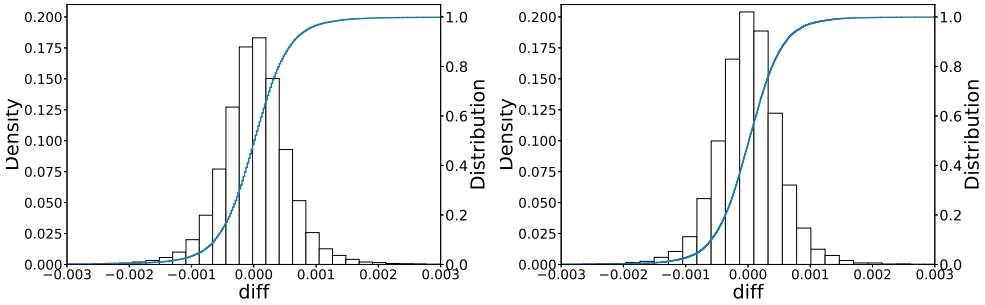


Figure 2.10: Two approaches of computing implied volatility for Heston model.

Table 2.11: Out-of-sample performance of the Heston-ANN plus the IV-ANN.

Heston-ANN & IV-ANN	RMSE	MAE	MAPE	R^2
Case 1: $\tau \in [0.3, 1.1]$, $m \in [0.7, 1.3]$	7.12×10^{-4}	4.19×10^{-4}	1.46×10^{-3}	0.999966
Case 2: $\tau \in [0.4, 1.0]$, $m \in [0.75, 1.25]$	5.53×10^{-4}	3.89×10^{-4}	1.14×10^{-3}	0.999980



(a) (b)

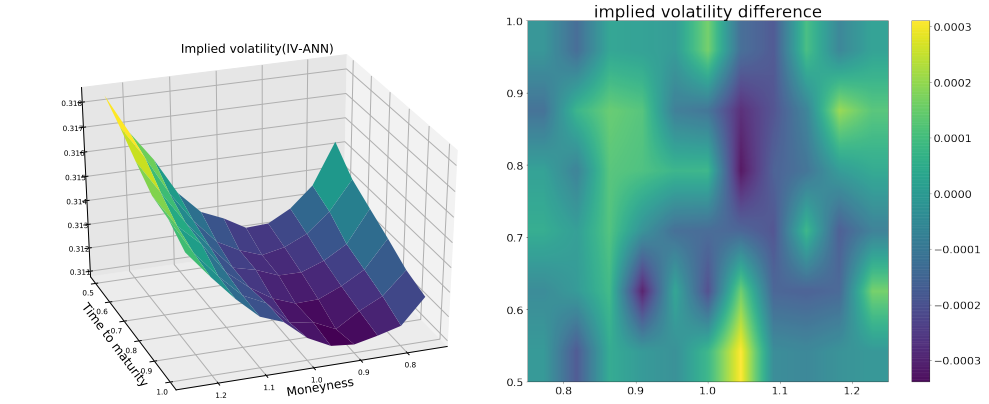
Figure 2.11: The error distribution of the implied volatility: The combined Heston-ANN and IV-ANN techniques for implied volatility. (**a**) Case 1: The error distribution; (**b**) Case 2: The error distribution.

Another experiment is to show that the IV-ANN can generate a complete implied volatility surface for the Heston model. With the following Heston parameters (an example to generate a smile surface), $\rho = -0.05$, $\kappa = 1.5$, $\gamma = 0.3$, $\bar{v} = 0.1$, $v_0 = 0.1$ and $r = 0.02$, we calculate the option prices by the COS method for the moneyness $m \in [0.7, 1.3]$ and time to maturity $\tau \in [0.5, 1.0]$. The implied volatility surface approximated by means of the IV-ANN is shown in Figure 2.12a, and Figure 2.12b shows that the maximum deviation between the ground-truth and the predicted values is no more than 4×10^{-4}.

Concluding, the ANN can approximate the Heston option prices as well as the implied volatilities accurately. The characteristic function of the financial model is not required during the test phase of the ANN.

2.5. CONCLUSION

In this chapter we have proposed an ANN approach to reduce the computing time of pricing financial options, especially for high-dimensional financial mod-



(a) (b)

Figure 2.12: (**a**) Heston implied volatility surface generated by IV-ANN. (**b**) Heston implied volatility difference between Brent's method and IV-ANN.

els. We test the ANN approach on three different solvers, including the closed-form solution for the Black-Scholes equation, the COS method for the Heston model and Brent's root-finding method for the implied volatilities. Our numerical results show that the ANN can compute option prices and implied volatilities efficiently and accurately in a robust way. This means, particularly for asset price processes leading to much more time-consuming computations, that we are able to provide a highly efficient approximation technique by means of the ANN. Although the off-line training will take longer then, the on-line prediction will be fast. Moreover, parallel computing allows the ANN solver to process derivative contracts "in batch mode" (i.e., dealing with many observed market option prices simultaneously during calibration), and this property boosts the computational speed by a factor of around 100 on GPUs over the original solver in the present case. We have shown that the boundaries of parameter values have an impact when applying the ANN solver. It is recommended to train the ANN on a data set with somewhat wider ranges than the values of interest. Regarding high-dimensional asset models, as long as the option values can be obtained by any numerical solver (Fourier technique, finite differences or Monte Carlo method), we may speed up the calculation by employing a trained ANN.

Although we focus on European call options in this work, it should be possible to extend the approach to pricing more complex options, like American, Bermuda or exotic options (see also Chapter 4). This chapter's work initially demonstrates the feasibility of learning a data-driven solver to speed up solving parametric financial models. The model accuracy can be further improved,

for example, by using deeper neural networks, more complex NN architectures. The solver's speed may also improve, for example, by designing a more shallow neural network, extracting insight from the complex network [50].

Furthermore, the option Greeks, representing the sensitivity of option prices with respect to the market or model parameters, are important in practice (i.e., for hedging purposes). As ANNs approximate the solution to the financial PDEs, the related derivatives can also be recovered from the trained ANN. There are several ways to calculate Greeks from the ANN solver. A straightforward way is to extract the gradient information directly from the ANN, since the approximation function in Equation (2.25) is known analytically. Alternatively, a trained ANN may be interpreted as an implicit function, where Auto-Differentiation [51] can help to calculate the derivatives accurately. In Chapter 4, we will combine these two neural networks, the Heston-ANN and the IV-ANN, into a single neural network, which makes it more efficient while computing the implied volatility surface during the calibration of the Heston model.

3

A NEURAL NETWORK-BASED FRAMEWORK FOR FINANCIAL MODEL CALIBRATION

A data-driven approach called CaNN (Calibration Neural Network) is proposed to calibrate financial asset price models using an Artificial Neural Network (ANN). Determining optimal values of the model parameters is formulated as training hidden neurons within a machine learning framework, based on available financial option prices. The framework consists of two parts: a forward pass in which we train the weights of the ANN off-line, valuing options under many different asset model parameter settings; and a backward pass, in which we evaluate the trained ANN-solver on-line, aiming to find the weights of the neurons in the input layer. The rapid on-line learning of implied volatility by ANNs, in combination with the use of an adapted parallel global optimization method, tackles the computation bottleneck and provides a fast and reliable technique for calibrating model parameters while avoiding, as much as possible, getting stuck in local minima. Numerical experiments confirm that this machine-learning framework can be employed to calibrate parameters of two-dimensional stochastic volatility models efficiently and accurately.

3.1. INTRODUCTION

Model calibration can be formulated as an inverse problem, where, based on observed output results, the input parameters need to be inferred. Previous work

This chapter is based on the article 'A neural network-based framework for financial model calibration', published in *Journal of Mathematics in Industry*, 2019, 9(9).

on solving inverse problems includes research on adjoint optimization methods [52, 53], Bayesian methods [54, 55], and sparsity regularization [56].

In a financial context, e.g., in the pricing and risk management of financial derivative contracts, asset model calibration means recovering the model parameters of the underlying stochastic differential equations (SDEs) from observed market data. In other words, in the case of stocks and financial options, the calibration aims to determine the stock model parameters such that heavily traded, liquid option prices can be recovered by the mathematical model. The calibrated asset models are subsequently used to either determine a suitable price for over-the-counter (OTC) exotic financial derivatives products, or for hedging and risk management purposes.

Calibrating financial models is a critical task within finance, and may need to be performed numerous times every day. Relevant issues in this context include accuracy, speed and robustness of the calibration. Real-time pricing and risk management require a fast and accurate calibration process. Repeatedly computing the values using mathematical models and at the same time fitting the parameters may be a computationally heavy burden, especially when dealing with multi-dimensional asset price models.

The calibration problem is not necessarily a convex optimization problem, and it often gives rise to multiple local minima. For example, the authors in [57] vary two parameters of the Heston model (keeping the other parameters unchanged), and find that the objective function exhibits multiple local minima. Also in [58] it is stated that multiple local minimal are common for calibration in the foreign exchange or commodities markets. A local optimization technique is generally relatively cheap and fast, but a key factor is to choose an accurate initial guess. Otherwise, it may fail to converge and get stuck in a local minimum. To address robustness, global optimizers are becoming popular to calibrate financial models, like Differential Evolution (DE) [59], Particle Swarm optimization and Simulated Annealing, as their convergence does not depend on specific initial values. Parallel computing may help to reduce the computing time of global calibration problems.

A generic, *robust* calibration framework may be based on a global optimization technique in combination with a highly efficient pricing method, in a parallel computing environment. To meet these requirements, we will employ the machine learning technology and develop an artificial neural network (ANN) method for a generic calibration framework. The basic idea of our approach is to connect model calibration with machine learning from an optimization point of view. Estimating the model parameters is converted into finding the values of the ANN's hidden units, so that the network output matches the observed option prices or volatility.

The proposed ANN-based framework comprises three phases, i.e., training, prediction and calibration. During the training phase, the hidden layer parameters of the ANNs are optimized by means of supervised learning. This training phase builds a mapping between the model parameters and the output of interest. During the prediction phase, the hidden layers are kept unchanged (frozen) to compute the output quantities (e.g., option prices) given various input parameters of the asset price model. The prediction phase can also be used to evaluate the model performance (namely testing). Together these steps are called the *forward pass*. Finally, during the calibration phase, given the observed output data (e.g., market option prices), the original input layer becomes a learnable layer again, whereas all previously learned hidden layers are kept fixed. This latter stage, which is also called the *backward pass*, inverts the already trained neural network conditional on certain known input. The overall calibration framework we name *CaNN (Calibration Neural Network)* here. The CaNN establishes a connection between machine learning and model calibration.

There are several interesting aspects to the proposed approach. First of all, the machine learning approach may significantly accelerate classical option pricing techniques, particularly when involved asset price models are of interest. Recently there has been increasing interest in applying machine-learning techniques for fast pricing and calibration, see [60–66]. For example, the paper [62] used Gaussian process regression methods for derivative pricing. Other work, including this chapter, employs artificial neural networks to learn the solution of the financial SDE system [60, 63, 66], that do not suffer much from the curse of dimensionality.

Secondly, the CaNN is a generic ANN-based framework, and views the three phases, training/prediction/calibration, as a whole, the difference between them being just to change the learnable units. Furthermore, the proposed ANN approach can handle a flexible number of input market data. In other papers, like [65], [64], the number of input observed samples had to be fixed in order to fit the employed Convolutional Neural Networks.

Moreover, there is inherent parallelism in our ANN approach, so we will also take advantage of modern processing units (like GPUs). Unlike the paper [63] which presented a neural network-based method to calibrate rough volatility models, our CaNN incorporates a parallel global search method to achieve higher calibration speed, as calibrating financial models often gives rise to non-convex optimization problems, for which local optimization algorithms may have convergence issues. As a global searcher, DE has been used to calibrate financial models [57, 67] and to train neural networks [68], making it also suitable in the ANN-based calibration framework.

The contributions of the work in this chapter are three-fold. First, we design

a generic ANN-based framework for calibration. Apart from data generators, all the components and tasks are implemented on a unified computing platform. Second, a parallel global searcher is adopted based on a population-based optimization algorithm (here DE), an approach that fits well within the ANN-based calibration framework. Both the forward and backward passes run in parallel, tackling the computational bottleneck of global optimization and making the calibration time reasonable, even in the case of employing a large neural network. Third, the key components are robust and stable: using a robust data generator and the global optimization technique help the ANN-based calibration method not getting stuck in local minima.

The rest of this chapter is organized as follows. In Section 3.2, the Heston and Bates stochastic volatility models and their calibration requirements are briefly introduced. These models will be used in the numerical experiments. In Section 3.3, artificial neural networks are introduced as function approximators, in the context of parametric financial models. Furthermore, a generic machine learning framework for model calibration to find the global solution is presented. In Section 3.4, numerical experiments are presented to demonstrate the performance of the proposed calibration framework.

3.2. FINANCIAL MODEL CALIBRATION

We start by explaining the stochastic models for the asset prices, the corresponding partial differential equations for the option valuation and the standard ways of calibrating these models. The open parameters in these models, that need to be calibrated with the help of an objective function, are also discussed.

3.2.1. ASSET PRICING MODELS

In the following subsections we present the financial asset pricing models that will be used in this chapter, the Heston and Bates stochastic volatility models. European option contracts are used as examples to derive the pricing models, however, other types of financial derivatives can be taken into consideration in a similar way.

THE HESTON MODEL

As discussed in Chapter 2, one of the most popular stochastic volatility asset pricing models is the Heston stochastic volatility model [31], see Section 2.2.3 for more details.

THE BATES MODEL

Next to the Heston model, we will also consider its generalization, the Bates model [69], by adding jumps to the Heston stock price process. The model is

described by the following system of SDEs:

$$\frac{dS(t)}{S(t)} = \left(r - \lambda_J \mathbb{E}[e^J - 1]\right) dt + \sqrt{v(t)} dW_s(t) + \left(e^J - 1\right) dX_{\mathscr{P}}(t), \tag{3.1a}$$

$$dv(t) = \kappa(\bar{v} - v(t))dt + \gamma\sqrt{v(t)} dW_v(t), \ v_{t_0} = v_0, \tag{3.1b}$$

$$dW_s(t) dW_v(t) = \rho dt, \tag{3.1c}$$

with $X_{\mathscr{P}}(t)$ a Poisson process with intensity λ_J, and J being normally distributed jump sizes with expectation μ_J and variance ν_J^2, i.e. $J \sim \mathcal{N}(\mu_J, \nu_J^2)$. The Poisson process $X_{\mathscr{P}}(t)$ is assumed to be independent of the Brownian motions and of the jump sizes. Clearly, we have three more parameters, λ_J, μ_J and ν_J^2, to calibrate in this case, in comparison with calibrating the Heston model. The corresponding option pricing equation is a so-called Partial Integro-Differential Equation (PIDE),

$$\begin{aligned}\frac{\partial V}{\partial t} + \frac{1}{2} v S^2 \frac{\partial^2 V}{\partial S^2} + \rho\gamma v S \frac{\partial^2 V}{\partial S \partial v} + \frac{1}{2}\gamma^2 v \frac{\partial^2 V}{\partial v^2} + (r - \frac{1}{2} v(t) - \lambda_J(e^{\mu_J} - 1)) \frac{\partial V}{\partial S} \\ + \kappa(\bar{v} - v)\frac{\partial V}{\partial v} - (r + \lambda_J) V + \lambda_J \int_0^\infty V(x) P_J(x)\, dx = 0,\end{aligned} \tag{3.2}$$

with the given terminal condition $V(T, S, v; T, K)$, where $P_J(x)$ is the log-normal probability density function of the jump magnitudes.

Both the Heston and Bates models do not give rise to analytic option value solutions and the governing P(I)DEs thus have to be solved numerically. There are several possibilities for this, like by means of finite difference PDE techniques, Monte Carlo, or numerical integration methods. We will employ a Fourier-type method, the COS method from [38], to obtain highly accurate option values, for the details we refer to Section 2.2.5 in Chapter 2. A prerequisite to using Fourier methods is the availability of the asset price's characteristic function, which is available for both Heston and Bates. From the resulting option values, the corresponding Black-Scholes' implied volatilities will be determined by means of a robust root-finding iteration known as Brent's method [37].

3.2.2. THE CALIBRATION PROCEDURE

Calibration refers to estimating the model parameters (i.e., the constant coefficients in the PDEs) given the samples of the market data. The market value of either option prices or implied volatilities, with moneyness $m := S_0/K$ and time to maturity $\tau := T - t$, is denoted here by $Q^*(\tau, m)$, and the corresponding model-based value is $Q(\tau, m; \Theta)$, with the parameter vector $\Theta \in \mathbb{R}^n$, where n denotes the

number of parameters to calibrate. For the Heston model, $\Theta := [\rho, \kappa, \gamma, \bar{v}, v_0]$, while for the Bates model we have, $\Theta := [\rho, \kappa, \gamma, \bar{v}, v_0, \lambda_J, \mu_J, \sigma_J]$.

The difference between the observed values and the ones given by the model is indicated by an error measure,

$$e_i := ||Q(\tau_i, m_i; \Theta) - Q^*(\tau_i, m_i)||, \; i = 1, ..., N \tag{3.3}$$

where $||\cdot||$ measures the distance, and N is the number of available calibration instruments. The total difference is represented by the following target function,

$$J(\Theta) := \sum_{i=1}^{N} \omega_i e_i + \bar{\lambda}||\Theta||, \tag{3.4}$$

where ω_i are the corresponding weights and $\bar{\lambda}$ is a regularization parameter. When $\omega_i = \frac{1}{N}$ and $\bar{\lambda} = 0$ with squared errors in Equation (3.4), we obtain a well-known error measure, the MSE (Mean Squared Error). When people wish to guarantee perfect calibration for ATM options (the options are most liquid in the market), the corresponding weight value ω_i is increased. Usually calibrating financial models reduces to the following minimization problem,

$$\arg\min_{\Theta \in \mathbb{R}^n} J(\Theta), \tag{3.5}$$

which gives us a set of parameter values making the difference between the market and the model quantities as small as possible.

The above formula is over-determined in the sense that $N > n$, i.e., the number of data samples is larger than the number of to-calibrate parameters. Equation (3.5) is usually solved iteratively to minimize the residual. Initially, a set of parameter values is assigned and the corresponding model values are determined; these values are compared with market data, and the corresponding error is computed, after which a search direction is determined to find a next parameter set. The above steps are repeated until a stopping criterion is met. While evaluating Equation (3.4), an array of options with different strikes and maturities needs to be valued thousands of times and therefore this valuation should be performed highly efficiently. *Here, we will employ ANNs that can deal with a complete array of option prices in parallel.*

3.2.3. CHOICES WITHIN CALIBRATION

Typically, the objective function is highly nonlinear and even non-convex, for example, the authors in [70] discuss the impact of the objective function and the calibration method for the Heston model. This issue becomes worse when being faced with a high-dimensional optimization problem. A way to address this

problem is to smooth the objective function and employ traditional local optimization methods. Another difficulty when calibrating the model is that the set Θ includes multiple parameters that need to be determined, and that these model parameters are not completely "independent", for example, the effect of different parameters on the shape of the implied volatility smile may be quite similar. For this reason, one may encounter several "local minima" when searching for optimal parameter values. In most cases, a global optimization algorithm should be preferred during calibration.

Regarding the target objective function, there are two popular choices in the financial context, namely either based on observed option prices or based on computed implied volatilities. Option prices can be collected directly from the market, and implied volatility should be computed based on the collected option prices. The most common choices without regularization terms thus include,

$$\min_{\Theta} \sum_i \sum_j \omega_{i,j} \left(V_c^*(T_j - t_0, S_0/K_i) - V_c(T_j - t_0, S_0/K_i; \Theta) \right)^2, \tag{3.6}$$

and

$$\min_{\Theta} \sum_i \sum_j \omega_{i,j} \left(\sigma_{imp}^*(T_j - t_0, S_0/K_i) - \sigma_{imp}(T_j - t_0, S_0/K_i; \Theta) \right)^2, \tag{3.7}$$

where $V_c^*(T_j - t_0, S_0/K_i)$ is the call option price for strike K_i and maturity T_j with instantaneous stock price S_0 at time t_0 as observed in the market; $V_c(T_j - t_0, S_0/K_i; \Theta)$ is the call option value computed from the model using model parameters Θ; similarly $\sigma_{imp}^*(\cdot)$, $\sigma_{imp}(\cdot)$ are the implied volatilities from the market and from the Heston/Bates model, respectively; $\omega_{i,j}$ is some weighting function. The notation i and j is to distinguish the two factors impacting the target quantity. A third approach is to calibrate the model to both prices and implied volatility. For option prices, weighting the target quantity by Vega (the derivative of the option price with respect to the volatility) is a technique to remedy model risk. When taking implied volatility into account, a numerical root-finding method is often employed to invert the Black-Scholes formula in addition to computing option prices. That is to say, two numerical methods are required, one for pricing options, the other one for calculating the Black-Scholes implied volatility. Nevertheless, calibrating to an implied volatility surface can help to specify prices of all vanilla options regardless of their types (e.g., call or put), given the current term structure of interest rates. This is one of the reasons why the practitioners prefer implied volatility during calibration. Besides, we will mathematically discuss the difference between calibrating to option prices and implied volatilities in Section 3.4.3. Moreover, it is well known that OTM instruments are liquid or heavily traded in the market. Calibrating the financial models to OTM instruments is common practice in reality.

The calibration performance (e.g., speed and accuracy) is also influenced by the employed method while solving the financial models. An analytic solution is not necessarily available for the model to be calibrated, and different numerical methods have therefore been developed to solve the corresponding option pricing models. Alternatively, based on some existing solvers, ANNs can be used as a numerical method to learn the solution [60].

3.3. AN ANN-BASED APPROACH TO CALIBRATION

This section presents the framework to calibrate a financial model by means of machine learning. Training the ANNs and calibrating financial models both boil down to optimization problems, which motivates the present machine learning-based approach to model calibration.

3.3.1. ARTIFICIAL NEURAL NETWORKS

ANNs are powerful universal function approximators and can be used without assuming any pre-specified relation between the input and the output. Training the ANNs is learning the optimal weights and biases in Equation (2.18) to make the loss function as small as possible. As discussed in Chapter 2, the process of training neural networks can be formulated as an optimization problem,

$$\arg\min_{\hat{\Theta}} L(\hat{\Theta}|(\acute{\mathbf{X}},\acute{\mathbf{Y}})), \tag{3.8}$$

given the input-output pairs $(\acute{\mathbf{X}},\acute{\mathbf{Y}})$ and a user-defined loss function $L(\hat{\Theta})$.

In order to perform the optimization in Equation (3.8), the composite function from Equation (2.18) is differentiated using the chain rule. The first- and second-order partial derivatives of the loss function with respect to any weight w (or bias b) are easily computable; for more details we refer to [71]. This differentiation enables us to not only train ANNs with gradient-based methods, but also the sensitivity of the approximated functions using the trained ANN can be investigated. For this latter task, the Hessian matrix will be derived in Section 3.4 to study the sensitivity of the objective function with respect to the calibrated parameters.

3.3.2. THE FORWARD PASS: LEARNING THE SOLUTION WITH ANNS

The first part of the CaNN, the forward pass, employs an ANN, in the form of an MLP, to learn the solution generated by different numerical methods and subsequently maps the input to the output of interest (i.e., neglecting the intermediate variables). For example, in order to approximate the Black-Scholes implied

volatilities based on the Heston input parameters, two numerical methods are required, i.e., the COS method to calculate the Heston option prices and Brent's root-finding algorithm to determine the corresponding implied volatility, as presented in Figure 3.1. Using two separate ANNs to map the Heston parameters to implied volatility has been studied in Chapter 2. In this chapter, we merge these two ANNs, see Figure 3.1. In other words, the Heston-IV-ANN is used as the forward pass to learn the mapping between the model parameters and the implied volatility. Note that a similar ANN model will be employed for the Bates model, however then based on the Bates model parameters.

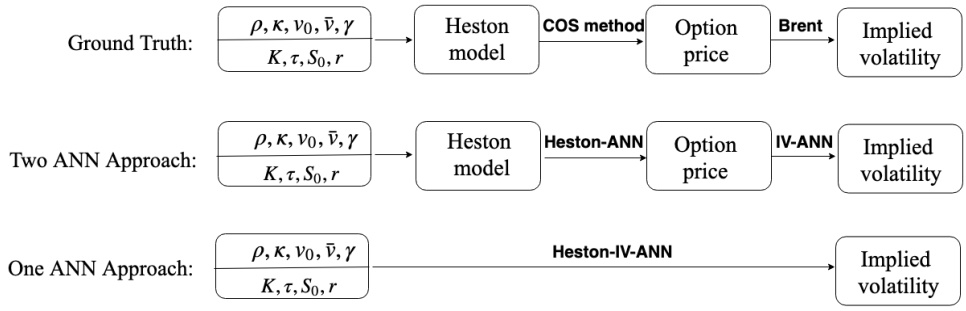


Figure 3.1: The Heston-implied volatility ANN.

The forward pass consists of training and prediction, and in order to do so the network architecture and optimization method have to be defined. Generally, an increasing number of neurons, or a deeper structure, may lead to better approximations, but may also result in a computationally heavy optimization and evaluation of the network. In [8] it is proved that a deep NN can approximate a function for which a shallow NN may need a very large number of neurons to reach the same accuracy. Different residual neural networks have been trained and tested as a validation of our work. They may improve the predictive power while using a similar number of weights as in an MLP, but they typically take significantly more computing time during the training and testing phases. Very deep network structures may reduce the parallel efficiency, because the operations within a layer have to wait for the output of previous layers. With the limitation of computing resources available, a trade-off between ANN's computation speed and approximation capacity may be considered.

Many techniques have been put forward to train ANNs, especially for deep networks. Most of the neural network training relies on gradient-based methods. A proper *random initialization* ensures the network to start with suitable initial weight values. *Batch normalization* scales the output of a layer by subtracting the batch mean and dividing it by the batch standard deviation. This can often

speed up the training process. A *dropout operation* randomly selects a proportion of the neurons and deactivates them, which forces the network to learn more generalized features and prevents over-fitting. The dropout rate p refers to the proportion of deactivated neurons in a layer. In the testing phase, in order to take into account the missing activation during training, each activation in the entire network is reduced by a factor p. As a consequence, the ANNs prediction slows down, which has been verified during our numerical experiments on GPUs. We found that our ANNs model did not encounter over-fitting, not even when using a zero dropout rate, as long as sufficient training data were provided. In our neural network we employ the Stochastic Gradient Descent method, as further described in Section 3.3.4.

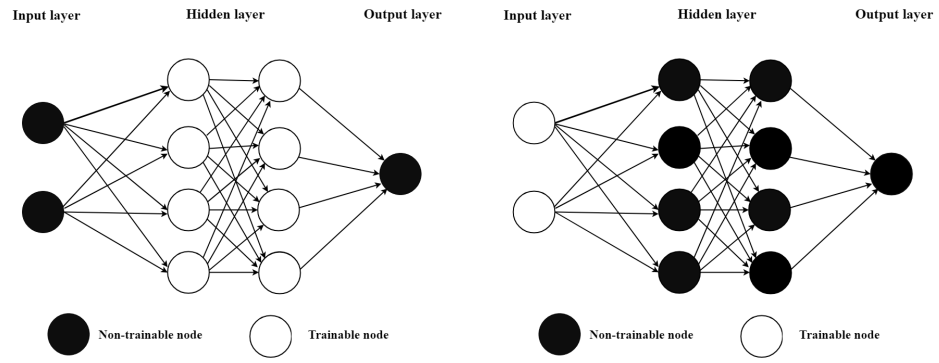
3.3.3. THE BACKWARD PASS: CALIBRATION USING ANNS

This section discusses the connection between training the ANN and calibrating the financial model. First of all, both Equations (3.5) and (3.8) aim at estimating a set of parameters to minimize a particular objective function. For the calibration problem, these are the parameters of the financial model and the objective function is the error measure between the market quantity and the model-based quantity. For the neural networks, the parameters correspond to the learnable weights and biases in the artificial neurons, and the objective function is the user-defined loss. This connection forms an inspiration for the machine learning-based approach to calibrate financial models.

As mentioned before, the ANN approach comprises three phases, training, prediction and calibration. During training, given the input-output pairs and a loss function as in Equation (3.8), the hidden layers are optimized to determine the appropriate values of the weights and biases, as shown in Figure 3.2a, which results in a trained ANN approximating the option solutions of the financial model (the forward pass, as explained in the previous section).

During the prediction phase, the hidden layers of the trained ANN are fixed (frozen), and new input parameters enter the ANN to yield the output quantities of interest. This phase is used to evaluate the performance of the trained ANN (the so-called model testing) or to accelerate option pricing by replacing the original solver.

During the calibration phase (or *the backward pass*), the original input layer of the ANN is transformed into a learnable layer, while all hidden layers remain unchanged. These layers are the ANN layers obtained from the forward pass with the already trained weights, as shown in Figure 3.2b. By providing the output data, here consisting of market-observed option prices and implied volatilities, and changing to an objective function for model calibration, see Equation (3.5), the ANN can be used to find the input values that match the given output. The



(a) Training phase (b) Calibration phase

Figure 3.2: The different phases of the ANNs.

task is thus to solve the inverse problem by learning a certain set of input values, here the model parameters Θ, either for the Heston or Bates model. The option's strike price K, as an example, belongs to the input layer, but is not estimated in this phase. Note that the training phase in the forward pass is time-consuming but done off-line and only once. The calibration phase is computationally cheap, and is performed on-line. The calibration phase thus results in model parameters that best match the observed market data, provided the model has been trained sufficiently.

The gradients of the objective function, with respect to the input parameters, can be derived based on Formula (2.18). This is useful when employing gradient-based optimization algorithms to conduct model calibration with the trained ANNs. Compared to the classical calibration methods, in the ANN-based approach it is also possible to incorporate the gradient information from the trained ANNs to compute the search direction (without external numerical techniques). As mentioned, we focus on a general calibration framework in which we integrate both gradient-based and gradient-free algorithms. Importantly, within the proposed calibration framework we may insert any number of market quotes, without requiring a fixed structure or a grid of input parameters.

3.3.4. NUMERICAL OPTIMIZATION

The optimization method plays a key role in training ANNs and calibrating financial models, but there are different requirements on the solutions for different phases. When training the neural network to learn the mapping between input and output values, we aim for a good performance on a test data set while optimizing the model on a training data set (this concept is called generalization).

Calibration is regarded as an optimization problem with only a training data set, where the objective is to fit the market-observed prices as well as possible. In this work, the Stochastic Gradient Descent (SGD) is used when training the ANN, and Differential Evolution is preferred in the phase of calibration to address the problem of multiple local minima. [1]

STOCHASTIC GRADIENT DESCENT

A popular optimizer to train ANNs to learn the mapping function is SGD, as discussed in Chapter 2. Artificial neural networks contain thousands of weights, which give rise to a high-dimensional, non-convex optimization problem. The local minima appear not to be problematic for this involved black-box system, as long as the cost function reaches a sufficiently low value. The loss function of training the ANN solver is based on MSE.

DIFFERENTIAL EVOLUTION

Differential Evolution [59] is a population-based, derivative-free optimization algorithm, which does not require any specific initialization. With DE, a global optimum can be found, even when the objective function is non-convex. The general form of the DE algorithm usually comprises the following four steps:

1. Initialization: Generate the population with N_p individuals and locate each member with random positions in the search space,

$$(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, ..., \boldsymbol{\theta}_{N_p})$$

2. Mutation: Once initialized, a randomly sampled difference is added to each individual, named differential mutation,

$$\boldsymbol{\theta}'_i = \boldsymbol{\theta}_a + \tilde{F} \cdot (\boldsymbol{\theta}_b - \boldsymbol{\theta}_c) \tag{3.9}$$

where i represents the i-th candidate, and the indices a, b, c are randomly selected from the population with $a \neq i$. The resulting $\boldsymbol{\theta}'$ is called a mutant. The differential weight $\tilde{F} \in [0, \infty)$ determines the step size of the evolution. Generally, large F values increase the search radius, but may cause DE to converge slowly. There are several mutation strategies, for example, when $\boldsymbol{\theta}_a$ is always the best candidate of the previous population, the mutation strategy is called *best1bin*, which will be used in the following numerical experiments; when $\boldsymbol{\theta}_a$ is randomly chosen, it is called *rand1bin*. After this step, an intermediary (or donor) population, consisting of N_p mutant candidates, is generated.

[1] We have tested SGD during calibration. SGD is faster but may fail in some cases without good initial guess.

3. Crossover: During the crossover stage, mutated candidates that may enter the next evaluation stage are determined. For each $i \in \{1,\dots,N_p\}$, a uniformly distributed random number $\hat{p}_i \sim U(0,1)$ is selected. Some samples are filtered out by setting a user-defined crossover possibility $Cr \in [0,1]$,

$$\boldsymbol{\theta}_i'' = \begin{cases} \boldsymbol{\theta}_i', \text{ if } \hat{p}_i \le Cr, \\ \boldsymbol{\theta}_i, \text{ otherwise.} \end{cases} \tag{3.10}$$

   If the probability is greater than Cr, the donor candidate will be discarded. Increasing Cr allows more mutants to enter the next generation, but at the expense of population stability. Here, a trial population $(\boldsymbol{\theta}_1'', \boldsymbol{\theta}_2'', ..., \boldsymbol{\theta}_{N_p}'')$ has been defined.

4. Selection: Comparing each new trial candidate with the corresponding target individual on the objective function,

$$\boldsymbol{\theta}_i \leftarrow \begin{cases} \boldsymbol{\theta}_i'', \text{ if } g(\boldsymbol{\theta}_i'') \le g(\boldsymbol{\theta}_i), \\ \boldsymbol{\theta}_i, \text{ otherwise,} \end{cases} \tag{3.11}$$

   where $g(\cdot)$ is the user-defined objective function, for example, Formula (3.6) or (3.7) for model calibration. If the trail individual has improved performance, the selected individual is replaced. Otherwise, the offspring individual inherits the parameters from its parent. This gives birth to a next generation population.

The Steps (2)-(4) are repeated until the algorithm converges or until a pre-defined criterion is satisfied. Adjusting the control parameters may impact the performance of DE. For example, a large population size and mutation rate can increase the probability of finding the global minimum. An additional parameter, convergence tolerance, is used to measure the diversity within a population, and determines when to stop DE. The control parameters can also change over time, which is out of our scope here.

