Pricing Options and Computing Implied Volatilities using Neural Networks

Liu, Shuaiqiang; Oosterlee, Cornelis W.; Bohte, Sander M.

DOI
10.3390/risks7010016

Publication date
2019

Document Version
Final published version

Published in
Risks

risks

MDPI

Article

Pricing Options and Computing Implied Volatilities using Neural Networks

Shuaiqiang Liu [1,*], Cornelis W. Oosterlee [1,2] and Sander M. Bohte [2]

[1] Delft Institute of Applied Mathematics (DIAM), Delft University of Technology, Building 28, Mourik Broekmanweg 6, 2628 XE Delft, The Netherlands; C.W.Oosterlee@cwi.nl

[2] Centrum Wiskunde & Informatica, Science Park 123, 1098 XG Amsterdam, The Netherlands; S.M.Bohte@cwi.nl

* Correspondence: s.liu-4@tudelft.nl

Received: 8 January 2019; Accepted: 6 February 2019; Published: 9 February 2019

Abstract: This paper proposes 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.

Keywords: machine learning; neural networks; computational finance; option pricing; implied volatility; GPU; Black-Scholes; Heston

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. finite 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 a large data set. There are different neural network variants for particular tasks, for example, convolutional neural networks for image recognition LeCun et al. (2010) and recurrent neural networks for time series analysis Lipton (2015).

It is well-known that ANNs can approximate nonlinear functions Cybenko (1989); Hornik (1991); Hornik et al. (1990), and can thus be used to approximate solutions to PDEs Lagaris et al. (1998); Sirignano and Spiliopoulos (2018). Recent advances in data science have shown that using deep learning techniques even highly nonlinear multi-dimensional functions can be accurately represented LeCun et al. (2015). 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) Oh and Jung (2004) or even Tensor Processing Units(TPUs) Jouppi et al. (2017).

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, especially with GPUs or TPUs. 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 paper that this data-driven approach is highly promising.

The proposed approach in this paper 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 the observed market option prices and the underlying asset value, ANN-based regression techniques have been applied to fit a model-free, non-parametric pricing function, see, for example, Hutchinson et al. (1994); Yao et al. (2000); Gencay and Qi (2001); Garcia and Gençay (2000). Furthermore, the authors of Dugas et al. (2001); Yang et al. (2017) 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 Han et al. (2017); Weinan et al. (2017); Beck et al. (2017) take advantage of reinforcement learning to speed up solving high-dimensional stochastic differential equations. The author of Sirignano and Spiliopoulos (2017) 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 Fan and Mancini (2009) 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 Hesthaven and Ubbiali (2018); Raissi and Karniadakis (2018); Sirignano and Spiliopoulos (2018); Tompson et al. (2016). The work in this paper 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 Lagaris et al. (1998) or Sirignano and Spiliopoulos (2018), 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 paper is organized as follows. In Section 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 3, the employed ANN is presented with suitable hyper-parameters. 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 4.

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

2.1. The Black-Scholes PDE

A first model for asset prices is GBM,

$$dS_t = \mu S_t dt + \sqrt{\nu} S_t dW_t^s, \tag{1}$$

where S is the price of a 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 $\sigma = \sqrt{\nu}$. 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,

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

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,

$$V(t = T, S) = (S_0 - K)^+, \tag{3}$$

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), (3) exists for European plain vanilla options, i.e.,

$$V_c(t, S) = SN(d_1) - Ke^{-r\tau}N(d_2), \tag{4a}$$

$$d_1 = \frac{\log(S/K) + (r - 0.5\sigma^2)\tau}{\sigma\sqrt{\tau}}, \quad d_2 = d_1 - \sigma\sqrt{\tau}, \tag{4b}$$

where $\tau := T - t$, $V_c(t, S)$ is the European call option value at time t for stock value S, and $N(\cdot)$ represents the normal distribution. This solution procedure (4) is denoted by $V(\cdot) = BS(\cdot)$.

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,

$$\sigma^*(K, T) = BS^{-1}(V^{mkt}; S, K, \tau, r), \tag{5}$$

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 Cont and da Fonseca (2002).

For simplicity, we denote here $\sigma^*(m, \tau)$ by σ^*. An analytic solution for Equation (5) does not exist. The value of σ^* is determined by means of a numerical iterative technique, since Equation (5) can be converted into a root-finding problem,

$$g(\sigma^*) = BS(S, \tau, K, r, \sigma^*) - V^{mkt}(S, \tau; K) = 0. \tag{6}$$

2.2. The Heston Model

One of the limitations of using the Black-Scholes model is the assumption of a constant volatility σ in (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 Heston (1993), for which the system of stochastic equations under the risk-neural measure reads,

$$dS_t = rS_t dt + \sqrt{\nu_t} S_t dW_t^s, S_{t_0} = S_0, \tag{7a}$$
$$d\nu_t = \kappa(\bar{\nu} - \nu_t)dt + \gamma\sqrt{\nu_t} dW_t^\nu, \nu_{t_0} = \nu_0, \tag{7b}$$
$$dW_t^s dW_t^\nu = \rho dt, \tag{7c}$$

with ν_t the instantaneous variance, and W_t^s, W_t^ν are two Wiener processes with correlation coefficient ρ. The second equation in (7) models a mean reversion process for the variance, with the parameters, r the risk-free interest rate, $\bar{\nu}$ the long term variance, κ the reversion speed; γ is the volatility of the variance, determining the volatility of ν_t. There is an additional parameter ν_0, the t_0-value of the variance.

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

$$\begin{aligned} \frac{\partial V}{\partial t} &+ rS\frac{\partial V}{\partial S} + \kappa(\bar{\nu} - \nu)\frac{\partial V}{\partial \nu} + \frac{1}{2}\nu S^2 \frac{\partial^2 V}{\partial S^2} \\ &+ \rho\gamma S\nu \frac{\partial^2 V}{\partial S \partial \nu} + \frac{1}{2}\gamma^2 \nu \frac{\partial^2 V}{\partial \nu^2} - rV = 0. \end{aligned} \tag{8}$$

The typically observed implied volatility shapes in the market, e.g., smile or skew, can be reproduced by varying the above parameters $\{\kappa, \rho, \gamma, \nu_0, \bar{\nu}\}$. 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. 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 Feynman-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 Feynman-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 Fang and Oosterlee (2009), which utilizes Fourier-cosine series expansions to approximate the asset price's probability density 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.

