Preprints.org

Article

Not peer-reviewed version

Study on Pricing of High Dimensional Financial Derivatives Based on Deep Learning

Xiangdong Liu and Yu Gu *

Posted Date: 14 April 2023

doi: 10.20944/preprints202304.0369.v1

Keywords: Deep learning; Backward stochastic differential equation; Nonlinear Feynman-Kac formula; High dimensional PDE; Derivatives pricing; Neural network

Preprints.org is a free multidiscipline platform providing preprint service that is dedicated to making early versions of research outputs permanently available and citable. Preprints posted at Preprints.org appear in Web of Science, Crossref, Google Scholar, Scilit, Europe PMC.

Copyright: This is an open access article distributed under the Creative Commons Attribution License which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 April 2023 doi:10.20944/preprints202304.0369.v1

Disclaimer/Publisher's Note: The statements, opinions, and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions, or products referred to in the content.

Article

Study on Pricing of High Dimensional Financial Derivatives Based on Deep Learning

Xiangdong Liu [1] and Yu Gu [2,*]

[1,2] Department of Statistics and Data Science, Jinan University, Guangzhou 510632, China
* Correspondence: 1437092373@qq.com

Abstract: Many problems in the fields of finance and actuarial science can be transformed into the problem of solving backward stochastic differential equations (BSDE) and partial differential equations (PDE) with jumps, which are often difficult to solve in high-dimensional cases. To solve this problem, this paper applies the deep learning algorithm to solve a class of high-dimensional nonlinear partial differential equations with jump terms and their corresponding backward stochastic differential equations (BSDE) with jump terms. Using the nonlinear Feynman-Kac formula, the problem of solving this kind of PDE is transformed into the problem of solving the corresponding backward stochastic differential equations with jump terms, and the numerical solution problem is turned into a stochastic control problem. At the same time, the gradient and jump process of the unknown solution are separately regarded as the strategy function and they are approximated respectively by using two multilayer neural networks as function approximators. Thus, deep learning-based method is used to overcome the "curse of dimensionality" caused by high-dimensional PDE with jump, and the numerical solution is obtained. In addition, this paper proposes a new optimization algorithm based on the existing neural network random optimization algorithm, and compares the results with the traditional optimization algorithm, and achieves good results. Finally, the proposed method is applied to three practical high-dimensional problems: Hamilton-Jacobi-Bellman equation, bond pricing under the jump Vasicek model and high-dimensional option pricing model with default risk. The proposed numerical method has obtained satisfactory accuracy and efficiency. The method has important application value and practical significance in investment decision-making, option pricing, insurance and other fields.

Keywords: deep learning; backward stochastic differential equation; nonlinear feynman-kac formula; high dimensional PDE; derivatives pricing; neural network

1. Introduction

High-dimensional (dimension ≥ 3) nonlinear partial differential equations (PDEs) are one of the topics that attract much attention and have been widely used in many fields. Many practical problems require the use of high-dimensional PDE, for example: the Schrodinger equation of the quantum many-body problem, whose PDE dimension is about three times that of the electron or quantum (particle) in the system; Black Scholes equation used to price financial derivatives, where the dimension of PDE is the number of relevant financial assets under consideration; Hamilton-Jacobi-Bellman equation in dynamic programming. Although high-dimensional nonlinear partial differential equations have strong practicability, there are few explicit solutions, or the analytical expressions are too complex, so they often need to be solved by some numerical methods. However, in practical applications, high-dimensional nonlinear partial differential equations are usually very difficult to solve, which is still one of the challenging topics in the academic community. The difficulty is that, due to the " curse of dimensionality " [1], the time complexity of traditional numerical solutions will increase exponentially with the increase of dimensions, thus requiring a lot of computing resources. However, because these equations can solve many practical problems, we urgently need

© 2023 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 April 2023 doi:10.20944/preprints202304.0369.v1

to approximate the numerical solution of this high-dimensional nonlinear partial differential equation.

In recent years, deep learning algorithms have gradually emerged and achieved success in many application fields. Therefore, many scientists try to apply deep learning algorithms to solve high-dimensional PDE problems, and have achieved good results. In 2017, E W. Han et al. [2,3] proposed the deep BSDE method and systematically applied the deep learning to general high-dimensional PDE for the first time. After that, Han et al. (2018) [4] further extended the deep BSDE method to the field of random games, and also achieved good results. Beck et al. (2019) [5] extended the method to the case of 2BSDE and the corresponding fully nonlinear PDE. Raissi et al.(2017)[6] use the latest development of probabilistic machine learning to infer the control equation represented by a parametric linear operator, and modifies the prior value of the Gaussian process according to the special form of such operator, which is then used to infer the parameters of the linear equation from the scarce and possibly noisy observations. Sirignano et al. (2018) [7] proposed to use a deep neural network approach to approximate the solution of high-dimensional PDE. In a batch of randomly sampled time and space, the network was trained to meet differential operators, initial conditions and boundary conditions. The above series of achievements show that it is feasible to solve high-dimensional PDE with deep learning-based method.

In this paper, we mainly apply the deep learning algorithm to a special class of high-dimensional nonlinear partial differential equations with jumps, and obtain a numerical solution of the equation. Specifically, through mathematical derivation and equivalent formula expression of high-dimensional PDE and backward stochastic differential equation, the problem of solving partial differential equation is equivalent to the problem of solving BSDE, and then it is transformed into a stochastic control problem. Then, a deep neural network framework is designed to fit the problem. Therefore, this method can be used to solve high-dimensional PDE and corresponding backward stochastic differential equations simultaneously.

The method introduced in this paper to solve a kind of high-dimensional PDE with jump by using deep learning is mainly applied in the financial field, and has important applications in financial derivatives pricing, insurance investment decision-making, small and micro enterprise financing, risk measurement mechanism and other issues.