ACCELERATION OF CALIBRATION

In this section we develop DE into a parallel version which is beneficial within the ANNs. Generally, matrix multiplications and element-wise operations in a neural network can be implemented in parallel to reduce the computing time, especially when a large number of arguments is involved. As a result, several components of the calibration procedure can be accelerated. First, for the ANN solver in the forward pass, all observed market samples can be evaluated at once. Second, in the selection stage of the DE, an entire population can be treated simultaneously. Third, the ANN solver itself is able to run parallelly on any machine learning platform.

Table 3.1: The setting of DE

Parameter	Option
Population size	50
Strategy	best1bin
Mutation	(0.5, 1.0)
Crossover recombination	0.7
Convergence tolerance	0.01

An example of the parameter settings for DE is shown in Table 3.1 and Figure 3.3, where the population of one generation comprises 50 vector candidates for the calibrated parameters (e.g., a vector candidate contains five parameters to calibrate in the Heston model), and each candidate generates a number of market samples (here 35, i.e., 7 strike prices K and 5 time points). So, there are 50×35=1650 input samples for the Heston model each generation. Traditionally, all these input samples (here 1650) are computed individually, except for those with the same maturity time T. The first speed-up is achieved because 35 sample output quantities from each parameter candidate can be computed by the ANN solver at the same time, even if these samples have different maturity times and strike prices. The second speed-up is based on the parallel DE combined with the ANN, where all parameter candidates in one generation enter the ANN solver at once, that is, all 1650 input samples in one generation can be included in the ANN solver simultaneously, giving 1650 output values (e.g., implied volatilities). Note that the batch size of the ANN solver should be adapted to the limitations of the specific processor, here 2048 in our used processor. We find that with the population size being around 50, the parallel CaNN is at least 10 times faster than the conventional CaNN, on either a CPU or a GPU. It is believed that a larger population size should lead to a higher parallel computing performance, especially on a GPU.

Remark. *There are basically two error sources in this framework. One is a consistency error which comes from the employed numerical methods to solve the financial model, and it is found while generating the training data set. The other one is an optimization error during training and calibration. These errors will influence the performance of the forward pass.*

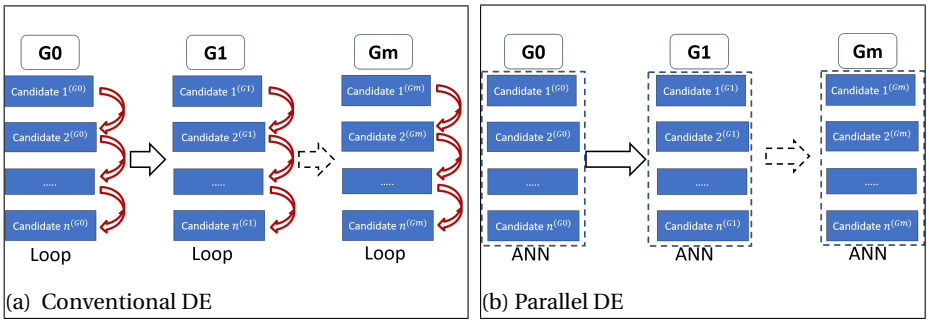


Figure 3.3: Illustration of the parallel CaNN version.

3.4. NUMERICAL RESULTS

In this section we show the performance of the proposed CaNN. We begin with calibrating the Heston model, a special case of the Bates model. Some insights into the effect of the Heston parameters on the implied volatility are discussed to give some intuition on the relation, since no explicit mapping between them exists. Then, the forward pass is presented where an ANN is trained to build a mapping between the model parameters and implied volatilities. It is also demonstrated that the trained forward pass can be used as a tool for performing the sensitivity analysis of the model parameters. After that, we implement the backward pass of the Heston-CaNN to calibrate the model and evaluate the CaNN performance. We end this section by considering the calibration of the Bates model, a model that consists of more parameters than the Heston model, using the Bates-CaNN.

3.4.1. PARAMETER SENSITIVITIES FOR HESTON MODEL

This section discusses the sensitivity of the implied volatility to the Heston parameters. This sensitivity analysis can be used to estimate a set of initial parameters, as is used in traditional calibration methods. In our calibration method this will not be required, however, we can gain some insights in the case of no explicit formulas.

The typically observed implied volatility shapes in the market, e.g., the implied volatility smile or skew, can be reproduced by varying the above parameters $\{\kappa, \rho, \gamma, v_0, \bar{v}\}$. We will give some intuition about the parameter values and their impact on the implied volatility shape.

From a PDE viewpoint, the calibration problem consists of finding appropriate values of PDE coefficients $\{\kappa, \rho, \gamma, v_0, \bar{v}\}$ to make the Heston model reproduce the observed option/implied volatility data. The authors in [72] reduce the calibration time by giving smart initial values for asset models, whereas in [73] an

approximation formula for the Heston dynamics was employed to determine a satisfactory initial set of parameters, followed by a local optimization to reach the final parameters. The paper [74] derived a Heston model characteristic function to analytically obtain gradient information of the option prices during the search for an optimal solution. In Section 3.4.3 we will use the ANN to extract gradient information of the implied volatility with respect to the Heston parameters.

EFFECT OF INDIVIDUAL PARAMETERS

To analyze the parameter effects numerically, we use the following set of reference parameters,

$$T = 2, S_0 = 100, \kappa = 0.1, \gamma = 0.1, \bar{\nu} = 0.1, \rho = -0.75, \nu_0 = 0.05, r = 0.05.$$

A numerical study is performed by varying individual parameters while keeping the others fixed. For each parameter set, Heston stochastic volatility option prices are computed (by means of the numerical solution of the Heston PDE) and the Black-Scholes implied volatilities are subsequently determined.

Two important parameters that are varied are the correlation parameter ρ and the volatility-of-variance parameter γ. Figure 3.4 (left side) shows that, when $\rho = 0\%$, an increasing value of γ gives a more pronounced implied volatility *smile*. A higher volatility-of-variance parameter thus increases the implied volatility *curvature*. We also see, in Figure 3.4 (right side), that when the correlation between stock and variance process gets increasingly negative, the slope of the *skew* in the implied volatility curve increases. Furthermore, it is found that parameter κ

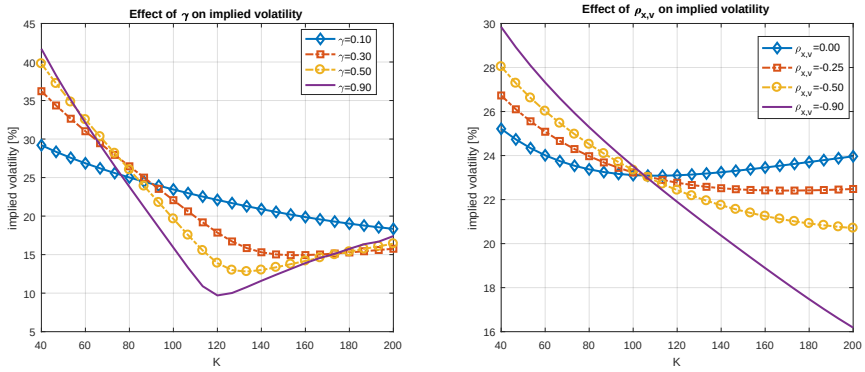


Figure 3.4: Impact of variation of the Heston parameter γ (left side), and correlation parameter ρ (right side), on the implied volatility which varies as a function of strike price K.

has a limited effect on the implied volatility smile or skew, up to $1\% - 2\%$ only. It determines the speed at which the volatility converges to the long-term volatility $\bar{\nu}$.

The calibration procedure can be accelerated by a reduction of the set of parameters to be optimized. By comparing the impact of the speed of mean reversion parameter κ and the curvature parameter γ, it is observed that these two parameters have a similar effect on the shape of the implied volatility. It is therefore common (industrial) practice to prescribe (or fix) one of them. Practitioners often fix κ and optimize parameter γ, for example $\kappa = 0.5$. By this, the optimization reduces to four parameters.

Another parameter which may be determined in advance, using heuristics, is the initial value of the variance process v_0. For maturity time T "close to today" (i.e., $T \to 0$), one expects the stock price to behave *like in the Black-Scholes case*. The impact of a stochastic variance process should reduce to zero, in the limit $T \to 0$. For options with short maturities, the process may therefore be approximated by a process of the following form:

$$dS(t) = rS(t)dt + \sqrt{v_0}S(t)dW_s(t). \tag{3.12}$$

This suggests that for initial variance v_0 one may use the square of the ATM implied volatility of an option with the shortest maturity, $v_0 \approx \sigma^2_{imp}$, for $T \to 0$, as an accurate approximation for the initial guess for the parameter. One may also use the connection of the Heston dynamics to the Black-Scholes dynamics with a time-dependent volatility function. In the Heston model we may, for example, *project* the variance process onto its expectation, i.e.,

$$dS(t) = rS(t)dt + \mathbb{E}\left[\sqrt{v(t)}\right] S(t)dW_s(t).$$

By this projection the parameters of the variance process $v(t)$ may be calibrated similarly to the case of the time-dependent Black-Scholes model. The Heston parameters are then determined, such that

$$\sigma^{ATM}(T_i) = \sqrt{\int_0^{T_i} \left(\mathbb{E}\left[\sqrt{v(t)}\right]\right)^2 dt},$$

where $\sigma^{ATM}(T_i)$ is the ATM implied volatility for maturity T_i.

Another classical calibration technique for the Heston parameters is to use VIX index market quotes [2]. With different market quotes for different strike prices K_i and for different maturities T_j, we may determine the optimal parameters by solving the following equalities, for all pairs (i, j),

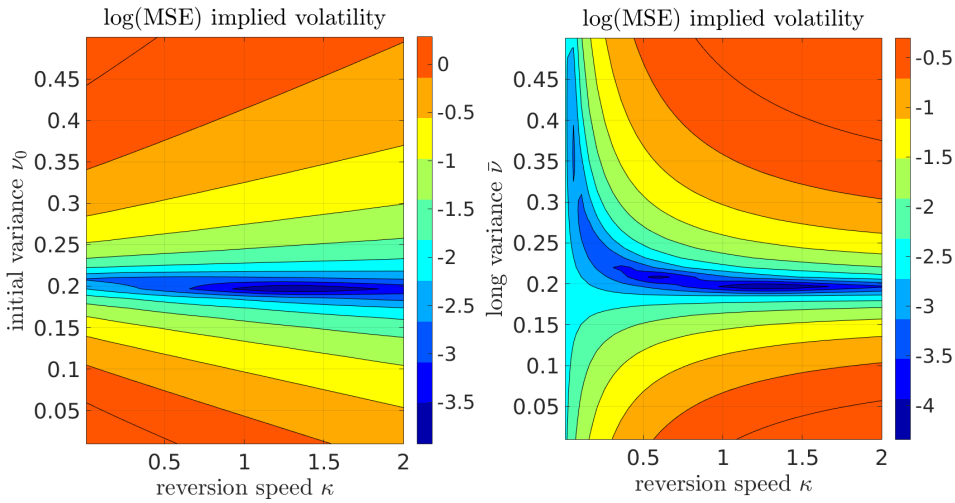
$$K_{i,j} = \bar{v} + \frac{v_0 - \bar{v}}{\kappa(T_j - t_0)}\left(1 - e^{-\kappa(T_i - t_0)}\right). \tag{3.13}$$

[2] VIX stands for the Chicago Board Options Exchange's Volatility Index.

When the initial values of the parameters have been determined, one can use the whole implied volatility surface to determine the optimal model parameters. To conclude, the number of Heston parameters to be calibrated depends on different scenarios. The flexibility of our CaNN is that it can handle varying numbers of to-calibrate parameters.

Effect of two combined parameters

In this section, two parameters are varied simultaneously in order to understand the joint impact on the objective function. Figure 3.5a presents the landscape of the objective function, here the logarithm of the MSE, when varying v_0 and κ but keeping the other parameters fixed in the Heston model. It is observed that the valley is narrow in the direction of v_0 but flat in the direction of κ. Several values of these parameters thus result in similar values of the objective function, which means that there may be no unique global minimum above a certain error threshold. Furthermore, for $\bar{v}$ and κ we observe also a flat minimum, with multiple local minima giving rise to similar MSEs, see Figure 3.5b.



(a) v_0 vs. κ (b) $\bar{v}$ vs. κ

Figure 3.5: Landscape of the objective function for the implied volatility. The true values are $\kappa^* = 1.0$ and $v_0^* = 0.2$ in the left plot, and $\kappa^* = 1.0$ and $\bar{v}^* = 0.2$ in the right plot. There are 35 market samples. The objective function is MSE. The contour plot is rendered by a log-transformation.

A similar observation holds for κ and γ: small values of κ and large γ values will, in certain settings, give essentially the same option prices as large values of κ and small γ values. This may give rise to multiple local minima for the objective function, as shown in Figure 3.6.

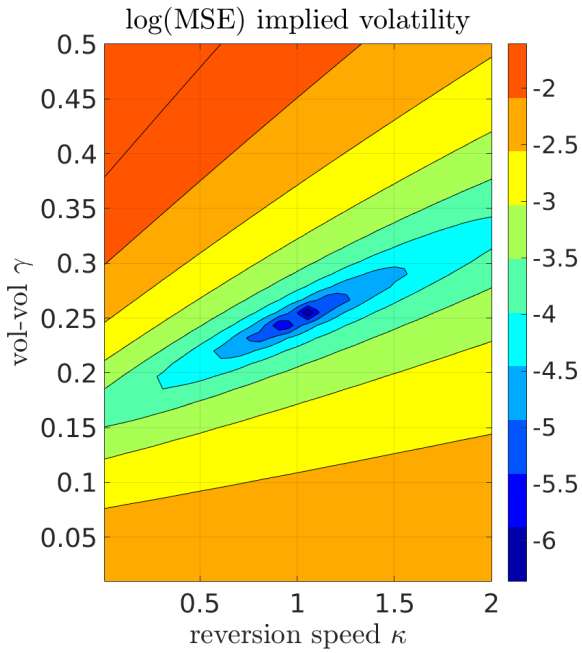


Figure 3.6: The objective function when varying γ and κ. The true values are $\kappa^* = 1.0$ and $\gamma^* = 0.25$.

For higher-dimensional objective functions, the structure becomes even more complex. This is a preliminary study of the sensitivities, and advanced tools are required for studying the effect of more than two parameters. We will show that the ANN can be used to obtain the sensitivities for more than two parameters to present the bigger picture of the dependencies and sensitivities. For this task the Hessian matrix of the five Heston parameters will be extracted (see Section 3.4.3).

3.4.2. THE FORWARD PASS

In this section, we discuss the forward pass, i.e., Heston-IV-ANN. A relatively large neural network is chosen so that in the forward pass the network is over-parametrized in terms of its expressive power and should be able to fit the pricing model well enough. This in turn comes at the cost of a more expensive computation, but provides a suitable forward pass to demonstrate that the parallel backward pass, in Section 3.4.3, can handle computation-intensive model calibration in a fast way. The selected hyper-parameters are listed in Table 3.2. Please note that increasing the number of neurons or using a deeper structure may lead to better approximations, but gives rise to an expensive-to-compute network. With our computing resources, we choose to employ 200 neurons each hidden layer to

balance the calibration speed and accuracy. We use 4 hidden layers and a linear output (regression) layer, so that the network contains 122,601 trainable parameters. MSE is used as the loss function measure to train the forward pass. The global structure is depicted in Figure 3.7. More details on the ANN solver can be found in [60] and in the previous chapter.

Table 3.2: Details and parameters of the selected ANN.

Parameters	Options
Hidden layers	4
Neurons(each layer)	200
Activation	ReLu
Dropout rate	0.0
Batch-normalization	No
Initialization	Glorot_uniform
Optimizer	Adam
Batch size	1024

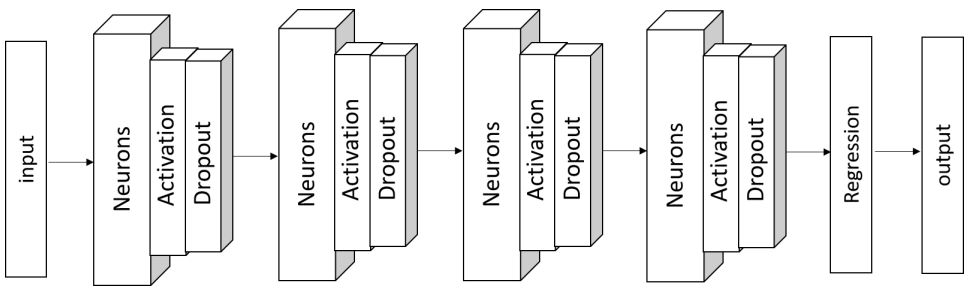


Figure 3.7: The structure of the ANN.

As a data-driven method, the samples from the parameter set for which the ANN is trained are randomly generated for the pricing of European options. The input contains eight variables, and Table 3.3 presents the range of six Heston input parameters (r, ρ, κ, $\bar{v}$, γ, v_0) as well as two option contract-related parameters (τ, m), with a fixed strike price $K = 1$. There are around one million data points. The complete data set is randomly divided into three parts, with 10% as the testing set, 10% as validation and 80% as the training data set.

After sampling the parameters, a robust version of the COS method is used to determine the option prices under the Heston model numerically. The default setting with $L_{COS} = 50$ and $N_{COS} = 1500$ will provide highly accurate option solutions for most of the samples, but it may end up with insufficient precision

in some extreme parameter cases. In such cases, the integration interval $[a, b]$ will be enlarged automatically, by increasing L_{COS} until the lower bound a and the upper bound b have different signs. Subsequently, the Black-Scholes implied volatility is calculated by Brent's method.

The option prices are just intermediate variables during training in the forward pass. The overall Heston-IV-ANN solver does not depend on the type of European option (e.g., call or put), since during the computation of the Black-Scholes implied volatilities the European options with identical Heston parameters should give rise to the same implied volatilities, independent of call or put prices. The forward pass can handle both call and put implied volatilities without requiring additional efforts. Here we are using European put options, since the COS method is more robust for pricing put than call options.

Table 3.3: Sampling range for the Heston parameters; LHS means Latin Hypercube Sampling, COS stands for the COS method (see Chapter 2), and Brent for the root-finding iteration.

ANN	Parameters	Value Range	Generating Method
ANN Input	Moneyness, $m = S_0/K$	[0.6, 1.4]	LHS
	Time to maturity, τ	[0.05, 3.0] (year)	LHS
	Risk free rate, r	[0.0%, 5%]	LHS
	Correlation, ρ	[-0.90, 0.0]	LHS
	Reversion speed, κ	(0, 3.0]	LHS
	Volatility of volatility, γ	(0.01, 0.8]	LHS
	Long average variance, $\bar{\nu}$	(0.01, 0.5]	LHS
	Initial variance, ν_0	(0.05, 0.5]	LHS
-	European put price, V	$(0, 0.6)$	COS
ANN Output	Black-Scholes IV, σ	$(0, 0.76)$	Brent

The ANN takes as input parameters $(r, \rho, \kappa, \bar{\nu}, \gamma, \nu_0, \tau, m)$, and approximates the Black-Scholes implied volatility σ. As mentioned in Table 3.2, the optimizer Adam is used to train the ANN on the generated data set. The learning rate is halved every 500 epochs. The training consists of 8000 epochs, both the training and validation losses have converged. The performance of the trained model is shown in Table 4.5.

We observe that the forward pass is able to obtain a very good accuracy and therefore learns the mapping between model parameters and implied volatility in a robust and accurate manner. The test performance is very similar to the train performance, showing that the ANN is able to generalize well.

Table 3.4: The trained forward pass performance. The default float type is float32 on the GPU. The measures are defined as follows: MSE = $\frac{1}{n}\sum(y_i-\hat{y}_i)^2$, MAE = $\frac{1}{n}\sum|y_i-\hat{y}_i|$, MAPE = $\frac{1}{n}\sum\frac{|y_i-\hat{y}_i|}{y_i}$, where y represents the true value, and $\hat{y}$ represents the predicted value with n being the number of samples.

Heston-IV-ANN	MSE	MAE	MAPE	R^2
Training	8.07×10^{-8}	2.15×10^{-4}	5.83×10^{-4}	0.9999936
Testing	1.23×10^{-7}	2.40×10^{-4}	7.20×10^{-4}	0.9999903

3

3.4.3. THE BACKWARD PASS

We will perform calibration using the CaNN based on the trained ANN from the previous section and evaluate its performance. We will work with the full set of Heston parameters to calibrate, but we will also study the impact of reducing the number of parameters to calibrate, as discussed in Section 3.4.1.

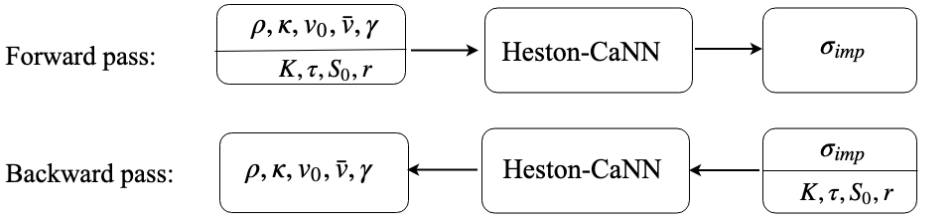


Figure 3.8: The Calibration Neural Network for the Heston model.

The aim is to check how accurately and efficiently the ANN approach can recover the input values. In order to investigate the performance of the proposed calibration approach, as shown in Figure 3.8, we generate synthetic samples by means of Heston-IV/COS-Brent, where the 'true' values of the parameters are known in advance. In other words, the parameters used to obtain the IV's from the COS-Brent's method, are now taken as output of the backward pass of the neural network, with σ_{imp} being the input conditional on (K,τ,S_0,r). Different financial models correspond to different CaNNs. Here we distinguish the Heston-CaNN (based on the Heston model, studied in this section), from the Bates-CaNN (based on the Bates model, studied in Section 3.4.4).

There are $5\times7=35$ 'observed' European option prices, that are made up of European OTM puts and calls. As shown in Table 3.5, the moneyness ranges from 0.85 to 1.15, and the maturity times vary from 0.5 to 2.0. Each implied volatility surface contains moneyness levels (85%, 90%, 95%, 100%, 110%, 115%) and maturities (0.5, 0.75, 1, 1.25, 1.5, 1.75, 2.0) with a prescribed risk-free interest rate of 3%. The samples with $m<1$ correspond to European call OTM options, while

those ones with $m > 1$ and $m = 1$ are OTM and ATM put options, respectively.

Table 3.5: The range of market quotes.

-	Parameters	Range	Samples
Market data	Moneyness, $m = S_0/K$	[0.85, 1.15]	5
	Time to maturity, τ	[0.5, 2.0](year)	7
	Risk free rate, r	0.03	Fixed
	European call/put price, V/K	(0.0, 0.6)	-
Black-Scholes	Implied Volatility	(0.2, 0.5)	35

We use the total squared error measure $J(\Theta)$ as the objective function during the calibration,

$$J(\Theta) = \sum \omega \left(\sigma_{imp}^{\text{ANN}} - \sigma_{imp}^{*} \right)^2 + \bar{\lambda} ||\Theta||, \tag{3.14}$$

where $\sigma_{imp}^{\text{ANN}}$ is the ANN-model-based value and σ_{imp}^{*} is the observed one. We give a small penalty parameter $\bar{\lambda}$ depending on the dimensionality of the calibration[3]. The forward pass has been trained with implied volatility as the output quantity, as described in Section 3.4.1. The parameter settings of the DE optimization is shown in Table 3.1.

CALIBRATION TO HESTON OPTION QUOTES

In this section we focus on two scenarios for the Heston model, calibrating either three parameters, with a fixed κ and a known v_0, or calibrating five parameters. In order to create synthetic calibration data, we choose five equally-spaced points between the lower and upper bound for each parameter, and there are $5^5 = 3125$ combination cases in total, as shown in Table 3.6. For each experiment, five different random seeds of DE are tested, because the DE optimization involves random operations which may cause the performance to fluctuate. In addition, all quotes have the equal weight $\omega = 1$ in this section.

First, the scenario of three parameters is studied, fixing κ and v_0 during calibration. We compare the averaged results by implementing each test case five times. The wording "function evaluation" refers to how many times the model has been compared to the observed implied volatility. The population size in the DE is $15 \times N_v$, that is, $15 \times 3 = 45$. With the population ratio increasing further, no significant benefits were observed. As shown in Table 3.7, the time on the GPU is around half of that on the CPU.

[3]When calibrating three parameters, we set $\bar{\lambda}$ to zero. When calibrating more than three parameters, $\bar{\lambda}$ is a small value 1.0×10^{-6}, which is close to the MSE of the trained ANN. In this case, the regularization term only has a limited effect on the objective function during calibration.

Table 3.6: Uniformly distributed points between the lower and upper bounds of the Heston parameters.

parameter	lower	upper	points	CaNN search space
ρ	-0.75	-0.25	5	[-0.85,-0.05]
$\bar{v}$	0.15	0.35	5	[0.05, 0.45]
γ	0.3	0.5	5	[0.05, 0.75]
v_0	0.15	0.35	5	[0.05, 0.45]
κ	0.5	1.0	5	[0.1, 2.0]

Table 3.7: Averaged performance of the backward pass of the Heston-CaNN, calibrating **3** parameters on a CPU (Intel i5, 3.33GHz with cache size 4MB) and on a GPU (NVIDIA Tesla P100), over 3125×5 (random seeds) test cases, where † stands for CaNN estimated value, and $*$ stands for the true value, with MJ = $J(\Theta)/N$.

Absolute deviation from Θ^*		Error measure		Computational cost	
$\lvert\bar{v}^\dagger - \bar{v}^*\rvert$	1.60×10^{-3}	$J(\Theta)$	1.45×10^{-6}	**CPU** time (seconds)	0.29
$\lvert\gamma^\dagger - \gamma^*\rvert$	1.79×10^{-2}	MJ	4.14×10^{-8}	**GPU** time (seconds)	0.15
$\lvert\rho^\dagger - \rho^*\rvert$	2.44×10^{-2}	Data points	35	Function evaluations	59221

In the case of five parameters $(\rho, \bar{v}, \gamma, v_0, \kappa)$, the calibration problem is more likely to give rise to a many-to-one problem; that is, many sets of parameter values may correspond to the same volatility surface. A regularization factor $\bar{\lambda} = 1.0\times10^{-6}$ is added to guide CaNN to a set of values for which the sum of their magnitudes is the smallest among the feasible solutions, as shown in Equation (3.4). Here the DE population size is $50 = 10\times5$ parameters. As shown in Table 3.8, the Heston-CaNN finds the values of these parameters in approximately 0.5 seconds on a GPU, with around 20,000 function evaluations. *There are several reasons why the CaNN with DE performs fast and efficiently.* First of all, the forward pass runs faster compared to a two-step computation from the Heston parameters to the implied volatilities, since an iterative root-finding algorithm for the implied volatility takes some computing time. In addition, the entire group of observed data can be evaluated at once in the framework. Other benefits come from the acceleration due to the parallelized DE optimization, where the whole population is computed simultaneously in the selection stage.

Table 3.8: Performance of Heston-CaNN, calibrating **5** parameters on a GPU, over 3125×5 (random seeds) test cases.

Absolute deviation from Θ^*		Error measure		Computational cost	
$\lvert v_0^\dagger - v_0^*\rvert$	4.39×10^{-4}	$J(\Theta)$	2.52×10^{-6}	CPU time (seconds)	0.85
$\lvert \bar{v}^\dagger - \bar{v}^*\rvert$	4.54×10^{-3}	MJ	7.18×10^{-8}	GPU time (seconds)	0.48
$\lvert \gamma^\dagger - \gamma^*\rvert$	3.28×10^{-2}			Function evaluations	193249
$\lvert \rho^\dagger - \rho^*\rvert$	4.84×10^{-2}			Data points	35
$\lvert \kappa^\dagger - \kappa^*\rvert$	4.88×10^{-2}				

SENSITIVITY ANALYSIS BASED ON ANNS

The gradients of the objective function can be extracted from the trained model, as mentioned in Section 3.3.1. These can be used to gain some insights into the complex structure of the loss surface and thus into the complexity of the optimization problem for calibration. We use here the Hessian matrix, which describes the local curvature of the loss function. No explicit formula is available for the relations the neural network learns between the implied volatilities and the model parameters, however, it is feasible to extract the Hessian from the trained ANN, giving insight into this relation and the sensitivities. Table 3.9 shows a Hessian matrix, where the Hessian is defined as $\partial_{y_i y_j} L(\Theta)$, where y_i and y_j are output of the neural network ($y \in \Theta$, the to-be calibrated parameters). The Hessian is computed by differentiating the Heston-IV-ANN loss for computing the Black-Scholes implied volatility with respect to the Heston parameters on 35 market data points based on the parameter ranges in Table 3.5. Here the objective function is the MSE to exclude the effects of a regularization factor.

Table 3.9: A Hessian matrix at the true value set Θ^*.

	$\partial\rho$	$\partial\kappa$	$\partial\gamma$	$\partial\bar{v}$	∂v_0
$\partial\rho$	2.79×10^{-2}	–	–	–	–
$\partial\kappa$	1.14×10^{-2}	8.20×10^{-3}	–	–	–
$\partial\gamma$	-2.88×10^{-2}	-1.76×10^{-2}	4.11×10^{-2}	–	–
$\partial\bar{v}$	7.45×10^{-2}	5.51×10^{-2}	-1.19×10^{-1}	3.76×10^{-1}	–
∂v_0	2.16×10^{-1}	1.27×10^{-1}	-3.10×10^{-1}	8.77×10^{-1}	2.66

We can understand how the parameters affect the loss surface around the optimum with help of the Hessian matrix, by analyzing the sensitivities of the implied volatility with respect to the five parameters. Observe that the value of

the Hessian with respect to κ is the smallest among the sensitivities. As shown in Table 3.9, the ratio between $\partial^2 J(\Theta^*)/\partial v_0^2$ and $\partial^2 J(\Theta^*)/\partial \kappa^2$ is around 323, which suggests that changing 1 unit of v_0 is approximately equivalent to changing 323 units of κ for the objective function. When the Hessian value is small in absolute value, the loss surface at that point exhibits flatness in the corresponding direction. As visible in Figure 3.5, the ground-truth loss surface gets increasingly stretched along the axis with κ, resulting in a narrow valley with a flat bottom. This also indicates that there is no unique global minimum above a certain non-zero convergence tolerance, since multiple values of κ would result in similar values of the loss function. In addition, the convergence performance, especially for the steepest descent method, depends on the ratio of the smallest to the largest eigenvalue of the Hessian; this ratio is also known as the condition number in the case of symmetric positive matrices. The ratio between $\partial^2 J(\Theta^*)/\partial \bar{v}^2$ and $\partial^2 J(\Theta^*)/\partial \kappa^2$ is around 45, as visible in Figure 3.5b. From the results in [74], when the target quantity is based on the option prices, this ratio between $\partial^2 J(\Theta^*)/\partial \bar{v}^2$ and $\partial^2 J(\Theta^*)/\partial \kappa^2$ is sometimes found to be of order 10^6, which makes the calibration problem increasingly complex due to a great disparity in sensitivity. Calibrating to the implied volatility appears to reduce the ratio between different Hessian entries compared to the option prices, thus decreasing Hessian's condition number and resulting in a more efficient and accurate calibration performance.

Table 3.9 also suggests that the entries $|\partial^2 J(\Theta^*)/\partial \kappa^2|$ and $|\partial^2 J(\Theta^*)/\partial \rho^2|$ are among the smallest ones around the optimum. These two parameters thus have the smallest effect on the objective function. Therefore, the DE method can converge to values that are in a wide area of the search space, since these parameters do not impact the error measure significantly. A straightforward way to address this issue is by adding a regularization term to choose a particular solution, for example, like Equation (3.14). Another way is to take advantage of the population-based algorithm DE. Since there are several candidates in each generation, we can select the top few candidates to get an averaged solution when DE converges. This averaged solution may lead to wider optima and better robustness. Some recent papers, like [75] have used similar ideas to improve the generalization of the neural network. The parameter v_0 is the most sensitive one and it appears to dominate the ANN calibration process. Therefore, the predicted parameter v_0 is the most precise among all parameters in order to achieve the desired accuracy.

The above analysis explains the behavior of the absolute deviation of the five parameters as shown in Table 3.8. The error measure MJ can not drop significantly below 7.18×10^{-8}, as this value is close to the testing accuracy, MSE= 1.23×10^{-7}, of the Heston-IV-ANN model. In other words, any further explo-

ration of the DE optimization can not distinguish the parameters impact on the loss anymore.

3.4.4. THE BATES MODEL

In this section, we compute the option prices using the Bates model. First, we use Bates to simulate the quotes in the market data and evaluate Heston-CaNN. Second, we develop Bates-CaNN to generate more complex shapes of implied volatility curves.

CALIBRATION TO BATES QUOTES

In this section, the Bates model is employed to create the synthetic market data, in order to generate a more realistic (complex) volatility shape by adding some 'perturbations' to the previous Heston data. It is then followed by a calibration based on the Heston model. The aim is to check whether the resulting implied volatilities can be recovered by the machine learning calibration framework.

So, the observed data set in Table 3.10 is from the Bates option prices. During the calibration, we will employ the backward pass based on the Heston model to determine a set of parameter values which approximate the generated implied volatility function.

There are two sets of experiments, based on either rare jumps or common jumps in the stock price process. Figure 3.9 compares the implied volatility from the Bates model (forward) computations and the CaNN-based Heston implied volatilities. Clearly, when the impact of the jumps is small, the Heston model can accurately mimic the implied volatility generated by the Bates parameters. In this case, many different input parameters for the Bates model will give very similar implied volatility surfaces. With an increasing jump intensity, the deviation between the two models can become significant, especially for short maturity options.

In financial practice, a perfect calibration to the ATM options is often required. We can enforce this, by increasing the weights of the ATM options in the objective function. The third figure from Figure 3.9 and Table 3.10 both compare the differences when the ATM options in the objective function are weighted. The two curves fit very well ATM, however, in this case the total error increases with unequal weighting. The results demonstrate the robustness of the CaNN framework. It is however well-known that the Heston model can not fit short-maturity market implied volatility very well, and therefore we will also employ a higher-dimensional model, e.g., calibrating directly the Bates model, which will be discussed in the next section.

Table 3.10: The Heston parameters are estimated with the CaNN by calibrating to a data set generated by the Bates model. 'Ground total squared error' refers to the sum of the differences between σ^*_{imp} and σ_{imp}, where σ_{imp} is obtained using the COS and Brent methods with already calibrated Heston parameter values. For a single calibration case, the computing time fluctuates slightly, as the CPU or GPU performance may be influenced by external factors. Function evaluations should be a reliable measure to estimate the time.