2.3. Numerical Methods for Implied Volatility

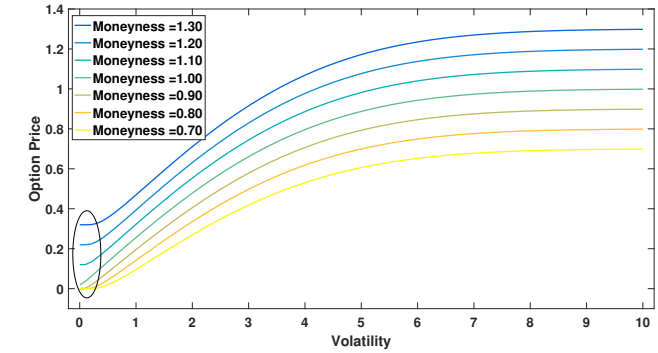
Focussing on the implied volatility σ^*, there are several iterative numerical techniques to solve (6), for example, the Newton-Raphson method, the bisection method or the Brent method. The Newton-Raphson iteration reads,

$$\sigma^*_{k+1} = \sigma^*_k - \frac{V(\sigma^*_k) - V^{mkt}}{g'(\sigma^*_k)},\ k = 0, \ldots. \tag{9}$$

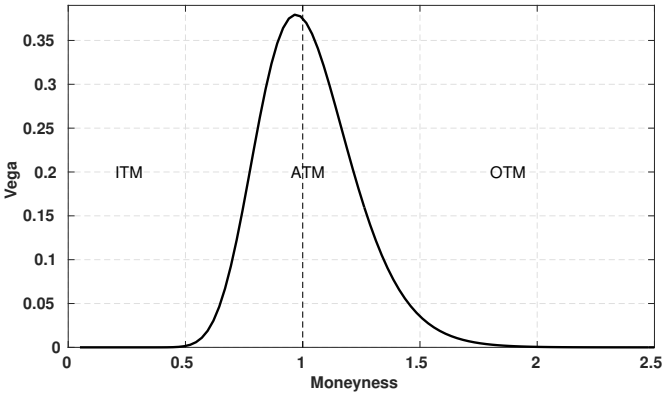
Starting with an initial guess, σ^*_0, the approximate solutions, σ^*_{k+1}, $k = 0, \ldots$, iteratively improve, until a certain criterion is satisfied. The first derivative of Black-Scholes option value with respect to the volatility, named the option's Vega, in the denominator of (9) can be obtained analytically for European options.

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 1a. As a result, the near-flat function shapes 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 1b shows that the option's Vega can be very close to zero in the regions with small or large volatilities in deep ITM or OTM options, although its value is relatively large in the at-the money (ATM) region.

A possible robust root-finding algorithm for solving this problem is to employ a hybrid of the Newton-Raphson and the bisection methods. Alternatively, the author of Jäckel (2015) proposed to select a suitable initial value at the beginning of the iteration to avoid divergence. In the next subsection, we will introduce a derivative-free, robust and efficient algorithm to find the implied volatility.



(**a**)



(**b**)

Figure 1. Vega tends to be zero in certain regions of deep ITM or OTM options. (**a**) Option price vs. volatility; (**b**) Vega vs. Moneyness.

Brent's Method for Implied Volatility

As a derivative-free, robust and efficient algorithm, Brent's method Brent (1973) 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.,

$$\begin{aligned}\sigma_{k+1} = & \frac{\sigma_k g(\sigma_{k-1}) g(\sigma_{k-2})}{(g(\sigma_{(k)}) - g(\sigma_{k-1}))(g(\sigma_k) - g(\sigma_{k-2}))} \\ & + \frac{\sigma_{k-1} g(\sigma_{k-2}) g(\sigma_k)}{(g(\sigma_{k-1}) - g(\sigma_{k-2}))(g(\sigma_{k-1}) - g(\sigma_k))} \\ & + \frac{\sigma_{k-2} g(\sigma_{k-1}) g(\sigma_k)}{(g(\sigma_{k-2}) - g(\sigma_{k-1}))(g(\sigma_{k-2}) - g(\sigma_k))}.\end{aligned} \tag{10}$$

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,

$$\sigma_{k+1} = \sigma_{k-1} - g(\sigma_{k-1}) \frac{\sigma_{k-1} - \sigma_{k-2}}{g(\sigma_{k-1}) - g(\sigma_{k-2})}. \tag{11}$$

In this paper, Brent's method is used to compute the BS implied volatility related to the Heston option prices in Section 4.4.2. We will develop an ANN to approximate the implicit function relating the volatility to the option price.

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),

Algorithm 1 Model framework

1: – Generate the sample data points for input parameters,
2: – Calculate the corresponding output (option price or implied volatility) to form a complete data set with inputs and outputs,
3: – Split the above data set into a training and a test part,
4: – Train the ANN on the training data set,
5: – Evaluate the ANN on the test data set,
6: – Replace the original solver by the trained ANN in applications.

3.1. Artificial Neural Network

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 signals. 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 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.