2. Background Knowledge

2.1. A class of PDE

The Materials and Methods should be described with sufficient details to allow others to replicate and build on the published results. Please note that the publication of your manuscript implicates that you must make all materials, data, computer code, and protocols associated with the publication available to readers. Please disclose at the submission stage any restrictions on the availability of materials or information. New methods and protocols should be described in detail while well-established methods can be briefly described and appropriately cited.

We consider a class of semilinear parabolic PDE with jump term in the following form:

Let $T \in (0, \infty)$, $d \in \mathbb{N}$, and $f: \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}$ and $g: \mathbb{R}^d \to \mathbb{R}$ be continuous functions. For all $t \in [0, T]$, $x \in \mathbb{R}^d$, function to be solved $u = u(t, x) \in C^{1,2}([0, T] \times \mathbb{R}^d, \mathbb{R})$has a terminal value condition $u(T, x) = g(x)$ and satisfies:

$$\frac{\partial u}{\partial t}(t, x) - Lu(t, x) - f\big(t, x, u(t, x), \sigma^T(t, x)\nabla_x u(t, x), Bu(t, x)\big) = 0 \tag{2.1}$$

Where we introduce two operators:

$$Lu(t, x) = \nabla_x u(t, x)[\mu(x) - \lambda\beta(x)] + \frac{1}{2}Tr\big(\sigma\sigma^T(t, x)(Hess_x u)(t, x)\big) + \lambda\{u\big(t, x + \beta(x)\big) - u(t, x)\} \tag{2.2}$$

$$Bu(t, x) = u\big(t, x + \beta(x)\big) - u(t, x) \tag{2.3}$$

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 April 2023 doi:10.20944/preprints202304.0369.v1

Here t is the time variable, x is the d-dimensional space variable, f is the nonlinear part of the equation, $\nabla_x u$ represents the gradient of u with respect to x, $Hess_x u$ represents the Hessian matrix of u with respect to x. In particular, this equation can be regarded as a special case of partial integro-differential equation. What we are interested in is the solution at $t = 0$, $x = \xi \in \mathbb{R}^d$, which is $u(0, \xi)$.

2.2. Backward Stochastic Differential Equations with Jumps

Let $(\Omega, \mathcal{F}, P)$ be a complete probability space, $W: [0, T] \times \Omega \to \mathbb{R}^d$ be the d-dimensional standard Brownian motion in this probability space, $\{\mathcal{F}_t\}_{t \in [0,T]}$ be the normal filtration generated by W in space $(\Omega, \mathcal{F}, P)$. $\{X_t\}_{0 \le t \le T}$, $\{Y_t\}_{0 \le t \le T}$, $\{Z_t\}_{0 \le t \le T}$, $\{U_t\}_{0 \le t \le T}$ are integrable $\mathbb{F}$-adapted stochastic processes. For a class of forward backward stochastic differential equations with jump terms, it has the following form:

$$\begin{cases} X_t = X_0 + \int_0^t \mu(s, X_s) ds + \int_0^t \sigma(s, X_s) dW_t + \int_0^t \beta(s, X_s) d\tilde{N}_s \\ Y_t = g(X_T) + \int_t^T f(s, X_s, Y_s, Z_s, U_s) ds - \int_t^T Z_s dW_s - \int_t^T U_s d\tilde{N}_s \end{cases} \tag{2.4}$$

Where $d\tilde{N}_t$ is the compensated Poisson measure, $\tilde{N}_t$ is a centralized Poisson process with the compensator intensity λt such that $\tilde{N}_t = N_t - \lambda t$, N_t is the Poisson process in the probability space $(\Omega, \mathcal{F}, P)$ such that $E(N_t) = \lambda t$.

Under the standard Lipschitz assumptions on the coefficients μ, σ, β, f, g, the existence and uniqueness of the solution have been proved. [8]

2.3. The Generalized Nonlinear Feynman-Kac Formula

Under the appropriate regularization assumption, and $u(t, x) \in C([0, T] \times R^d)$ satisfies equation (1) and linear growth condition $|u(t, x)| \le K(1 + |x|)$, $|\nabla_x u(t, x)| \le K(1 + |x|)$, $(t, x) \in C([0, T] \times R^d)$, the following relationships are established almost everywhere:

$$Y_t = u(t, x) \in R \tag{2.5}$$

$$Z_t = \nabla_x u(t, x) \sigma(t, x) \in R^d \tag{2.6}$$

$$U_t = u\big(t, x + \beta(x, \cdot)\big) - u(t, x) \in R \tag{2.7}$$

(Y_t, Z_t, U_t) is the only solution of BSDE. [9,10]

2.4. Improvement of Neural Network Parameter Optimization Algorithm

Adam (Adaptive Moment Estimation) algorithm is an algorithm that combines RMSProp algorithm with classical momentum in physics. It dynamically adjusts the learning rate of each parameter by using the first-order moment estimation and second-order moment estimation of gradient. Adam optimizer is one of the most popular classical optimizers in deep learning, and it also shows excellent performance in practice. [11]

Although Adam combines RMSprop with momentum, the adaptive moment estimation with Nesterov acceleration is often better than momentum. Therefore, we consider introducing Nesterov acceleration effect [12] into Adam algorithm, that is, using Nadam (Nesterov-accelerated Adaptive Moment Estimation) optimization algorithm. The calculation formula is as follows:

$$\hat{g}_t = \frac{g_t}{1 - \prod_{i=1}^{t} \mu_i} \tag{2.8}$$

$$m_t = \mu * m_{t-1} + (1 - \mu) * g_t \tag{2.9}$$

$$\hat{m}_t = \frac{m_t}{1 - \prod_{i=1}^{t+1} \mu_i} \tag{2.10}$$

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 April 2023 doi:10.20944/preprints202304.0369.v1

$$n_t = \nu * n_{t-1} + (1-\nu) * g_t^2 \tag{2.11}$$

$$\hat{n}_t = \frac{n_t}{1-\nu^t} \tag{2.12}$$

$$\bar{m}_t = (1-\mu_t)\hat{g}_t + \mu_{t+1}\hat{m}_t \tag{2.13}$$

$$\theta_t = \theta_{t-1} - \eta \frac{\bar{m}_t}{\sqrt{\hat{n}_t} + \varepsilon} \tag{2.14}$$

For parameters of neural network θ_t , g_t is the gradient of θ_t.Besides, m_t and n_t are the first order moment estimate and the second order moment estimate of the gradient respectively, which can be regarded as the estimation of the expectation $E|g_t|$ and $E|g_t^2|$, here μ and ν are their attenuation rates respectively. Moreover, $\hat{m}_t$ and $\hat{n}_t$ are the correction for m_t and n_t, which can be approximated as an unbiased estimate of the expectation. It can be seen that Nadam has a stronger constraint on the learning rate, and has a more direct impact on the update of the gradient. Therefore, we try to apply the new optimization algorithm to our method and compare the results with the traditional Adam algorithm.