–	Calibration	Rare	Jump	Common	Jump	Weighting	ATM
Parameters	Search space	Bates	Heston	Bates	Heston	Bates	Heston
Intensity of jumps, λ_J	-	0.1	-	1.0	-	1.0	-
Mean of jumps, μ_J	-	0.1	-	0.1	-	0.1	-
Variance of jumps, ν_J^2	-	0.1^2	-	0.1^2	-	0.1^2	-
Correlation, ρ	[-0.9, 0.0]	-0.3	-0.284	-0.3	-0.135	-0.3	-0.164
Reversion speed, κ	[0.1, 3.0]	1.0	1.140	1.0	1.050	1.0	1.205
Long variance, $\bar{\nu}$	[0.01, 0.5]	0.1	0.100	0.1	0.120	0.1	0.114
Volatility of volatility, γ	[0.01, 0.8]	0.7	0.728	0.7	0.701	0.7	0.604
Initial variance, ν_0	[0.01, 0.5]	0.1	0.103	0.1	0.119	0.1	0.115
Function evaluations	CaNN	-	162890	-	155680	-	258300
Time(seconds)	GPU	-	0.45	-	0.40	-	0.7
Total Squared Error	Ground	-	1.38×10^{-6}	-	5.19×10^{-6}	-	5.95×10^{-5}

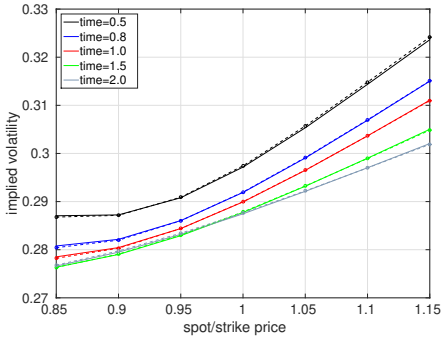
CALIBRATING THE BATES MODEL

Next we show the ability of Bates-CaNN to calibrate the Bates model parameters. The Bates model calibration is a higher-dimensional problem, since the Bates model is based on more parameters than the Heston model. The proposed CaNN framework is used to calibrate eight parameters in the Bates model, a setting in which we are dealing with more complex implied volatility surfaces.

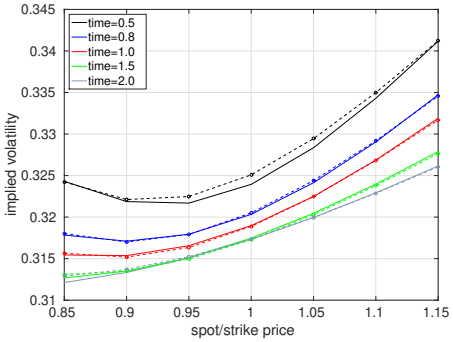
Initially, the Bates-IV-ANN forward pass is trained on the training data set consisting of one million samples that are generated by the Bates model. Compared to the forward pass of the Heston model, merely a different characteristic function is inserted in the COS method, and three additional model parameters have been varied. The Bates-CaNN is employed to calibrate the Bates model, aiming to recover the eight Bates model parameters possibly well. All the samples have equal weight, and the regularization factor is $\bar{\lambda} = 1.0 \times 10^{-6}$.

Table 3.11 shows an example with high intensity, large variance jumps, for which the Heston model can not capture the corresponding implied volatility accurately. There are still 35 market samples as shown in Table 3.5. Estimating eight parameters is a challenging task, including very many comparisons between the model and the market values during calibration.

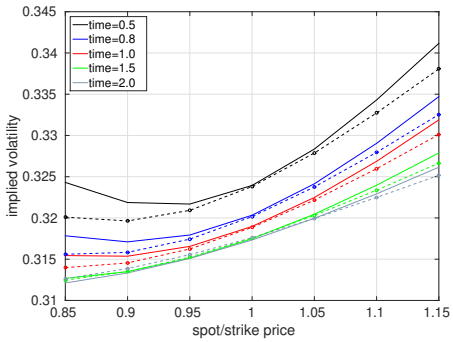
Figure 3.10 compares the implied volatilities from the synthetic market and the calibrated Bates model. These volatilities resemble each other very well, even when the curvature is high with short time to maturity.



(a) rare jump, equal weighting



(b) common jump, equal weighting



(c) common jump, $\omega_{ATM} = 5000$

Figure 3.9: Implied volatilities from the 'market' and calibration. The solid lines represent the Bates implied volatilities, while the dashed lines are the calibrated Heston-based volatilities. The impact of weighting ATM options can be seen in the third figure.

3.5. CONCLUSION

In this chapter we proposed a machine learning-based framework to calibrate pricing models, in particular focusing on the high-dimensional calibration problems of the Heston and Bates models. The proposed approach has several favorable features, where an important one is robustness. Without choosing specific initial values, the DE global optimizer prevents the model calibration getting stuck in a local minimum.

Fast calibration results from several factors. An ANN is efficient in computing the output values for a single input setting. When calibrating, the market data can be computed by ANNs simultaneously. Using DE, during the selection stage, ANNs can calculate a whole population in each generation at once, in parallel on a parallel computing architecture. The numerical experiments show that opti-

Table 3.11: The Bates parameters are estimated with Bates-CaNN, by calibrating to a data set (35 samples) generated by the Bates model. In DE, the random seed is 2 and the population size is $10 \times N_v = 80$.

Parameters	CaNN Search space	Bates	Calibrated
Intensity of jumps, λ_J	[0, 3.0]	1.0	1.065
Mean of jumps, μ_J	[0, 0.4]	0.1	0.087
Variance of jumps, v_J^2	[0, 0.3]	0.160	0.146
Correlation, ρ	[-0.9, 0.0]	-0.3	-0.228
Reversion speed, κ	[0.1, 3.0]	1.0	0.598
Long average variance, $\bar{v}$	[0.01, 0.5]	0.1	0.128
Volatility of volatility, γ	[0.01, 0.8]	0.7	0.776
Initial variance, v_0	[0.01, 0.5]	0.1	0.102
Total Squared Error	-	-	4.95×10^{-6}
Function evaluation	-	-	842800
Time(seconds)	-	-	1.8

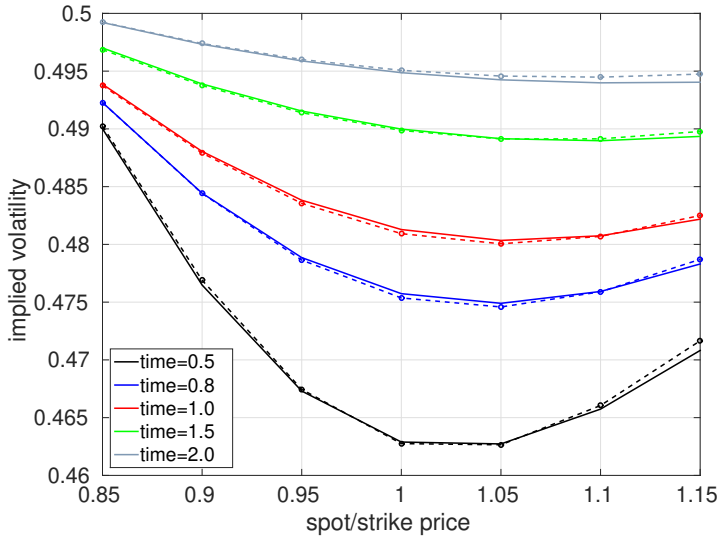


Figure 3.10: The solid lines represent the observed implied volatilities, with the dashed lines being the model calibrated ones. This plot shows the result with equal weights and $\bar{\lambda} = 1.0 \times 10^{-6}$. The random seed is 2 during calibration.

mal values can be found within a second even when using a global optimization algorithm.

The ANN-based approach provides new tools to gain insight into the calibration problem. We used the Hessian matrix to perform a sensitivity analysis, where the sensitivities can efficiently be extracted for large numbers of model parameters. The Hessian matrix also explained why implied volatility, used in our work, is preferred over option prices, used in previous works, from an optimization perspective.

The calibration framework furthermore is generic, and does neither require characteristic functions, nor explicit gradients of financial models. The number of market data or to-calibrate parameters is also flexible. With this framework, the model can be extended to multiple quantities, e.g., calibrating to both option prices and implied volatility. To conclude, the ANN combined with DE provides an efficient and accurate framework for calibrating financial models.

To look forward, the above ANN calibration process does not rely on the quality of the initial guess. However, because the market does not change dramatically in a short time period, it may make sense to take the last available values as starting points of the calibration. There are several possible strategies for the calibration framework in this situation. One is switching to the gradient-based local optimization algorithms and another one is narrowing the search space of the DE, which will further reduce the computational time considerably. Further future improvements include combining gradient-based optimization with the DE, since the gradient information is readily accessible. It is also feasible to employ a small neural network to reduce the computing time, like in the paper [63] which builds a three-hidden-layers ANN and each layer has 30 nodes during the calibration.

Finally, a follow-up paper recently appeared by researchers from Germany, see [76], which used our proposed CaNN to calibrate financial models on the basis of real observed market data, and they achieved highly competitive results. We also extended the CaNN ourselves to calibrate so-called rough volatility models, see the MSc thesis [77].

4

EXTRACTING IMPLIED INFORMATION FROM AMERICAN OPTIONS

Extracting implied information, like volatility and dividend, from observed option prices is a challenging task when dealing with American options, because of the complex-shaped early-exercise regions and the computational costs to solve the corresponding mathematical problem repeatedly. We will employ a data-driven machine learning approach to estimate the Black-Scholes implied volatility and the dividend yield for American options in a fast and robust way. To determine the implied volatility, the inverse function is approximated by an artificial neural network on the computational domain of interest, which decouples the offline (training) and online (prediction) stages and thus eliminates the need for an iterative process. In the case of unknown dividend yield, we formulate the inverse problem as a calibration problem and determine simultaneously the implied volatility and dividend yield. For this, a generic and robust calibration framework, CaNN as introduced in Chapter 3, is used to estimate multiple parameters. It is shown that machine learning can be used as an efficient numerical technique to extract implied information from American options, particularly when considering multiple early-exercise points due to negative interest rates.

This chapter is based on the article 'On a neural network to extract implied information from American options', submitted for publication.

4.1. Introduction

So-called implied financial information, which is obtained from financial derivatives prices, is useful information for risk management, for the valuation of other financial derivatives, for hedging, and forecasting [78, 79]. Different from historical volatility (which is computed from past known asset prices), implied volatility (which is computed from the market option prices) reflects the market implied uncertainty in the underlying asset prices. Similarly, implied dividend can be seen as a measure which indicates how much market participants expect an asset price will be reduced under the so-called pricing, or risk-neutral, measure. Compared to implied volatility, the implied dividend is typically a relatively weak signal, but there are several applications for the implied dividend information. An accurate implied dividend yield may result in accurate theoretical option values, Greeks and implied volatilities. One may choose a specific trading strategy according to the difference between the market implied and announced actual dividends, see [80]. The authors in [79, 81, 82] give evidence for the fact that implied dividends are a significant factor for forecasting actual dividend changes. In other words, implied dividend is shown to have important predictive power compared to historical dividends.

American options, i.e. options with early-exercise features, are commonly traded. The underlying asset may be any financial asset, such as currencies, commodities, bonds, stocks, and so on. With American options, the holder has the right (but not the obligation) to exercise the contract at any time before the contract's expiry, whereas a European option can only be exercised at the expiry time. Computing the implied volatility and implied divided from observed European option prices has often been addressed in the literature, for example, in [34–36, 80]. The computation of American option prices is generally more expensive than pricing European options, because of early-exercise features. Deriving the implied information from American option values is therefore also a more challenging task [83–86].

There are different ways to compute implied information. For European options, a closed-form expression may be derived, for example, to approximate the implied volatility in certain parameter ranges, see [34–36]. Such expressions are typically based on a Taylor series expansion and on the analytical solution of the European option pricing model. One of the drawbacks, however, is that the resulting formulas are only accurate near ATM, and may give rise to inaccurate implied volatility values for deep ITM and OTM options. To estimate the implied dividend, a popular way is by means of the put-call parity, which holds, however, only for European options, and is not valid for American options due to the early-exercise premium [80, 81]. A second approach is by formulating the computation of the implied information as a minimization problem, which is then

based on an iterative search technique. Traditionally, this methodology requires the repeated solution of the American option pricing problem before reaching the stop criterion. Under a minimization framework, there are essentially two popular approaches of determining the American option implied volatility. The first one is by means of a de-Americanization technique, which translates an American option price into the corresponding European prices [83, 84, 87]. Significant pricing errors may arise due to inaccurate incorporation of the early-exercise premium. The higher the early-exercise premium, typically the larger the error can get. Taking dividends into consideration may further increase the error of the de-Americanization technique. A second approach is to conduct a direct calibration of the American pricing model, see [83, 85, 86, 88, 89]. Unlike the European options, the derivative of the option value with respect to the volatility does not have a closed-form expression in the case of American options. In addition, other complicating factors, such as a negative interest rate [90], may lead to a complex-shaped early-exercise region (e.g. we may encounter different continuation regions)[91].

Recently, artificial neural networks have been emerging as advanced computational techniques to obtain solutions to possibly complicated problems in computational finance, for example, in [22, 92–96]. We refer to [97] for a review. Nowadays, deep neural networks are also used to speed up the computation of prices and sensitivities of American options, see [98–101]. To date, those neural network-based algorithms have not been computationally efficient to enable fast American option model calibration.

In order to accelerate the time-consuming solution of the American option pricing model, we can take advantage of supervised learning of ANNs. Training ANNs to learn the implied information from observed option prices is non-trivial. For example, the steep gradient of implied volatility with respect to the option price may cause inaccurate ANN results [92]. Moreover, in the early-exercise region, the American option price and the volatility are not bijective, as the gradient of the option price with respect to volatility, Vega, equals zero in the early-exercise region. A robust, global optimization plays an important role when exploring the solution space. Consequently, the ANN efficiency may suffer due to the global optimization. In Chapter 3 we proposed a neural network-based calibration framework, i.e., the Calibration Neural Network (CaNN), to address the speed issue of model calibration with a global optimization algorithm. A recent study [76] shows that the CaNN can achieve a competitive performance when dealing with real market data. Usually, however, the implied dividend yield is also unknown, so that there are two open implied model parameters to determine, see for example, [102, 103].

In this work, we will employ the CaNN to extract implied information from

American option prices. Negative interest rates as well as a negative dividend yield are taken into consideration to cover a broad variety of market conditions. We thus propose a data-driven machine learning method to address the American option implied information. More specifically, the proposed CaNN is composed of three components, an efficient option pricing method, a global optimization technique and an implementation which runs efficiently on a parallel computing platform. There are two separate stages in CaNN, the forward pass to learn the pricing model and the backward pass to estimate the model parameters. Here the forward pass of CaNN will give us two output quantities, American put and call prices, which originate from one ANN. When a dividend yield is already known, the ANN simplifies as it can be used to computing the American option implied volatility by directly approximating the inverse function. In such case, an iterative numerical method is not needed.

The remainder of this chapter is organized as follows. In Section 4.2, the mathematical American option pricing models are introduced. Furthermore, in Section 4.2.3, the implied volatility and implied dividend yield from American options are discussed. In Section 4.4, we describe the data-driven ANN to extract implied information from American options. When a dividend yield is known, the CaNN simplifies as it can be used to computing the American option implied volatility by directly approximating the inverse function. In such case, an iterative numerical method is not needed. When the dividend yield is known, we can use the ANN to approximate the inverse function. In other cases, the CaNN is employed to determine both the implied dividend and implied volatility. In Section 4.5, numerical experiments are presented to demonstrate the performance of the proposed methods.

4.2. American options

In this section we will discuss the mathematical model used to price the American options, and the implied information in market option prices.

4.2.1. Problem formulation

Although other pricing models would easily fit into our framework, for clarity we will concentrate on the Black-Scholes pricing framework. The underlying asset price thus follows a Geometric Brownian Motion (GBM) process, under the risk-neutral measure,

$$dS(t) = (r-q)S(t)dt + \sigma S(t)dW(t), \;\; S(0) = S_0, \tag{4.1}$$

where $S(t)$ is the underlying spot price at time t, and σ is the volatility parameter and $W(t)$ a Wiener process under the risk-neutral measure, S_0 the starting

point at time $t = 0$. The two parameters r and q can be interpreted in different ways. For example, r and q are the risk-less interest rate and dividend yield, respectively, for stocks. In the context of currencies, r and q may be two different interest rates, and q would represent the cost of carry in the case of commodities. In this chapter, we stay with a stock option description, for convenience, but we will also discuss $q < 0$, which is found in commodity modeling. With a risk-less asset $B(t)$,

$$dB(t) = rB(t)dt, \tag{4.2}$$

the arbitrage-free value of an American option at time t is given by

$$V_{am}(t,S) = \sup_{u\in[0,T]} \mathbb{E}_t^{\mathbb{Q}}[e^{-r(T-t)}H(K,S(u))|S(u)], \tag{4.3}$$

where $H(\cdot)$ is the payoff function, with strike price K; $\mathbb{E}_t^{\mathbb{Q}}$ represents the expectation under the risk-neutral measure $\mathbb{Q}$, with T being the maturity time. An optimal exercise boundary $S_t^* \equiv S^*(t)$, which depends on the time to maturity $T-t$, divides the domain into early-exercise (stopping) regions Ω_s and continuation (or holding) regions Ω_h. In general, early-exercise will be triggered when the discounted expected value drops below the value of exercising the option.

As an American option can be exercised anytime before the expiry time, a corresponding early-exercise premium should be added to the European option counterpart. For example, an American put option [104] can be decomposed into the corresponding European put price and the early-exercise premium, i.e.,

$$V_{am}^P(t,S) = \mathbb{E}_t^{\mathbb{Q}}[e^{-r(T-t)}\max(K-S(T),0)] + \int_t^T \mathbb{E}_u^{\mathbb{Q}}[(rK-qS(u))\mathbb{1}_{\{S(u)\in\Omega_s\}}]du, \tag{4.4}$$

where Ω_s represents the stopping region, and the whole domain is $\Omega = \Omega_s + \Omega_h$ with Ω_h being the holding region. The first term in Equation (4.4) is indeed equivalent to the European Black-Scholes put solution. The above problem can be formulated as a Black-Scholes inequality, on a domain Ω with a free boundary S_t^*. At the free boundary, we have,

$$V_{am}(t,S_t^*) = H(K,S_t^*), \quad \frac{\partial V_{am}}{\partial S} = \alpha, \tag{4.5}$$

where $\alpha = 1$ for American calls, $\alpha = -1$ for American puts, and $S(t)^*$ is the asset price for which the option value equals the payoff function. For American put options, the payoff function equals $H(K,S(t)) = \max(K-S(t),0)$. The above conditions at the free boundary can be used to distinguish continuation from stopping regions, and we will use these conditions in Section 4.4.2. In this chapter, the American Black-Scholes solution and corresponding pricing model are denoted by $V_{am} = BS_{am}(\sigma,S,K,t,T,r,q,\alpha)$, with the time to maturity $\tau := T-t$.

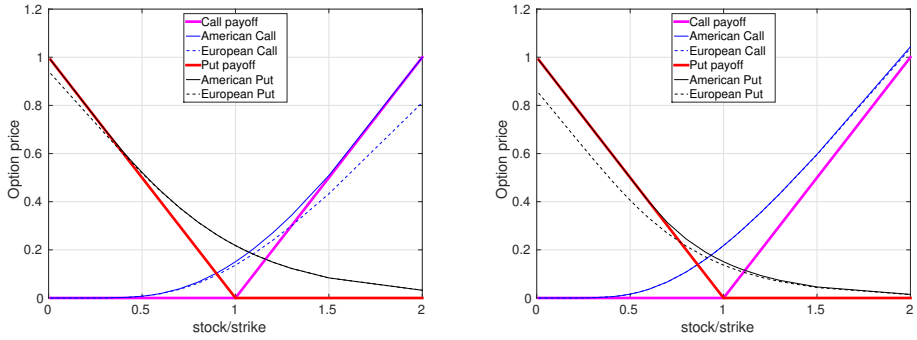
4.2.2. THE PUT-CALL SYMMETRY

The *put-call symmetry* relation holds for both European options and American options, and is given by,

$$V^P(t,S;K,T,\sigma,r,q) = V^C(t,K;S,T,\sigma,q,r). \tag{4.6}$$

The above relation allows us to value a call or put option by means of its counterpart, where the role of stock and cash values is interchanged. By swapping the strike with the spot price and the interest rate with the dividend yield, an American call value equals the corresponding American put. The relationship is also valid under negative discount rates [91]. Because with Equation (4.6) we can get two option prices from one computation, only one function evaluation is required to compute American call and put prices. We will focus on American put options in the following sections, and compute American call options using the put-call symmetry relation.

There are two types of computational regions when dealing with American options, the continuation (or holding) and the early-exercise (or stopping) region. The continuation region boundaries are not known a-priori. Figure 4.1 illustrates the difference between European and American Black-Scholes option solutions, with two sets of parameters. The American put option price, in the case of no dividend payment, should not be less than the put payoff, while the value of a European put option may be less than the payoff before expiry. In Figure 4.1, there only appears one early-exercise point.



(a) $r > q$ (b) $r < q$

Figure 4.1: Left: American vs. European Black-Scholes put and call option prices ($r > q$): $r = 0.10$, $q = 0.04$. Right: American vs. European Black-Scholes put and call option prices ($r < q$): $r = 0.04$, $q = 0.10$. The option values are at initial time $t = 0.0$, with the expiry time $T = 1.5$, volatility $\sigma = 0.4$.

However, two continuation regions may arise, when both the interest rate and dividend yield become negative [91, 105] in the case of American options.

Figure 4.2 presents an American put solution with two early-exercise points, so that the continuation regions are discontinuous. There are several reasons why we consider a negative dividend yield in our pricing model. The interest rate may be negative in practice. When we use the put-call symmetry to compute American calls by means of puts (the counterpart), we switch the interest rate and the dividend yield in the pricing equation. For that reason, the dividend yield may also be negative in the corresponding formula. An American Black-Scholes model is also used in foreign exchange or commodity markets, where negative q may be interpreted from an economic point-of-view.

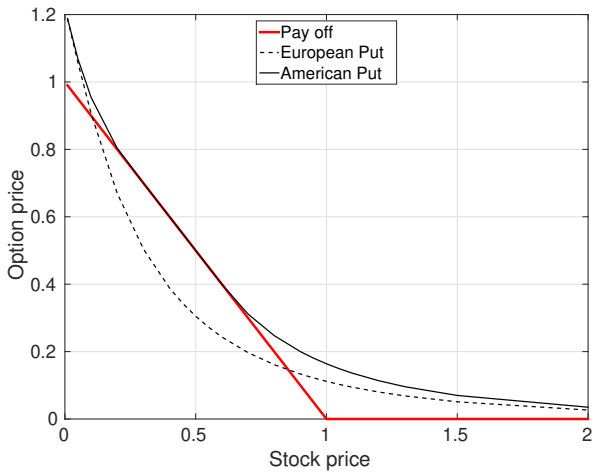


Figure 4.2: The setting: $r = -0.01$, $q = -0.06$, $\sigma = 0.2$, $T = 20$, $K = 1.0$. The option value in the solid black line hits the payoff function twice. The stopping region is between the two early-exercise points.

4.2.3. IMPLIED VOLATILITY AND DIVIDEND YIELD

The information implied in option prices provides participants' expectations about future market conditions. We will discuss the implied volatility and implied dividend yield.

IMPLIED VOLATILITY

Implied volatility represents a specific measure of the future uncertainty from the market point of view. Mathematically, the implied volatility of an option is the level of volatility which, when inserted in the (Black-Scholes) pricing model, makes the market and model prices match. In that sense, the implied volatility computed from market option prices is viewed as an indication for the look-

forward uncertainty of the underlying asset prices as estimated by market participants.

Computing the implied volatility can be formulated as an inverse problem. The American option's implied volatility is then written as,

$$\sigma^* = BS_{am}^{-1}(V_{am}^{mkt}; S, K, t, T, r, q, \alpha), \tag{4.7}$$

where $BS_{am}^{-1}(\cdot)$ denotes the inversion of the American Black-Scholes pricing problem, and V_{am}^{mkt} is an American option price observed in the market, with S being the underlying asset spot price at time t.

One often solves the implied volatility problem by means of a nonlinear root-finding method, and employs an iterative algorithm to obtain its solution. Given an American option market price, the implied volatility σ^* is determined by solving the following

$$V_{am}^{mkt} - BS_{am}(\sigma^*; S, K, t, T, r, q, \alpha) = 0. \tag{4.8}$$

Existence of σ^* can be guaranteed by the monotonicity of the Black-Scholes equation with respect to the volatility in the holding region. Unlike for European options, a closed-from expression for the derivative of the American option value with respect to the volatility is not available. Various solutions have been proposed to solve the implied volatility of American options, see, for example, [83, 85, 86, 88]. As stated in [86], these solutions may have difficulties especially with deep in-the-money options. One of the reasons is that option prices are insensitive to the underlying volatility deep in the money. Gradient-free methods, like bisection, do not rely on gradient information, but they may converge slowly because of the stopping regions. An important aspect when extracting the implied volatility is that the derivative of the option price with respect to the volatility, the option's Vega, becomes zero in the stopping region for American call and put options. It is well-known that,

$$|\Delta| = |\frac{\partial V_{am}}{\partial S}| = 1, \quad \text{Vega} = \frac{\partial V_{am}}{\partial \sigma} = 0. \tag{4.9}$$

In other words, the American option prices do not depend on the volatility in the stopping regions. As shown in Figure 4.3, Vega is positive in the holding region and zero in the stopping region. Consequently,

$$\frac{\partial \sigma}{\partial V_{am}} = \frac{1}{\text{Vega}} \to \infty.$$

When we invert the American Black-Scholes pricing problem in the stopping regions, there is no unique solution for the implied volatility. Therefore, the definition domain of Formula (4.7) should be the continuation region.

Remark. *When gradient-based minimization algorithms (e.g. Newton's method) are used to extract implied information, an initial guess for the solution has to be specified. An inappropriate starting point may cause the algorithm to fall into a "different" continuation region from the "envisioned" region. Special rules have to be designed to help the algorithm reach the "correct" continuation region and explore the solution space. This makes it challenging to define a suitable starting point for the minimization algorithm when inverting the pricing model.*

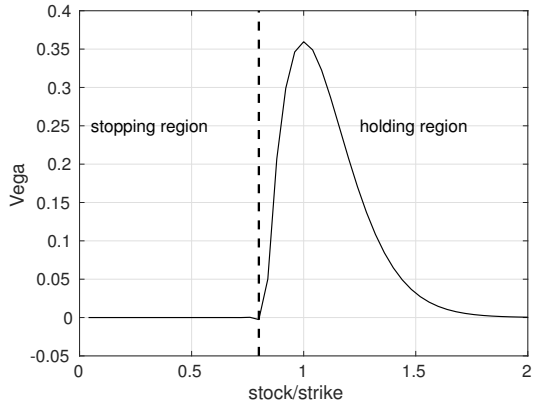


Figure 4.3: The Vega for American puts in different regions. The strike price is $K = 1.0$.

In our approach, we will use the ANNs to approximate the inverse function of Formula (4.7) on the continuation region, without relying on any iterative technique, which will be explained in Section 4.4.2.

IMPLIED DIVIDEND

Many companies pay a share of the stock value to the share holder on the ex-dividend date, which causes the stock price to drop. The option prices are also impacted by the changes in the underlying stock price. Generally, option prices may rise (in the case of the put) or drop (the call) slightly due to the dividend payment. This dividend is called the actual dividend (denoted by δ), while *implied dividend* reflects how the market anticipates future dividend payments of stocks. It is extracted from option prices, and thus a quantity under the risk-neutral measure.

The difference between actual dividends and implied dividends is similar to that between historical and implied volatility. The two parameters reflect different market aspects. An implied dividend may be modeled by means of multiple components, for example,

$$q = \epsilon_r + \delta + b, \tag{4.10}$$

where ϵ_r reflects the difference between the employed and the market interest rate, δ the historical dividend and b the borrowing costs of the underlying asset. Some companies do not pay dividends, but the corresponding options still imply a non-zero dividend, which may reflect the borrowing level of the stock, see [80]. The borrowing costs are seen as a factor that influences the implied dividend as a function of the time or the strike price.

Our approach is to estimate implied dividend and implied volatility at the same time, assuming the implied dividend is not constant over strike prices [89]. In the case of European stock options, the implied dividend can be estimated by the put-call *parity relation* [80],

$$V_{eu}^C(t,S) - V_{eu}^P(t,S) = S(t)e^{-q\tau} - Ke^{-r\tau}, \tag{4.11}$$

so that,

$$q = -\frac{1}{\tau}\log\left(\frac{V_{eu}^C - V_{eu}^P + Ke^{-r\tau}}{S(t)}\right). \tag{4.12}$$

For American options, the put-call parity does not hold. In certain works [80, 81] the authors employ Formula (4.12) to roughly estimate the implied dividend yield for American options. Obviously, this may result in inaccuracies when the put-call parity deviation gets large.

In order to eliminate this error, the authors in [106] take the early-exercise premium (EEP) into account for American options. For example, $V_{am}^C = V_{eu}^C + \Delta V^C$ and $V_{am}^P = V_{eu}^P + \Delta V^P$, where ΔV^C and ΔV^P stand for the American call and put early-exercise premiums, respectively. Given the early-exercise premiums ΔV^C and ΔV^P, Equation (4.11) can be used to calculate the implied dividend yield from the American option prices. A requirement is that the American and European options are available, under the same parameter set. It is however usually not easily possible to deduce the early-exercise premiums from market option prices.

Next we will numerically investigate how early-exercise premiums affect the put-call parity. As mentioned, the American option price can be viewed as the sum of two components, the corresponding European option price and the early-exercise premium. Let $f(S(u);S(t))$ be the transition density function of $S(u)$ conditional on $S(t)$ for $u \geq t$. Then, Equation (4.4) can be rewritten as follows [107],

$$\begin{aligned} V_{am}^P(t,S) =& e^{-r(T-t)}\int_0^K (K - S(T))f(S(T);S(t))dS(T) \\ &+ \int_t^T e^{-r(u-t)}\int_{\Omega_s}(rK - qS(u))f(S(u);S(t))dS(u)du \\ =& V_{eu}^P(t,S) + \Delta V^P, \end{aligned} \tag{4.13}$$

with constant r and q, and where Ω_s is the stopping region. If the holder of an American put chooses to exercise the put option in the case of $S(u)$ being in the stopping region, he/she would gain interest $rKdt$ from the cash received, and lose dividend $qS(t)dt$ from selling the asset. Similarly, the American call price is made up of two components,

$$\begin{aligned} V^C_{am}(t,S) =& e^{-r(T-t)} \int_K^\infty (S(T)-K) f(S(T);S(t)) dS(T) \\ &+ \int_t^T e^{-r(u-t)} \int_{\Omega_s} (qS(u)-rK) f(S(u);S(t)) dS(u) du \\ =& V^C_{eu}(t,S) + \Delta V^C. \end{aligned} \tag{4.14}$$

Equations (4.13) and (4.14) can be substituted into the European put-call parity,

$$(V^C_{am} - \Delta V^C) - (V^P_{am} - \Delta V^P) = S(t)e^{-q\tau} - Ke^{-r\tau},$$

and the deviation from the put-call parity is found to be,

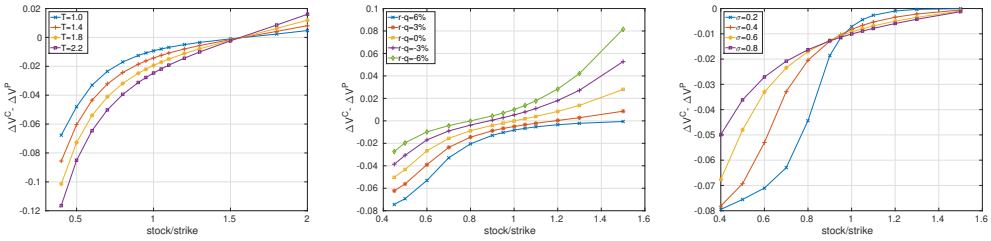
$$\Delta V^C - \Delta V^P = V^C_{am} - V^P_{am} - S(t)e^{-q\tau} + Ke^{-r\tau}. \tag{4.15}$$

We can measure the "deviation" from the European put-call parity relation by EED $:= \Delta V^C - \Delta V^P$, i.e., the difference between two EEPs. The larger the deviation, the more the American and European implied dividend yields will differ. For European options, EED = 0.

We can assess the corresponding early-exercise premium by calculating the difference between the European and American option prices. In the following figures, we will demonstrate how the early-exercise premiums vary with respect to the following factors, maturity time T (Fig. 4.4a), difference between interest rate and dividend yield $r-q$ (Fig. 4.4b), volatility σ (Fig. 4.4c). Roughly speaking, the absolute deviation is monotonically increasing when an option goes deeper into the money (OTM or ITM), based on Figure 4.4. Therefore, significant errors may occur when using the European put-call parity to compute the implied dividends from American options.

Remark. *The early-exercise premium is not observable for most underlying securities, unless both American and European options with the same strike price and time to maturity are available. Empirical studies have been conducted to analyze how the EEP varies in the market using regression techniques, like in [108]. The basic idea is to fit a function for the EEP and other observed factors, i.e.*

$$EEP = \beta_0 + \beta_1(r-q) + \beta_2(T-t) + \beta_3(S/K) + \beta_4(\sigma_{t-1}) + \epsilon$$



(a) (b) (c)

Figure 4.4: Left: EED versus the ratio of stock over strike price for different maturities T, with $\sigma = 0.4$, $r = 0.1$, $q = 0.05$, $K = 1.0$. Middle: EED versus the ratio of stock over strike price for different values $r - q$, with $T = 1.0$, $r = 0.1$, $\sigma = 0.4$, $K = 1.0$. Right: EED versus the ratio of stock over strike price for different volatilities σ, with $T = 1.0$, $r = 0.1$, $q = 0.04$, $K = 1.0$.

4

In the Swedish equity market, for example, the early-exercise premiums for American puts are empirically found to be positive [109], increasing with option moneyness, and decreasing with time to maturity and the underlying asset's volatility. For American-style currency options, it was observed [110] that the early-exercise premiums equal approximately 5% for puts and 4.6% for calls. Our numerical results coincide with these empirical studies of EEP.