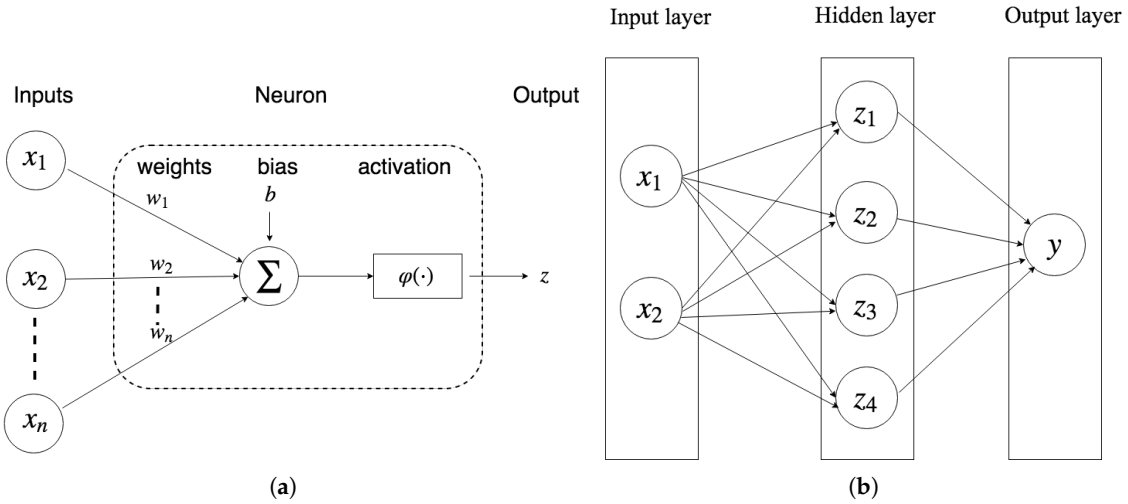


Figure 2. Illustration of a MLP configuration. (**a**) A neuron; (**b**) An example of MLP.

A multi-layer perceptron (MLP), consisting of at least one hidden layer is the simplest version of an ANN. Mathematically, MLPs are defined by the following parameters,

$$\boldsymbol{\theta} = (\mathbf{W}_1, \mathbf{b}_1, \mathbf{W}_2, \mathbf{b}_2, ..., \mathbf{W}_L, \mathbf{b}_L) \tag{12}$$

where $\mathbf{W}_j$ is a weight matrix with $\mathbf{b}_j$ being a bias vector in the L-th neural layer. A function can then be expressed as follows,

$$y(\mathbf{x}) = F(\mathbf{x}|\boldsymbol{\theta}). \tag{13}$$

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

$$z_j^{(l)} = \varphi^{(l)}\left(\sum_i w_{ij}^{(l)} z_i^{(l-1)} + b_j^{(l)}\right), \tag{14}$$

where $z_i^{(l-1)}$ is the output value of the i-th neuron in the $(l-1)$-th layer and $\varphi(\cdot)$ is an activation function, with $w_{ij}^{(l)} \in \mathbf{W}_l$, $b_j^{(l)} \in \mathbf{b}_l$. When $l = 0$, $z^{(0)} = x$ is the input layer; When $l = L$, $z^{(L)} = y$ is the output layer; Otherwise, $z^{(l)}$ represents an intermediate variable. The activation function $\varphi(\cdot)$ adds non-linearity to the system, for example, the following activation functions may be employed,

- Sigmoid, $\varphi(x) = \dfrac{1}{1+e^{-x}}$,
- ReLU, $\varphi(x) = \max(x, 0)$;

see LeCun et al. (2015) for more activation functions. For example, a MLP with "one-hidden-layer" is shown in Figure 2b, and Equation (15) presents its mathematical formula,

$$\begin{cases} y = \varphi^{(2)}\left(\sum_j w_j^{(2)} z_j^{(1)} + b^{(2)}\right) \\ z_j^{(1)} = \varphi^{(1)}\left(\sum_i w_{ij}^{(1)} x_i + b_j^{(1)}\right). \end{cases} \tag{15}$$

According to the Universal Approximation Theorem Cybenko (1989), 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 $||\cdot||$,

$$D(f(\mathbf{x}), F(\mathbf{x})) = ||f(\mathbf{x}) - F(\mathbf{x})||, \tag{16}$$

where $f(\mathbf{x})$ is the objective function, $F(\mathbf{x})$ is the neural network approximated function. For example, the p-norm reads,

$$||f(\mathbf{x}) - F(\mathbf{x}|\boldsymbol{\theta})||_p = \sqrt[p]{\int_{\mathbf{X}} |f(\mathbf{x}) - F(\mathbf{x}|\boldsymbol{\theta})|^p d\mu(\mathbf{x})},$$

where $1 \leq p < \infty$ and $\mu(\mathbf{x})$ is a measure over the space $\mathbf{X}$. We choose $p = 2$ to evaluate the averaged accuracy, which corresponds to the mean squared error (MSE). Within supervised learning, the loss function is equivalent to the above distance,

$$L(\boldsymbol{\theta}) := D(f(\mathbf{x}), F(\mathbf{x}|\boldsymbol{\theta})). \tag{17}$$

The training process aims to learn the optimal weights and biases in Equation (13) to make the loss function as small as possible. The process can be formulated as an optimization problem,

$$arg\min_{\boldsymbol{\theta}} L(\boldsymbol{\theta}|(\mathbf{x}, \mathbf{y})), \tag{18}$$

given the known input-output pairs $(\mathbf{x}, \mathbf{y})$ and a loss function $L(\boldsymbol{\theta})$.

Several back-propagation gradient descent methods Ruder (2016) have been successfully applied to solve Equation (18), for instance, Stochastic Gradient Descent (SGD) and its variants Adam and RMSprop. These optimization algorithms start with initial values and move in the direction in which the loss function decreases. The formula for updating the parameters reads,

$$\begin{cases} \mathbf{W} \leftarrow \mathbf{W} - \eta(i)\frac{\partial L}{\partial \mathbf{W}}, \\ \mathbf{b} \leftarrow \mathbf{b} - \eta(i)\frac{\partial L}{\partial \mathbf{b}}, \\ i = 0, 1, 2, ..., \end{cases} \tag{19}$$

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 detail will be given in Section 3.3.

3.2. Hyper-Parameters Optimization

Training deep neural networks involves numerous choices for the commonly called "hyper-parameters". These include the number of layers, neurons, and the specific activation function, etc. First, 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 a 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 using automatic machine learning Bergstra et al. (2011). 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 Bergstra and Bengio (2012) concluded that random search is more efficient for hyper-parameters optimization. Recently, Bayesian hyper-parameter optimization Snoek et al. (2012) has been developed to efficiently reduce the computational cost by navigating through the hyper-parameters space. However, it is still difficult to outperform random search in combination with certain expert knowledge at the current stage.

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 this model selection process, over-fitting can be reduced by adopting the k-fold cross validation as follows (Algorithm 2).