3. Main Theorem

3.1. Basic Ideas

In this paper we propose a deep learning-based PDE numerical solution for nonlinear PDE with jump terms in the form of Equation (2.1). The basic idea of the algorithm are as follows:

By using the generalized nonlinear Feynman-Kac formula, the PDEs to be solved can be equivalently constructed using BSDEs with jumps.

Taking the gradient operator and the jump term of the solution of the function to be solved as policy functions, the problem of solving the numerical solution of BSDEs can be considered as a stochastic control problem, which can be further considered as a reinforcement learning problem.

Two different depth neural networks are used to approximate this pair of high-dimensional strategy functions, and the neural networks are trained by deep learning method to obtain the numerical solution of the original equation.

3.2. Transforming the Nonlinear PDE Numerical Solution Problem into a Stochastic Control Problem

It is well known that PDE and BSDE can be linked by the generalized nonlinear Feynman-Kac formula under appropriate assumptions. Thus, the solution $Y_t(x)$ of the above equation is equivalent to the viscous solution $u(t,x)$ of the semilinear parabolic PDE partial differential equation with jump term in (2.1). [13,14] Because the numerical solution of the traditional partial differential equation does not perform well in high dimensions, we can estimate the solution $u(t,x)$ of equation (2.1) by estimating the solution of equation (2.4).

For the stochastic control problem related to equation (2.4), under the appropriate regularity assumption, for the nonlinear function f, it holds that a group of solution functions composed of $u(0,\xi) \in \mathbb{R}$, $\nabla_x u(t,x) \in \mathcal{A}$ and $u\big(t,x+\beta(x)\big) - u(t,x) \in \mathbb{R}$ is the unique global minimum of the following functions $(y,Z,U) \in \mathbb{R} \times \mathcal{A} \times \mathbb{R}$:

$$(y,Z,U) \longmapsto \mathbb{E}\left[\left|Y_T^{y,Z,U} - g(X_T)\right|^2\right] \in [0,\infty] \tag{3.1}$$

We regard Z and U as the policy function of deep learning problem, and use DNN to approximate Z and U. In this way, the stochastic process $u(t,x)$ corresponds to the solution function of the stochastic control problem and can be solved by using the strategy functions Z and U, and thus we can transform the nonlinear partial differential equation into a stochastic control problem.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 April 2023 doi:10.20944/preprints202304.0369.v1

3.3. Forward Discretization of the Backward Stochastic Differential Equations with Jumps

For the following BSDE:

$$\begin{cases} X_t = x + \int_0^t \mu(s, X_s)ds + \int_0^t \sigma(s, X_s)dW_t + \int_0^t \beta(s, X_s)d\tilde{N}_s \\ Y_t = g(X_T) + \int_t^T f(s, X_s, Y_s, Z_s)ds - \int_t^T Z_s dW_s - \int_t^T U_s d\tilde{N}_s \end{cases} \tag{3.2}$$

We will discretize it in the time dimension and divide the time interval $[0, T]$ into N partitions, so that $t_0, t_1, \dots, t_N \in [0, T]$ and $0 = t_0 < t_1 < \cdots < t_N = T$, assuming $t_i - t_{i-1} = h$, we use Euler scheme for discretization:

$$X_{t_{n+1}} - X_{t_n} = \mu(t_n, X_{t_n})\Delta t_n + \sigma(t_n, X_{t_n})\Delta W_{t_n} + \beta(t_n, X_{t_n})\Delta\tilde{N}_{t_n} \tag{3.3}$$

$$Y_{t_{n+1}} = Y_{t_n} - f\left(t_n, X_{t_n}, Y_{t_n}, (\nabla u\sigma)(t_n, X_{t_n})\right)\Delta t_n + \langle(\nabla u\sigma)(t_n, X_{t_n}), W_{t_{n+1}} - W_{t_n}\rangle + \langle U(t_n, X_{t_n}), \tilde{N}_{t_{n+1}} - \tilde{N}_{t_n}\rangle \tag{3.4}$$

Where $\Delta t_n = t_{n+1} - t_n$, $\Delta W_{t_n} = W_{t_{n+1}} - W_{t_n}$, $\Delta\tilde{N}_{t_n} = \tilde{N}_{t_{n+1}} - \tilde{N}_{t_n}$.

In this way we can start from the initial value X_0 of (3.2) and finally obtain the approximation of the Euler format of (3.2) at N partitions.

3.4. General Framework for Neural Networks

Two feedforward neural networks are established at each time step $t = t_n$ in (3.3) and (3.4). One is to approximate the gradient of the unknown solution, which means to approximate the function $X_{t_n} \to Z_{t_n}$: $\nabla_x u(t, x)\sigma(t, x)$, and this neural network is recorded as $\mathrm{NN}_{\theta_{z_n}}(\mathrm{x})$, such that θ_{z_n} represents all parameters of this neural network and Z_n indicates that this neural network is used to approximate Z_{t_n} at time t_n. Another is to approximate the jump process of the unknown solution, which means to approximate the function $X_{t_n} \to U_{t_n}: u(t, x + \beta(x,\cdot)) - u(t, x)$ and this neural network is recorded as $\mathrm{NN}_{\theta_{U_n}}(\mathrm{x})$, such that θ_{U_n} represents all parameters of this neural network, and U_n indicates that this network is used to approximate U_{t_n} at time t_n. For the convenience of expression, we can suppose $\theta_Z = \{\theta_{Z_1}, \dots, \theta_{Z_{N-1}}\}, \theta_U = \{\theta_{U_1}, \dots, \theta_{U_{N-1}}\}$, $\theta = \{\theta_Z, \theta_U\}$. As shown in Figure 1, all sub neural networks are stacked together to form a complete neural network.