We will employ a model-based approach, i.e., the American option Black-Scholes pricing model, including a dividend yield, is inverted in order to extract the implied dividend. American options can be priced as follows,

$$\begin{cases} V_{am}^{C} = BS_{am}(\sigma^*, q^*; S_0, K, t = 0, T, r, \alpha = 1), \\ V_{am}^{P} = BS_{am}(\sigma^*, q^*; S_0, K, t = 0, T, r, \alpha = -1), \end{cases} \tag{4.16}$$

where, BS_{am} is the corresponding pricing model. Assuming the implied volatility and the implied dividend are the same for calls and puts with the same parameter set K, S_0, τ and r, there appears to be a unique solution of the system with two equations and two unknowns. However, this is not always the case, as option prices do not depend on the volatility in the stopping regions. The system of equations (4.16) is usually formulated as a minimization problem and a numerical optimization algorithm is employed to search the solution space. Note that a local search based optimization method will most likely not converge when traversing those early-exercise regions. For that reason, a global searcher is preferred.

4.3. PRICING AMERICAN OPTIONS BY THE COS METHOD

In this section, we explain the pricing of American options by means of the COS method, which forms the basis for the data set that the ANN should learn. First a brief introduction on using the COS method to price American options [111] is given, based on the derivation of pricing European options in Section 2.2.5 of Chapter 2. Then we generalize the above COS method to deal with two separate early-exercise points, which is encountered when pricing American options with negative interest rates.

As aforementioned, the COS method is an efficient numerical technique and provides the derivative information (i.e. the Greeks) essentially without any additional costs. A popular way to approximate the American option value is solving a small series of Bermudan options with different numbers of exercises opportunities, and subsequently applying an extrapolation technique based on these obtained option prices. In other words, for the valuation of American options, we can use a series of Bermudan options with an increasing number of exercise opportunities to approximate the limit. Thus the basic procedure includes computing Bermudan options using COS under the Black-Scholes framework, followed by employing a four-point Richardson extrapolation to compute the price of the American options.

We start with a Bermudan option, where the holder has the right to exercise the contract at pre-specified dates before maturity. With t_0 being initial time, we assume there are M pre-specified exercise dates, and have $\{t_1, \cdots, t_M\}$ as the collection of all exercise dates. The regular time interval reads $\Delta t := (t_m - t_{m-1})$, $t_0 < t_1 < \cdots < t_M = T$. With the help of the risk-neutral valuation, we arrive at the pricing formula for a Bermudan option with M exercise dates, for $m = M, M-1, \ldots, 2$:

$$\begin{cases} c(t_{m-1}, x) &= e^{-r\Delta t} \int_{\mathbb{R}} V(t_m, y) f(y|x) dy, \\ V_{ber}(t_{m-1}, x) &= \max\left(h(t_{m-1}, x), c(t_{m-1}, x)\right), \end{cases} \tag{4.17}$$

followed by

$$V_{ber}(t_0, x) = e^{-r\Delta t} \int_{\mathbb{R}} V(t_1, y) f(y|x) dy. \tag{4.18}$$

where $f(\cdot)$ is the conditional density function, and the state variables x and y are the log-prices and separately defined as

$$x := \log(S(t_{m-1})/K) \quad \text{and} \quad y := \log(S(t_m)/K),$$

where $S(t_m)$ stands for the stock price at time t_m, and K for the strike price. Functions $V(t,x)$, $c(t,x)$ and $h(t,x)$ represent the option value, the continuation value

and the log-price payoff at time t, respectively, for example,

$$h(T,x) = \max[\alpha K(e^x - 1), 0], \quad \alpha = \begin{cases} 1 & \text{for a call,} \\ -1 & \text{for a put.} \end{cases} \tag{4.19}$$

4.3.1. PRICING BERMUDAN OPTIONS

The COS method is generally based on employing a Fourier cosine expansion to approximate the density function on a truncated domain $[a,b]$,

$$f(y|x) \approx \frac{2}{b-a} \sum_{j=0}^{N_{COS}-1}{}' \Re\left(\hat{f}(\Delta t, \frac{j\pi}{b-a}) \exp(-i\frac{ak\pi}{b-a})\right) \cos(k\pi \frac{y-a}{b-a}), \tag{4.20}$$

where $\hat{f}(u;x,t)$ represents the characteristic function of the log-asset price, $x := \log(S(t)/K)$, and the notation $\sum'$ means that the first term in the summation is weighted by one-half.

With the center of the interval $x_0 := \log(S_0/K)$, the integration range, $[a,b]$, is defined as follows,

$$[a,b] := \left[(\xi_1 + x_0) - L_{COS}\sqrt{\xi_2}, (\xi_1 + x_0) + L_{COS}\sqrt{\xi_2}\right] \tag{4.21}$$

where L_{COS} is a user-defined parameter to achieve a certain integration accuracy, and parameters ξ_i represent the corresponding cumulants of the underlying stochastic process, see [111].

We define the following formula,

$$\phi(t,u) := \hat{f}(u; x = 0, t). \tag{4.22}$$

For the Black-Scholes dynamics in Formula (4.1) under the log-asset price, we have

$$\phi(t,u) = e^{iut(r-q-\frac{1}{2}\sigma^2) - \frac{1}{2}\sigma^2 u^2 t},$$

$$\xi_1 = (r - q - \frac{1}{2}\sigma^2)t, \ \xi_2 = \frac{1}{2}\sigma^2 t, \ \xi_4 = 0.$$

The continuation value in (4.17), which resembles a European option between two consecutive exercise dates, can be computed through the COS formula,

$$c(t_{m-1},x) = e^{-r\Delta t} \sum_{k=0}^{N_{COS}-1}{}' \Re\left(\phi\left(\Delta t, \frac{k\pi}{b-a}\right) e^{ik\pi\frac{x-a}{b-a}}\right) \mathcal{V}_k(t_m), \tag{4.23}$$

where N is the number of Fourier cosine items. The $\mathcal{V}_k(t_m)$ terms are the so-called option coefficients, to be computed depending on the early-exercise region.

An early-exercise point, x_m^*, at time t_m, is a point where the continuation value equals the payoff, i.e. $c(t_m, x_m^*) = H(t_m, x_m^*)$. We will first derive the induction formula for $\mathcal{V}_k(t_1)$ for a single early-exercise point, and then extend it to the case of two early-exercise points. The early-exercise point is determined by means of a root-finding algorithm, for example, Newton's method. With x_m^*, the option coefficients $\mathcal{V}_k(t_m)$ can be split into two components: One on the interval $[a, x_m^*]$ and the other on $(x_m^*, b]$ (i.e. on the holding or stopping region),

$$\mathcal{V}_k(t_m) = \begin{cases} \tilde{C}_k(t_m, a, x_m^*) + \tilde{G}_k(x_m^*, b), & \text{for a call,} \\ \tilde{G}_k(a, x_m^*) + \tilde{C}_k(t_m, x_m^*, b), & \text{for a put,} \end{cases} \tag{4.24}$$

for $m = M-1, M-2, \cdots, 1$. When $t_m = t_M$ at the terminal time,

$$\mathcal{V}_k(t_M) = \begin{cases} \tilde{G}_k(0, b), & \text{for a call,} \\ \tilde{G}_k(a, 0), & \text{for a put.} \end{cases} \tag{4.25}$$

With the COS method, we have

$$\tilde{G}_k(x_1, x_2) := \frac{2}{b-a} \int_{x_1}^{x_2} h(t_m, x) \cos\left(k\pi \frac{x-a}{b-a}\right) dx, \tag{4.26}$$

and

$$\tilde{C}_k(t_m, x_1, x_2) := \frac{2}{b-a} \int_{x_1}^{x_2} c(t_m, x) \cos\left(k\pi \frac{x-a}{b-a}\right) dx, \tag{4.27}$$

for $k = 0, 1, \cdots, N-1$ and $m = 1, 2, \cdots, M$. There are analytic solutions for $\tilde{G}_k(x_1, x_2)$ in (4.26), since the payoff function $h(t_m, x)$ is known. The terms $\tilde{C}_k(t_m, x_1, x_2)$ can be computed in $O(N \log_2 N)$ operations under the Black-Scholes dynamics.

At times t_m, $m = 1, 2, \cdots, M$, from Equations (4.17) and (4.23), we obtain an approximation for $c(x, t_m)$. Afterwards, $c(t_m, x)$ is inserted into (4.27). Interchanging summation and integration gives the following formula,

$$\tilde{C}_k(t_m, x_1, x_2) := e^{-r\Delta t} \sum_{j=0}^{N_{COS}-1}{}' \Re\left(\phi\left(\Delta t, \frac{j\pi}{b-a}\right) \mathcal{V}_j(t_{m+1}) \cdot \tilde{H}_{k,j}(x_1, x_2)\right), \tag{4.28}$$

where $\tilde{H}_{k,j}(x_1, x_2)$ is computed in the following integrals,

$$\tilde{H}_{k,j}(x_1, x_2) = \frac{2}{b-a} \int_{x_1}^{x_2} e^{ij\pi \frac{x-a}{b-a}} \cos(k\pi \frac{x-a}{b-a}) dx.$$

With the help of basic calculus, the term $\tilde{H}_{k,j}(x_1, x_2)$ can be further divided into two parts,

$$\tilde{H}_{k,j}(x_1, x_2) = -\frac{i}{\pi}(\tilde{H}_{k,j}^s(x_1, x_2) + \tilde{H}_{k,j}^c(x_1, x_2)).$$

Because $\tilde{H}^s_{k,j}(x_1,x_2) = \tilde{H}^s_{k+1,j+1}(x_1,x_2)$ and $\tilde{H}^c_{k,j}(x_1,x_2) = \tilde{H}^c_{k+1,j-1}(x_1,x_2)$, we get a Toeplitz and Hankel structure in matrices $\tilde{H}_s$ and $\tilde{H}_c$, respectively. Therefore the Fast Fourier Transform can be employed for highly efficient matrix-vector multiplication, and the resulting computational complexity of $\tilde{C}_k(x_1,x_2,t_m)$ is reduced to $O(N\log_2 N)$. In addition, the Greeks of American Black-Scholes option prices can be easily approximated based on Equation (4.23), for example,

$$\text{Vega} \approx \frac{\partial c(t_{m-1},x)}{\partial \sigma} = e^{-r\Delta t} {\sum_{k=0}^{N_{COS}-1}}' \Re\left(\frac{\partial \phi\left(\Delta t, \frac{k\pi}{b-a}\right)}{\partial \sigma} e^{ik\pi\frac{x-a}{b-a}}\right) \mathcal{V}_k(t_m), \tag{4.29}$$

4

Here we generalize this Bermudan COS method to deal with two early-exercise points. Suppose there are at most two early-exercise points $x^*_{m1} < x^*_{m2}$ at time t_m. Thus, we have three intervals while computing $\mathcal{V}_k(t_m)$, that is, $[a, x^*_{m1}]$, $[x^*_{m1}, x^*_{m2}]$, and $[x^*_{m2}, b]$. Taking a Bermudan put as an example, the option value is

$$\mathcal{V}_k(t_m) = \tilde{C}^{(1)}_k(t_m, a, x^*_{m1}) + \tilde{G}_k(x^*_{m1}, x^*_{m2}) + \tilde{C}^{(2)}_k(t_m, x^*_{m2}, b), \tag{4.30}$$

for $m = M-1, M-2, ..., 1$ and $\mathcal{V}_k(t_M) = \tilde{G}_k(a,0)$ when $t_m = t_M$. There are two continuation values $\tilde{C}^{(1)}_k$ and $\tilde{C}^{(2)}_K$ correspondingly, which are computed by Equation (4.28). In such case, two roots x^*_{m1} and x^*_{m2} are computed, with different, smartly selected, starting points for the Newton's method.

4.3.2. PRICING AMERICAN OPTIONS

The extrapolation-based pricing method with the COS computations for Bermudan options to price American options has been described in [112], and a brief description is given in this section.

Let $V_{ber}(M)$ denote the value of a Bermudan option with M exercise dates, considering maturity T and $\Delta t = T/M$ which is a time interval between two consecutive exercise dates, the American option value V_{am} can be approximated by applying the following 4-point Richardson extrapolation scheme,

$$V_{am}(\ell) \approx \frac{\ell}{21}\left(64V_{ber}(2^{\ell+3}) - 56V_{ber}(2^{\ell+2}) + 14V_{ber}(2^{\ell+1}) - V_{ber}(2^{\ell})\right), \tag{4.31}$$

where parameter $\ell \in \mathbb{N}^+$ determines the number of the exercise dates considered for each Bermudan option involved.

Remark. *The binomial trees technique is also a suitable candidate for American option valuation, since it can accurately deal with two early-exercise points when pricing American options. However, the COS method provides faster computation and the Greeks come with no extra cost.*

4.4. METHODOLOGY

Artificial neural networks, ANNs, have been used to approximate the solution of European option pricing models, for instance, under the Black-Scholes and Heston models in Chapter 2. Here, we will use the ANN to address the numerical solution of American options, and in particular use it for computing the implied information. For the inverse problem, when there is one parameter to calibrate, we can employ the ANN to build a mapping from the observed market data to the target parameter (as a unique mapping). When there are multiple parameters to calibrate, like the implied volatility and the implied dividend, the CaNN (Calibration Neural Network) [94] is more flexible to handle the minimization problem. The former case can be viewed as a simplified CaNN which approximates a single model parameter.

4.4.1. ARTIFICIAL NEURAL NETWORKS

It is well-known that neural networks are powerful function approximators, and more details can be found in Section 2.3.1 in Chapter 2. In the previous chapters, the ANN function is used to approximate a single variable. Actually an ANN can have multiple outputs provided the same input, which means we can approximate a pair of American call/put option prices at the same time. Next, we will discuss how the ANNs are used to approximate the inverse or pricing functions.

4.4.2. ANN FOR IMPLIED VOLATILITY

When we focus solely on extracting the implied volatility, the basic technique is to employ the ANN to approximate the inverse function of the American Black-Scholes model on a suitable effective definition domain Ω_h,

$$\begin{aligned} \sigma^* &= BS_{am}^{-1}(V_{am}^{mkt}; S, K, t, T, r, q, \alpha) \\ &\approx \mathrm{NN}(V_{am}^{mkt}; S, K, t, T, r, q, \alpha), \ [V, S, K, t, T, r, q] \in \Omega_h. \end{aligned} \tag{4.32}$$

The ANN is trained based on the above known model variables, which are observable in the market, to approximate the unique target variable σ^*.

DEFINITION DOMAIN SELECTION

The continuation regions are not known initially or are so complicated that there is no analytic formula to describe them. However, the counterpart, the early-exercise regions, can be found implicitly in a data-driven approach. In our method we wish to only train the neural network on the points in the continuation region. Overall, our aim is to find the inverse function of the American-style pricing model on the continuation region, represented by the shaded domain in Figure

4.5. There is an off-line stage, where the continuation regions are being determined, and an on-line stage where we use the problem parameters to compute the implied volatility. We build the mapping function via the ANN in the "irregular" continuation region. The resulting trained ANN solver is typically much faster to determine the implied volatility than an iterative numerical solver.

First, random parameter values are generated as ANN samples in the entire input domain Ω, followed by detecting the parameter samples that are in the early-exercise region Ω_s according to Equation (4.5).

Second, we use a robust version of the COS method [112] to calculate American option values while generating the data set during the off-line ANN stage. With a variety of asset and option parameters, we need to make sure that the COS pricing technique is robust, i.e., it should work under all occurring parameter sets. This might imply a large integration domain within the COS method, and a relatively large number of Fourier terms in the cosine expansion. We generalized the COS method to deal with two separate early-exercise regions, which is encountered when pricing American options with negative interest rate. More details can be found in Section 4.3.

Third, we obtain the approximate continuation region, $\Omega_h = \Omega - \Omega_s$, as shown in Figure 4.5. There are two indicators to detect the samples in the early-exercise region, the difference between the option value and the payoff, and the option's sensitivity Vega in Equation (4.9). We obtain the approximate continuation region, $\Omega_h = \Omega - \Omega_s$, as shown in Figure 4.5. The procedure requires additional computations, but these happen in the off-line stage without affecting the on-line approximation. To control numerical errors, threshold values are prescribed for the two indicators. A threshold ϵ_1 is set for the difference between the payoff function value and the generated option value,

$$V_{am}(t,S) - H(K,S(t)) > \epsilon_1. \tag{4.33}$$

The appropriate training samples for the continuation region are selected based on Formula (4.33). We also set a threshold ϵ_2 for the value of Vega,

$$\text{Vega} > \epsilon_2, \tag{4.34}$$

with $\epsilon_1 \in \mathscr{R}^+$ and $\epsilon_2 \in \mathscr{R}^+$. As early-exercise takes place with options that are ITM, the above two criteria only apply to ITM samples. In principle, Criterion (4.33) is equivalent to Criterion (4.34), but for robustness reasons, both will be enforced to mitigate the influence of numerical errors.

Let $\hat{\Omega}_s$ represent the region where the generated samples do not meet the requirements (4.33) and (4.34), and the remaining region $\hat{\Omega}_h = \Omega - \hat{\Omega}_s$. The effective definition domain is found numerically by taking $\Omega_h \approx \hat{\Omega}_h$.

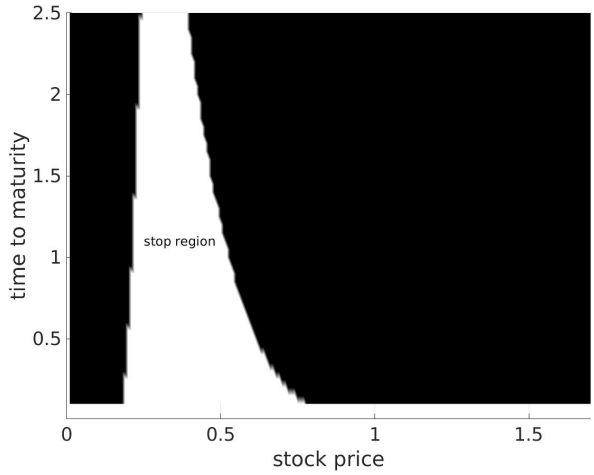


Figure 4.5: Schematic diagram: An example of two continuation regions for an American put. The shaded area represents the holding region, while the white area represents the stopping region. There are two isolated continuation regions. Here the strike price is fixed $K = 1$.

GRADIENT-SQUASHING AND OPTION PRICES

Generally, ANNs are not accurate when functions with steep gradients need to be approximated. Therefore, we adapt the requested output function. To obtain the implied volatility from the option prices, we need to employ the gradient-squashing technique as proposed in [92]. We subtract the intrinsic value from the American option price to obtain the corresponding *time value*. A brief derivation how to compute the intrinsic value of an American option with the dividend yield is given. Taking the put as an example, recall the put-call parity for European options,

$$V_{eu}^{C} - S(t)e^{-q\tau} = V_{eu}^{P} - Ke^{-r\tau}.$$

The following lower bound can then be deduced,

$$V_{eu}^{P}(t,S) = V_{eu}^{C}(t,S) + Ke^{-r\tau} - S(t)e^{-q\tau} \geq Ke^{-r\tau} - S(t)e^{-q\tau}, \tag{4.35}$$

where the right-hand side is called the European option's intrinsic value. As an American option is at least as expensive as its European counterpart, we have

$$V_{am}^{P}(t,S) \geq V_{eu}^{P}(t,S) \geq Ke^{-r\tau} - S(t)e^{-q\tau}. \tag{4.36}$$

Additionally, American option prices should not be worth less than the pay-off function at any time, as for example shown in Figure 4.2, and the time value of an American put is computed by

$$\hat{V}_{am}^{P} = V_{am}^{P}(t,S) - \max(K - S(t), Ke^{-r\tau} - S(t)e^{-q\tau}, 0). \tag{4.37}$$

After that, the gradient-squashing technique [92] is applied as follows,

$$\tilde{V}_{am}^{P} = \log(\hat{V}_{am}^{P}). \tag{4.38}$$

The gradient-squashing technique for computing implied volatility using the ANNs, means taking the logarithm of the time value to obtain a quantity that can be well approximated with ANNs, because its gradient is not too steep to approximate accurately.

4.4.3. DETERMINING IMPLIED DIVIDEND AND IMPLIED VOLATILITY

When the American option implied dividend yield is unknown, we will determine both implied volatility and implied dividend simultaneously by means of the CaNN calibration methodology. We assume the implied volatility and the implied dividend are identical for American calls and puts with the same K, S_0, T, t and r values,

$$\begin{cases} V_{am}^{C,mkt} - BS_{am}(\sigma^*, q^*; S_0, K, t=0, T, r, \alpha = 1) = 0, \\ V_{am}^{P,mkt} - BS_{am}(\sigma^*, q^*; S_0, K, t=0, T, r, \alpha = -1) = 0, \end{cases} \tag{4.39}$$

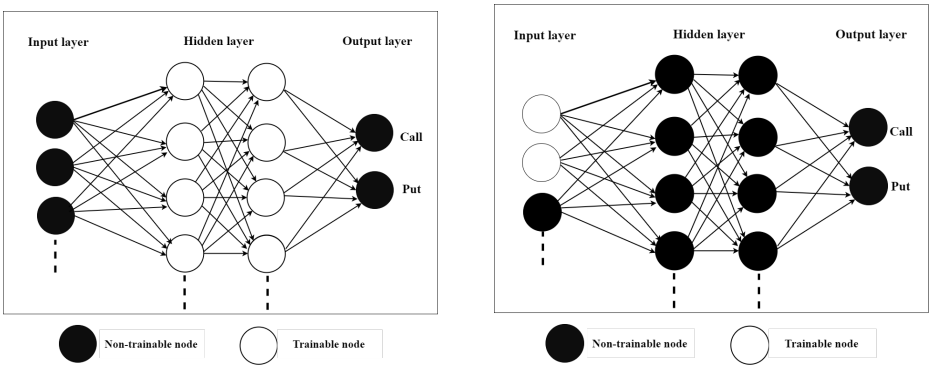
so that there are two unknown parameters to calibrate, implied volatility σ^* and the implied dividend yield q^*, given a pair of American option prices, $V_{am}^{C,mkt}$ and $V_{am}^{P,mkt}$. The above system is reformulated as a minimization problem,

$$\underset{\sigma^* \in R^+, q^* \in R}{\operatorname{argmin}} \; (BS_{am}(\sigma^*, q^*; \alpha = 1) - V_{am}^{C,mkt})^2 + (BS_{am}(\sigma^*, q^*; \alpha = -1) - V_{am}^{P,mkt})^2. \tag{4.40}$$

We adapt a fast, generic and robust calibration framework, the CaNN (Calibration Neural Networks) developed in [94]. The basic idea of the methodology is to convert the calibration of model parameters into an estimation of the neural network's hidden units. The reason for this is that model calibration and training ANNs (here supervised learning) can be reduced to solving an optimization problem according to Formula (4.40) and (2.24) (see Chapter 2). It enables parallel GPU computing to speed up the computations, which enables us to employ a global optimization technique to search the solution space. The gradient-free optimization algorithm, Differential Evolution, typically does not get stuck in local minima or in the stopping region. Another benefit of DE is that it is an inherently parallel technique where populations of possible optimal solutions can be evaluated simultaneously within the ANN.

There is a calibration (the backward pass) in the CaNN, in addition to the training and testing stages. The training is to determine suitable weights and

biases in the hidden layers to map model input to output, while the calibration estimates the model input parameters to optimally match a given output. We will view the different stages as one framework, and just change the learnable units between the original layers (i.e. the hidden, output and input layers) within the different stages. More specifically, the CaNN consists of two passes. The forward pass, including the training and testing stages, approximates the American Black-Scholes prices. We have developed one neural network providing two output values in the forward pass, the American call and the put prices, as illustrated in Figure 4.6. The backward pass, on the other hand, aims to find the two parameters, (σ^*, q^*), to match the two observed American option prices, $V_{am}^{P,mkt}$ and $V_{am}^{C,mkt}$, with strike price K, maturity time T, spot price S_0, interest rate r.



(a) Training (b) Calibration

Figure 4.6: Left: In the forward pass of the CaNN, the output layer gives us two option prices. Right: During the calibration, the CaNN estimates the two parameters, implied volatility and implied dividend, in the original input layer.

This is different from the network in [94], where one neural network corresponds to one output quantity. Therefore Equation (4.40) is written as an objective function for the model calibration, as follows,

$$\arg\min_{\sigma^*,q^*} (\mathrm{NN}(\sigma^*, q^*; \alpha = 1) - V_{am}^{C,mkt})^2 + (\mathrm{NN}(\sigma^*, q^*; \alpha = -1) - V_{am}^{P,mkt})^2, \quad (4.41)$$

where $\sigma^* \in R^+, q^* \in R$. Formula 4.41 is used as the loss function for the backward pass in the CaNN. The (group-based) DE global optimization can be implemented in an efficient way together with ANNs, see Figure 3.3 in Chapter 3, on a central processing unit (CPU) or a graphics processing unit (GPU). In such way, the for-loop is replaced by an efficient matrix multiplication, and searching the implied information globally can be completed efficiently.

Remark. *Because of a generic calibration framework, the CaNN can easily deal with more complex situations, for example, the objective function* (4.41) *including more than a pair of American price quotes which share the same implied dividend, for example, as in the work [102].*

4.4.4. The ANN configuration

Our chosen ANN architecture in the forward pass constitutes four hidden layers and two parallel output layers. Some particularly useful operations in deep neural networks, e.g. dropout and batch normalization, did not bring any significant benefits to our ANN. The proposed configuration has been demonstrated to be able to fit the pricing model with acceptable accuracy in [94]. We choose the activation function Softplus here, i.e.

$$\varphi(x) = \log(1 + e^x),$$

where x is the input. The smooth derivative of Softplus fits well to the smoothness of the pricing function, especially in the continuation region. According to the universal approximation theorem, a one-layer based ANN can be used to approximate any continuous function to any desired precision, but with the rate which linearly depends on the number of neurons involved. The depth of the ANN (i.e. the number of hidden layers) can increase the function's representation accuracy exponentially, but deep ANNs are difficult to implement in parallel (e.g. a current hidden layer has to wait for output signals of a previous one), resulting in long computation time. Considering the approximation power and the computation efficiency, we choose four hidden layers and 200 neurons in each layer. Both ANNs in the forward and backward passes of the CaNN will make use of the hyper-parameters that are shown in Table 4.1. In the backward pass, the DE algorithm, a gradient-free global optimizer, replaces Adam. These values have shown to result in a robust neural network which converges well for a variety of problem parameters.

4.5. Numerical results

This section presents numerical experiments for using the ANN to extract implied information from American options. We begin with the simplified CaNN, focusing on the implied volatility, and later employ the CaNN to extract implied volatility and dividend. Note that we use the COS method described in Section 4.3 to compute American option prices for the training data set and for the simulated market data in this section.

Table 4.1: The ANN configuration.

Hyper-parameters	Options
Hidden layers	4
Neurons (each layer)	200
Activation	Softplus
Initialization	Glorot_uniform
Optimizer	Adam
Batch size	1024

4.5.1. COMPUTING IMPLIED VOLATILITY

In this section, we approximate only implied volatility. We use one forward pass to approximate the inverse function, from an American option price to its corresponding implied volatility assuming the other parameters to be known.

Without loss of generality, we use a fixed spot price $S_0 = 1.0$. Then, the input for the ANN is made up of five parameters $\{\log(\hat{V}^P_{am}), K, r, q, \tau\}$. The two thresholds in Equations (4.33) and (4.34) are $\epsilon_1 = 0.0001$ and $\epsilon_2 = 0.001$ for the data set.

Table 4.2: Train dataset for American options under the Black-Scholes model; The spot price $S_0 = 1$ is fixed. Here we use τ instead of the two variables T and $t = 0$. The upper bound of American put price is 1.2. LHS stands for Latin Hypercube Sampling.

ANN	Parameters	Value range	Employed method
ANN Input	Strike, K	[0.6, 1.4]	LHS
	Time value, $\log(\hat{V}^P_{am})$	$(-11.51, -0.24)$	COS
	Time to maturity, τ	[0.05, 3.0]	LHS
	Interest rate, r	[-0.05, 0.1]	LHS
	Dividend yield, q	[-0.05, 0.1]	LHS
ANN output	Implied volatility, σ^*	$(0.01, 1.05)$	LHS

The ANN is trained with American put options to learn the weights of the ANN-based solver for the computation of the implied volatility from American options. Similarly to Chapters 2 and 3, the following measures are used,

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \quad \text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|, \quad \text{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_i - \hat{y}_i|}{y_i},$$

where y represents the true values of American option prices, and $\hat{y}$ represents the approximated values, with n being the number of involved samples. Dur-

ing training, the MSE is used to find the weights and biases, while the other two measures MAE and MAPE are monitored. The goodness of fit, R^2, is also provided, which describes the closeness between the predicted values and the true values. By using different measures we evaluate the quality of approximation from different angles.

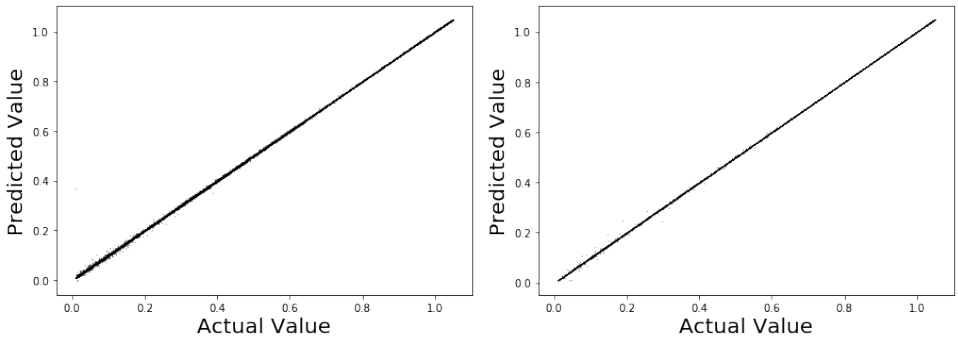
After the model input parameters are sampled (here by LHS) over the specific domain, the COS method is used to solve the corresponding American Black-Scholes pricing model, resulting in the data collection $\{S_0, K, \tau, r, q, \sigma, V_{am}\}$, where σ is considered the ground-truth value. Afterwards, the variable σ is placed into the output layer of the ANN, as the implied volatility $\sigma^* \equiv \sigma$ for the data collection. Meanwhile, the other variables in the collection are included in the input layer of the ANN, and more details are in Table 4.2. The validation samples help avoid over-fitting during training the ANN. The test samples, which the ANN did not encounter during training, are subsequently used to evaluate the generalization performance of the trained ANN. There are around one million samples, with 80% being used as training, 10% as validation, 10% as test samples. The learning rate is halved every 400 epochs during training. After 4000 epochs, the training and validation losses have converged.

Table 4.3 and Figure 4.7 present the performance of the trained ANN. The test performance is close to the train performance, suggesting that the trained ANN generalizes well for unseen data, as shown in Table 4.3. The ANN predicted implied volatility values approximate the true values accurately for both the train and test datasets, as is indicated by the R^2 measure in Figure 4.7. Moreover, the online prediction stage for the American option implied volatility, requiring only the evaluation of the trained ANN, is much faster than traditional iterative root-finding algorithms.

It is observed that the trained model performance tends to decrease when the pricing model parameters gets close to the upper or lower bounds of the values in Table 4.2. In other words, outliers are most likely to appear near the boundary. Thus the training data set is recommended to have a wider parameter range than the test range of interest.

Table 4.3: Multiple measures are used to evaluate the performance.

-	MSE	MAE	MAPE	R^2
Training	$4.33 \cdot 10^{-7}$	$2.44 \cdot 10^{-4}$	$1.11 \cdot 10^{-3}$	0.999994
Testing	$4.60 \cdot 10^{-7}$	$2.51 \cdot 10^{-4}$	$1.15 \cdot 10^{-3}$	0.999993



(a) Training (b) Testing

Figure 4.7: Predicted versus ground-truth implied volatility value on the test data set. Here the actual values are prescribed. Left: R^2=0.999994; Right: R^2=0.999993

4.5.2. COMPUTING IMPLIED INFORMATION

Next we will approximate the implied volatility and the implied dividend yield simultaneously from the observed American option prices using the CaNN. The CaNN is based on a forward and a backward pass, that are implemented in a sequential way.

Here we extend the original CaNN by using one forward pass to approximate two American option prices, a put and a call value. The input for the neural network in the forward pass consists of $(S_0, K, r, q, \sigma, \tau)$, and the output compromises a pair of American prices, that is $(\tilde{V}^P_{am}, \tilde{V}^C_{am})$. As the neural network gives us two output variables, the loss function of the forward pass includes two components,

$$\text{MSE} = \frac{1}{2n}\sum_{i=1}^{n}\{(\tilde{V}^P_{am,i} - V^{P,mod}_{am,i})^2 + (\tilde{V}^C_{am,i} - V^{C,mod}_{am,i})^2\} \tag{4.42}$$

where $V^{P,mod}_{am}$ and $V^{C,mod}_{am}$ stand for the American put and call prices, respectively, that are generated by the American option Black-Scholes model. This rule also applies to MAE and MAPE. There are four hidden layers with 200 neurons each layer, as shown in Table 4.1. The total number of hidden parameters is 122,202, and the loss function, Equation (4.42), is used to update the hidden layers of the CaNN during the training stage. The training data set is constructed according to the parameter ranges in Table 4.4, where the COS method is used to compute American option prices for both the training data set and the simulated market data. Based on a good training phase, the performance of the CaNN's forward pass is presented in Table 4.5. The results, for both calls and puts, are highly satisfactory, achieving very good levels of precision in all the considered

measures.

Table 4.4: Training data set for the forward pass. We fix $S_0 = 1$, and sample strike prices K to generate different moneyness levels. The total number of the data samples is nearly one million, with 80% training, 10% validation, 10% test samples.

ANN	Parameters	Value range	Method
Forward input	Strike, K	[0.45, 1.55]	LHS
	Time to maturity, τ	[0.08, 3.05]	LHS
	Risk-free rate, r	[-0.1, 0.25]	LHS
	Dividend yield, q	[-0.1, 0.25]	LHS
	Implied volatility, σ	(0.01, 1.05)	LHS
Forward output	American put, V_{am}^{P}	(0, 1.8)	COS
	American call, V_{am}^{C}	(0, 1.2)	COS

Table 4.5: The performance of the CaNN forward pass with two outputs.

–	Option	MSE	MAE	MAPE	R^2
Training	call	1.40×10^{-7}	3.00×10^{-4}	1.25×10^{-3}	0.9999965
	put	2.54×10^{-7}	4.24×10^{-4}	1.64×10^{-3}	0.9999959
Testing	call	1.43×10^{-7}	3.02×10^{-4}	1.27×10^{-3}	0.9999964
	put	2.55×10^{-7}	4.26×10^{-4}	1.64×10^{-3}	0.9999959

After performing the forward pass, in the CaNN's backward (calibration) pass the implied parameters are determined, in our current setting. Supposing each option quote in the market includes American call and put prices, the idea behind the backward pass is to determine two parameters (σ^*, q^*) within the American Black-Scholes model to best match the pair of market option prices, given the interest rate r, maturity time T, strike price K, and spot price S_0. The objective function for the calibration procedure is found in Formula (4.41), which is equivalent to Criterion (4.42) in the case of $n = 1$. In practice, the market price is taken to be the mid-price of the bid and ask prices.