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.

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. As shown in Table 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 growing (more related details will be discussed in Section 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 application.

Table 1. The setting of random search for hyper-parameters optimization.

Parameters	Options or Range
Activation	ReLU, Tanh, Sigmoid, ELU
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]

Table 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

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 2. As Table 2 shows, the optimal parameter values, i.e., neurons and batch size, do not lie at the boundaries of the search space (except for the drop out rate). Compared to Sigmoid, ReLU is more likely to cause a better convergence (e.g., overcome the gradient vanishing in a deep neural network). As an extension to SGD, Adam can handle an optimization problem more adaptively. However, batch normalization and drop-out do not improve the model accuracy in this regression problem, and one possible reason is 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 can result in an ANN with sufficient accuracy to approximate the financial option values.

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 Smith (2015) for a cyclical learning rate (CLR) and Loshchilov and Hutter (2016) 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 Smith (2015) to determine the lower and upper bound of learning rates. 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 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 10^{-3}, where the slope is the steepest and the training loss reduces quickly. Therefore, the learning rate is reduced from 10^{-3} to 10^{-5} in our experiments.

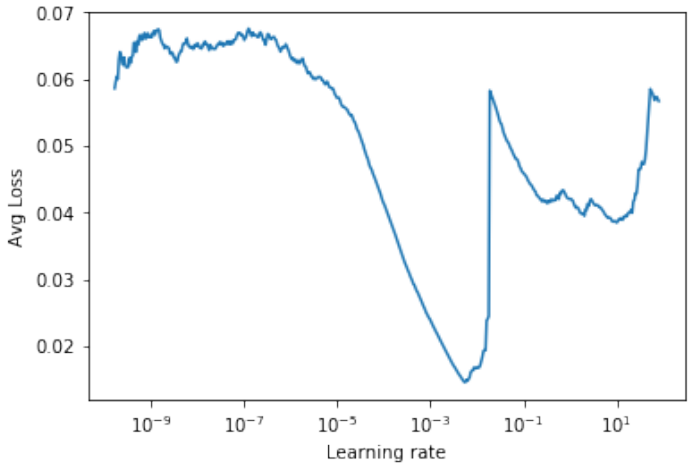
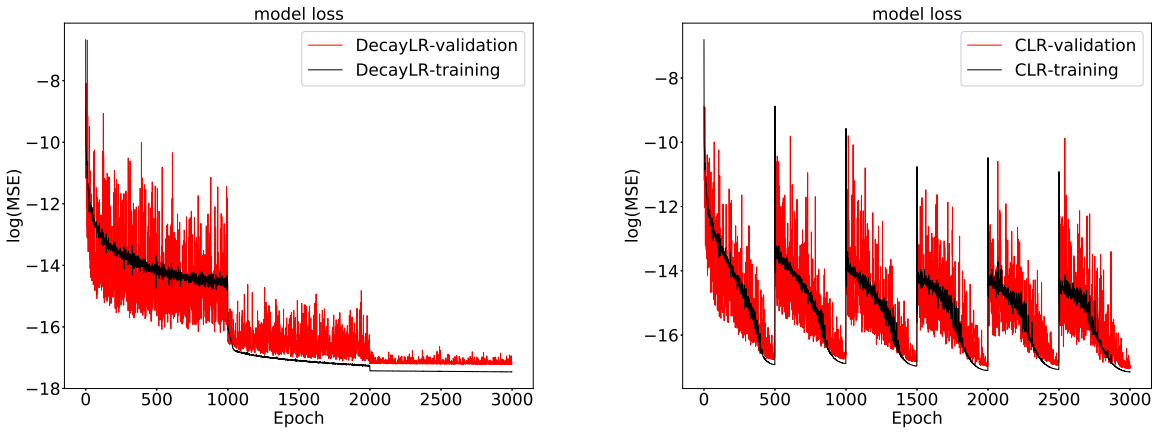


Figure 3. Average training loss against varying learning rates.

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 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 5, in this case a decay rate-based schedule outperforms the CLR with the same learning rate bounds, although with the CLR the difference between training and validation losses are smaller. This is contrary to the conclusion in Smith (2015), but their network included batch normalization and L2 regularization. For the tests in this paper, 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.



(**a**) (**b**)

Figure 4. The history of training and validation losses for the Heston model. (**a**) Losses with decaying learning rates; (**b**) Losses with cyclical learning rates.

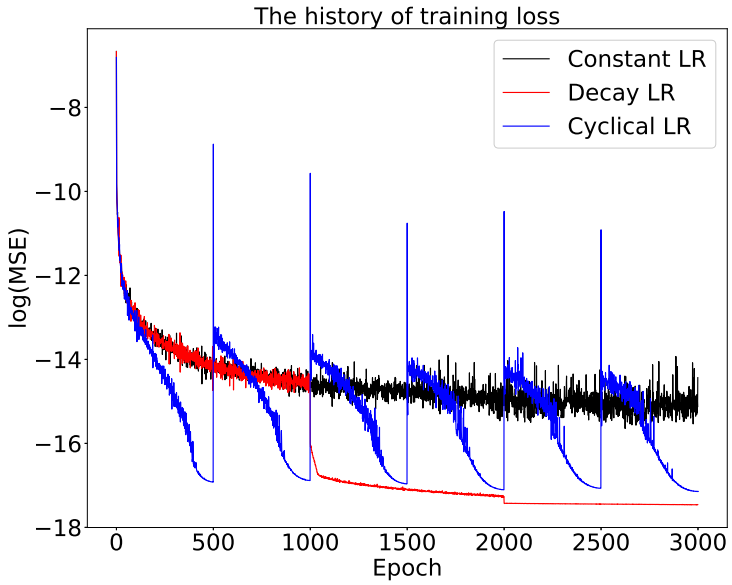


Figure 5. Different learning rate schedules for training ANNs on Heston model.