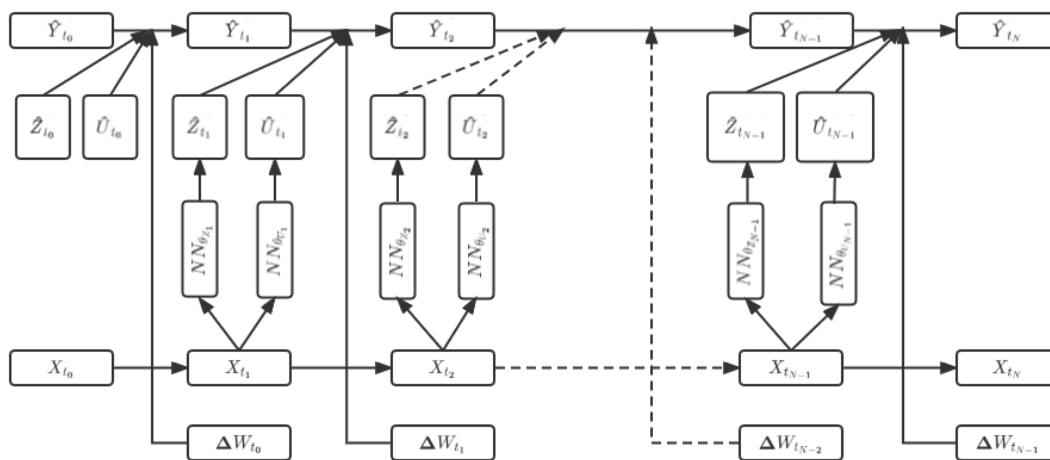

Figure 1. Deep neural network framework.

$X_{t_0} = \xi \in \mathbb{R}^d$ in the initial layer is a random variable. $u(t_0, X_{t_0})$, $\nabla_x u(t_0, X_{t_0})$ and $u\left(t_0, X_{t_0} + \beta(X_{t_0}, \cdot)\right) - u(t_0, X_{t_0})$ in the initial layer is unknown, and we treat it as parameters of the neural network. And $u(t_0, X_{t_0})$, $\nabla_x u(t_0, X_{t_0})$ and $u\left(t_0, X_{t_0} + \beta(X_{t_0}, \cdot)\right) - u(t_0, X_{t_0})$ corresponds to the value of the BSDE with jumps as follows:

$$Y_0 = u(0, \xi) \tag{3.5}$$

$$Z_0 = \nabla_x u(t_0, \xi) \tag{3.6}$$

And

$$U_0 = u\big(t_0, \xi + \beta(\xi, \cdot)\big) - u(t_0, \xi) \tag{3.7}$$

In this neural network, the X_{t_n} of the current layer is related to the $X_{t_{n-1}}$ of the previous layer, and at the same time the $u(t_n, X_{t_n})$ of the current layer is related to the $X_{t_{n-1}}$, $u(t_{n-1}, X_{t_{n-1}})$, $\nabla_x u(t_{n-1}, X_{t_{n-1}})$ and $u\left(t_{n-1}, X_{t_{n-1}} + \beta(X_{t_{n-1}}, \cdot)\right) - u(t_{n-1}, X_{t_{n-1}})$ of the previous layer. However, there are no $\nabla_x u(t_{n-1}, X_{t_{n-1}})$ and $u\left(t_{n-1}, X_{t_{n-1}} + \beta(X_{t_{n-1}}, \cdot)\right) - u(t_{n-1}, X_{t_{n-1}})$ in the current layer. Therefore, as shown in the Figure 1, our solution is to start from the X_{t_n} of the current layer to build two sub neural networks to represent these two values. In addition, the construction of the final loss function can be obtained from a given terminal value condition $u(T, x)$, that is, $u(t_N, X_{t_N})$ in the neural network, which also corresponds to $g(X_T)$ in the nonlinear Feynman-Kac formula.

Specifically, for $n = \{0,1, \dots, N-1\}$, we can set $\hat{Y}_0$, $\hat{Z}_0$ as parameters to obtain the appropriate Euler approximation $\hat{Y}$ of the process forward:

$$\hat{Y}_{t_{n+1}} = \hat{Y}_{t_n} - f(t_n, X_{t_n}, \hat{Y}_{t_n}, \hat{Z}_{t_n})\Delta t_n + \langle \hat{Z}_{t_n}, \Delta W_{t_n} \rangle + \langle \hat{U}_{t_n}, \Delta \tilde{N}_{t_n} \rangle \tag{3.8}$$

For all appropriate θ_{z_n} , we have $\hat{Z}_{t_n} = \mathrm{NN}_{\theta_{z_n}}(X_{t_n}) \approx (\nabla u \sigma)(t_n, X_{t_n})$. For all appropriate θ_{U_n} , we have $\hat{U}_{t_n} = \mathrm{NN}_{\theta_{U_n}}(X_{t_n}) \approx u\left(t_n, X_{t_n} + \beta(X_{t_n}, \cdot)\right) - u(t_n, X_{t_n})$. Then we can get a suitable approximation of $u(0, \xi)$:

$$\hat{Y}_0 \approx u(0, \xi) \tag{3.9}$$

The mean square error between the final output $\hat{Y}_{t_N}$ of neural network and the true value $g(X_{t_N})$ at the terminal time is selected as the loss function:

$$\theta \longmapsto loss(\theta) = \mathbb{E}\left[\left|\hat{Y}_{t_N} - g(X_{t_N})\right|^2\right] \in [0, \infty] \tag{3.10}$$

θ is the set of all training parameters of the above system, such that $\theta = \{\hat{Y}_0, \hat{Z}_0, \hat{U}_0, \theta_{z_1}, \dots \theta_{z_{N-1}}, \theta_{U_1}, \dots, \theta_{U_{N-1}}\}$. This loss function is used since $Y_T = g(X_T)$. And the expectation in the loss function in (3.10) is the expectation for all sample paths, but due to the infinite number of sample paths, it is not possible to traverse the entire training set. Therefore, we adopt an optimization method based on stochastic gradient descent. In each iteration of gradient descent, only a portion of samples are selected to estimate the loss function, thereby obtaining the gradient of the loss function over all parameters, and obtaining a one-step neural network parameter update.