Remark. *The objective function, under the CaNN, can also be defined differently taking into account the bid-ask spread, the Delta (the sensitivity of an option's value to the underlying asset price) weighting, and other factors. This is however out of our scope here.*

In order to evaluate the approach, we prescribe model parameters and investigate how accurately the CaNN can recover them. Table 4.6 presents a set of

examples, including many different scenarios, e.g. ITM and OTM scenario's, are considered. The results in Table 4.6 suggest that the CaNN can recover the implied volatility and implied dividend highly accurately from our "artificial market option data". Even when interest rates and dividend yields are negative, the CaNN recovers the true values without any stalling of the convergence. The method's robustness may be attributed to the robust numerical solver generating accurate option prices for a wide range of model parameters, the designed neural network providing sufficient approximation capacity, and the gradient-free optimizer (i.e. DE) to globally search the solution space.

Table 4.6: Examples of using CaNN to extract implied volatility and implied dividend. † represents the prescribed values, $*$ represents the calibrated values.

K/S_0	T	r	$\sigma^\dagger$	$q^\dagger$	C_{am}^{mkt}	P_{am}^{mkt}	σ^*	q^*
1.0	0.5	-0.04	0.1	0.06	0.0146	0.0597	0.099	0.059
1.1	0.5	-0.04	0.2	-0.06	0.0255	0.1181	0.198	-0.061
1.0	0.75	0.0	0.3	-0.02	0.1119	0.0976	0.300	-0.020
1.2	1.0	-0.04	0.4	0.08	0.0603	0.3810	0.40	0.080
0.8	1.0	0.02	0.3	0.02	0.2322	0.03472	0.299	0.020
0.7	1.25	0.0	0.4	-0.04	0.3886	0.0378	0.399	-0.040

For DE, a search interval is required for each parameter, and we provide $q \in [-0.08, 0.1]$ and $\sigma^* \in (0, 1.0)$. During the mutation operation in DE, the population size of each generation is taken as a small number, here 10. We choose 'best1bin' as the mutation strategy, that is, the best candidate of the previous generation enters the mutation. During the crossover stage, the crossover possibility is set to 0.7. During the selection stage, all new trial candidates with the objective function can be processed in parallel, and the DE convergence tolerance is set to 0.01. As the number of calibration parameters is only two, the computation time on a CPU is around 0.37 seconds using the sequential DE and is less than 0.1 second using the parallel version of the DE method.

Furthermore, a systemic test is conducted to assess the averaged performance over a large number of cases. We generate equally-spaced samples over a certain interval according to Table 4.7, but remove the samples that are connected to early-exercise region option prices. The experiment ends up with 9271 test cases. The results in Table 4.8 suggest that the proposed approach performs very well under a wide variety of option market conditions at a reduced computational cost. The calibration speed is due to the efficient forward pass and the parallel, gradient-free DE optimizer. Basically the forward pass serves as a fast numerical solver for the American pricing model. Additionally, with two output prices, the

forward pass requires half of the computation cost.

Table 4.7: The parameter range for the systemic experiment.

Parameter	interval	step	number
σ	[0.1, 0.45]	0.05	8
q	[-0.06, 0.08]	0.02	8
K	[0.7, 1.2]	0.1	6
τ	[0.5, 1.5]	0.25	5
r	[-0.04, 0.06]	0.02	6
P_{am}^{mkt}	[0.7, 1.2]	-	9271
C_{am}^{mkt}	[0.7, 1.2]	-	9271

Table 4.8: With a CPU (Intel i5) and a GPU(NVIDIA Tesla P100), the averaged performance of the CaNN estimating implied volatility and implied dividend based on 9271 different test cases. The averaged number of function evaluations is 1060.

Absolute deviation		Computational cost	
$\lvert\sigma^{\dagger}-\sigma^{*}\rvert$	6.65×10^{-3}	**CPU** time (seconds)	0.08
$\lvert q^{\dagger}-q^{*}\rvert$	8.56×10^{-4}	**GPU** time (seconds)	0.04

4.6. Conclusion

We studied the problem of pricing American options and extended a data-driven machine learning method to extract the implied volatility and implied dividend yield from observed market American option prices in a fast and robust way.

For computing the American implied volatility, we explained that the domain of the inverse function should be equivalent to the continuation regions of the American options. The ANN-based approach builds an approximating function and addresses complex boundaries of the definition domain, by means of the different off-line and on-line stages. More specifically, we used two conditions to classify the random data samples in the domain in the off-line stage. The definition domain was represented by data points which lie in the continuation regions. Subsequently, a neural network was trained on those samples to approximate the inverse function. This data-driven approach avoids an iterative algorithm which may suffer from convergence problems. Due to the off-line definition of the domain, our approach also successfully dealt with negative interest

rates and dividend yields, where two early-exercise regions may appear. In short, the offline-online decoupling brings much flexibility.

Furthermore, we presented a method for finding simultaneously implied dividend and implied volatility from American options using a calibration approach. The CaNN, which consists of an efficient solver and a fast global optimizer, is employed to carry out the calibration procedure. As a result, the early-exercise premiums, which the European option put-call parity relation fails to deal with, are handled successfully. The parallel global optimizer prevents the CaNN from stopping in the early-exercise regions and allows to achieve a good quality solution in a short amount of time. The numerical experiments demonstrate that the CaNN is able to accurately extract multiple pieces of implied information from American options. A continuous dividend yield is considered in this chapter, and it should be feasible to extend the approach to deal with time-dependent or discrete dividends.

5

THE SEVEN-LEAGUE SCHEME

In this chapter[1]*, we present another application of the supervised learning methodology. We propose an accurate data-driven numerical scheme to solve Stochastic Differential Equations (SDEs), by taking large time steps. The SDE discretization is built up by means of a polynomial chaos expansion method, on the basis of accurately determined stochastic collocation (SC) points. By employing an artificial neural network to learn these SC points, we can perform Monte Carlo simulations with large time steps. Error analysis confirms that this data-driven scheme results in accurate SDE solutions in the sense of strong convergence, provided the learning methodology is robust and accurate. With a variant method called the compression-decompression collocation and interpolation technique, we can drastically reduce the number of neural network functions that have to be learned, so that computational speed is enhanced. Numerical experiments confirm a high-quality strong convergence error when using large time steps, and the novel scheme outperforms some classical numerical SDE discretizations. Some applications, here in financial option valuation, are also presented.*

5.1. INTRODUCTION

The highly successful deep learning paradigm [113] receives a lot of attention in science and engineering, in many different forms and flavors. As an example of one such flavor, so-called Physics-Informed Neural Networks (PiNN) in [114], combining physical and mathematical insights with the machine learning methodology, have now successfully entered the classical field of numerically solving or-

This chapter is based on the article 'The Seven-League Scheme: Deep learning for large time step Monte Carlo simulations of stochastic differential equations', submitted for publication, 2020.

[1] *Note that, in this chapter we may use the notation different from that of the previous chapters.*

dinary (ODEs) and Partial Differential Equations (PDEs) [22, 115]. Recent progress includes universal differential equations [116], which incorporates machine-learnable structures for scientific computing. The specific aim with PiNN is then to either speed up the solution process or to solve high-dimensional problems that are not easily handled by the traditional numerical methods. In the spirit of PiNN, in this chapter we will develop a highly accurate numerical discretization scheme for stochastic differential equations (SDEs), which is based on taking possibly large discrete time steps. We "learn" to take large time steps, with the help of the Stochastic Collocation Monte Carlo sampler (SCMC) proposed by [117], and by using an artificial neural network (ANN) to approximate target functions [see, for example, 118].

Stochastic differential equations are widely used to describe uncertain phenomena, in physics, finance, epidemics, amongst others, as a means to model and quantify uncertainty. This type of differential equation therefore contains terms that are stochastic processes. As a result, the corresponding solution is also a stochastic process.

Numerical approximation of the solution to an SDE is unavoidable, as an analytic solution is typically not available. The most commonly known technique to solve SDEs is based on Monte Carlo (MC) simulation, for which the SDE first needs to be discretized. Our focus lies on this time discretization, which we develop in a data-driven way.

Basically, there are two ways to measure the convergence rate of discrete solutions to SDEs, by means of the approximation to the sample path or by approximation to the corresponding distribution. This way, strong and weak convergence of a numerical SDE solution have respectively been defined [see 119]. Weak convergence, the convergence in distributional sense, is often addressed in the literature. Moment-matching, for example, is a basic technique to improve weak convergence. Strong, path-wise, convergence is particularly challenging, and requires accurate *conditional distributions*. There are natural approaches to improve strong convergence properties, i.e. by adding higher order terms or by using finer time grids. However, these are nontrivial and costly, especially when considering multi-dimensional SDEs.

We aim to develop highly accurate numerical schemes by means of deep learning, for which the strong error of the discretization does not depend on the size of the simulation time step. For this, we will employ the SCMC method as an efficient approach for approximating (conditional) distribution functions. The distribution function of interest is then expanded as a polynomial in terms of a random variable which is *cheap to sample from* at given collocation points, and interpolation takes place between these points. The resulting big time steps discretization, in which the SCMC methodology is combined with deep learning, is

called *the Seven-League scheme* [2], and we abbreviate it by *the 7L scheme* here.

There are different reasons to learn stochastic collocation points instead of the sample paths directly. Stochastic collocation points have a specific physical meaning, which makes the data-driven scheme explainable. More importantly, Monte Carlo sample paths are random, while collocation points are deterministic (i.e., representing key features of a probability distribution), which simplifies learning and using neural networks. Moreover, the SCMC methodology enables us to highly efficiently generate samples from a complex distribution.

We list some related work in the area of using deep learning for discretization schemes. With the help of learning, the authors of [120], for example, derived data-driven high-order discretizations for PDEs. When the depth of a neural network is viewed as a virtual time-wise direction, the dynamics of deep neural networks can be described by ODEs [121]. The papers [121, 122] developed a continuous-time generative model to turn a simple distribution into a complex one, which is then employed to obtain the solution of an SDE in [123]. Generative Adversarial Networks, GANs [124], have been used to generate high-resolution images based on low-resolution versions, and this technique has been adopted within Computational Fluid Dynamics [125], where high-fidelity PDE solutions have been obtained using GANs. Recently stochastic PDEs have been addressed using GANs, for example, see [126].

The remainder of this chapter is organized as follows. In Section 5.2, SDEs, their discretization, stochastic collocation and the connection between SDE discretizations and the SCMC method are introduced. In Section 5.3, the data-driven methodology is explained to address large time step simulation, i.e. the *7L scheme*, for SDEs. ANNs will be used as function approximators to learn the stochastic (conditional) collocation points. A brief description of their details is placed in Section 5.3.3. In Section 5.4, we introduce a decompression-compression technique to accelerate the computation. This latter efficient variant is named the *7L-CDC scheme* (i.e., seven-league compression-decompression scheme). Section 5.5 presents numerical experiments to show the performance of the proposed approach. Furthermore, the corresponding error is analyzed. Section 5.6 concludes.

5.2. STOCHASTIC DIFFERENTIAL EQUATIONS AND STOCHASTIC COLLOCATION

We first describe the basic, well-known SDE setting, and explain our notation.

[2] With seven-league boots, we are marching through the time-wise direction, see also `https://en.wikipedia.org/wiki/Seven-league_boot`.

5.2.1. SDE BASICS

We work with a real-valued random variable $Y(t)$, defined on the probability space $(\Omega, \Sigma, \mathbb{P})$ with filtration $\mathscr{F}_{t\in[0,T]}$, sample space Ω, σ-algebra Σ and probability measure $\mathbb{P}$. For the time evolution of $Y(t)$, consider the generic scalar Itô SDE,

$$\mathrm{d}Y(t) = a(t, Y(t), \boldsymbol{\theta})\mathrm{d}t + b(t, Y(t), \boldsymbol{\theta})\mathrm{d}W(t), \quad 0 \le t \le T, \tag{5.1}$$

with the drift term $a(t, Y(t), \boldsymbol{\theta})$, the diffusion term $b(t, Y(t), \boldsymbol{\theta})$, model parameters $\boldsymbol{\theta}$, Wiener process $W(t)$, and given initial value $Y_0 := Y(t = 0)$. When the drift and diffusion terms satisfy some regularity conditions (e.g., the global Lipschitz continuity [127, p.289]), existence and uniqueness of the solution of (5.1) are guaranteed. The cumulative distribution function of $Y(t)$, $t \in [0, T]$, $F_{Y(t)}(\cdot)$, is available and the corresponding density function, evolving over time, is described by the Fokker-Planck equation [128].

With a discretization in time interval $[0, T]$, $t_i = i \cdot T/N$, $i = 0, \ldots N$, with equidistant time step $\Delta t = t_{i+1} - t_i$, the discrete random variable at time t_i is denoted by $Y(t_i)$. Traditional numerical schemes have been designed based on Itô's lemma, in a similar fashion as the Taylor expansion is used to discretize deterministic ODEs and PDEs. The basic discretization, for each Monte Carlo path, is the Euler-Maruyama scheme [119], which reads,

$$\hat{Y}_{i+1} = \hat{Y}_i + a(t_i, \hat{Y}_i, \boldsymbol{\theta})\Delta t + b(t_i, \hat{Y}_i, \boldsymbol{\theta})\sqrt{\Delta t}\hat{X}_{i+1}, \tag{5.2}$$

where $\hat{Y}_{i+1} := \hat{Y}(t_{i+1})$ is a realization (i.e., a number) from random variable $\tilde{Y}(t_{i+1})$, which represents the numerical approximation to exact solution $Y(t_{i+1})$ at time point t_{i+1}, and a realization $\hat{X}_{i+1}$ is drawn from the random variable X, which here follows the standard normal distribution $\mathcal{N}(0, 1)$. Moreover, $\check{Y}(t_i)$ (a number) will be used as the notation for a realization of $Y(t_i)$.

In addition, the Milstein discretization [129] reads,

$$\hat{Y}_{i+1} = \hat{Y}_i + a(t_i, \hat{Y}_i, \boldsymbol{\theta})\Delta t + b(t_i, \hat{Y}_i, \boldsymbol{\theta})\sqrt{\Delta t}\hat{X}_{i+1} + \frac{1}{2}b'(t_i, \hat{Y}_i, \boldsymbol{\theta})b(t_i, \hat{Y}_i, \boldsymbol{\theta})\Delta t(\hat{X}_{i+1}^2 - 1), \tag{5.3}$$

where $b'(t_i, \cdot, \boldsymbol{\theta})$ represents the derivative with respect to $\hat{Y}$ of $b(\cdot, \hat{Y}, \boldsymbol{\theta})$. When the drift and diffusion terms are independent of time t, the SDE is called time-invariant.

Two error convergence criteria are commonly used to measure the SDE discretization accuracy, that is, the convergence in the weak and strong sense. Strong convergence, which is of our interest here, is defined as follows.

Definition 1. *Let $Y(t_i)$ be the exact solution of an SDE at time t_i, its discrete approximation $\tilde{Y}(t_i)$ with time step $\Delta t \in \mathbb{R}^+$ converges in the strong sense, with order*

$\beta_s \in \mathbb{R}^+$, if there exists a constant K such that

$$\mathbb{E}|\tilde{Y}(t_i) - Y(t_i)| \leq K(\Delta t)^{\beta_s}. \tag{5.4}$$

It is well-known that the Euler-Maruyama scheme (5.2) has strong convergence $\beta_s = 0.5$, while the Milstein scheme (5.3) has $\beta_s = 1.0$. When deriving high order schemes for SDEs, the rules of Itô calculus must be respected [119]. As a result, there will be eight terms in a Taylor SDE scheme with $\beta_s = 1.5$, and twelve with $\beta_s = 2.0$, and the computational complexity increases. As a consequence, higher order schemes are involved and somewhat expensive. Convergence of the numerical solution for $\Delta t \to 0$ is guaranteed, but the computational costs increase significantly to achieve accurate solutions.

The generic form of the above mentioned numerical schemes to solve the Itô SDE is as follows,

$$\hat{Y}_{i+1}|\hat{Y}_i = \sum_{j=0}^{m-1} \alpha_j \hat{X}_{i+1}^j, \tag{5.5}$$

where m represents the number of polynomial terms, the coefficients α_j are predefined and equation-dependent. For example, for the Euler-Maruyama scheme (5.2), with $m = 2$, we have

$$\begin{cases} \alpha_0 = \hat{Y}_i + a(t_i, \hat{Y}_i, \boldsymbol{\theta})\Delta t, \\ \alpha_1 = b(t_i, \hat{Y}_i, \boldsymbol{\theta})\sqrt{\Delta t}, \end{cases} \tag{5.6}$$

while for the Milstein scheme, with $m = 3$, it follows that

$$\begin{cases} \alpha_0 = \hat{Y}_i + a(t_i, \hat{Y}_i, \boldsymbol{\theta})\Delta t + \frac{1}{2}b'(t_i, \hat{Y}_i, \boldsymbol{\theta})b(t_i, \hat{Y}_i, \boldsymbol{\theta}), \\ \alpha_1 = b(t_i, \hat{Y}_i, \boldsymbol{\theta})\sqrt{\Delta t}, \\ \alpha_2 = \frac{1}{2}b'(t_i, \hat{Y}_i, \boldsymbol{\theta})b(t_i, \hat{Y}_i, \boldsymbol{\theta}). \end{cases} \tag{5.7}$$

With these explicit coefficients we arrive at the probability distribution of the random variable,

$$Y(t_{i+1})|Y(t_i) \approx \tilde{Y}(t_{i+1})|\tilde{Y}(t_i) \overset{d}{=} \sum_{j=0}^{m-1} \alpha_j X^j. \tag{5.8}$$

These discrete SDE schemes are based on a series of transformations of the previous realization to approximate the conditional distribution,

$$\mathbb{P}\big[Y(t+\Delta t) < y|Y(t)\big] = F_{Y(t+\Delta t)|Y(t)}(y) \approx F_{\tilde{Y}(t+\Delta t)|\tilde{Y}(t)}(y). \tag{5.9}$$

A numerical scheme is thus essentially based on conditional sampling of $Y(t+\Delta t)|Y(t)$. The Euler-Maruyama scheme draws from a normal distribution, with a specific mean and variance, to approximate the distribution in the next time point, while the Milstein scheme combines a normal and a chi-squared distribution. Similarly, we can derive the stochastic collocation methods.

5.2.2. STOCHASTIC COLLOCATION METHOD

Let's assume two random variables, Y and X, where the latter one is cheaper to sample from (e.g., X is a Gaussian random variable). These two scalar random variables are connected, via,

$$F_Y(Y) \overset{d}{=} U \overset{d}{=} F_X(X), \tag{5.10}$$

where $U \sim \mathcal{U}([0,1])$ is a uniformly distributed random variable, $F_Y(\bar{y}) := \mathbb{P}[Y \leq \bar{y}]$ and $F_X(\bar{x}) := \mathbb{P}[X \leq \bar{x}]$ are cumulative distribution functions (CDF). Note that $F_X(X)$ and $F_Y(Y)$ are random variables following the same uniform distribution. $F_Y(\bar{y}_n)$ and $F_X(\bar{x}_n)$ are supposed to be strictly increasing functions, so that the following inversion holds true,

$$\bar{y}_n = F_Y^{-1}(F_X(\bar{x}_n)) =: g(\bar{x}_n). \tag{5.11}$$

where $\bar{y}_n$ and $\bar{x}_n$ are samples (numbers) from Y and X, respectively. The mapping function, $g(\cdot) = F_Y^{-1}(F_X(\cdot))$, connects the two random variables and guarantees that $F_X(\bar{x}_n)$ equals $F_Y(g(\bar{x}_n))$, in distributional sense and also element-wise. The mapping function should be approximated, i.e., $g(\bar{x}_n) \approx g_m(\bar{x}_n)$, by a function which is cheap. When function $g_m(\cdot)$ is available, we may generate "expensive" samples, $\bar{y}_n$ from Y, by using the cheaper random samples $\bar{x}_n$ from X.

The Stochastic Collocation Monte Carlo method (SCMC) developed in [117] aims to find an accurate mapping function $g(\cdot)$ in an efficient way. The basic idea is to employ Equation (5.11) at specific collocation points and approximate the function $g(\cdot)$ by a suitable monotonic interpolation between these points. This procedure, see Algorithm I, reduces the number of expensive inversions $F_Y^{-1}(\cdot)$ to obtain many samples from $Y(\cdot)$.

The SCMC method parameterizes the distribution function by imposing probability constraints at the given collocation points. Taking the Lagrange interpolation as an example, we can expand function $g_m(\cdot)$ in the form of polynomial chaos,

$$Y \approx g_m(X) = \sum_{j=0}^{m-1} \hat{\alpha}_j X^j = \hat{\alpha}_0 + \hat{\alpha}_1 X + \ldots + \hat{\alpha}_{m-1} X^{m-1}. \tag{5.12}$$

Monotonicity of interpolation is an important requirement, particularly when dealing with peaked probability distributions.

Algorithm I: SCMC Method

Taking an interpolation function of degree $m-1$ (with $m \geq 2$, as we need at least two collocation points), as an example, the following steps need to be performed:

1. Calculate CDF $F_X(x_j)$ on the points $(x_1, x_2, ..., x_m)$, that are obtained, for example, from Gauss-Hermite quadrature, giving m pairs $(x_j, F_X(x_j))$;
2. Invert the target CDF $y_j = F_Y^{-1}(F_X(x_j))$, $j = 1, \ldots, m$, and form m pairs (x_j, y_j);
3. Define the interpolation function, $y = g_m(x)$, based on these m point pairs (x_j, y_j);
4. Obtain sample $\hat{Y}$ by applying the mapping function $\hat{Y} = g_m(\hat{X})$, where sample $\hat{X}$ is drawn from X.

The Cameron-Martin Theorem [130] states that any distribution can be approximated by a polynomial chaos approximation based on the normal distribution, but also other random variables may be used for X (see, for example, [117]).

Clearly, Equation (5.8) can be compared to Equation (5.12), as a discretization scheme to approximate the realization in the next time point.

5.3. METHODOLOGY

For our purposes, given $Y(t)$, the conditional variable $Y(t+\Delta t)$ can be written as,

$$Y(t+\Delta t)|Y(t) \approx g_m(X) = \sum_{j=0}^{m-1} \hat{\alpha}_j X^j, \tag{5.13}$$

where the coefficients $\hat{\alpha}_j \equiv \hat{\alpha}_j(\hat{Y}_i, t_i, t_{i+1}, \Delta t, \boldsymbol{\theta})$ are now functions of realization $\hat{Y}_i$. Equation (5.13), with large m-values, holds for any Δt, particularly also for large Δt. As such the scheme can be interpreted as an *almost exact simulation scheme* for an SDE under consideration. By the scheme in (5.13) we can thus take large time steps in a highly accurate discretisation scheme. More specifically, a sample from the known distribution X can be mapped onto a corresponding unique sample of the conditional distribution $Y(t+\Delta t)$ by the coefficient functions.

There are essentially two possibilities for using an ANN in the framework of the stochastic collocation method, the first being to directly learn the (time-dependent) polynomial coefficients, $\hat{\alpha}_j$, in (5.13), the second to learn the collocation points, y_j. The two methods are equivalent mathematically, but the latter,

our method of choice, appears more stable and flexible. Here, we explain how to learn the collocation points, y_i, which is then followed by inferring the polynomial coefficients. When the stochastic collocation points at time $t + \Delta t$ are known, the coefficients in (5.13) can easily be computed.

An SDE solution is represented by its cumulative distribution at the collocation points, plus a suitable accurate interpolation $g_m(x)$. In other words, the SCMC method forces the distribution functions (the target and the numerical approximation) to strictly match at the collocation points over time. The collocation points are dynamic and evolve with time.

5.3.1. DATA-DRIVEN NUMERICAL SCHEMES

Calculating the conditional distribution function requires generating samples conditionally on previous realizations of the stochastic process. Based on a general polynomial expression, the conditional sample, in discrete form, is defined as follows,

$$\hat{Y}_{i+1}|\hat{Y}_i = \sum_{j=0}^{m-1} \hat{\alpha}_{i+1,j}\left(\hat{Y}_i, t_i, t_{i+1} - t_i, \boldsymbol{\theta}\right) \hat{X}_{i+1}^j, \tag{5.14}$$

where $\Delta t = t_{i+1} - t_i$, and the coefficients $\hat{\alpha}_{i+1,j}$ are functions of the variables $\hat{Y}_i, t_i, t_{i+1} - t_i, \boldsymbol{\theta}$, for example, see Formulas (5.6) and (5.7).

In the case of a Markov process, the future doesn't dependent on past values. Given $\hat{Y}(t_i)$, the random variable $\hat{Y}(t_{i+1})$ only depends on the increment $Y(t_{i+1}) - Y(t_i)$. The process has independent increments, and the conditional distribution at time t_{i+1} given information up to time t_i only depends on the information at t_i.

Similar to these coefficient functions, the *m conditional* stochastic collocation points at time t_{i+1}, $y_j(t_{i+1})|\hat{Y}_i$, with $j = 0, \ldots, m-1$, can be written as a functional relation,

$$y_j(t_{i+1})|\hat{Y}_i = H_j\left(\hat{Y}_i, t_i, t_{i+1} - t_i, \boldsymbol{\theta}\right). \tag{5.15}$$

A closed-form expression for function $H(\cdot)$ is generally not available. Finding the conditional collocation points can however be formulated as a regression problem.

It is well-known that neural networks can be utilized as universal function approximators [11]. We then generate random data points in the domain of interest and the ANN should "learn the mapping function $H_j(\cdot)$", in an off-line ANN training stage. The SCMC method is here used to compute the corresponding collocation points at each time point, which are then stored to train the ANN, in a supervised learning fashion (see, for example, [71]).

5.3.2. THE SEVEN-LEAGUE SCHEME

Next, we detail the generation of the stochastic collocation points to create the training data. Consider a stochastic process $Y(\tau)$, $\tau \in [0, \tau_{max}]$, where τ_{max} represents the maximum time horizon for the process that we wish to sample from. When the analytical solution of the SDE is not available (and we cannot use an exact simulation scheme with large time steps), a classical numerical scheme will be employed, based on *tiny constant time increments* $\Delta\tau = \tau_{i+1} - \tau_i$, a discretization in the time-wise direction with grid points $0 < \tau_1 < \tau_2 < \ldots < \tau_N \leq \tau_{max}$, to generate a sufficient number of highly accurate samples at each time point τ_i, to approximate the corresponding cumulative functions highly accurately. With the obtained samples, we approximate the collocation points, as follows,

$$\hat{y}_j(\tau_i) = F^{-1}_{\hat{Y}(\tau_i)}(F_X(x_j)) \approx F^{-1}_{Y(\tau_i)}(F_X(x_j)) \tag{5.16}$$

where $\hat{y}_j(\cdot)$ represents the approximate collocation points of $Y(t)$ at time τ_i, and x_j, $j = 1, \ldots, M_s$, are collocation points of variable X. For simplicity, consider $X \sim \mathcal{N}(0,1)$, so that the points x_j are known analytically and do not depend on time point τ_i. In the case of a normal distribution, these points are known quadrature points, and tabulated, for example, in [117]. After this first step, we have the set of collocation points, $\hat{y}_j(\tau_i)$, for $i = 1, \ldots, N$ and $j = 1, \ldots, M_s$. Subsequently, the $\hat{y}_j(\cdot)$ from (5.16) are used as the ground-truth to train the ANN.

In the second step, we determine the *conditional* collocation points. For each time step τ_i and collocation point indexed by j, a *nested Monte Carlo simulation* is then performed to generate the conditional samples. Similar to the first step, we obtain the conditional collocation points from each of these sub-simulations using (5.16). This yields the following set of M_c conditional collocation points,

$$\hat{y}_{k|j}(\tau_{i+1}) := \hat{y}_k(\tau_{i+1})|\hat{y}_j(\tau_i) = F^{-1}_{\hat{Y}(\tau_{i+1})|\hat{Y}(\tau_i)=\hat{y}_j(\tau_i)}\Big(F_X(x_{k|j})\Big), \tag{5.17}$$

where $x_{k|j}$ is a conditional collocation point, and $i \in \{0, 1, \ldots, N-1\}$, $j \in \{1, \ldots, M_s\}$, $k \in \{1, \ldots, M_c\}$. Note that, in the case of Markov processes, the above generic procedure can be simplified by just varying the initial value Y_0 instead of running a nested Monte Carlo simulation. Specifically, we then set $\hat{Y}_i = \hat{Y}_0$, $\tau_i = \tau_0$ and $\tau_{i+1} = \tau_0 + \Delta\tau$ to generate the corresponding conditional collocation points.

The inverse, $F^{-1}_{\hat{Y}(\tau_{i+1})|\hat{Y}(\tau_i)}(\cdot)$, is often not known analytically, and needs to be derived numerically. An efficient procedure for this is presented in [131]. Of course, it is well-known that the computation of $F^{-1}(p)$ is equivalent with the computation of the quantile at level p.

We encounter essentially four types of stochastic collocation (SC) points: x_j are called the original SC points, $\hat{x}_j$ are original conditional collocation points, $\tilde{y}_j$ are the marginal SC points, and $\hat{y}_k|\cdot$ are the conditional SC points. For example,

$\hat{y}_k|\hat{Y}_i$ is conditional on a realization $\hat{Y}_i$. When a previous realization happens to be a collocation point, e.g., $\hat{Y}_i = \hat{y}_j$, we have $\hat{y}_{k|j} := \hat{y}_k|\hat{y}_j$.

When the data generation is completed, the ANNs are trained on the generated SC points to approximate the function H in (5.15), giving us a learned function $\hat{H}$. This is called the training phase. With the trained ANNs, we can approximate new collocation points, and develop a numerical solver for SDEs, which is the Seven-League scheme (7L), see Algorithm II. Figure 5.1 gives a schematic illustration of Monte Carlo sample paths that are generated by the 7L scheme.

When the approximation errors from ANN and SCMC are negligible, the strong convergence properties of the 7L scheme are defined, as follows,

$$\mathbb{E}|\tilde{Y}(t_i) - Y(t_i)| \leq \epsilon(\Delta\tau) \ll K(\Delta t)^{\beta_s}, \tag{5.18}$$

where time step $\Delta\tau$ is used to define the ANN training data-set, and the actual time step Δt is used for ANN prediction, with $\Delta\tau \ll \Delta t$. Based on the trained 7L scheme, the strong error, $\epsilon(\Delta\tau)$, does thus not grow with the actual time step Δt. Particularly, let's assume $\Delta\tau = \Delta t/\kappa$, for example $\kappa = 100$, when employing the Euler-Maruyama scheme with time step $\Delta\tau$ during the ANN learning phase, we expect a strong convergence of $O(\sqrt{\Delta\tau})$, which then equals $O(\sqrt{\Delta t/\kappa})$, while the use of the Milstein scheme during training would result in $O(\Delta t/\kappa)$ accuracy. When $\kappa = 100$, the time step during the learning phase is 100 times smaller than Δt, which has a corresponding effect on the overall scheme's accuracy in terms of its strong, path-wise convergence. *Moreover, the maximum value of the time step Δt in the 7L scheme can be set up to τ_{max} for a Markov process.*

Remark (Lagrange interpolation issue)**.** *In the case of classical Lagrange interpolation, matrix $A(x_{k|i})$ in Algorithm II (on the next page) would be the Vandermonde matrix. In that case, it should not get too large, as the matrix would then suffer from ill-conditioning. However, when employing orthogonal polynomials, this drawback is removed. More details can be found in [117].*

Algorithm II: 7L Scheme

1. Offline stage: Train the ANNs to learn the stochastic collocation points. At this stage, we choose different $\boldsymbol{\theta}$ values, simulate corresponding Monte Carlo paths, with small constant time increments $\Delta\tau = \tau_{i+1} - \tau_i$ in $[0, \tau_{max}]$, generate the $\hat{y}_j$ and $\hat{y}_{k|j}$ collocation points, and learn the relation between input and output. So, we actually "learn" $H_k \approx \hat{H}_k$. See Section 5.3.3 for the ANN details.

2. Online stage: Partition time interval $[0, T]$, $t_i = i \cdot T/N$, $i = 0, \dots N$, with equidistant time step $\Delta t = t_{i+1} - t_i$. Given a sample $\hat{Y}_i$ at time t_i, compute m collocation points at time t_{i+1} using

$$\hat{y}_j(t_{i+1})|\hat{Y}_i = \hat{H}_j(\hat{Y}_i, t_i, t_{i+1} - t_i, \boldsymbol{\theta}), j = 1, 2, \dots, m, \tag{5.19}$$

   and form a vector $\hat{\mathbf{y}}_{i+1} = (\hat{y}_1(t_{i+1})|\hat{Y}_i, \hat{y}_2(t_{i+1})|\hat{Y}_i, \dots, \hat{y}_m(t_{i+1})|\hat{Y}_i)$.

3. Compute the interpolation function $g_m(\cdot)$, or calculate the coefficients $\hat{\boldsymbol{\alpha}}_{i+1}$ (if necessary):

$$A(x_{k|i+1})\hat{\boldsymbol{\alpha}}_{i+1} = \hat{\mathbf{y}}_{i+1}, \tag{5.20}$$

   see Algorithm I for details on the computation of original collocation points. We will compare monotonic spline, Chebyshev and the barycentric formulation of Lagrange interpolation for this purpose. See Section 5.4.2 for a detailed discussion.