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),

$$\text{MSE} = \frac{1}{n}\sum(y_i - \hat{y}_i)^2, \tag{20}$$

where y_i is the actual value and $\hat{y}_i$ is the ANN predicted value. For completeness, however, we also report the other well-known metrics,

$$\text{RMSE} = \sqrt{\text{MSE}}, \tag{21a}$$

$$\text{MAE} = \frac{1}{n}\sum|y_i - \hat{y}_i|, \tag{21b}$$

$$\text{MAPE} = \frac{1}{n}\sum\frac{|y_i - \hat{y}_i|}{y_i}. \tag{21c}$$

The MSE is used as the training metric to update the weights, and all above metrics are employed to evaluate the selected ANN.

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.

4.1. Details of the Data Set

As a data-driven approach, the quality of a data set has an 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) McKay et al. (1979) 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 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 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 3. Meanwhile, the test data is kept unchanged. The example here is learning the implied volatility. We first train the ANN on each data set separately by using a decaying learning rate, as described in Section 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 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 3.3. The training and validation losses remain close, which indicates that there is no over-fitting.

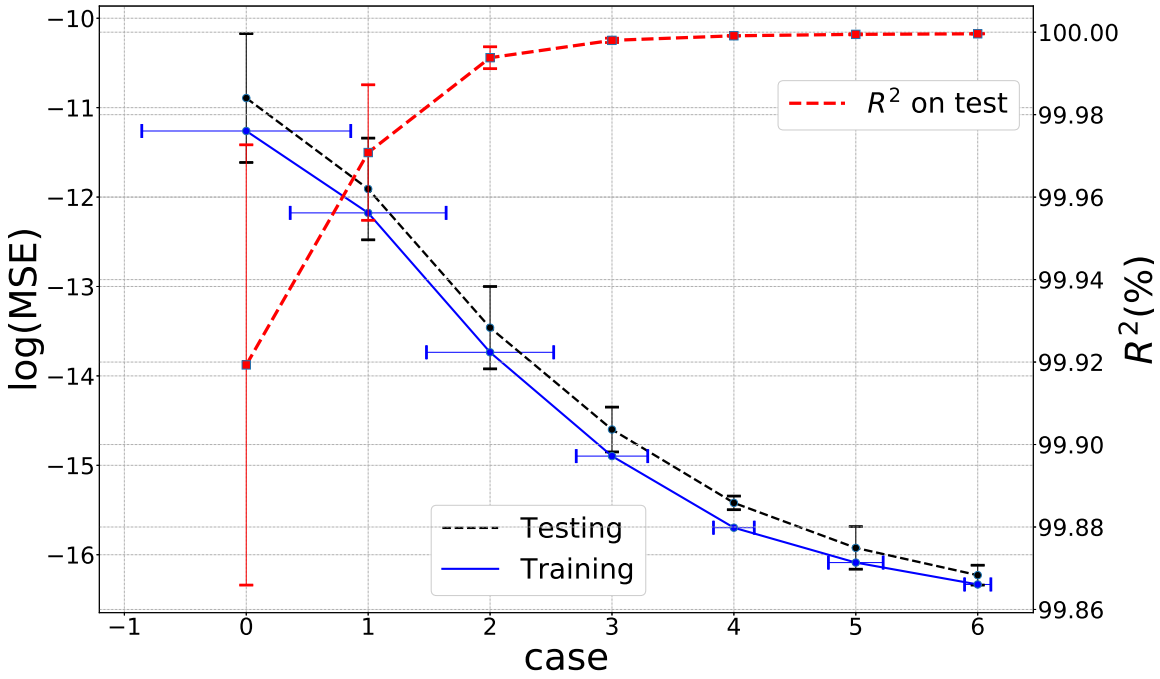


Figure 6. R^2 and MSE vs. size of the training set. The figure shows improvement in the loss, which is an indicator of the model performance.

4.2. Black-Scholes Model

Focussing on European call options, we generate 1,000,000 random samples for the input parameters, see Table 4. We calculate the corresponding European option prices $V(S,t)$ of Equation (2) with the solution in (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 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.

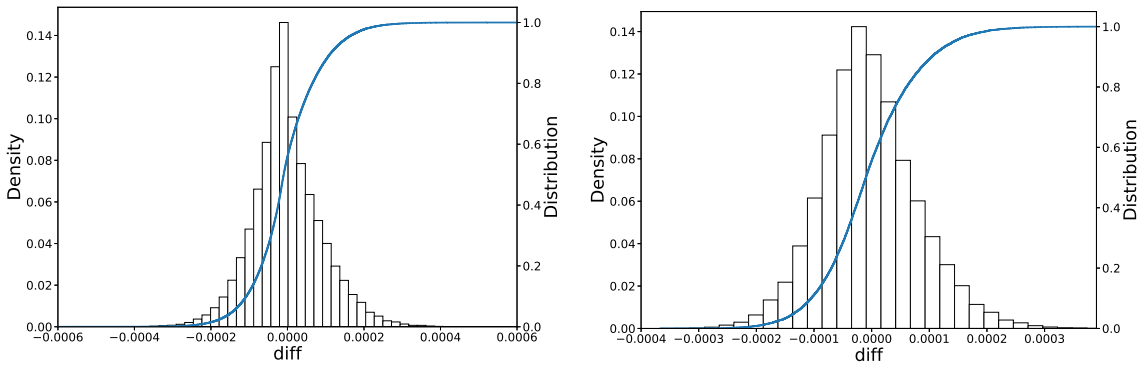
The wide test data set is based on the same parameter ranges as the training data set. As shown in Table 5, the root averaged mean-squared error (RMSE) is around 9×10^{-5}, which is an indication that the average pricing error is 0.009% of the strike price. Figure 7a shows the histogram of prediction errors, where it can be seen that the error approximately exhibits a normal distribution, and the maximum absolute error is around 0.06%.

The narrow test set is based on a somewhat more narrow parameter range than the training data set. As Table 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 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 with R^2 being close to 1.0.

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. In the next sections, however, we list the performance of the ANN on the wide test data set.

Table 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}



(**a**) (**b**)

Figure 7. BS-ANN performance. (**a**) Error distribution on the wide test data set; (**b**) Error distribution on the narrow test data set.