Like this, the back propagation of this neural network uses optimizer to update the parameters layer by layer. When DNN is trained, $\hat{Y}_0$ is fixed as the parameter value. Take this parameter value out and it is the required value.

3.5. Details of the Algorithms

The detailed steps of our proposed algorithm based on deep learning to numerically solve the BSDE with jumps is presented as follows.

According to our algorithm, which uses a deep learning-based neural network solver, the BSDE with jumps in the form of equation (2.4) can be numerically solved in the following ways:

- Simulate sample paths using standard Monte Carlo methods
- Use a deep neural network (DNN) to approximate $Z = \nabla_x u(t, x)\sigma(t, x)$ and $U = u\big(t, x + \beta(x, \cdot)\big) - u(t, x)$, and then plug them into the BSDE with jumps to perform a forward iterative operation in time

For simplicity, here we use the one-dimensional case as an example, and the high-dimensional case is similar. We divide the time interval $[0, T]$ into N partitions, so that $t_0, t_1, \dots, t_N \in [0, T]$ and

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 April 2023 doi:10.20944/preprints202304.0369.v1

$0 = t_0 < t_1 < \cdots < t_N = T$, assuming $h_i = t_{i+1} - t_i$, $dW_i = W_{i+1} - W_i$ and $d\widetilde{N}_i = \widetilde{N}_{i+1} - \widetilde{N}_i$. The detailed steps are as follows:

(1) N Monte Carlo paths of the diffusion process $\left\{X_i^{(j)}: i = 0,1,\ldots,N;\ j = 1,2,\ldots,M;\ X_0^{(j)} \equiv X_0\right\}$ are sampled by Euler scheme:

$$X_{i+1}^{(j)} = X_i^{(j)} + \mu\left(t_i, X_i^{(j)}\right) h_i + \sigma\left(t_i, X_i^{(j)}\right) dW_i^{(j)} + \beta\left(t_i, X_i^{(j)}\right) d\widetilde{N}_i^{(j)} \tag{3.11}$$

This step is the same as the standard Monte Carlo method. Other discretization schemes can also be used, such as logarithmic Euler discretization or Milstein discretization.

(2) At $t_0 = 0$, the initial value Y_0, $Z_0 = \frac{\partial Y}{\partial X}(0, X_0)\sigma(0, X_0)$ and $U_0 = u\left(0, X_0 + \beta(X_0,\cdot)\right) - u(0, X_0)$ is randomly selected as parts of the neural network parameters, and Y_0, Z_0 and U_0 are all constant random numbers selected from an empirical range.

(3) At each time step t_i, given $\left\{X_i^{(j)}: j = 1,2,\ldots,M\right\}$, $\left\{Z_i^{(j)}: j = 1,2,\ldots,M\right\}$ and $\left\{U_i^{(j)}: j = 1,2,\ldots,M\right\}$ are approximated using deep neural networks, respectively. Note that every time $t = t_i$, two sub neural networks are used for all Monte Carlo paths. In our one-dimensional case, as described in Section 3.4, we introduce two sub deep neural networks: $\theta_{Z_i}: \mathbb{R} \to \mathbb{R}$, such that $\left\{Z_i^{(j)} = \theta_{Z_i}\left(X_i^{(j)}\right): j = 1,2,\ldots,M\right\}$ and $\theta_{U_i}: \mathbb{R} \to \mathbb{R}$, such that $\left\{U_i^{(j)} = \theta_{U_i}\left(X_i^{(j)}\right): j = 1,2,\ldots,M\right\}$. And $\theta_Z = \left\{\theta_{Z_1},\ldots,\theta_{Z_{N-1}}\right\}$, $\theta_U = \left\{\theta_{U_1},\ldots,\theta_{U_{N-1}}\right\}$, $\theta = \{\theta_Z, \theta_U\}$.

(4) For $t_i \in [0, T]$, we have

$$Y_{i+1}^{(j)} = Y_i^{(j)} - f\left(t_i, X_i^{(j)}, Y_i^{(j)}, Z_i^{(j)}, U_i^{(j)}\right) h_i + \langle Z_i^{(j)}, dW_i^{(j)}\rangle + \langle U_i^{(j)}, d\widetilde{N}_i^{(j)}\rangle \tag{3.12}$$

According to this formula, we can directly calculate the $Y_{i+1}^{(j)}$ at the next time step. This process does not require any parameters to be optimized. In this way, the BSDE with jumps propagates forward in time direction from t_i to t_{i+1}. Along each Monte Carlo path, as the BSDE with jumps propagating forward in time from 0 to T, $Y_N^{(j)}$ is estimated as $Y_N^{(j)}(Y_0, Z_0, U_0, \theta')$, where $\theta' = (\theta_Z, \theta_U)$ is all the hyper-parameters for the N-1 layers neural networks.

(5) Calculate the following loss function:

$$L_{Forward} = \frac{1}{M}\sum_{j=1}^{M}\left(Y_N^{(j)}(Y_0, Z_0, U_0, \theta) - g\left(X_N^{(j)}\right)\right)^2 \tag{3.13}$$

Where g is the terminal function.