4. Sample from X and obtain a sample in the next time point, $\hat{Y}_{i+1}$, by $\hat{Y}_{i+1}|\hat{Y}_i = g_m(\hat{X}_{i+1})$, or the coefficient form as follows,

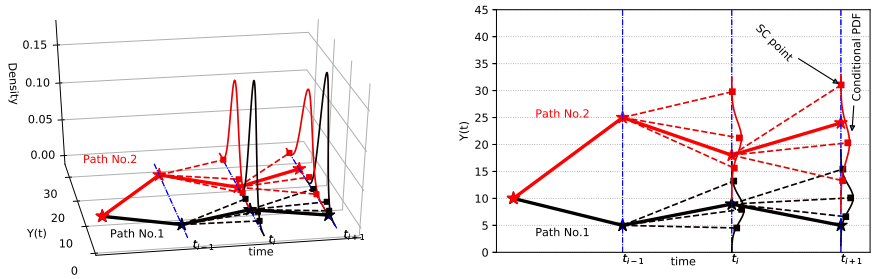
$$\hat{Y}_{i+1}|\hat{Y}_i = \sum_{j=0}^{m-1} \hat{\alpha}_{i+1,j} \hat{X}_{i+1}^j.$$

5. Return to Step 2 by $t_{i+1} \rightarrow t_i$, iterate until terminal time T.

6. Repeat this procedure for a number of Monte Carlo paths.

5.3.3. THE ARTIFICIAL NEURAL NETWORK

The ANN to learn the conditional collocation points is detailed in this subsection. As shown in the previous chapters, neural networks can be utilized as powerful



(a) Sample paths by 7L

(b) The 2D projection

Figure 5.1: Schematic diagram of the 7L scheme. Left: Sample paths generated by 7L. Right: The 2D projection of Figure 5.1a. Here conditional SC points, represented by ■, are conditional on a previous realization, denoted by ★. "Conditional PDF" is the conditional probability density function, defined by these conditional SC points. The density function, which is not required by 7L, is plotted only for illustration purposes.

5

functions to approximate a nonlinear relationship. Next we will briefly recall the derivation of ANNs as function approximates.

A fully connected neural network, without skip connections, can be described as a composition function, i.e.,

$$\hat{H}(\acute{\mathbf{x}}|\hat{\boldsymbol{\Theta}}) = \hat{h}_{L_A}(...\hat{h}_2(\hat{h}_1(\acute{\mathbf{x}};\hat{\boldsymbol{\theta}}_1);\hat{\boldsymbol{\theta}}_2);...\hat{\boldsymbol{\theta}}_{L_A}), \tag{5.21}$$

where $\acute{\mathbf{x}}$ represents the input variables, $\boldsymbol{\Theta}$ being the hidden parameters (i.e. weights and biases), L_A the number of hidden layers. We can expand the hidden parameters as,

$$\hat{\boldsymbol{\Theta}} = (\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_2, \ldots, \hat{\boldsymbol{\theta}}_{L_A}) = (\mathbf{w}_1, \mathbf{b}_1, \mathbf{w}_2, \mathbf{b}_2, \ldots, \mathbf{w}_{L_A}, \mathbf{b}_{L_A}), \tag{5.22}$$

where $\mathbf{w}_\ell$ and $\mathbf{b}_\ell$ represent the weight matrix and the bias vector, respectively, in the ℓ-th hidden layer.

Each hidden-layer function, $\hat{h}_\ell(\cdot), \ell = 1, 2, \ldots, L_A$, takes input signals from the output of a previous layer, computes an inner product of weights and inputs, and adds a bias. It sends the resulting value in an activation function to generate the output. Let $z_j^{(\ell)}$ denote the output of the j-th neuron in the ℓ-th layer. Then,

$$z_j^{(\ell)} = \varphi^{(\ell)}\left(\sum_i w_{i,j}^{(\ell)} z_i^{(\ell-1)} + b_j^{(\ell)}\right), \tag{5.23}$$

where $w_{i,j}^{(\ell)} \in \mathbf{w}_\ell$, $b_j^{(\ell)} \in \mathbf{b}_\ell$, and $\varphi^{(\ell)}$ is a nonlinear transfer function (i.e. activation function). With a specific configuration, including the architecture, the hidden

parameters, activation functions and other specific operations (e.g., drop out), the ANN in (5.21) becomes a deterministic, complicated, composite function.

Supervised machine learning [71] is used here to determine the weights and biases, where the ANN should learn the mapping from a given input to a given output, so that for a new input, the corresponding output will be accurately approximated. Such ANN methodology consists of basically two phases. During the (time-consuming, but off-line) training phase the ANN learns the mapping, with many in- and output samples, while in the testing phase, the trained model is used to very rapidly approximate new output values for other parameter sets, in the on-line stage.

In a supervised learning context, the loss function measures the distance between the target function and the function implied by the ANN. During the training phase, there are many known data samples available, which are represented by input-output pairs $(\acute{\mathbf{X}}, \acute{\mathbf{Y}})$. With a user-defined loss function $\mathrm{L}(\hat{\boldsymbol{\Theta}})$, training neural networks is formulated as

$$\arg\min_{\hat{\boldsymbol{\Theta}}} \mathrm{L}(\hat{\boldsymbol{\Theta}}|(\acute{\mathbf{X}}, \acute{\mathbf{Y}})), \tag{5.24}$$

where the hidden parameters are estimated to approximate the function of interest in a certain norm. For example, using the $L2$-norm, the loss function reads,

$$\mathrm{L}(\hat{\boldsymbol{\Theta}}|(\acute{\mathbf{X}}, \acute{\mathbf{Y}})) = ||\hat{H}(\acute{\mathbf{X}}|\hat{\Theta}) - \acute{\mathbf{Y}})||_2. \tag{5.25}$$

In this chapter, the input, $\acute{\mathbf{x}}$, equals $\{\hat{Y}_i, t_i, t_{i+1} - t_i, \boldsymbol{\theta}\}$, and the output, $\acute{\mathbf{y}}$, represents the collocation points $\hat{\mathbf{y}}_{i+1}$, as in Equation (5.19). In the domain of interest $\acute{\Omega}$, we have a collection of data points $\{\acute{\mathbf{x}}_k\}$, $k = 1, \ldots, M_D$, and their corresponding collocation points $\{\acute{\mathbf{y}}_k\}$, which form a vector of input-output pairs $(\acute{\mathbf{X}}, \acute{\mathbf{Y}}) = \{(\acute{\mathbf{x}}_k, \acute{\mathbf{y}}_k)\}_{k=1,\ldots,M_D}$. A popular approach for training ANNs is to optimize the hidden parameters via back-propagation, for instance, using stochastic gradient descent [71].

5.4. AN EFFICIENT LARGE TIME STEP SCHEME: COMPRESSION-DECOMPRESSION VARIANT

The 7L scheme employs the ANNs to generate the conditional collocation points for all samples of a previous time point, see Figure 5.1b. The extensive use of ANNs in the methodology has an impact on the method's computational complexity.

In order to speed up the data-driven 7L scheme procedure, we introduce a compression-decompression (CDC) variant, *in the on-line validation phase*. Please note that the off-line learning phase is identical for both variants. The so-called 7L-CDC scheme, to be developed in this section, only uses the ANNs to

determine the conditional collocation points for the optimal collocation points of a previous time point. All other samples will be computed by means of accurate interpolation. The computational complexity is reduced when the chosen interpolation is computationally cheaper than using ANNs.

By the compression-decompression procedure, Monte Carlo sample paths based on SDEs can be recovered from a 3D matrix. We then employ the 7L scheme procedure only to compute the entries of the encoded matrices C_i at time point t_i, which leads to a reduction of the computational cost in many cases.

Next, we will explain the process of recovering the sample paths from a known matrix C using the decompression method.

5.4.1. CDC VARIANT

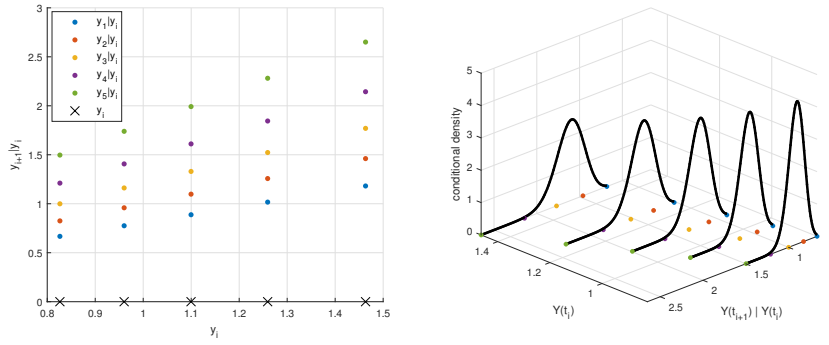
With a time discretization $\{t_0, t_1, t_2, \dots, t_N\}$, we define a three-dimensional matrix $\hat{C} = \{\hat{C}_0, \hat{C}_1, \dots, \hat{C}_{N-1}\}$, which consists of $N \times (M_s + 1) \times (M_c + 2)$ entries in total. Recall that M_s represents the number of collocation points and M_c the number of conditional collocation points. M_s and M_c may vary with time points t_i (in case of an adaptive scheme, for example), but we use constant values for M_s and M_c. For each time point t_i, we construct a 2D matrix $\hat{C}_i$,

$$\hat{C}_i = \begin{pmatrix} - & - & \hat{x}_1 & \hat{x}_2 & \dots & \hat{x}_{M_c} \\ x_1 & \tilde{y}_1(t_i) & \hat{y}_{1|1}(t_i) & \hat{y}_{2|1}(t_i) & \dots & \hat{y}_{M_c|1}(t_i) \\ x_2 & \tilde{y}_2(t_i) & \hat{y}_{1|2}(t_i) & \hat{y}_{2|2}(t_i) & \dots & \hat{y}_{M_c|2}(t_i) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{M_s} & \tilde{y}_{M_s}(t_i) & \hat{y}_{1|M_s}(t_i) & \hat{y}_{2|M_s}(t_i) & \dots & \hat{y}_{M_c|M_s}(t_i) \end{pmatrix}_{(M_s+1)\times(M_c+2)}, \tag{5.26}$$

with x_i, $i = 1, \dots, M_s$, the original SC points, $\hat{x}_k$, $k = 1, \dots, M_c$, the k-th original conditional SC points, and the conditional SC points $\hat{y}_{k|j}(t_i) = \hat{y}_k(t_{i+1})|\hat{y}_j(t_i)$. We thus represent the conditional SC points, $\hat{y}_k(t_{i+1})|\hat{y}_j(t_i)$, by matrix elements $c_{i,j,k}$. The two empty cells in (5.26) are not addressed in the computation. Moreover, at the last time point, t_N, $\hat{C}_N$ is not needed.

Remark (Time-dependent elements)**.** *As the original collocation points, x_i and $\hat{x}_k$, do not depend on time, we can remove the first row and the first column of matrix $\hat{C}_i$ to obtain a time-dependent version, $C = \{C_0, C_1, \dots, C_{N-1}\}$, with the following elements,*

$$C_i = \begin{pmatrix} \tilde{y}_1(t_i) & \hat{y}_{1|1}(t_i) & \hat{y}_{2|1}(t_i) & \dots & \hat{y}_{M_c|1}(t_i) \\ \tilde{y}_2(t_i) & \hat{y}_{1|2}(t_i) & \hat{y}_{2|2}(t_i) & \dots & \hat{y}_{M_c|2}(t_i) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \tilde{y}_{M_s}(t_i) & \hat{y}_{1|M_s}(t_i) & \hat{y}_{2|M_s}(t_i) & \dots & \hat{y}_{M_c|M_s}(t_i) \end{pmatrix}_{M_s\times(M_c+1)}. \tag{5.27}$$



(a) Marginal and Cond. SC points (b) Cond. SC points and densities

Figure 5.2: Schematic illustration of matrix C, with five marginal SC points and five conditional SC points. The conditional SC points are dependent on the realization connected to the corresponding marginal SC point.

An entry in matrix $\hat{C}$ can be computed by the trained ANNs, as follows,

$$c_{i,j,k} := \hat{y}_{k|j}(t_i) = \hat{H}_k^{(M_c)}\Big(\hat{y}_{i,j}, t_i, t_{i+1} - t_i, \boldsymbol{\theta}\Big), \tag{5.28}$$

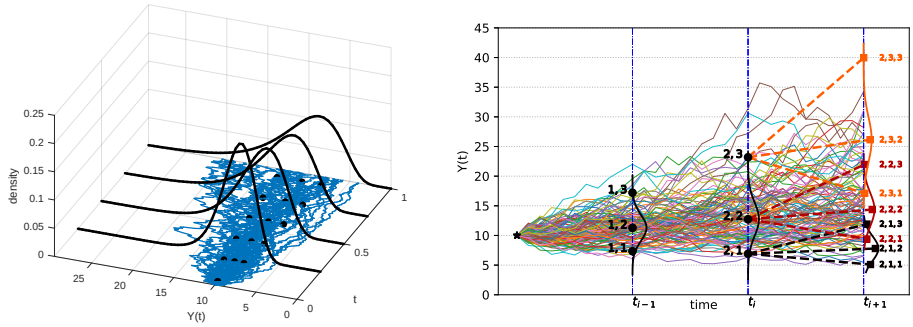
using the marginal SC points,

$$\tilde{y}_j(t_i) = \hat{H}_j^{(M_s)}\Big(Y_0, t_0, t_i - t_0, \boldsymbol{\theta}\Big), \tag{5.29}$$

where $\hat{H}_j^{(\Lambda)}(\cdot)$, $j = 1, \ldots, m$, $\Lambda = \{M_s, M_c\}$, represents the ANN function which approximates the j-th collocation point when $\Lambda = M_s$, and the j-th conditional collocation point when $\Lambda = M_c$. When $M_c = M_s$, $\hat{H}_j^{(M_s)} = \hat{H}_j^{(M_c)}$. Figure 5.2 shows an example of the distribution of the conditional SC points when $M_c = 3$ and $M_s = 3$. When the matrices have been defined, all sample paths are compressed into a structured matrix. In other words, matrix $\hat{C}$ contains all the information needed to perform the Monte Carlo simulation of the SDEs, apart from the interpolation technique.

The resulting matrix C will be *decompressed* to generate Monte Carlo sample paths with the help of an interpolation. The process of decompression is straightforward given a matrix $\hat{C}$. In addition to the interpolation process $g_{M_c}(\cdot)$ in SCMC (see Equation (5.20)), an interpolation $\tilde{g}(\cdot)$ is needed to compute conditional collocation points for previous realizations, based on the matrix $\hat{C}$.

Suppose a vector of samples $\hat{\mathbf{Y}}_i$ at time t_i, and we wish to generate samples of $\hat{\mathbf{Y}}_{i+1}$. For a specific sample $\hat{Y}_i^*$, we need to calculate M_c conditional SC points. To obtain the k-th ($1 \le k \le M_c$) conditional SC point, we take marginal



(a) Paths for marginal SC points (b) Sample paths by 7L-CDC

Figure 5.3: Schematic diagram of the 7L-CDC scheme at time t_i. Left: Marginal SC points, corresponding to Equation (5.29). Right: Sample paths generated by 7L-CDC. The triple {2,1,3}, in the picture, represents the third conditional SC point, dependent on the first marginal SC point at time point t_2. The above procedure is also applicable to other time points.

5

collocation points and their k-th conditional collocation points to form M_s pairs $\{(\tilde{y}_1(t_i), \hat{y}_{k|1}(t_i)), (\tilde{y}_2(t_i), \hat{y}_{k|2}(t_i)), \ldots, (\tilde{y}_{M_s}, \hat{y}_{k|M_s}(t_i))\}$. This combination gives us the interpolation function $\hat{\mathbf{y}} = \hat{g}(\hat{\mathbf{x}})$. Then we can obtain the k-th conditional SC point of $\hat{Y}_i^*$,

$$\hat{y}_k^*(t_{i+1})|\hat{Y}_i^* = \hat{g}(\hat{Y}_i^*). \tag{5.30}$$

As a result, for each sample $\hat{Y}_i^*$, we obtain M_c interpolation nodes, that form a set of pairs, $\left(\hat{x}_1, \hat{y}_1^*(t_{i+1})|\hat{Y}_i^*\right), \left(\hat{x}_2, \hat{y}_2^*(t_{i+1})|\hat{Y}_i^*\right)$, up to $\left(\hat{x}_k, \hat{y}_{M_c}^*(t_{i+1})|\hat{Y}_i^*\right)$, which are used to determine the interpolation function $g_{M_c}(\cdot)$ required by SCMC. Afterwards, to generate a new sample $\hat{Y}_{i+1}^*|\hat{Y}_i^*$, the mapping function g_{M_c} produces a conditional sample by taking in a random sample from X,

$$\hat{Y}_{i+1}^*|\hat{Y}_i^* = g_{M_c}(\hat{X}_{i+1}).$$

The choice of the appropriate number of (conditional) collocation points is a trade-off between the computational cost and the required accuracy. When the number of collocation points tends to infinity, the 7L-CDC scheme will resemble the 7L scheme from Section 5.3.1. A schematic picture is presented in Figure 5.3.

Remark (Computation time)**.** *During the on-line phase of the method, the total computation time of the large time step schemes consists of essentially two parts, calculation of the conditional SC points, and generating random samples by interpolation (the second part). The difference between the 7L and 7L-CDC schemes is found in the computation of the conditional SC points, the generation of the samples is identical for both schemes.*

In this first part, for the 7L-CDC scheme, the work consists of setting up matrix C by the ANNs and computing the conditional SC points by the interpolation. In matrix C, there are $M_s \times M_c \times N$ elements that are computed by the ANNs, where N represents the number of time points, M_s the number of collocation points and M_c the number of conditional collocation points. Based on the M_s collocation points, the interpolation is based on M_c conditional collocation points for each path. For the 7L scheme, $M \times M_c \times N$ elements, where M is the total number of paths, are computed by the ANNs. The time ratio between the 7L-CDC and 7L schemes is found as

$$\gamma = \frac{t_I M + t_A M_s}{t_A M} = \frac{t_I}{t_A} + \frac{M_s}{M}, \tag{5.31}$$

with t_A the computational time of the ANN (i.e., the function $\hat{H}(\cdot)$), t_I for the interpolation (i.e., the function $\hat{g}(\cdot)$ in (5.30)), which is a polynomial function of M_s. Given the fact that the number of sample paths is typically much larger than the number of SC points $M \gg M_s$,

$$\gamma \approx \frac{t_I}{t_A}.$$

When the employed interpolation is computationally cheaper than the ANNs, $\gamma < 1$, so that the 7L-CDC scheme needs fewer computations than the 7L scheme.

5.4.2. INTERPOLATION TECHNIQUES

To define the function $g_m(x)$ in (5.13) or $\hat{g}(x)$ in (5.30), we will compare three different interpolation techniques.

A bijective mapping function is obtained by the *monotonic* Piecewise Cubic Hermite Interpolating Polynomial (PCHIP) [132]. Assuming there are multiple data points, (x_k, y_k), using,

$$h_k := x_{k+1} - x_k,\ d_k := \frac{y_{k+1} - y_k}{x_{k+1} - x_k},$$

the derivatives f'_k at the points x_k are computed as a weighted average,

$$\frac{\hat{w}_1 + \hat{w}_2}{f'_k} = \frac{\hat{w}_1}{d_{k-1}} + \frac{\hat{w}_2}{d_k}, \quad \text{if } d_k \cdot d_{k-1} > 0,$$

where $\hat{w}_1 := 2h_k + h_{k-1}$ and $\hat{w}_2 := h_k + 2h_{k-1}$. At each data point the first derivative is guaranteed to be continuous, and a cubic spline is used to interpolate between the data points. If $d_k \cdot d_{k-1} \leq 0$, then $f'_k = 0$. PCHIP requires more computations than a Lagrange interpolation, but it results in a monotonic function which is generally advantageous.

The convergence of the stochastic collocation method is not really dependent on the monotonicity of the mapping function, so an interpolation based

on *Lagrange polynomials* is possible in practice. The barycentric version of Lagrange interpolation [133], our second interpolation technique, provides a rapid and stable interpolation scheme, which is applied when using Lagrange interpolation in our numerical experiments. With help of the basic Lagrange interpolation expressions, however, we can conveniently perform theoretical analysis.

The third technique is based on choosing the interpolation points carefully (e.g., as the Chebyshev zeros) to achieve a stable interpolation. The *Chebyshev interpolation* [134] is of the form,

$$g_m(x) = \sum_{j=0}^{m-1} \alpha_j p_j(x) = \alpha_0 + \alpha_1 p_1(x) + \ldots + \alpha_{m-1} p_{m-1}(x), \tag{5.32}$$

where $p_{m-1}(x)$ are interpolation basis functions, here Chebyshev orthogonal polynomials, up to degree $m-1$. The Chebyshev nodes in the interval $[x_a, x_b]$ are computed as,

$$\tilde{x}_k = x_a + \frac{1+\cos(\frac{\pi k}{m-1})}{2}(x_b - x_a), k = 0, 1, \ldots, m-1.$$

When the polynomial degree increases, the Chebyshev interpolation retains uniform convergence. In financial mathematics, Chebyshev interpolation has been successfully used, for example, to compute parametric option prices and implied volatility in [135–137]. When the interpolation points are not Chebyshev nodes (e.g., Gauss quadrature points), the Chebyshev coefficients can be estimated by means of a least squares regression, which is also called the Chebyshev fit. In such case, the coefficients in (5.14) can be explicitly computed, in contrast to the barycentric Lagrange interpolation. The selection of a suitable interpolation technique depends on various factors, for instance, speed, monotonicity, availability of coefficients. These three interpolation methods will be compared in the numerical section.

5.4.3. PATH-WISE SENSITIVITY

Often in computations with stochastic variables, we wish to determine the derivatives of the variables of interest, the so-called pathwise sensitivities. This is generally not a trivial exercise in a Monte Carlo setting, see, for example, the discussions in [138–141]. With our new large time step schemes, we determine the pathwise sensitivities of the computed stochastic variables in a natural way, based on the available information in the (conditional) SC points and the interpolation. In this section, we derive the pathwise sensitivity of the state variable $Y(t)$ with respect to model parameters θ.

The first derivative with respect to parameter θ of the conditional distribution in Equation (5.13) reads,

$$\frac{\partial Y(t_{i+1})}{\partial \theta} = \frac{\partial g(X)}{\partial \theta} \approx \frac{\partial}{\partial \theta}\left(\sum_{j=1}^{m} \hat{y}_j(t) p_j(X)\right) = \frac{\partial}{\partial \theta}\left(\sum_{j=1}^{m} \hat{H}_j p_j(x)\right) = \sum_{j=1}^{m}\left(\frac{\partial \hat{H}_j}{\partial \theta} p_j(X)\right), \tag{5.33}$$

where $p_j(X)$ are basis functions, which do not depend on the model parameters. For the derivative $\frac{\partial \hat{H}_j}{\partial \theta}$ in (5.33) at time t_i, the expression of the ANN (5.21), given the specific activation function, is available. So, the function $\hat{H}$ is analytically differentiable. As a result, $\frac{\partial \hat{H}_j}{\partial \theta}$ can be easily computed, by means of automatic differentiation in the machine learning framework. Thus, we arrive at the sensitivity of a sample path with respect to model parameters, as follows,

$$\frac{\partial \hat{Y}_{i+1}}{\partial \theta} = \sum_{j=1}^{m} \frac{\partial \hat{H}_j}{\partial \theta} p_j(\hat{X}_{i+1}). \tag{5.34}$$

5

5.5. NUMERICAL EXPERIMENTS

In this section with numerical experiments we will give evidence of the high quality of our numerical SDE solver, by analyzing first in detail its components. For this purpose, we mainly focus on the Geometric Brownian Motion SDE, which reads,

$$\mathrm{d}Y(t) = \mu Y(t)\mathrm{d}t + \sigma Y(t)\mathrm{d}W(t), \quad 0 \le t \le T. \tag{5.35}$$

The model parameters are the constant drift and volatility coefficients, i.e., $\boldsymbol{\theta} = \{\mu, \sigma\}$, and the initial value is given by Y_0. For (5.35) a continuous-time analytic expression for the asset price at time t is available, i.e.,

$$Y(t) = Y_0 e^{(\mu - \frac{1}{2}\sigma^2)(t-t_0) + \sigma(W(t) - W(t_0))} \overset{d}{=} Y_0 e^{(\mu - \frac{1}{2}\sigma^2)(t-t_0) + \sigma\sqrt{t-t_0}X}, \tag{5.36}$$

where $X \sim \mathcal{N}(0,1)$, and $Y(t)$ is governed by the lognormal distribution. The derivative of the stock price with respect to volatility σ is available in closed form, and reads,

$$\frac{\partial Y(t)}{\partial \sigma} \overset{d}{=} Y(t)(-\sigma(t-t_0) + \sqrt{t-t_0}X). \tag{5.37}$$

This expression will be used as the reference value of the sensitivity obtained from the 7L discretization.

Furthermore, the Ornstein-Uhlenbeck process is explained and also analyzed, in Subsection 5.5.3. We will employ the large time step discretization, in which the conditional collocation points are computed by the trained ANN, and compare the results of the novel scheme with those obtained by the Milstein SDE discretization.

5.5.1. ANN TRAINING DETAILS

GBM and the OU process are Markov processes, so the conditional distribution at time t_{i+1} given information up to time t_i only depends on the information at time t_i. The ANN (5.15) will therefore be used for the conditional collocation stochastic points, with $\boldsymbol{\theta} = \{\mu, \sigma\}$, for GBM, and $\boldsymbol{\theta} = \{\overline{Y}, \sigma, \lambda\}$ for the OU process (as will be discussed in Subsection 5.5.3).

Regarding the size of the compression-decompression matrix, the more conditional collocation points, the better the accuracy of the 7L-CDC method. A 5x5 matrix size (i.e., five marginal and five conditional SC points) is preferred, taking into account the computing effort and the accuracy. In [117] it has been discussed and shown that highly accurate approximations could already be obtained with a small number of collocation points.

As the first method component, we evaluate the quality of the ANN which defines the collocation points, for the GBM dynamics. For this purpose, M_L random points are generated by using Latin Hyper-cube Sampling (LHS) in the domain of interest for the three parameters (Y_0, μ, σ), see Table 5.1. As the second step, for each point a Monte Carlo method is employed to simulate the discretized SDE based on the tiny time step $\Delta\tau$. We use an Euler-Maruyama time discretization for this purpose, with N_τ the number of time points and the time horizon $\tau_{max} = N_\tau \cdot \Delta\tau$. At each time step, $j = 1, ..., N_\tau$, the conditional distribution function $F_{Y(t_j)|Y_0,\mu,\sigma}(\cdot)$ is computed, based on the many generated MC paths. This way, the resulting collocation points for the "big time step", $\Delta t = j \cdot \Delta\tau$, are also obtained, to form the required training data set.

We set $\tau_{max} = 1.6$, $N_\tau = 500$, $M_L = 160$. The amount of training data used is given by $M_{train} = M_L \cdot N_\tau = 80{,}000$ samples in total, which are divided into an ANN training (90%) and an ANN testing (10%) set.

Table 5.1: Training data, $\Delta\tau = 0.01$. Here is an example for training on five SC points.

ANN	Parameters	Value range	Method
input	drift, μ	(0.0, 0.10]	LHS
	volatility, σ	[0.05, 0.60]	LHS
	value, Y_0	[0.10, 15.0]	LHS
	time, τ_{max}	(0.0, 1.60]	Equidistant
$\hat{H}_1(\cdot)$ output	point, $\hat{y}_1$	(0.0, 25.65)	SCMC
$\hat{H}_2(\cdot)$ output	point, $\hat{y}_2$	(0.0, 25.98)	SCMC
$\hat{H}_3(\cdot)$ output	point, $\hat{y}_3$	(0.0, 27.84)	SCMC
$\hat{H}_4(\cdot)$ output	point, $\hat{y}_4$	(0.0, 54.67)	SCMC
$\hat{H}_5(\cdot)$ output	point, $\hat{y}_5$	(0.0, 154.35)	SCMC

The ANN hyper-parameters impact the optimization errors when training the ANN. The approximation accuracy depends on the width and depth of the network and on the number of hidden parameters. Deep neural networks have more powerful expression capabilities than shallow neural networks. We use one input, one output and four hidden layers. Each hidden layer consists of 50 neurons, with Softplus as the activation function[142]. Before training the ANN, the hidden parameters are initialized via the Glorot technique [143]. Training goes in batches. At each iteration, the stochastic gradient based optimizer, Adam[43], randomly selects a portion of the training samples according to the batch size, to calculate the gradient for updating the hidden parameters. In an epoch, all training samples have been processed by the optimizer. The mean squared error, which measures the distance between the ground-truth and the model values in supervised learning, is used to update the hidden parameters during training. The measure MAE (Mean Absolute Error), i.e.,

$$\text{MAE} = \frac{1}{M_D}\sum_j |y_j - \hat{y}_j|,$$

is also estimated, as the path-wise error of the 7L scheme is related to the maximum absolute difference in the approximated collocation points, $\hat{y}_j$, for which the relevant derivation can be found in Section 5.5.3.

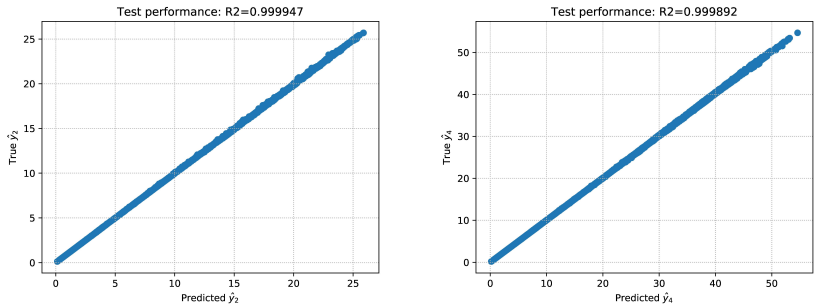
The training process starts with a relatively large learning rate (i.e 10^{-3}) to avoid getting stuck in local optima. After 1000 epochs, the learning rate is reduced to 10^{-4}, followed by training 500 more epochs, to achieve a steady convergence. Afterwards, the trained ANN is evaluated on the testing data set, with the results presented in Figure 5.4 (for two of the collocation points) and Table 5.2. Clearly, the predicted values fit very well with the true values of the stochastic collocation points. This implies that the trained ANNs reach a highly satisfactory generalization, and generate accurate and robust approximation results for all five collocation points.

Table 5.2: The approximation performance on test data set.

SC points	$\hat{y}_1$	$\hat{y}_2$	$\hat{y}_3$	$\hat{y}_4$	$\hat{y}_5$
R^2	0.999891	0.999947	0.999980	0.999892	0.999963
MAE	0.026	0.027	0.021	0.071	0.066

5.5.2. ERROR ANALYSIS, THE LAGRANGIAN CASE

There are essentially two approximation errors in the 7L scheme, a neural network approximation error when generating the collocation points, and an SCMC



(a) $\hat{y}_2$ (b) $\hat{y}_4$

Figure 5.4: The goodness of fit on test data set. Two scatter plots show the relation between the predicted values and the ground truth.

error when representing the conditional distribution function.

Considering d inputs, the neural network may approximate any function $\zeta_{d,n}$, from the function space $C^{n-1}([0,1]^d)$, where the derivatives up to order $n-1$ are Lipschitz continuous [144]. The input and output variables can be normalized to the unit interval $[0,1]$. With a fixed network architecture during training, the approximation error can be assessed, as follows.

Theorem 1. *From [144]. Given any $\hat{\epsilon} \in (0,1)$, there exists a neural network which is capable of approximating any function $\zeta_{d,n}$ with error $\hat{\epsilon}$, based on the following configuration:*

- *at least piece-wise activation functions,*
- *at least $\tilde{c}(\ln(1/\hat{\epsilon})+1)$ hidden layers and $\tilde{c}\hat{\epsilon}^{-d/n}(\ln(1/\hat{\epsilon})+1)$ weights and computation units, where $\tilde{c} := \tilde{c}(d,n)$ depends on the parameters d and n.*

When the architecture is dynamic, the error bound can be further reduced, as shown in [145] and [144]. One of the assumptions is that the ANNs are sufficiently trained, so that the optimization error is negligible.

The error from the SCMC methodology was derived in [117]. The optimal collocation points, x_i, $i = 1,\ldots,m$, correspond to the zeros of an orthogonal polynomial. In the case of Lagrange interpolation, when the collocation method can be connected to Gauss quadrature, we have

$$\int_{\mathbb{R}} \Psi(x) f_X(x) \mathrm{d}x = \sum_{i=1}^{m} \Psi(x_i)\omega_i + \epsilon_m = \epsilon_m, \tag{5.38}$$

with $\Psi(x) = \big(g(x) - g_m(x)\big)^2$, the difference between the target and the SC approximated function, $f_X(x)$ the weight function, and ω_i the quadrature weights.

When the Gauss-Hermite quadrature is used with m collocation points. the approximation error of the CDF can be estimated as,

$$\epsilon_m = \frac{m!\sqrt{\pi}}{2^m}\frac{\Psi^{(2m)}(\xi_1)}{(2m)!}, \tag{5.39}$$

where $\xi_1 \in (-\infty, \infty)$ and the distance function

$$\Psi(x) = \big(g(x) - g_m(x)\big)^2 \approx \left(\frac{1}{m!}\frac{\partial^m g(x)}{\partial x^m}\bigg|_{x=\xi_2}\prod_{k=1}^{m}(x - x_k)\right)^2,$$

with $\xi_2 \in [x_1, x_{m-1}]$. In other words, the error of approximating the target CDF converges exponentially to zero when the number of corresponding collocation points increases.