4.3. Implied Volatility

The aim here is to learn the implicit relationship between implied volatilities and option prices, which is guided by Equation (5). The option Vega can become arbitrarily small, which also 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,

$$\tilde{V} = V_t - \max(S_t - Ke^{-r\tau}, 0),$$

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.

4.3.1. 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 $\tilde{V} < 10^{-7}$.

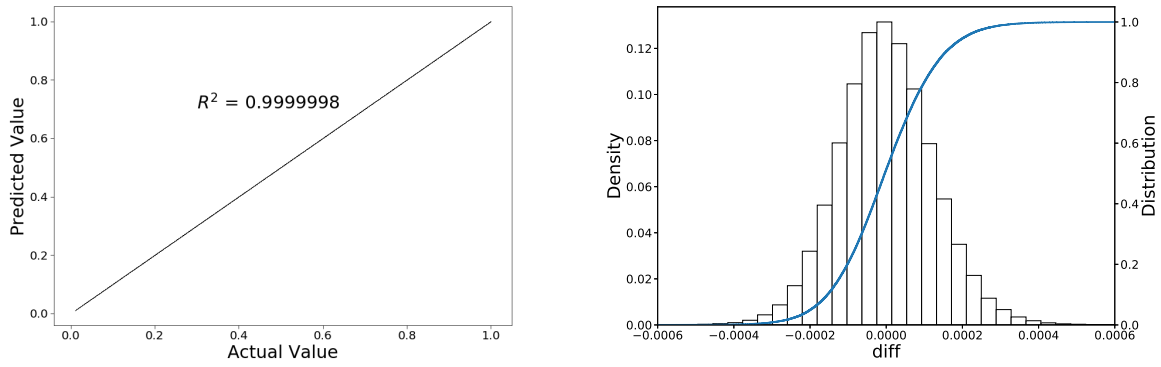
Table 6. Parameter range of data set.

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)	-

Two implied volatility ANN (IV-ANN) solvers are trained based on the dataset in Table 6. After that, Table 7 compares the performance of the trained ANNs with the scaled and original (unscaled) input, where it is clear that scaling improves the ANN performance significantly. Figure 8 shows the out-of-sample performance of the trained ANN on the scaled inputs. The error distribution also approximately follows a normal distribution, where the maximum deviation is around 6×10^{-4}, and most of implied volatilities equal their true values.

Table 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 8. Out-of-Sample IV-ANN performance on the scaled input. (**a**) Comparison of implied volatilities; (**b**) The error distribution.

4.3.2. 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 method, 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 guess for the Newton-Raphson and secant method is chosen $\sigma_0^* = 0.5$. 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 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 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

4.4. The Heston Stochastic Volatility Model

This section presents the quality of the ANN predictions of the Heston option prices and the corresponding implied volatilities. The performance of the Heston-ANN solver is also evaluated.

4.4.1. 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.

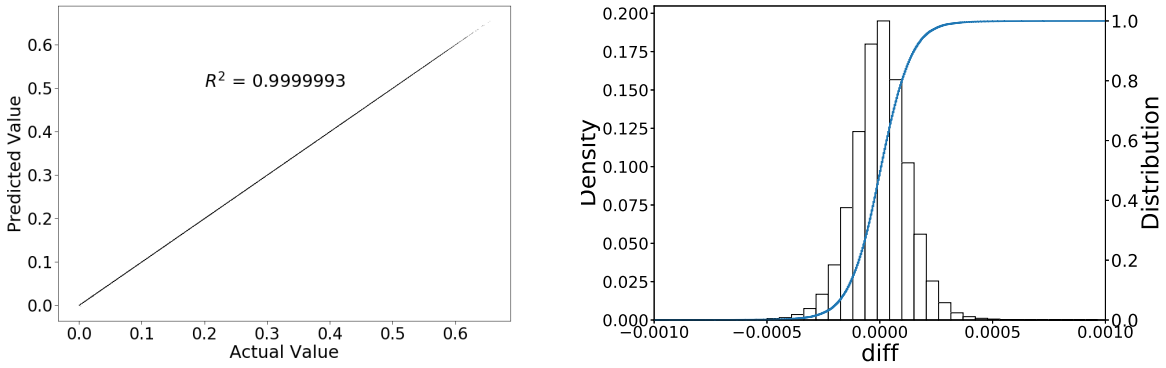
According to the given ranges of Heston parameters in Table 9, for the COS method, the integration interval is based on $L_{COS} = 50$, with the number of Fourier cosine terms in the expansion being $N_{COS} = 1500$. The prices of deep OTM European call options are calculated using the put-call parity, as the COS method call prices that are close to zero may be inaccurate due to truncation errors. In Table 9, we list the range of the six Heston input parameters (r, ρ, κ, $\bar{v}$, γ, v_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 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 9.

Table 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, $\bar{v}$	(0.0, 0.5)	LHS
	Volatility of volatility, γ	(0.0, 0.5)	LHS
	Initial variance, v_0	(0.05, 0.5)	LHS
Output	European call price, V	(0, 0.67)	COS

Table 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 9. Out-of-sample Heston-ANN performance. (**a**) COS vs. Heston-ANN prices; (**b**) The error distribution.

4.4.2. 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 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 as follows.

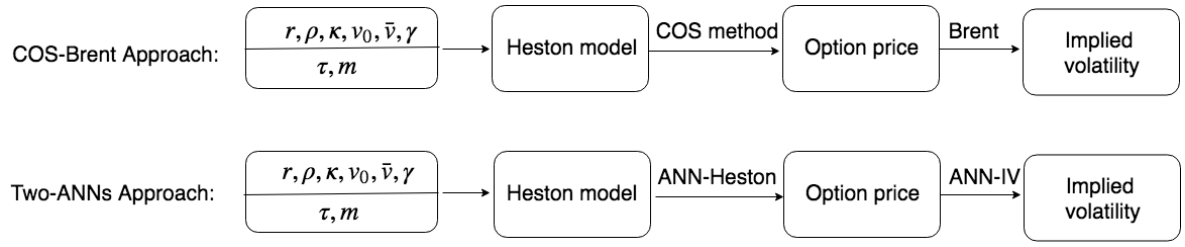


Figure 10. Two approaches of computing implied volatility for Heston model.