(6) Use stochastic optimization algorithms to minimize the loss function:

$$\left(\widetilde{Y}_0, \widetilde{z}_0, \widetilde{\theta}\right) = \underset{(Y_0, z_0, \theta)}{\operatorname{argmin}} \frac{1}{M}\sum_{j=1}^{M}\left(Y_N^{(j)}(Y_0, z_0, \theta) - g\left(X_N^{(j)}\right)\right)^2 \tag{3.14}$$

The estimated value $\widetilde{Y}_0$ is the desired initial value at time t=0.

4. Numerical Results

In this section, we will use the depth neural network to code the theoretical framework of the proposed algorithm. In this paper, three classical high-dimensional PDEs with important applications in finance-related fields are selected for numerical simulation, they are: financial derivative pricing under jump-diffusion model, Bond pricing under Vasicek model with jump, Hamilton-Jacobi-Bellman equation.

The numerical experiments in this paper uses a 64-bit Windows 10 operating system, based on the PyTorch deep learning framework in Python. All examples in this section are calculated based on 1000 sample paths, each chosen for a time partition of N = 100. We ran each example 10 times independently to calculate the average result. All neural network parameters will be initialized by uniform distribution. Each sub neural network of each time node contains 4 layers: one d-dimensional input layer, two (d+10)-dimensional hidden layers, one d-dimensional output layer. We

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 April 2023 doi:10.20944/preprints202304.0369.v1

use the rectifier function (ReLU) as the activation function. Batch Normalization (BN) techniques are used after each linear transformation and before activation.

4.1. Pricing of Financial Derivatives under Jump Diffusion Model

In this section, we will apply the proposed method to the pricing of derivatives related to a 100-dimensional jump diffusion model. In this model, the stock price X_t satisfies the following jump diffusion model [15]:

$$dX_t = (r - \lambda k)X_t dt + \sigma X_t dW_t \quad \text{if an asset price does not jump}$$

$$= (r - \lambda k)X_t dt + \sigma X_t dW_t + (V - 1)X_t \quad \text{if an asset price jumps}$$

Where r is the constant discount rate, λ is the average number of jumps per unit time of the stock price, σ is the constant volatility. V represents the jump magnitude. Assuming that the jump amplitude is fixed and V is constant, we can let $k = V - 1$ and $k > -1$. This equation can be written as $dX_t = rX_t dt + \sigma X_t dW_t + kX_t d\tilde{N}_t$.

It is known that the European call option with the stock as the underlying asset satisfies the following partial differential equation:

$$\frac{\partial u}{\partial t}(t,x) + \frac{1}{2}\sigma^2 x^2 \Delta_x u(t,x) + (r - \lambda k)x\nabla_x u(t,x) + \lambda[u(t,xV) - u(t,x)] - ru(t,x) = 0$$

$$u(T,x) = (x - K)^+ \tag{4.1}$$

For all $t \in [0,T]$, $x, \omega \in \mathbb{R}^d$, $y \in \mathbb{R}$, $z \in \mathbb{R}^{1\times d}$. Suppose $d = 100$, $T = 1$, $N = 100$, $\lambda = 1$, $X_0 = (1, \dots, 1)$, $\mu(t,x) = rX_t$, $\sigma(t,x) = \sigma X_t$, $\beta(t,x) = (V-1)X_t = kX_t$, $f(t,x,y,z) = -rY$, $g(x) = \left(\max_{1\le i\le 100} x_i - 5\right)^+$, here r^i, σ^i, k^i are randomly selected in $[0,1]$, $i = 1, \dots 100$.

After calculation, Tables 1 and 2 show the important numerical results of solving the related derivative pricing problem of the 100-dimensional jump diffusion model with Adam and NAdam optimizers respectively, including that with the change of iteration steps, mean and standard deviation of loss function Y_0, mean and standard deviation of loss function, and running time. Only the numerical results of iteration steps $n \in \{1000,2000,3000,4000\}$ are selected as typical examples for display.

Table 1. Numerical results with Adam optimizer.

Number of iteration steps m	Mean of $u(0, X_0)$	Standard deviation of $u(0, X_0)$	Mean of the loss function	Standard deviation of the loss function	Runtime in second
1000	0.5989	0.1719	1.2743	7.7509	474
2000	0.4664	0.1235	1.2532	4.3095	891
3000	0.4183	0.1623	1.0894	3.7375	1271
4000	0.3940	0.1465	0.6269	2.4771	1676

Table 2. Numerical results with NAdam optimizer.

Number of iteration steps m	Mean of $u(0, X_0)$	Standard deviation of $u(0, X_0)$	Mean of the loss function	Standard deviation of the loss function	Runtime in second
1000	0.2821	0.0625	1.1528	2.7450	549
2000	0.3019	0.0484	0.6747	2.0031	828
3000	0.3084	0.0406	0.4911	1.6566	1206
4000	0.3117	0.0356	0.4134	1.4535	1651

It can be seen from Tables 1 and 2 that when the iteration steps are the same, the calculation time required for using the two optimizers is not very different. As the iteration progresses, the loss

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 April 2023 doi:10.20944/preprints202304.0369.v1

function value of the Nadam optimizer is smaller than that of the Adam optimizer at the same iteration steps, and the final loss function value is also smaller.

Figures 2 and 3 show the changes of the initial value estimate and loss function with the number of iteration steps when Adam and NAdam optimizers are used for solving. It can be seen that the initial value estimates and loss functions of the numerical solutions of the two methods converge rapidly. However, the convergence speed of the latter is obviously faster than that of the former, whether it is the initial value estimate or the loss function. Among them, the former tends to converge after about 1500 iterations, while the latter tends to converge after about 700 iterations. When the iteration is 4000 times, the numerical solution obtained by Adam algorithm is 0.32143775, while the numerical solution obtained by NAdam algorithm is 0.32273737, which is similar.