At each time point t_i, the process $Y(t_i)$ is approximated using the collocation method, by a polynomial $g_m(X)$, i.e., in the case of classical Lagrange interpolation, using $\ell_j(\bar{x}) = p_j(\bar{x})$,

$$Y(t_i) \approx \tilde{Y}(t_i) = g_m(X) = \sum_{j=1}^{m} y_j(t_i)\ell_j(X),\ \ \ell_j(\bar{x}) = \prod_{k=1}^{m}\frac{X - x_k}{x_j - x_k}, \tag{5.40}$$

where the collocation points $y_j(t_i) = F^{-1}_{Y(t_i)}(F_X(x_j))$. Because of the use of an ANN, the collocation points are not exact, but they are approximated with $y_j(t_i) - \hat{y}_j(t_i) = \epsilon^A_j$, where $\hat{y}_j(t_i)$ represents the ANN approximated value. The error associated with ϵ^A_j can be estimated as in [145]. The impact of ϵ^A_j on the obtained output distribution needs to be assessed. Let $\tilde{g}_m$ denote the approximate function based on the predicted ANN collocation points $\hat{y}_j(t_i)$, and x a random sample from the standard normal distribution X. The approximation error, in the strong sense, is given by

$$\begin{aligned}
\mathbb{E}\big[|g_m(x) - \tilde{g}_m(x)|\big] &= \mathbb{E}\Big|\sum_{j=1}^{m} y_j(t_i)\ell_j(x) - \sum_{j=1}^{m}\hat{y}_j(t_i)\ell_j(x)\Big| \\
&= \int_{\mathbb{R}}\Big|\sum_{j=1}^{m} y_j(t_i)\ell_j(x) - \sum_{j=1}^{m}\hat{y}_j(t_i)\ell_j(x)\Big| f_X(x)\mathrm{d}x \\
&= \int_{\mathbb{R}}\Big|\sum_{j=1}^{m}\epsilon^A_j\ell_j(x)\Big| f_X(x)\mathrm{d}x.
\end{aligned} \tag{5.41}$$

Note that the $\ell_j(x)$ interpolation functions are identical as they depend solely on

the x values. We arrive at the following error related to the ANNs,

$$\begin{aligned}\int_{\mathbb{R}} \Big| \sum_{j=1}^{m} \epsilon_j^A \ell_j(x) \Big| f_X(x)\mathrm{d}x &\leq \int_{\mathbb{R}} \sum_{j=1}^{m} \max\{|\epsilon_1^A|,\ldots,|\epsilon_m^A|\}\ell_j(x) f_X(x)\mathrm{d}x \\ &= \int_{\mathbb{R}} \max\{|\epsilon_1^A|,\ldots,|\epsilon_m^A|\} f_X(x)\mathrm{d}x \\ &= \max\{|\epsilon_1^A|,\ldots,|\epsilon_m^A|\} \qquad (5.42)\end{aligned}$$

Considering the error introduced by SCMC in (5.39), the total path wise error reads

$$\begin{aligned}\mathbb{E}\left[|g(x)-\tilde{g}_m(x)|\right] &\leq \mathbb{E}\left[|g(x)-g_m(x)|\right] + \mathbb{E}\left[|g_m(x)-\tilde{g}_m(x)|\right] \\ &\leq \sqrt{|\epsilon_m|} + \max\{|\epsilon_1^A|,\ldots,|\epsilon_m^A|\}. \qquad (5.43)\end{aligned}$$

In other words, the expected pathwise error can be bounded by the approximation CDF error $\sqrt{|\epsilon_m|}$ plus the largest difference in the ANN approximated collocation points.

KOLMOGOROV-SMIRNOV TEST

The Kolmogorov-Smirnov test, calculating the supremum of a set of distances, is used to measure the nonparametric distance between two empirical cumulative distribution functions. We perform the two-sample Kolmogorov-Smirnov test, as follows,

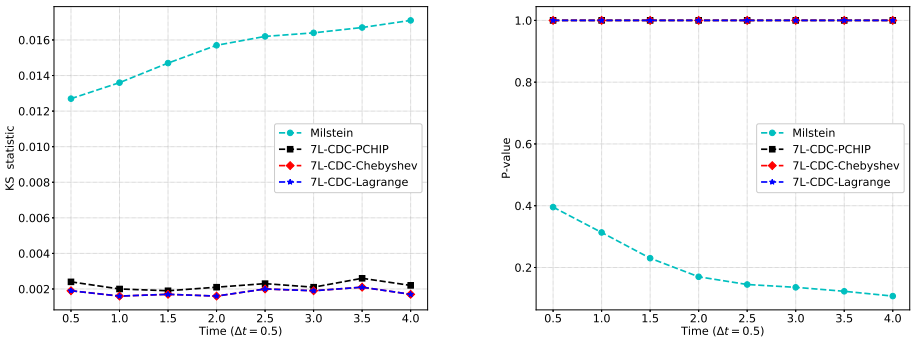
$$KS = \sup_z |F_Y(z) - \hat{F}_Y(z)|,$$

where $\hat{F}_Y(\cdot)$ and $F_Y(\cdot)$ are two empirical cumulative distribution functions, one from the 7L-CDC solution and the other one from the reference distribution. We take the analytic solution of the GBM as the reference distribution.

Remark (Time horizon for 7L-CDC)**.** *The information in Table 5.1 is used to train the mapping function between a realization (including marginal SC points) and its conditional SC points, via Equation* (5.28)*. For the marginal SC points in Equation* (5.29)*, however, we need training data up to terminal time T. So, we generate a second data set in which the time reaches τ_{max} (the terminal time of interest) and the upper value for Y_0 equals 5. The data sets are combined to train the ANNs for the 7L-CDC methodology.*

Figure 5.5 shows the Kolmogorov-Smirnov test at different time points based on 10000 samples. We focus on the CDC methodology here, and compare the accuracy with the different interpolation methods in the figure. Clearly, the KS statistic and also the corresponding P-values for the 7L-CDC schemes are much

better than those of the Milstein scheme in Figure 5.5. This is an indication that the CDFs that originate from the 7L-CDC schemes resemble the target CDF much better, with high confidence. In addition, unlike the Milstein scheme the 7L-CDC schemes exhibit an almost constant difference between the approximated and target CDFs with increasing time.



(a) KS statistic (b) P-value

Figure 5.5: The Kolmogorov-Smirnov test: $\Delta t = 0.5, \mu = 0.1, \sigma = 0.3, Y_0 = 1.0$, with 10000 samples. When we have a small KS statistic or a large P-value, the hypothesis that the distributions of the two sets of random samples are the same can not be rejected.

We will also analyze the costs of the different interpolation methods within 7L-CDC. The two steps which require interpolation are the computation of the conditional collocation points and the generation of conditional samples. The computational speed of the 7L-CDC scheme depends on the employed interpolation method, see Table 5.3.

Table 5.3: The CPU running time (seconds) to reach the same accuracy (CPU: E3-1240, 3.40GHz): simulating 10,000 sample paths until terminal time $T = 4.0$, based on 5×5 marginal/conditional SC points. Here, for the 7L scheme, PCHIP is used as the interpolant $g_m(\cdot)$ in Step 3 of Algorithm I.

Method /Time (Sec.)	$\Delta t = 1.0$			$\Delta t = 2.0$		
	Create C	Decom. C	Total	Create C	Decom. C	Total
7L-CDC Barycentric	0.054	4.93	4.98	0.027	2.48	2.51
7L-CDC Chebyshev	0.054	9.78	9.83	0.027	4.93	4.96
7L-CDC PCHIP	0.054	11.39	11.44	0.027	5.73	5.76
7L scheme	-	-	12.80	-	-	6.39
Milstein	-	-	27.01	-	-	27.70

To achieve a similar accuracy in the strong sense, the Milstein scheme needs a much finer time grid, by a factor of $\kappa = \Delta t/\Delta\tau$. When κ is sufficiently large, the 7L-CDC scheme outperforms Milstein in terms of both accuracy and speed. For example, in Table 5.3, $\kappa = 100$ when $\Delta t = 1.0$, and $\kappa = 200$ when $\Delta t = 2.0$. In addition, computational time of the new scheme can be further reduced by parallelization, for example, using Graphics Processing Units (GPUs).

5.5.3. PATH-WISE ERROR CONVERGENCE

In this section, we compare the path-wise errors of our proposed novel discretization with those of the classical discretization schemes.

GBM PROCESS

We analyze here the strong convergence properties of the new methodology for the GBM process. For GBM, the exact path is given by the expression (5.36). The random number, which is drawn from $X \sim N(0,1)$, is the same for the exact solution (5.36), the novel schemes (5.14) and the Milstein scheme (5.3). The path-wise differences between the numerical schemes and the exact simulation are plotted in Figure 5.6. When $\Delta t = 0.5$, the 7L-CDC scheme presents superior paths as compared to the Milstein scheme, in terms of its path-wise error comparing to the exact path.

As shown in Figure 5.7, the 7L-CDC scheme gives rise to flat, almost constant, strong and weak error convergence curves for many different Δt-values, suggesting a small, constant convergence error even with large time steps Δt. The Milstein scheme has the strong order of convergence $O(\Delta t)$, so that a larger time step gives rise to a larger error. When the time step becomes small, more time points are needed to reach a time T, and then the resulting recursive error of the 7L-CDC scheme increases.

The number of conditional collocation points, by which the conditional distribution at a next time point is mostly determined, has a significant contribution to the convergence order of the 7L-CDC scheme. As mentioned, we found empirically that five conditional collocation points are preferable in terms of computing effort versus accuracy. CDC matrix C is then of size $N \times 5 \times 5$, that is, at each time point, there are five collocation points and each of these has five conditional collocation points.

ORNSTEIN-UHLENBECK PROCESS

Any SDE which can be solved by the Euler-Maruyama discretization can be solved by our ANN methodology, with improved strong convergence properties. We also wish to confirm the strong convergence properties for another stochastic process in this section.

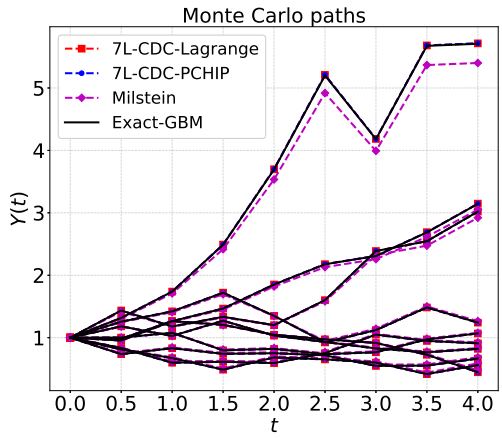
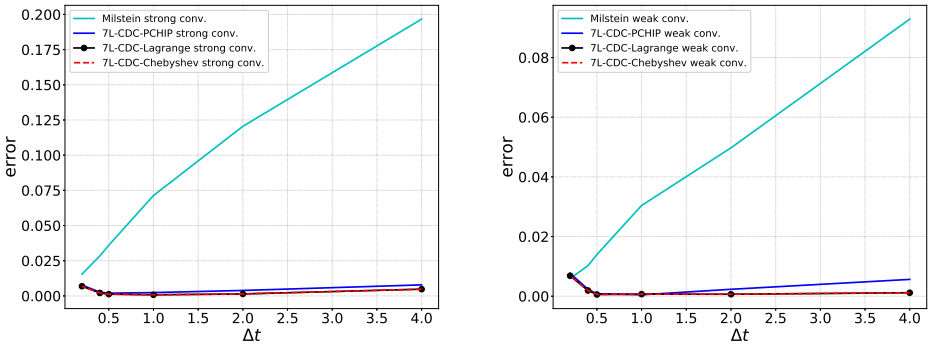


Figure 5.6: Paths generated by 7L-CDC: time step $\Delta t = 0.5$, GBM with $\sigma = 0.3$, $r = 0.1$, $Y_0 = 1.0$. The paths are with Chebyshev interpolation, which are not plotted, are identical to ones from Lagrange in this case.



(a) Strong convergence

(b) Weak convergence

Figure 5.7: The strong error is estimated as $\frac{1}{M}\sum|\check{Y}_k(T) - \hat{Y}_k(T)|$, see Equation (5.4) and the weak error by $\frac{1}{M}(\sum \check{Y}_k(T) - \sum \hat{Y}_k(T))$, see [140, page 261] for details on the computation of the convergence rate. There are $M = 1000$ sample paths in total.

The mean reverting Ornstein-Uhlenbeck (OU) process [146] is defined as,

$$\mathrm{d}Y(t) = \lambda(Y(t) - \overline{Y})\mathrm{d}t + \sigma \mathrm{d}W(t), \quad 0 \le t \le T, \tag{5.44}$$

with $\overline{Y}$ the long term mean of $Y(t)$, λ the speed of mean reversion, and σ the volatility. The initial value is Y_0, and the model parameters are $\boldsymbol{\theta} := \{\overline{Y}, \sigma, \lambda\}$. Its

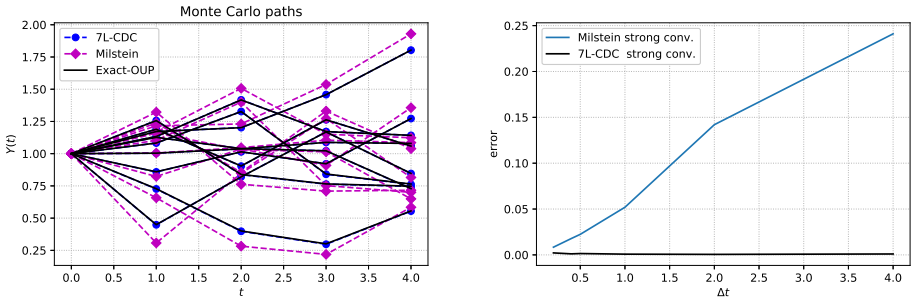
analytical solution is given by,

$$Y(t) \overset{d}{=} Y_0 e^{-\lambda t} + \overline{Y}(1 - e^{-\lambda t}) + \sigma \sqrt{\frac{1 - e^{-2\lambda t}}{2\lambda}} X, \tag{5.45}$$

with $t_0 = 0$, $X \sim \mathcal{N}(0,1)$. Equation (5.45) is used to compute the reference value to the path-wise error and the strong convergence.

We employ the same data-driven procedure as for GBM to discretize and solve the OU process. In the training phase, the Euler-Maruyama scheme (5.2) is used to discretize the OU dynamics and generate the data set. Note that the Milstein and Euler schemes are identical in the case of the OU process. As the OU process is a Markov process, we again can vary Y_0 to find the relation between the conditional SC points and the marginal SC points (i.e. as in Equation (5.28)). Similar to Table 5.1, we employ five SC points to learn within the ANN, with $\Delta\tau = 0.01$, $\tau_{max} = 4.1$, $N_\tau = 500$, $M_L = 410$, see Section 5.5.1 for the details of the training process.

After the training, the obtained ANNs will be applied to solve the OU process with specific parameters and details of our interest. We provide an example in Figure 5.8, which confirms that the sample paths generated by 7L-CDC are as accurate as the exact solution, and the error, in the sense of strong convergence, stays close to zero even with a large time step.



(a) Path-wise error ($\Delta t = 1.0$)

(b) Strong convergence

Figure 5.8: Paths and strong convergence for the OU process, using $\lambda = 0.5$, $\overline{Y} = 1.0$, $\sigma = 0.3$, $Y_0 = 1.0$. The sample paths with barycentric, Chebyshev and PCHIP interpolation overlap for the 7L-CDC scheme. There are five marginal and five conditional SC points at each time point.

5.5.4. APPLICATIONS IN FINANCE

The possibility to take large time steps and still get accurate SDE solutions, is certainly interesting in computational finance, as there are several financial prod-

ucts that are updated on a daily basis (think of an over-night interest rate), whereas monitoring of financial contracts and risk management is typically only done on a weekly, monthly of even yearly basis. In such situations, our novel scheme will be useful. Research into large time step simulations is state-of-the-art in computational finance, see the exact (and almost exact) Monte Carlo simulation papers, like [147, 148] for the SABR and Heston stochastic volatility asset dynamics, respectively.

THE ASIAN OPTIONS

Moreover, the strong convergence property of an SDE discretization is important in many cases. When valuing so-called path-dependent options, for example, improved strong convergence enhances the convergence of a Monte Carlo simulation. Options are governed by their pay-off function (i.e. the option value at the final time of the contract, $t = T$). Here we consider a path-dependent exotic option, the so-called European-style Asian option, which has a payoff that is based on a time-averaged underlying stock price. For example, the pay-off of a fixed strike Asian option is given by

$$V_A(T) = \max(A(T) - \tilde{K}, 0),$$

where T is the option contract's expiry time, and $\tilde{K}$ is the predetermined strike price. Here $A(T)$ denotes the discrete arithmetic average of the stock prices over N_b monitoring dates $\{t_1, \ldots, t_k\} \in [0, T]$,

$$A(T) = \frac{1}{N_b} \sum_{k=1}^{N_b} \hat{Y}(t_k),$$

where $\hat{Y}(t_k)$ is the observed stock price at time t_k, $1 \leq t_k \leq T$. Averaging thus takes place in the time-wise direction, and we consider pricing financial options based on the discrete arithmetic average of a number of stock prices.

We assume here that the underlying stock price follows Geometric Brownian motion, as in Equation (5.35), under the risk-neutral measure, meaning $\mu \equiv r$, where r is the risk-free interest rate. There is a cash account $M(t)$, governed by $\mathrm{d}M(t) = rM(t)\mathrm{d}t$. The value of European-style Asian option is then given by

$$V_A(t) = \mathrm{e}^{-r(T-t)} \mathbb{E}^{\mathbb{Q}} \Big[\max(A(T) - \tilde{K}, 0) \Big| \mathscr{F}(t) \Big]. \tag{5.46}$$

Because the pay-off is clearly a path-dependent quantity for such options, it is expected that an improved strong convergence, obtained with the variant 7L-CDC, will result in superior convergence, as compared to classical numerical discretization schemes.

Table 5.4: Pricing Asian European-style option with a fixed strike price, using $Y_0 = 1.0$, $\tilde{K} = Y_0$, $r = 0.1$, $T = \Delta t \times N_b$.

	method	$\Delta t = 1.0$, N_b=4	$\Delta t = 0.5$, N_b=8
σ=0.30	Analytic MC	0.24886257 (0.00%)	0.22403982 (0.00%)
	Milstein MC	0.23077000 (7.27%)	0.21558276 (3.77%)
	7L-CDC	0.24871446 (0.06%)	0.22404571 (0.00%)
σ=0.40	Analytic MC	0.28515109 (0.00%)	0.25723594 (0.00%)
	Milstein MC	0.26394277 (7.44%)	0.24717425 (3.91%)
	7L-CDC	0.28482371 (0.11%)	0.25647592 (0.30%)

The relative error is presented, which is defined as

$$\epsilon_{rel} = \Big| \frac{V_A^{ref}(t_0) - V_A(t_0)}{V_A^{ref}(t_0)} \Big|,$$

where $V_A^{ref}(t_0, S_0)$ is based on the exact GBM Monte Carlo simulation. As shown in Table 5.4, the 7L-CDC scheme gives highly accurate Asian option prices, compared to the Milstein scheme. As the accuracy of Asian option prices depends directly on the accuracy of the realized paths, an increasing number of monitoring dates will give rise to higher accuracy by 7L-CDC.

Next, we focus on the Asian option's sensitivity. The sensitivity of the option price with respect to volatility σ is called Vega, which can be computed in a pathwise fashion [see 149, Chapter 7], as follows,

$$\frac{\partial V}{\partial \sigma} = e^{-rT}\mathbb{E}^{\mathbb{Q}}\left[\sum_{i=1}^{N} \frac{\partial V(T, Y(t_i); \sigma)}{\partial Y(t_i)} \frac{\partial Y(t_i)}{\partial \sigma} | Y_0 \right]. \tag{5.47}$$

The chain rule is employed to derive the sensitivity. First of all, we compute the gradient of the payoff function with respect to the underlying stock price, by

$$\frac{\partial V(T, Y(t_i))}{\partial Y(t_i)} = \frac{1}{N} \mathbb{1}_{A(T) > \tilde{K}}. \tag{5.48}$$

Then, the derivative of the stock price at time t_i with respect to the model parameter, $\frac{\partial Y(t_i)}{\partial \sigma}$, can be found with the trained ANNs, as given by Equation (5.33). Vega can be estimated by,

$$\frac{\partial V}{\partial \sigma} \approx e^{-rT} \frac{1}{N} \mathbb{E}^{\mathbb{Q}}\left[\sum_{i=1}^{N} \left(\mathbb{1}_{A(T) > \tilde{K}} \sum_{j=0}^{m-1} \frac{\partial \hat{H}_j}{\partial \sigma} p_j(X) \right) | Y_0 \right]. \tag{5.49}$$

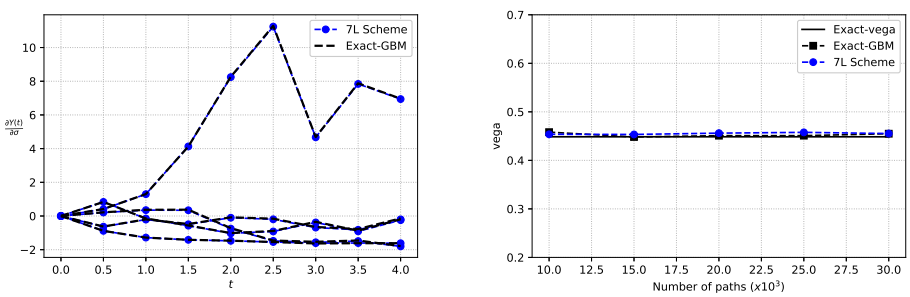
With there are M sample paths, we have,

$$\frac{\partial V}{\partial \sigma} \approx e^{-rT} \frac{1}{M} \frac{1}{N} \left[\sum_{k=1}^{M} \sum_{i=1}^{N} \left(\mathbb{1}_{A(T)>\tilde{K}} \sum_{j=0}^{m-1} \frac{\partial \hat{H}_j}{\partial \sigma} p_j(\hat{X}_{k,i+1}) \right) | Y_0 \right]. \tag{5.50}$$

As the realization $\hat{Y}_i$ is a function of the model parameters, at time t_{i+1}, the derivative with respect to the volatility in Equation (5.34) becomes

$$\frac{\partial \hat{H}_j(\hat{Y}_i, \sigma)}{\partial \sigma} = \frac{\partial \hat{H}_j(\sigma; \hat{Y}_i)}{\partial \sigma} + \frac{\partial \hat{H}_j(\hat{Y}_i; \sigma)}{\partial \hat{Y}_i} \frac{\partial \hat{Y}_i}{\partial \sigma}, \tag{5.51}$$

where $\frac{\partial \hat{Y}_i}{\partial \sigma}$ is known at the previous time point. Like simulating the Monte Carlo paths, the calculation of this derivative is done iteratively. Figure 5.9a compares the path-wise sensitivities obtained via Equations (5.37) and (5.51). Clearly, the path-wise derivative by the 7L scheme is very similar to the analytical solution. Figure 5.9b confirms that the ANN methodology computes a highly accurate Asian option Vega by means of the above path-wise sensitivity. Summarizing, the sensitivity with respect to model parameters can highly accurately be obtained from the trained ANNs. As the 7L-CDC scheme is composed of marginal and conditional collocation points, the above procedure of computing the path-wise sensitivity is also applicable to the variant 7L-CDC, by using the chain rule.



(a) Path-wise sensitivity

(b) Path-wise Vega

Figure 5.9: Path-wise estimator of Vega: Exact Vega is calculated by means of the central finite difference. The parameters are $Y_0 = 1.0$, $r = 0.05$, $\tilde{K} = Y_0$, $\sigma = 0.3$, $\Delta t = 1.0$, $N_b = 4$, $T = \Delta t \times N_b = 4.0$.

BERMUDAN OPTION VALUATION

When dealing with so-called Bermudan options, the option contract holder has the right (but not the obligation) to exercise the option contract at a finite number of pre-specified dates up to final time T. At an exercise date, when the holder

decides to exercise the Bermudan option, she immediately obtains the current payoff value of the contract. Alternatively, she may also wait until the next exercise opportunity. The Bermudan option can be exercised at the following set of exercise dates, $\{t_0, t_1, ..., t_{N_b}\}$, with a constant time difference, $\Delta t = t_i - t_{i-1}$, for any $0 < i \leq N_b$.

In this experiment, we compare the performance of the new 7L-CDC discretization scheme with a classical scheme. Valuation of the Bermudan option will take place by means of the well-known Longstaff-Schwartz Monte Carlo method (LSMC) [150], a least squares Monte Carlo method. Here the 7L scheme Longstaff-Schwartz algorithm is presented in Algorithm III.

The difference between a large time step simulation and a classical simulation, like the Milstein scheme, is that a classical scheme requires additional time steps to be taken between the early-exercise dates of the Bermudan option, while with the 7L-CDC scheme, we can perform one-step Monte Carlo simulation without any intermediate grid points between adjacent early-exercise dates.

We also assume here that the underlying stock price follows Geometric Brownian motion, as in Equation (5.35), under the risk-neutral measure, with $\mu \equiv r$. A Bermudan put option, with risk-free interest rate $r = 0.1$, pay-off function $V(t_j) = \max(K - Y(t_j), 0)$ with strike price $K = 1.1$ and initial stock price $Y_0 = 1.0$, is priced based on $M = 100,000$ Monte Carlo paths. The matrix size within the 7L-CDC scheme is set to $N_b \times 5 \times 5$. The terminal time is $T = \Delta t \times M_B$ with a constant time step Δt. The random seed is chosen to be zero when drawing random numbers. We compare the relative errors $|\frac{V^{ref}(t) - V(t_0)}{V^{ref}(t_0)}|$, where $V^{ref}(t_0)$ is computed with the help of a Monte Carlo method based on the exact simulation of GBM (5.36).

As shown in Table 5.5, the option prices based on the 7L-CDC Monte Carlo simulation are highly satisfactory, and the related error does not increase with larger time steps Δt. In contrast, a larger time step gives rise to significant pricing errors, in the case of the Milstein discretization.

Algorithm III: 7L scheme Longstaff-Schwartz algorithm

1. Divide the time horizon into N_b intervals.
2. Simulate M stock price paths $\hat{Y}_{i,j}$ ($0 \le i \le N_b$, $1 \le j \le M$), using the 7L-CDC methodology;
3. Price the Bermudan option by means of the Longstaff-Schwartz Monte Carlo method:
   (a) At terminal time T_{N_b}, calculate the payoff $\hat{V}_{N_b,j} = V\left(Y_{N_b,j}\right)$ for all paths j, where $V(\cdot)$ is the payoff function.
   (b) Perform a backward recursion, from $i = N_b - 1$ until $i = 0$ as follows:
   (c) Compute the discounted continuation value at time t_i, i.e.,

$$\hat{\eta}_{i,j} := \mathrm{e}^{-r\Delta t}\hat{V}_{i+1,j} \tag{5.52}$$

   (d) Perform least squares regression at time t_i, based on the cross-sectional information $\hat{Y}_{i,j}$ and $\hat{\eta}_{i,j}$ to estimate the conditional expectation function,

$$\bar{\eta}_i(\hat{Y}) = \sum_{k=1}^{M_k} \beta_k B_k(\hat{Y}) \tag{5.53}$$

   where M_k is the number of basis functions $B_k(S)$ (polynomial basis, here, $M_k = 3$), and the coefficients β_k are constant over different paths j. Note that only in-the-money paths are considered in Equations (5.52) and (5.53),
   (e) For each path j, compare the immediate exercise value $V\left(\hat{Y}_{i,j}\right)$ with the estimated continuation value $\bar{\eta}_i\left(\hat{Y}_{i,j}\right)$: If $V\left(\hat{Y}_{i,j}\right) \ge \bar{\eta}_i\left(\hat{Y}_{i,j}\right)$, then $\hat{V}_{i,j} = V\left(\hat{Y}_{i,j}\right)$; else $\hat{V}_{i,j} = \bar{\eta}_i\left(\hat{Y}_{i,j}\right)$.
4. Calculate the option price $V(t_0)$ at the initial time,

$$V(t_0) = \frac{1}{M}\sum_{j=1}^{M} \hat{V}_{0,j}. \tag{5.54}$$

Remark. *In principle, a sample value $\hat{Y}_i$ can be any rational number. So, a path*

Table 5.5: Bermudan put option prices based on large time step Monte Carlo simulations.

	method	$\Delta t = 1.0$, N_b=4	$\Delta t = 0.5$, N_b=4	$\Delta t = 0.5$, N_b=8
σ=0.30	Analytic MC	0.15213858(0.00%)	0.14620214(0.00%)	0.16161876(0.00%)
	Milstein MC	0.13872771(8.81%)	0.14065252(3.80%)	0.15429369(4.53%)
	7L-CDC	0.15234901(0.14%)	0.14648443(0.19%)	0.16196264(0.21%)
σ=0.40	Analytic MC	0.21459038(0.00%)	0.19552454(0.00%)	0.22340304(0.00%)
	Milstein MC	0.19598488(8.67%)	0.18790933(3.89%)	0.21297732(4.67%)
	7L-CDC	0.21474619(0.07%)	0.19590733(0.20%)	0.22389360(0.22%)

value may reach a larger stock price than the prescribed upper bound in Table 5.1. The probability of reaching boundaries of the training data set becomes high particularly when the volatility is large. We call the stock prices outside the training interval for Y_0 outliers. *When these outliers are used in the trained ANN, the approximation accuracy is not guaranteed due to the error in ANN extrapolation. Outliers did not appear in the experiments of Table 5.5. As a method to avoid the appearance of outliers, we may scale the asset price, to remove the dependence on the initial value.*

For example, GBM can be scaled, using the change of variables, as follows,

$$\bar{Y}(t) = \frac{Y(t)}{Y(t_0)\mathrm{e}^{rt}}.$$

Using Itô's lemma, we have a drift-less process,

$$\mathrm{d}\bar{Y}(t) = \bar{Y}(t)\sigma\mathrm{d}W,$$

where the initial value $\bar{Y}_0 = 1.0$. The following formula returns the original variable,

$$Y(t) = \bar{Y}(t)Y(t_0)\mathrm{e}^{rt}.$$

In such case, scaling guarantees a fixed initial value, for example, $Y_0 = 1.0$.

5.6. CONCLUSION

We develop a data-driven numerical solver for stochastic differential equations, by which large time step simulations can be carried out accurately in the sense of strong convergence. With a combination of artificial neural networks and the stochastic collocation Monte Carlo method, a small number of stochastic collocation points are learned by the ANN to approximate a nonlinear function which can be used to compute the unknown collocation points. Theoretical analysis indicates that the numerical error is controllable and does not increase when the simulation time step increases.

There are several advantages to the proposed approach. The powerful expressive ability of neural networks enables the ANNs to accurately approximate stochastic collocation points. The compression-decompression method reduces the computational costs, so that the numerical method can be applied in practice. In finance, the proposed big time step methodology will be highly beneficial for the generation for path-dependent financial option contracts or in risk management applications.

The introduced methodology can be extended for solving higher-dimensional or more involved SDE dynamics. Non-Markovian processes may also be solved with a large time step by the proposed ANN method, where the conditional collocation points are also dependent on past realizations. Fractional Brownian motion [151] forms a relevant example, which is used for the simulation of rough volatility in finance [152]. In such a context, advanced variants of fully connected neural networks, e.g., recurrent neural networks (RNN) or long short-term memory (LSTM) networks [see a review in 153], are recommended when approximating the nonlinear transition probability function, for example, Equation (5.13).

6

CONCLUSIONS AND OUTLOOK

6.1. CONCLUSIONS

In this dissertation, supervised learning techniques have been presented to address some computational challenges in the field of quantitative finance, for example, regarding option pricing, model calibration and Monte Carlo simulation. As powerful function approximators, deep neural networks are used to either enhance the efficiency of some classical numerical methods or to replace the original numerical method. It is the decoupling of the ANN off-line training and on-line prediction phases that results in flexible and highly efficient numerical methods.

In Chapter 2, artificial neural networks were successfully used for the fast and efficient pricing of financial derivatives and the computation of the implied volatility. To accurately approximate the inverse function for the implied volatility by means of the ANNs, steep-gradient approximation issues were handled by a gradient-squashing technique. The inherent parallel properties of the ANN approximation increased the computational speed by several orders of magnitude, as compared to the traditional numerical methods. The proposed guidelines turned out to be useful for designing, training and testing of neural networks, and resulted in robust approximations. This chapter demonstrated the feasibility of learning a deep neural network to quickly solve option pricing problems based on parametric asset models.

In Chapter 3, we developed the Calibration Neural Network (CaNN) to calibrate high-dimensional asset pricing models, based on available option prices, in a fast and efficient way. The CaNN is formed by a two-stage procedure, which includes a forward pass for fast option valuation, and a backward pass for the efficient and robust calibration. The CaNN addressed the drawback of long com-

puting times when a global optimization technique is employed for the optimization of the objective function during calibration. The results of calibrating the Heston and Bates stochastic volatility models suggested that it takes around one second to find a global solution on a CPU, and even less time on a GPU. The methodology is highly promising for practical use in calibration at financial institutions.

In Chapter 4, we dealt with the implied volatility and implied dividend yield from American options, where early-exercise features gave rise to numerical issues (e.g., the option Vega is equal to zero in certain regions) when inverting the American option pricing models. The neural network based method (mainly the CaNN) can accurately approximate the Black-Scholes implied volatility and the dividend yield for American options, even when multiple early-exercise regions appear due to negative interest rates. Because of the decoupling of the training and predicting phases, we found the effective definition domain during the off-line phase, using two criteria to ensure properly training of the ANNs. When determining the implied volatility and dividend yield at the same time, without getting stuck in the early-exercise regions, the CaNN with the global optimization explored the solution space globally in a short time. This showed the ability of the CaNN to deal with nontrivial cases.

In Chapter 5, large time step discretization Monte Carlo simulations of SDEs were performed using the 7L scheme. This scheme used the ANNs to learn the stochastic collocation points, after which the stochastic collocation Monte Carlo method generated the SDE paths. The related path-wise sensitivity could also be computed in a convenient and accurate way. Accompanied by theoretical analysis, the numerical error in the sense of strong convergence appeared controllable and did not grow when the simulation time step increased. Compared to an analytical solution (here for the Geometry Brownian Motion and the Ornstein-Uhlenbeck process), the 7L scheme appeared superior to the Euler-Maruyama and Milstein schemes. In the finance context, the 7L scheme was used for fast financial derivative pricing, like the valuation of exotic options.

6.2. OUTLOOK

In our current work, the ANNs are trained on a sufficiently large data set generated by the given model, and those data represent the model constraints in an implicit way. As an interesting topic, when training the ANNs, the loss function can incorporate explicit "physical" constraints, e.g., arbitrage-free conditions for option pricing.

When multiple solutions appear during calibration, an interesting topic is how to choose a suitable solution, for example, by regularizing the original objective function. In addition, the Calibration Neural Network could be used to

obtain a good initial guess for other calibration algorithms.

Regarding the 7L scheme, the parallelization on GPUs may further improve the computational speed. We also expect the generalisation towards accurate large time-step discretizations for systems of SDEs, for instance, the Heston or SABR stochastic volatility model.

As another outlook, Multilevel Monte Carlo (MLMC) methods, as developed by [154, 155], may achieve a convergence acceleration using our large time step accurate discretisation schemes. It is well-known that the strong convergence properties of SDE discretizations impact the efficiency of the MLMC methods.

The combination of differential equations (e.g., ODEs, SDEs) and artificial neural networks may further improve scientific computing or data-driven mathematical modelling. For example, the coefficients of an ODE could be expressed in the form of a neural network function, so that extra (prior) knowledge would be enforced into the ANN-and-ODE system, which could approximate a wider range of functions.

Moreover, next to supervised learning, the other two deep learning methodologies, i.e., unsupervised learning and reinforcement learning, also have good potential, like solving highly complicated models, in computational finance.

REFERENCES

REFERENCES

[1] K. Marko and T. K. Rajesh, *Big data and ai strategies: Machine learning and alternative data approach to investing,* J.P. Morgan Securities LLC , 1–280 (2017).

[2] K. Phoon and F. Koh, *Robo-Advisors and Wealth Management,* The Journal of Alternative Investments **20**, 79–94 (2017), https://jai.pm-research.com/content/20/3/79.full.pdf .