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 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 $\approx 7 \times 10^{-4}$. Then we reduce the range of the parameters, as listed in the third row of Table 11, and find that the prediction accuracy increases with the parameter ranges shrinking. Comparing the results in Figure 11 and Table 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.

Table 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

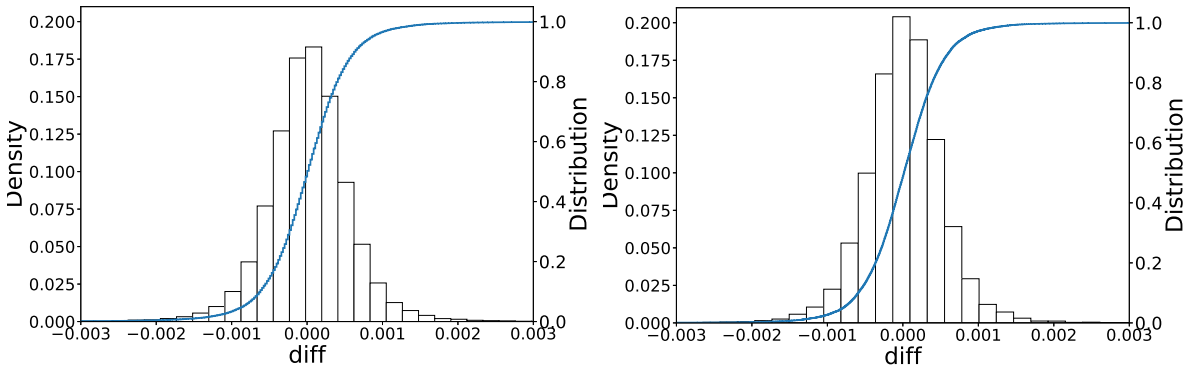
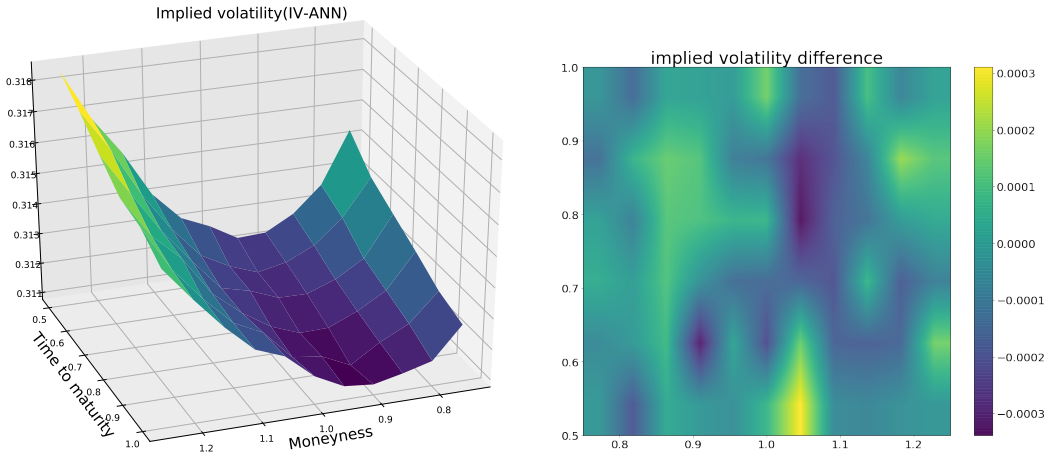


Figure 11. The error distribution of the implied volatility: The combined Heston-ANN and IV-ANN techniques for implied volatility. (**a**) Case 1: The error distribution; (**b**) Case 2: The error distribution.

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 produce 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 12a, and Figure 12b shows that the maximum deviation between the ground-truth and the predicted values is no more than 4×10^{-4}.



(**a**) (**b**)

Figure 12. (**a**) Heston implied volatility surface generated by IV-ANN. (**b**) Heston implied volatility difference between Brent's method and IV-ANN.

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.

5. Conclusions and Discussion

In this paper we have proposed an ANN approach to reduce the computing time of pricing financial options, especially for high-dimensional financial models. 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 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. This 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 Hinton et al. (2015).

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 (19) is known analytically. Alternatively, a trained ANN may be interpreted as an implicit function, where Auto-Differentiation Baydin et al. (2015) can help to calculate the derivatives accurately. Merging two neural networks, the Heston-ANN and the IV-ANN, into a single network should make it more efficient when computing the implied volatility surface for the Heston model.

Author Contributions: Supervision, C.W.O. and S.M.B.; Writing—original draft, S.L.

Funding: This research received no external funding.

Acknowledgments: The authors would like to thank the China Scholarship Council (CSC) for the financial support. This research was conducted using the supercomputer Little Green Machine II in the Netherlands.

Conflicts of Interest: The authors declare no conflict of interest.

References

Baydin, Atilim Gunes, Barak A. Pearlmutter, and Alexey Andreyevich Radul. 2015. Automatic Differentiation in Machine Learning: A survey. Available online: https://arxiv.org/abs/1502.05767 (accessed on 25 January 2018).

Beck, Christian, E. Weinan, and Arnulf Jentzen. 2017. Machine Learning Approximation Algorithms for High-Dimensional Fully Nonlinear Partial Differential Equations and Second-Order Backward Stochastic Differential Equations. Available online: https://arxiv.org/abs/1709.05963 (accessed on 2 February 2018).

Bergstra, James S., Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for Hyper-Parameter Optimization. Paper presented at the 24th International Conference on Neural Information Processing Systems, Granada, Spain, December 12–15.

Bergstra, James, and Yoshua Bengio. 2012. Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research* 13: 281–305.

Brent, Richard Peirce. 1973. An Algorithm with Guaranteed Convergence for Finding a Zero of a Function. In *Algorithms for Minimization without Derivatives*. New York: Dover.

Cont, Rama, and José da Fonseca. 2002. Dynamics of implied volatility surfaces. *Quantitative Finance* 2: 45–60. [CrossRef]

Cybenko, George. 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems* 2: 303–14. [CrossRef]

Dugas, Charles, Yoshua Bengio, François Bélisle, Claude Nadeau, and René Garcia. 2001. Incorporating second-order functional knowledge for better option pricing. In *Proceedings of the 13th International Conference on Neural Information Processing Systems*. Cambridge: MIT Press, pp. 451–7.