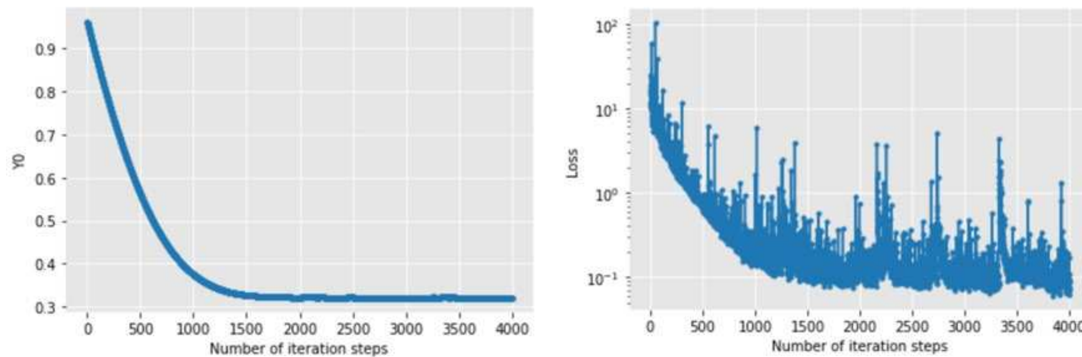

Figure 2. Changes of initial value estimation and loss function with iteration steps under Adam optimizer.

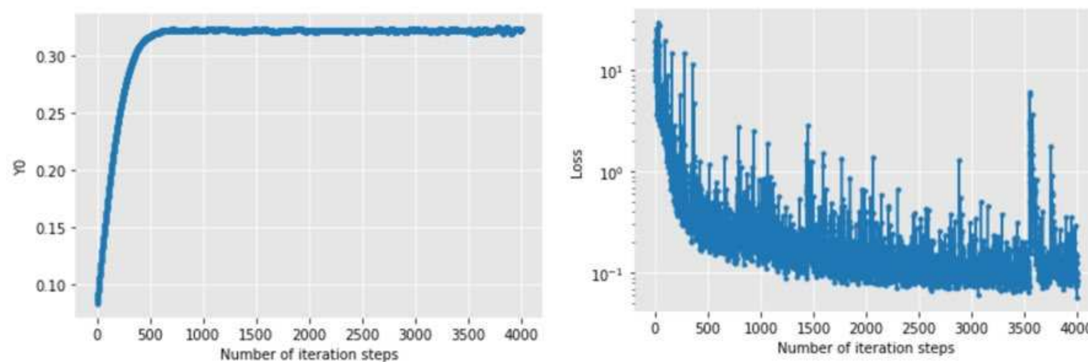

Figure 3. Changes of initial value estimation and loss function with iteration steps under NAdam optimizer.

4.2. Bond Pricing under the Jumping Vasicek Model

In this section, we use the proposed method to solve the pricing problem of a class of bonds with interest rates subject to the Vasicek model with jumps. [16,17] In this model, short-term interest rate X_t obeys the following stochastic differential equations:

$$dX_t = a(b - X_t)dt + \sigma dW_t + kd\tilde{N}_t \tag{4.2}$$

i.e., each part of X_t follows:

$$dX_t^i = a^i\left(b^i - X_t^i\right)dt + \sigma^i dW_t + k^i d\tilde{N}_t \tag{4.3}$$

For all $t \in [0,T]$, $x, \omega \in \mathbb{R}^d$, $y \in \mathbb{R}$, $z \in \mathbb{R}^{1\times d}$. Suppose $d = 100$, $T = 1$, $N = 100$, $\lambda = 1$, $X_0 = (1, \dots, 1)$, $\mu(t,x) = (\mu^1, \dots, \mu^i, \dots, \mu^d)$, $\mu^i = a^i(b^i - x^i)$, $\sigma(t,x) = (\sigma^1, \dots, \sigma^i, \dots, \sigma^d)$, $\beta(t,x) = (\beta^1, \dots, \beta^i, \dots, \beta^d)$, a^i, b^i, σ^i, $\beta^i$are randomly selected in[0,1], $f(t,x,y,z) = -\left(\max_{1\le i\le n} X^i\right)Y$, $g(x) = 1$. Then, for the zero-coupon bond price u paying 1 at maturity T under the above jump Vasicek model,

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 April 2023 doi:10.20944/preprints202304.0369.v1

it satisfies that $u(T,x)=1$ for all $t\in[0,T]$, $x\in\mathbb{R}^d$, and the following partial differential equation holds:

$$\frac{\partial u}{\partial t}(t,x)+\sum_{i=1}^{n}[a^i(b^i-x^i)-\beta^i]\frac{\partial u}{\partial x^i}+\frac{1}{2}\sum_{1\le i,j\le n}\sigma^i\sigma^j\sqrt{x^ix^j}\frac{\partial^2 u}{\partial x^i\partial x^j}+u\big(t,x+\beta(x)\big)-u(t,x)-\Big(\max_{1\le i\le n}X^i\Big)u=0 \quad (4.4)$$

After calculation, Tables 3 and 4 show the important numerical results of solving a class of bond pricing problems with interest rates subject to the Vasicek model with jumps using Adam and NAdam optimizers respectively, including that with the change of iteration steps, mean and standard deviation of loss function Y_0, mean and standard deviation of loss function, and running time. Only the numerical results of iteration steps $n\in\{1000,2000,3000,4000\}$ are selected as typical examples for display.

Table 3. Numerical results with Adam optimizer.

Number of iteration steps m	Mean of $u(0,X_0)$	Standard deviation of $u(0,X_0)$	Mean of the loss function	Standard deviation of the loss function	Runtime in second
1000	0.2096	0.0435	0.1405	0.4825	538
2000	0.2300	0.0370	0.0820	0.3464	924
3000	0.2396	0.0331	0.0585	0.2849	1296
4000	0.2450	0.0302	0.0463	0.2476	1779

Table 4. Numerical results with NAdam optimizer.