[3] Y. LeCun, Y. Bengio, and G. Hinton, *Deep learning,* Nature **521**, 436–444 (2015).

[4] W. S. McCulloch and W. Pitts, *A logical calculus of the ideas immanent in nervous activity,* Bulletin of Mathematical Biophysics **5**, 115–133 (1943).

[5] K. Hornik, M. Stinchcombe, and H. White, *Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks,* Neural Networks **3**, 551–560 (1990).

[6] H. Lin and S. Jegelka, *ResNet with one-neuron hidden layers is a Universal Approximator,* arXiv e-prints , arXiv:1806.10909 (2018).

[7] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (MIT Press, 2016).

[8] S. Liang and R. Srikant, *Why deep neural networks?* arXiv:1610.04161 (2016), arXiv:1610.04161 .

[9] Y. LeCun, K. Kavukcuoglu, and C. Farabet, *Convolutional networks and applications in vision,* in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems* (2010) pp. 253–256.

[10] Z. C. Lipton, J. Berkowitz, and C. Elkan, *A Critical Review of Recurrent Neural Networks for Sequence Learning,* arXiv e-prints , arXiv:1506.00019 (2015).

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

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

[13] I. Lagaris, A. Likas, and D. Fotiadis, *Artificial neural networks for solving ordinary and partial differential equations,* IEEE Transactions on Neural Networks **9**, 987–1000 (1998).

[14] J. Sirignano and K. Spiliopoulos, *DGM: A deep learning algorithm for solving partial differential equations,* Journal of Computational Physics (2018).

[15] N. P. Jouppi and et al., *In-Datacenter Performance Analysis of a Tensor Processing Unit,* arXiv e-prints , arXiv:1704.04760 (2017).

[16] J. M. Hutchinson, A. W. Lo, and T. Poggio, *A Nonparametric Approach to Pricing and Hedging Derivative Securities Via Learning Networks,* The Journal of Finance **49**, 851–889 (1994).

[17] J. Yao, Y. Li, and C. L. Tan, *Option price forecasting using neural networks,* Omega **28**, 455–466 (2000).

[18] R. Gencay and M. Qi, *Pricing and hedging derivative securities with neural networks: Bayesian regularization, early stopping, and bagging,* IEEE Transactions on Neural Networks **12**, 726–734 (2001).

[19] R. Garcia and R. Gençay, *Pricing and hedging derivative securities with neural networks and a homogeneity hint,* Journal of Econometrics **94**, 93 – 115 (2000).

[20] C. Dugas, Y. Bengio, F. Bélisle, C. Nadeau, and R. Garcia, *Incorporating second-order functional knowledge for better option pricing,* in *Proceedings of the 13th International Conference on Neural Information Processing Systems*, NIPS'00 (MIT Press, Cambridge, MA, USA, 2001) pp. 451–457.

[21] Y. Yang, Y. Zheng, and T. Hospedales, *Gated neural networks for option pricing: Rationality by design,* in *The Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)* (2017) pp. 52–58.

[22] J. Han, A. Jentzen, and W. E, *Solving high-dimensional partial differential equations using deep learning,* Proceedings of the National Academy of Sciences **115**, 8505–8510 (2018), https://www.pnas.org/content/115/34/8505.full.pdf .

[23] W. E, J. Han, and A. Jentzen, *Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations,* Communications in Mathematics and Statistics **5**, 349–380 (2017).

[24] C. Beck, W. E, and A. Jentzen, *Machine learning approximation algorithms for high-dimensional fully nonlinear partial differential equations and second-order backward stochastic differential equations,* ArXiv:abs/1709.05963 (2017), arXiv:1709.05963 [math.NA] .

[25] J. Sirignano and K. Spiliopoulos, *Stochastic gradient descent in continuous time,* SIAM Journal on Financial Mathematics **8**, 933–961 (2017), https://doi.org/10.1137/17M1126825 .

[26] J. Fan and L. Mancini, *Option Pricing With Model-Guided Nonparametric Methods,* Journal of the American Statistical Association **104**, 1351–1372 (2009).

[27] J. Hesthaven and S. Ubbiali, *Non-intrusive reduced order modeling of nonlinear problems using neural networks,* Journal of Computational Physics **363**, 55 – 78 (2018).

[28] M. Raissi and G. E. Karniadakis, *Hidden physics models: Machine learning of nonlinear partial differential equations,* Journal of Computational Physics **357**, 125 – 141 (2018).

[29] J. Tompson, K. Schlachter, P. Sprechmann, and K. Perlin, *Accelerating eulerian fluid simulation with convolutional networks,* ArXiv:abs/1607.03597 (2016), arXiv:1607.03597 .

[30] R. Cont and J. da Fonseca, *Dynamics of implied volatility surfaces,* Quantitative Finance **2**, 45–60 (2002).

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

[32] F. Fang and C. W. Oosterlee, *A Novel Pricing Method for European Options Based on Fourier-Cosine Series Expansions,* SIAM Journal on Scientific Computing **31**, 826–848 (2009).

[33] P. Jäckel, *Let's Be Rational,* Wilmott **2015**, 40–53 (2015).

[34] C. J. Corrado and T. W. Miller, *A note on a simple, accurate formula to compute implied standard deviations,* Journal of Banking Finance **20**, 595 – 603 (1996).

[35] D. M. Chance, *A Generalized Simple Formula to Compute the Implied Volatility,* Financial Review **31**, 859–867 (1996).

[36] M. Brenner and M. G. Subrahmanyam, *A Simple Formula to Compute the Implied Standard Deviation,* Financial Analysts Journal **44**, 80–83 (1988).

[37] R. P. Brent, *Algorithms for minimization without derivatives,* (NJ: Prentice-Hall, 1973) Chap. Chapter 4: An algorithm with guaranteed convergence for finding a zero of a function.

[38] F. Fang and C. W. Oosterlee, *A novel pricing method for European options based on Fourier-Cosine series expansions,* SIAM Journal on Scientific Computing **31**, 826–848 (2009).

[39] H. N. Mhaskar, *Neural networks for optimal approximation of smooth and analytic functions,* Neural Computation **8**, 164–177 (1996).

[40] V. Maiorov and A. Pinkus, *Lower bounds for approximation by mlp neural networks,* Neurocomputing **25**, 81 – 91 (1999).

[41] D. Yarotsky, *Error bounds for approximations with deep relu networks,* Neural Networks **94**, 103 – 114 (2017).

[42] S. Ruder, *An overview of gradient descent optimization algorithms,* arXiv e-prints , arXiv:1609.04747 (2016).

[43] D. P. Kingma and J. Ba, *Adam: A Method for Stochastic Optimization,* arXiv e-prints , arXiv:1412.6980 (2014).

[44] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, *Algorithms for Hyper-Parameter Optimization,* Advances in Neural Information Processing Systems (NIPS) , 2546–2554 (2011).

[45] J. Bergstra and Y. Bengio, *Random Search for Hyper-Parameter Optimization,* Journal of Machine Learning Research **13**, 281–305 (2012).

[46] J. Snoek, H. Larochelle, and R. P. Adams, *Practical bayesian optimization of machine learning algorithms,* in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'12 (Curran Associates Inc., Red Hook, NY, USA, 2012) p. 2951–2959.

[47] L. N. Smith, *Cyclical learning rates for training neural networks,* ArXiv:abs/1506.01186 (2015), arXiv:1506.01186 .

[48] I. Loshchilov and F. Hutter, *SGDR: stochastic gradient descent with restarts,* arXiv:abs/1608.03983 (2016), arXiv:1608.03983 .

[49] M. D. McKay, R. J. Beckman, and W. J. Conover, *A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code,* Technometrics **21**, 239 (1979).

[50] G. Hinton, O. Vinyals, and J. Dean, *Distilling the Knowledge in a Neural Network,* ArXiv:abs/1503.02531 (2015), arXiv:1503.02531 [stat.ML] .

[51] A. G. Baydin, B. A. Pearlmutter, and A. A. Radul, *Automatic differentiation in machine learning: a survey,* arXiv:abs/1502.05767 (2015), arXiv:1502.05767 .

[52] I. Bouchouev and V. Isakov, *The inverse problem of option pricing,* Inverse Problems **13**, L11–L17 (1997).

[53] Z.-C. Deng, J.-N. Yu, and L. Yang, *An inverse problem of determining the implied volatility in option pricing,* Journal of Mathematical Analysis and Applications **340**, 16 – 31 (2008).

[54] M. C. Kennedy and A. O'Hagan, *Bayesian calibration of computer models,* Journal of the Royal Statistical Society: Series B (Statistical Methodology) **63**, 425–464 (2001).

[55] R. Cont, *Inverse problems in option pricing: a statistical approach using minimal entropy random mixtures,* `https://studies2.hec.fr/jahia/webdav/site/hec/shared/site/statsinthechateau/sacces_anonyme/Lectures/Cont.pdf` (Accessed on 17/03/2019).

[56] I. Daubechies, M. Defrise, and C. De Mol, *An iterative thresholding algorithm for linear inverse problems with a sparsity constraint,* Communications on Pure and Applied Mathematics **57**, 1413–1457 (2004), https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpa.20042 .

[57] M. Gilli and E. Schumann, *Calibrating option pricing models with heuristics,* in *Natural Computing in Computational Finance: Volume 4*, edited by A. Brabazon, M. O'Neill, and D. Maringer (Springer Berlin Heidelberg, Berlin, Heidelberg, 2012) pp. 9–37.

[58] C. Homescu, *Implied volatility surface: Construction methodologies and characteristics,* arXiv e-prints , arXiv:1107.1834 (2011).

[59] R. Storn and K. Price, *Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces,* Journal of Global Optimization **11**, 341–359 (1997).

[60] S. Liu, C. W. Oosterlee, and S. M. Bohte, *Pricing options and computing implied volatilities using neural networks,* Risks **7** (2019), 10.3390/risks7010016.

[61] T. Poggio, H. Mhaskar, L. Rosasco, B. Miranda, and Q. Liao, *Why and when can deep-but not shallow-networks avoid the curse of dimensionality: A review,* International Journal of Automation and Computing **14**, 503–519 (2017).

[62] J. D. Spiegeleer, D. B. Madan, S. Reyners, and W. Schoutens, *Machine learning for quantitative finance: fast derivative pricing, hedging and fitting,* Quantitative Finance **18**, 1635–1643 (2018), https://doi.org/10.1080/14697688.2018.1495335 .

[63] B. Horvath, A. Muguruza, and M. Tomas, *Deep learning volatility,* arXiv e-prints , arXiv:1901.09647 (2019).

[64] G. Dimitroff, D. Röder, and C. P. Fries, *Volatility model calibration with convolutional neural networks,* http://dx.doi.org/10.2139/ssrn.3252432 (2018), http://dx.doi.org/10.2139/ssrn.3252432.

[65] A. Hernandez, *Model calibration with neural networks,* http://dx.doi.org/10.2139/ssrn.2812140 (2016).

[66] A. Hirsa, T. Karatas, and A. Oskoui, *Supervised Deep Neural Networks (DNNs) for Pricing/Calibration of Vanilla/Exotic Options Under Various Different Processes,* arXiv e-prints , arXiv:1902.05810 (2019).

[67] I. Vollrath and J. Wendland, *Calibration of interest rate and option models using differential evolution,* SSRN Electronic Journal (2009), 10.2139/ssrn.1367502.

[68] A. Slowik and M. Bialko, *Training of artificial neural networks using differential evolution algorithm,* in *2008 Conference on Human System Interactions* (2008) pp. 60–65.

[69] D. S. Bates, *Jumps and stochastic volatility: Exchange rate processes implicit in Deutsche mark options,* The Review of Financial Studies **9**, 69–107 (1996), http://oup.prod.sis.lan/rfs/article-pdf/9/1/69/24435185/090069.pdf .

[70] F. Guillaume and W. Schoutens, *Calibration risk: Illustrating the impact of calibration risk under the Heston model,* Review of Derivatives Research **15**, 57–79 (2012).

[71] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (MIT Press, 2016) `http://www.deeplearningbook.org`.

[72] P. Gauthier and P.-Y. H. Rivaille, *Fitting the smile, smart parameters for SABR and Heston,* SSRN Electronic Journal (2009), 10.2139/ssrn.1496982.

[73] M. Forde, A. Jacquier, and A. Mijatović, *Asymptotic formulae for implied volatility in the Heston model,* Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences **466**, 3593–3620 (2010).

[74] Y. Cui, S. del Baño Rollin, and G. Germano, *Full and fast calibration of the Heston stochastic volatility model,* European Journal of Operational Research **263**, 625–638 (2017).

[75] P. Izmailov, D. Podoprikhin, T. Garipov, D. Vetrov, and A. G. Wilson, *Averaging weights leads to wider optima and better generalization,* arXiv e-prints , arXiv:1803.05407 (2018).

[76] P. Büchel, M. Kratochwil, M. Nagl, and D. Rösch, *Deep calibration of financial models: Turning theory into practice,* (2020), available at SSRN:`https://ssrn.com/abstract=3667070`.

[77] K. E. Erkan, *European option pricing under the rough Heston model using the COS method,* MSc Thesis (2020).

[78] S. W. Seo and J. S. Kim, *The information content of option-implied information for volatility forecasting with investor sentiment,* Journal of Banking Finance **50**, 106 – 120 (2015).

[79] P. Christoffersen, K. Jacobs, and B. Y. Chang, *Handbook of Economic Forecasting*, edited by G. Elliott and A. Timmermann, Handbook of Economic Forecasting, Vol. 2 (Elsevier, 2013) pp. 581 – 656.

[80] J. C.Hull, *Derivative Securities: Options,* `https://www.math.nyu.edu/faculty/avellane/DSLecture3.pdf`, lecture 3, Accessed: 2019-06-30.

[81] A. Fodor, D. L. Stowe, and J. D. Stowe, *Option Implied Dividends Predict Dividend Cuts: Evidence from the Financial Crisis,* Journal of Business Finance and Accounting **44**, 755–779 (2017).

[82] J. F. Bilson, S. B. Kang, and H. Luo, *The term structure of implied dividend yields and expected returns,* Economics Letters **128**, 9 – 13 (2015).

[83] O. Burkovska, K. Glau, M. Mahlstedt, and B. Wohlmuth, *Complexity reduction for calibration to American options,* Journal of Computational Finance **23**, 25–60 (2019).

[84] O. Burkovska, M. Gass, K. Glau, M. Mahlstedt, W. Schoutens, and B. Wohlmuth, *Calibration to American options: numerical investigation of the de-Americanization method,* Quantitative Finance **18**, 1091–1113 (2018).

[85] Y. Achdou, G. Indragoby, and O. Pironneau, *Volatility calibration with American options,* Methods and Applications of Analysis **11**, 533–556 (2004).

[86] G. W. Kutner, *Determining the Implied Volatility for American Options Using the QAM,* The Financial Review **33**, 119–30 (1998).

[87] P. Carr and L. Wu, *Stock options and credit default swaps: A joint framework for valuation and estimation,* Journal of Financial Econometrics **8**, 409–449 (2009).

[88] R. Lagnado and S. Osher, *A Technique for Calibrating Derivative Security Pricing Models: Numerical Solution of an Inverse Problem,* Journal of Computational Finance **1** (1997), 10.21314/JCF.1997.002.

[89] M. Nardon and P. Pianca, *Extracting information on implied volatilities and discrete dividends from American options prices,* Journal of Modern Accounting and Auditing **9**, 112–129 (2013).

[90] L. H. Frankena, *Pricing and hedging options in a negative interest rate environment,* Master's thesis, Delft University of Technology, the Netherlands (2016).

[91] M. De Donno, Z. Palmowski, and J. Tumilewicz, *Double continuation regions for American and Swing options with negative discount rate in Lévy models,* Mathematical Finance **0**, 1–32, https://onlinelibrary.wiley.com/doi/pdf/10.1111/mafi.12218 .

[92] S. Liu, C. W. Oosterlee, and S. M. Bohte, *Pricing Options and Computing Implied Volatilities using Neural Networks,* Risks **7** (2019), 10.3390/risks7010016.

[93] V. Lokeshwar, V. Bhardawaj, and S. Jain, *Neural network for pricing and universal static hedging of contingent claims,* arXiv e-prints , arXiv:1911.11362 (2019).

[94] 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 **9**, 9 (2019).

[95] H. Bühler, L. Gonon, J. Teichmann, and B. Wood, *Deep Hedging,* arXiv e-prints , arXiv:1802.03042 (2018).

[96] S. Becker, P. Cheridito, and A. Jentzen, *Deep optimal stopping,* Journal of Machine Learning Research **20**, 1–25 (2019).

[97] J. Ruf and W. Wang, *Neural networks for option pricing and hedging: a literature review,* arXiv e-prints , arXiv:1911.05620 (2019).

[98] J. Sirignano and K. Spiliopoulos, *DGM: A deep learning algorithm for solving partial differential equations,* Journal of Computational Physics **375**, 1339 – 1364 (2018).

[99] S. Becker, P. Cheridito, and A. Jentzen, *Pricing and Hedging American-Style Options with Deep Learning,* Journal of Risk and Financial Management **13** (2020), 10.3390/jrfm13070158.

[100] Y. Chen and J. W. L. Wan, *Deep neural network framework based on backward stochastic differential equations for pricing and hedging American options in high dimensions,* Quantitative Finance **0**, 1–23 (2020), https://doi.org/10.1080/14697688.2020.1788219 .

[101] B. Salvador, C. W. Oosterlee, and R. van der Meer, *Financial option valuation by unsupervised learning with artificial neural networks,* arXiv e-prints , arXiv:2005.12059 (2020).

[102] Q. Cao, *Computation of implied dividend based on option market data,* Master's thesis, Delft University of Technology, the Netherlands (2005).

[103] J. Kragt, *Option Implied Dividends,* (2017), available at SSRN: `https://ssrn.com/abstract=2980275`.

[104] P. Carr, R. Jarrow, and R. Myneni, *Alternative characterizations of American put options,* Mathematical Finance **2**, 87–106 (1992).

[105] A. Battauz, M. De Donno, and A. Sbuelz, *Real Options and American Derivatives: The Double Continuation Region,* Management Science **61**, 1094–1107 (2015).

[106] R. Guerrero, *Essays on implied dividends,* PhD Thesis (2017).

[107] Y. K. Kwok, *Mathematical Models of Financial Derivatives*, 2nd ed. (Springer Verlag, 2008) chapter 5.

[108] W. Li and S. Chen, *The early exercise premium in American options by using nonparametric regressions,* International Journal of Theoretical and Applied Finance **21**, 1850039 (2018), https://doi.org/10.1142/S0219024918500395 .

[109] M. Engström and L. Nordén, *The early exercise premium in American put option prices,* Journal of Multinational Financial Management **10**, 461 – 479 (2000).

[110] G. Poitras, C. Veld, and Y. Zabolotnyuk, *European Put-Call Parity and the Early Exercise Premium for American Currency Options,* Multinational Finance Journal **13**, 39–54 (2009).

[111] B. Zhang and C. W. Oosterlee, *Fourier Cosine Expansions and Put-Call Relations for Bermudan Options,* in *Numerical Methods in Finance* (Springer Berlin Heidelberg, Berlin, Heidelberg, 2012) pp. 323–350.

[112] F. Fang and C. W. Oosterlee, *Pricing early-exercise and discrete barrier options by Fourier-cosine series expansions,* Numerische Mathematik **114**, 27 (2009).

[113] Y. LeCun, Y. Bengio, and G. Hinton, *Deep learning,* Nature **521**, 436–444 (2015).

[114] R. Maziar, P. Perdikaris, and G. E. Karniadakis, *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations,* Journal of Computational Physics **378**, 686–707 (2019).

[115] S. Becker, P. Cheridito, and A. Jentzen, *Deep Optimal Stopping,* Journal of Machine Learning Research **20**, 1–25 (2019).

[116] C. Rackauckas, Y. Ma, J. Martensen, C. Warner, K. Zubov, R. Supekar, D. Skinner, A. Ramadhan, and A. Edelman, *Universal Differential Equations for Scientific Machine Learning,* arXiv e-prints , arXiv:2001.04385 (2020).

[117] L. A. Grzelak, J. Witteveen, M. Suarez-Taboada, and C. W. Oosterlee, *The stochastic collocation Monte Carlo sampler: highly efficient sampling from expensive distributions,* Quantitative Finance **19**, 339–356 (2019), https://no$_d$oi.org/10.1080/14697688.2018.1459807 .

[118] A. Pinkus, *Approximation theory of the MLP model in neural networks,* Acta Numerica **8**, 143–195 (1999).

[119] E. Platen, *An introduction to numerical methods for stochastic differential equations,* Acta Numerica **8**, 197–246 (1999).

[120] Y. Bar-Sinai, S. Hoyer, J. Hickey, and M. P. Brenner, *Learning data-driven discretizations for partial differential equations,* Proceedings of the National Academy of Sciences **116**, 15344–15349 (2019), https://www.pnas.org/content/116/31/15344.full.pdf .

[121] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, *Neural Ordinary Differential Equations,* arXiv e-prints , arXiv:1806.07366 (2018).

[122] W. Grathwohl, R. T. Q. Chen, J. Bettencourt, I. Sutskever, and D. Duvenaud, *FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models,* arXiv e-prints , arXiv:1810.01367 (2018).

[123] X. Li, T.-K. L. Wong, R. T. Q. Chen, and D. K. Duvenaud, *Scalable gradients and variational inference for stochastic differential equations,* in *Proceedings of The 2nd Symposium on Advances in Approximate Bayesian Inference*, Proceedings of Machine Learning Research, Vol. 118, edited by C. Zhang, F. Ruiz, T. Bui, A. B. Dieng, and D. Liang (PMLR, 2020) pp. 1–28.

[124] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, *Generative adversarial nets,* in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14 (MIT Press, Cambridge, MA, USA, 2014) p. 2672–2680.

[125] Y. Xie, E. Franz, M. Chu, and N. Thuerey, *TempoGAN: A Temporally Coherent, Volumetric GAN for Super-Resolution Fluid Flow,* ACM Transactions on Graphics **37** (2018).

[126] L. Yang, D. Zhang, and G. E. Karniadakis, *Physics-Informed Generative Adversarial Networks for Stochastic Differential Equations,* arXiv e-prints , arXiv:1811.02033 (2018).

[127] I. Karatzas and S. E. Shreve, *Brownian Motion and Stochastic Calculus*, Graduate texts in mathematics (World Piblishing Company, 1988).

[128] H. Risken, *The Fokker-Planck Equation: Methods of Solution and Applications*, Springer series in synergetics (World Publishing Corporation, 1984).

[129] G. N. Milstein, *Approximate integration of stochastic differential equations,* Theory of Probability and Its Applications **19**, 557–562 (1975), https://no$_d$oi.org/10.1137/1119062 .

[130] R. H. Cameron and W. T. Martin, *The orthogonal development of nonlinear functionals in series of Fourier-Hermite functionals,* Annals of Mathematics **48**, 385–392 (1947).

[131] L. A. Grzelak, *The collocating local volatility framework – a fresh look at efficient pricing with smile,* International Journal of Computer Mathematics **96**, 2209–2228 (2019).

[132] F. N. Fritsch and R. E. Carlson, *Monotone piecewise cubic interpolation,* SIAM Journal on Numerical Analysis **17**, 238–246 (1980).

[133] J. P. Berrut and L. N. Trefethen, *Barycentric Lagrange Interpolation,* SIAM Review **46**, 501–517 (2004).

[134] Theodore J. Rivlin, *Chebyshev Polynomials: From Approximation Theory to Algebra and Number Theory*, Pure and Applied Mathematics: A Wiley Series of Texts, Monographs and Tracts (Wiley, 1990).

[135] M. Gaß, K. Glau, M. Mahlstedt, and M. Mair, *Chebyshev interpolation for parametric option pricing,* Finance and Stochastics **22**, 701–731 (2018).

[136] K. Glau, P. Herold, D. B. Madan, and C. Pötz, *The Chebyshev method for the implied volatility,* Journal of Computational Finance **23** (2019).

[137] K. Glau and M. Mahlstedt, *Improved error bound for multivariate Chebyshev polynomial interpolation,* International Journal of Computer Mathematics **96**, 2302–2314 (2019).

[138] L. Capriotti, *Fast Greeks by Algorithmic Differentiation,* Journal of Computational Finance **14**, 3–35 (2010).

[139] M. B. Giles and P. Glasserman, *Smoking adjoints: Fast Monte Carlo Greeks,* Risk **19**, 88–92 (2006).

[140] C. W. Oosterlee and L. A. Grzelak, *Mathematical Modeling and Computation in Finance* (World Scientific (EUROPE), 2019) https://worldscientific.com/no$_d$oi/pdf/10.1142/q0236 .

[141] Shashi Jain, Álvaro Leitao and Cornelis W. Oosterlee, *Rolling Adjoints: Fast Greeks along Monte Carlo scenarios for early-exercise options,* Journal of Computational Science **33**, 95–112 (2019).

[142] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, *Activation Functions: Comparison of trends in Practice and Research for Deep Learning,* arXiv e-prints , arXiv:1811.03378 (2018).

[143] X. Glorot and Y. Bengio, *Understanding the difficulty of training deep feedforward neural networks,* in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (PMLR, 2010) pp. 249–256.

[144] D. Yarotsky, *Error bounds for approximations with deep ReLU networks,* Neural Networks **94**, 103–114 (2017).

[145] H. Montanelli and Q. Du, *New Error Bounds for Deep ReLU Networks Using Sparse Grids,* SIAM Journal on Mathematics of Data Science **1**, 78–92 (2019).

[146] G. E. Uhlenbeck and L. Ornstein, *On the Theory of the Brownian Motion,* Physical Review **36**, 823–841 (1930).

[147] M. Broadie and O. Kaya, *Exact simulation of stochastic volatility and other affine jump diffusion processes,* Operations Research **54**, 217–231 (2006).

[148] A. Leitao, L. A. Grzelak, and C. W. Oosterlee, *On a one time-step Monte Carlo simulation approach of the SABR model: Application to European options,* Applied Mathematics and Computation **293**, 461–479 (2017).

[149] P. Glasserman, *Monte Carlo methods in financial engineering* (Springer, New York, 2004).

[150] F. A. Longstaff and E. S. Schwartz, *Valuing American Options by Simulation: A Simple Least-Squares Approach,* The Review of Financial Studies **14**, 113–147 (2015), https://academic.oup.com/rfs/article-pdf/14/1/113/24432078/113.pdf .

[151] B. B. Mandelbrot and J. W. Van Ness, *Fractional Brownian Motions, Fractional Noises and Applications,* SIAM Review **10**, 422–437 (1968).

[152] J. Gatheral, T. Jaisson, and M. Rosenbaum, *Volatility is rough,* Quantitative Finance **18**, 933–949 (2018).

[153] Y. Yu, X. Si, C. Hu, and J. Zhang, *A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures,* Neural Computation **31**, 1235–1270 (2019).

[154] M. B. Giles, *Multilevel Monte Carlo Path Simulation,* Operations Research **56**, 607–617 (2008).

[155] M. B. Giles, *Multilevel Monte Carlo methods,* Acta Numerica **24**, 259–328 (2015).

CURRICULUM VITÆ

Shuaiqiang LIU

18-Nov-1984	Born in Handan, Province Hebei, China.

EDUCATION

2004–2008	B.Sc in Mathematics and Applied Mathematics Northwestern Polytechnical University, Xi'an, China
2008–2011	M.Sc in Applied Mathematics Northwestern Polytechnical University, Xi'an, China
2010–2010	Visiting Student Technical University of Munich, Munich, Germany
2016–2020	Ph.D in Applied Mathematics Delft University of Technology, Delft, The Netherlands *Thesis*: Supervised deep learning in computational finance *Promotor:* Prof. dr. ir. Cornelis W. Oosterlee

WORK EXPERIENCE

2011-2016	Computational Fluid Dynamics Engineer, AECC Commercial Aircraft Engine CO. LTD, Shanghai, China
2016-2017	Chief Information Officer, Shuyun Puhui Digital Credit CO. LTD, Beijing, China

LIST OF PUBLICATIONS

- **Journal papers**

  1. **Shuaiqiang Liu**, Lech A. Grzelak and Cornelis W. Oosterlee. *The Seven-League Scheme: Deep learning for large time step Monte Carlo simulations of stochastic differential equations*, **under review**.
  2. **Shuaiqiang Liu**, Álvaro Leitao, Anastasia Borovykh, and Cornelis W. Oosterlee. *On a neural network to extract implied information from American options*, **under review**.
  3. **Shuaiqiang Liu**, Anastasia Borovykh, Lech A. Grzelak and Cornelis W. Oosterlee. *A neural network-based framework for financial model calibration*, Journal of Mathematics in Industry, **9 (9)**, 2019.
  4. **Shuaiqiang Liu**, Cornelis W. Oosterlee, and Sander M. Bohté. *Pricing Options and Computing Implied Volatilities using Neural Networks*, Risks, **7 (1)**, 16, 2019.

- **Proceedings or working papers**

  1. **Shuaiqiang Liu**, Álvaro Leitao, Anastasia Borovykh and Cornelis W. Oosterlee, *Machine Learning to Compute Implied Volatility from European/American Options Considering Dividend Yield*, Proceedings 2020 (the 3rd XoveTIC Conference, A Coruña, Spain), 54, 61.
  2. **Shuaiqiang Liu**, Cornelis W. Oosterlee and Sander M. Bohté. *Classify time series with space filling curves using deep Convolutional Neural Networks*, working paper, 2017.

List of Presentations

- **Oral Presentations**

  1. The 2020 ACM International Conference on AI in Finance (peer-reviewed, acceptance rate 41%), New York(online), USA, October 2020.
  2. Machine learning in quantitative finance and risk management, CWI, Amsterdam (online), July 2020.
  3. The 2nd international symposium on PDEs & Stochastic analysis in Mathematical Finance (Best Presentation), organized by University of Wollongong (Australia), January 2020.
  4. Centre for Translational Data Science (an invited talk), The University of Sydney, Sydney, Australia, December 2019.
  5. The Quantitative Methods in Finance 2019 Conference, Sydney, Australia, December 2019.
  6. The 3rd International Conference on Computational Finance, A Coruña, Spain, July 2019.
  7. SIAM Conference on Financial Mathematics & Engineering, Toronto, Canada, June 2019.
  8. The 18th Winter school on Mathematical Finance, Lunteren, The Netherlands, January 2019.

- **Poster Presentations**

  1. The 44th Woudschoten conference, Zeist, The Netherlands, 2019.
  2. The 43th Woudschoten conference, Zeist, The Netherlands, 2018.

ACKNOWLEDGEMENTS

This dissertation concludes my PhD research work that was done between December 2016 and November 2020 in TU Delft. Herewith I would like to acknowledge those people from whom I benefited over the past four years, although the names on the list are too many to mention everyone.

My most sincere gratitude goes to my supervisor Prof. Cornelis(Kees) Oosterlee. He offered me this valuable opportunity of pursuing a PhD, for which I will always be grateful. Without his constant guidance and help, the dissertation would have never been successfully finished. Pearls are everywhere but not the same as the eyes. Kees has the eyes, as well as the patience, to let his students shine. I have much freedom to explore new ideas or speak my mind. Kees is always the anchor of me when things are difficult. His positive attitude (e.g., doing his best, working hard, trust), professional supervision, broad and deep knowledge motivated and helped me to overcome those difficulties. When it was day or night, I always got his swift and helpful feedback, which kept my work going on without any interruption. Even when we had to stay at home during the COVID-19 pandemic lockdown, our research still went smoothly. Working together with Kees is more than enjoyable and fruitful. In addition to work, I have many good memories of him in life. It is unbelievable that he recognized me among many people at a glance on his way to the lecture room, when I first arrived and was sitting in the public area of the EWI building. Kees also gave me help with life in the Netherlands. Words are not enough to express my appreciation.

I am indebted to Prof. Sander Bohté, who guided me to the field of machine learning over my first two years, and Dr. Lech Grzelak, who provided many useful comments and ideas during my PhD research. Special thanks also go to my collaborators, Álvaro and Anastasia, for their contributions to this dissertation. Thank Prof. Pasquale Cirillo for being my co-promotor when he worked at TU Delft. Thank Dr. Damien Ackerer (Swissquote) for bringing into practice some of the work involved in this dissertation.

I am sincerely thankful to the members of my defence committee board for reading the dissertation and participating in my defence.

I learnt a lot from interaction with (former) members of Kees' excellent research group, for example, Anton, Andrea, Beatriz, Bowen, Bin, Dan, Fei, Fang, Jing, Kiwai, Kristoffer, Linlin, Nikolaj, Peiyao, Prashant, Qian, Thomas, ZaZa. I also had numerous useful discussions with Kees' MSc or Bachelor students, es-

pecially Daan, Erkan, Jorino, Maximo and Sultan. It was a great pleasure to have met them. Thank Nada for the arrangement when I stayed in CWI.

I would also like to thank my friendly office mates, Anne, Hugo, Jiao, Lisa, Luis, Luyu, Thomas, for a lot of pleasant moments. Thank Amey, Varun and Xiujie. Due to our close cooperation as the 2018-2019 board of SIAM Student Chapter Delft, we successfully organized academic or non-academic events. The help from the Chinese community is very appreciated, such as Senlei, Guoxin, Cong, Jie, Hongzhi, Lizhou, and so on. I feel very lucky to be a friend of a kind-hearted couple, Jiao Chen and Fei Xu, who shared countless practical tips on study & life in the Netherlands.

I am grateful to the colleagues of the Numerical Analysis Group in Delft Institute of Applied Mathematics for creating such a pleasant atmosphere, Baljinnyam, Behrouz, Fred, Gabriela, Jochen, Kees Lemmens, Kees Vuik, Kristof, Roel, Reinaldo, Prajatka, Mohamed, Marieke, Merel, Mousa, Menel, Xiwei, etc. It was fun playing with the team mates of Krylov Tiger, the soccer team participating in the Monday League of TU Delft. I also enjoyed the football night every Friday, together with my Chinese friends. Thank Qiyao for regularly organizing Badminton games. Without these colleagues and sport activities, the life in Delft would have never been so colorful.

I would like to express the special gratitude to my family, including my parents, my parents-in-law, my sister and her husband (all the best to your son YiChun), my younger brother, my brother-in-law. Their endless support gives me the courage to move every time I want to take a step forward in my career or life. My wife, Lu, deserves the most special acknowledgement. Had it not been for her support and love, I would have never gone so far.

感谢我的父亲和母亲！此书献给父亲六十岁生日。

Shuaiqiang Liu
Eindhoven, December 2020