Fan, Jianqing, and Loriano Mancini. 2009. Option Pricing With Model-Guided Nonparametric Methods. *Journal of the American Statistical Association* 104: 1351–72. [CrossRef]

Fang, Fang, and Cornelis W. Oosterlee. 2009. A Novel Pricing Method for European Options Based on Fourier-Cosine Series Expansions. *SIAM Journal on Scientific Computing* 31: 826–48. [CrossRef]

Garcia, René, and Ramazan Gençay. 2000. Pricing and hedging derivative securities with neural networks and a homogeneity hint. *Journal of Econometrics* 94: 93–115. [CrossRef]

Gencay, Ramazan, and Min Qi. 2001. Pricing and hedging derivative securities with neural networks: Bayesian regularization, early stopping, and bagging. *IEEE Transactions on Neural Networks* 12: 726–34. [CrossRef] [PubMed]

Han, Jiequn, Arnulf Jentzen, and E Weinan. 2017. Solving High-Dimensional Partial Differential Equations Using Deep Learning. Available online: https://arxiv.org/abs/1707.02568 (accessed on 10 December 2017).

Hesthaven, Jan S., and Stefano Ubbiali. 2018. Non-intrusive reduced order modeling of nonlinear problems using neural networks. *Journal of Computational Physics* 363: 55–78. [CrossRef]

Heston, Steven L. 1993. A Closed-Form Solution for Options with Stochastic Volatility with Applications to Bond and Currency Options. *Review of Financial Studies* 6: 327–43. [CrossRef]

Hinton, Geoffrey, Oriol Vinyals, and Jeff Dean. 2015. Distilling the Knowledge in a Neural Network. *arXiv* arXiv:1503.02531.

Hornik, Kurt. 1991. Approximation capabilities of multilayer feedforward networks. *Neural Networks* 4: 251–7. [CrossRef]

Hornik, Kurt, Maxwell Stinchcombe, and Halbert White. 1990. Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural Networks* 3: 551–60. [CrossRef]

Hutchinson, James M., Andrew W. Lo, and Tomaso Poggio. 1994. A Nonparametric Approach to Pricing and Hedging Derivative Securities Via Learning Networks. *The Journal of Finance* 49: 851–89. [CrossRef]

Jäckel, Peter. 2015. Let's Be Rational. *Wilmott* 2015: 40–53. [CrossRef]

Jouppi, Norman P., Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Nan Boden, Al Borchers, Rick Boyle, et al. 2017. In-datacenter performance analysis of a tensor processing unit. Paper presented at 44th Annual International Symposium on Computer Architecture, Toronto, ON, Canada, June 24–28, pp. 1–12. [CrossRef]

Lagaris, Isaac E., Aristidis Likas, and Dimitrios I. Fotiadis. 1998. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks* 9: 987–1000. [CrossRef] [PubMed]

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521: 436–44. [CrossRef] [PubMed]

LeCun, Yan, Koray Kavukcuoglu, and Clement Farabet. 2010. Convolutional networks and applications in vision. Paper presented at 2010 IEEE International Symposium on Circuits and Systems, Paris, France, May 30–June 2. [CrossRef]

Lipton, Zachary Chase. 2015. A Critical Review of Recurrent Neural Networks for Sequence Learning. Available online: https://arxiv.org/abs/1506.00019 (accessed on 12 November 2017).

Loshchilov, Ilya, and Frank Hutter. 2016. SGDR: Stochastic Gradient Descent with Restarts. Available online: https://arxiv.org/abs/1608.03983 (accessed on 10 October 2017).

McKay, Michael D., Richard J. Beckman, and William J. Conover. 1979. A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code. *Technometrics* 21: 239. [CrossRef]

Oh, Kyoung-Su, and Keechul Jung. 2004. Gpu implementation of neural networks. *Pattern Recognition* 37: 1311–14. [CrossRef]

Raissi, Maziar, and George Em Karniadakis. 2018. Hidden physics models: Machine learning of nonlinear partial differential equations. *Journal of Computational Physics* 357: 125–41. [CrossRef]

Ruder, Sebastian. 2016. An Overview of Gradient Descent Optimization Algorithms. Available online: https://arxiv.org/abs/1609.04747 (accessed on 18 October 2017).

Sirignano, Justin, and Konstantinos Spiliopoulos. 2017. Stochastic gradient descent in continuous time. *SIAM Journal on Financial Mathematics* 8: 933–61. [CrossRef]

Sirignano, Justin, and Konstantinos Spiliopoulos. 2018. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*. [CrossRef]

Smith, Leslie N. 2015. Cyclical lEarning Rates for Training Neural Networks. Available online: https://arxiv.org/abs/1506.01186 (accessed on 15 April 2018).

Snoek, Jasper, Hugo Larochelle, and Ryan P. Adams. 2012. Practical bayesian optimization of machine learning algorithms. Paper presented at the 25th International Conference on Neural Information Processing Systems, Lake Tahoe, NV, USA, December 3–6, pp. 2951–2959.

Tompson, Jonathan, Kristofer Schlachter, Pablo Sprechmann, and Ken Perlin. 2016. Accelerating Eulerian Fluid Simulation with Convolutional Networks. Available online: https://arxiv.org/abs/1607.03597 (accessed on 18 November 2017).

Weinan, E., Jiequn Han, and Arnulf Jentzen. 2017. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Communications in Mathematics and Statistics* 5: 349–80. doi:10.1007/s40304-017-0117-6. [CrossRef]

Yang, Yongxin, Yu Zheng, and Timothy Hospedales. 2017. Gated neural networks for option pricing: Rationality by design. Paper presented at The Thirty-First AAAI Conference on Artificial Intelligence, San Francisco, CA, USA, February 4–9.

Yao, Jingtao, Yili Li, and Chew Lim Tan. 2000. Option price forecasting using neural networks. *Omega* 28: 455–66. [CrossRef]