Number of iteration steps m	Mean of $u(0,X_0)$	Standard deviation of $u(0,X_0)$	Mean of the loss function	Standard deviation of the loss function	Runtime in second
1000	0.2828	0.0293	0.0323	0.1310	652
2000	0.2772	0.0215	0.0224	0.1033	1326
3000	0.2757	0.0177	0.0172	0.0855	1957
4000	0.2750	0.0154	0.0150	0.0753	2581

It can be seen from Tables 3 and 4 that with the iteration, the loss function value of the NAdam optimizer is smaller than that of the Adam optimizer at the same iteration steps, and the final loss function value is smaller, but the cost is that the operation time becomes longer.

Figures 4 and 5 show the changes of the initial value estimate and loss function with the number of iteration steps when Adam and NAdam optimizers are used for solving. It can be seen that the initial value estimates and loss functions of the numerical solutions of the two methods converge rapidly. However, the convergence speed of the latter is obviously faster and more stable than the former, whether it is the initial value estimate or the loss function. When the iteration is 4000 times, the numerical solution obtained by Adam algorithm is 0.25530547, while the numerical solution obtained by NAdam algorithm is 0.27403888. The difference is not significant.

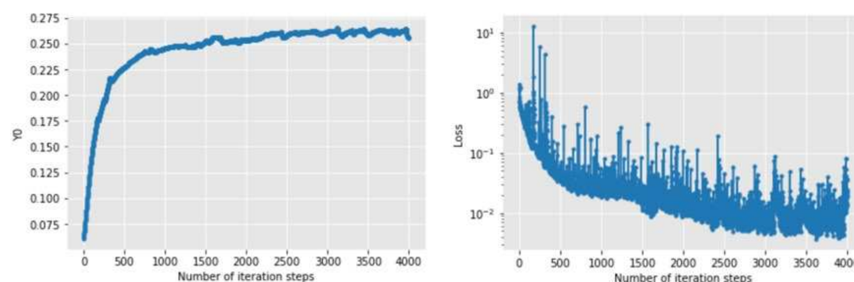

Figure 4. Changes of initial value estimation and loss function with iteration steps under Adam optimizer.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 April 2023 doi:10.20944/preprints202304.0369.v1

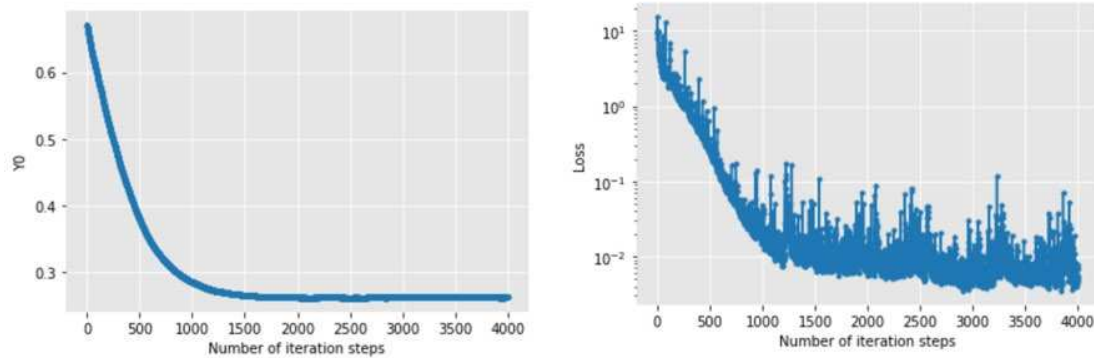

Figure 5. Changes of initial value estimation and loss function with iteration steps under NAdam optimizer.

4.3. Hamilton-Jacobi-Bellman (HJB) Equation

In fields such as finance, investment, and risk management, optimization and control problems are often involved, and these problems are often represented by stochastic optimal control models. One way to solve this kind of control problem is to obtain and solve the Hamilton-Jacobi-Bellman equation (HJB equation for short) of the corresponding control problem according to the principle of dynamic programming. In this section, we use the proposed method to solve a class of 100-dimensional HJB equations [9]:

For all $t \in [0,T]$, $x,\omega \in \mathbb{R}^d$, $y \in \mathbb{R}$, $z \in \mathbb{R}^{1\times d}$. Suppose $d = 100$, $T = 1$, $N = 100$, $\lambda = 1$, $X_0 = (1,\dots,1)$, $\mu(t,x) = 0$, $\sigma(t,x) = \sqrt{2}$, $\beta(t,x) = \beta^T x$, where $\beta^T = (\beta^1,\dots,\beta^i,\dots,\beta^d)$, β^i are randomly selected in $[0,1]$, $f(t,x,y,z) = -\|z\|^2_{\mathbb{R}^{1\times d}}$, $g(x) = ln\big((1+\|x\|^2)/2\big)$, it satisfies that $u(T,x) = ln\big((1+\|x\|^2)/2\big)$ for all $t \in [0,T]$, $x \in \mathbb{R}^d$, and the following partial differential equation holds:

$$\frac{\partial u}{\partial t}(t,x) + \Delta_x u(t,x) + u\big(t,x+\beta(x)\big) - u(t,x) - \nabla_x u(t,x)\beta(x) = \|\nabla_x u(t,x)\|^2 \tag{4.5}$$

After calculation, the important numerical results of solving a class of HJB equations with Adam and NADam optimizers are shown in Tables 5 and 6 respectively, including that with the change of iteration steps, mean and standard deviation of loss function Y_0, mean and standard deviation of loss function, and running time. Only the numerical results of iteration steps $n \in \{2000,4000,6000,8000\}$ are selected as typical examples for display.

Table 5. Numerical results with Adam optimizer.

Number of iteration steps m	Mean of $u(0,X_0)$	Standard deviation of $u(0,X_0)$	Mean of the loss function	Standard deviation of the loss function	Runtime in second
2000	1.2504	0.1521	1.3142	1.4709	857
4000	1.4850	0.2820	0.9090	1.1166	1770
6000	1.7704	0.4768	0.7447	0.9412	2906
8000	2.0139	0.5904	0.6314	0.8386	3864

Table 6. Numerical results with NAdam optimizer.

Number of iteration steps m	Mean of $u(0,X_0)$	Standard deviation of $u(0,X_0)$	Mean of the loss function	Standard deviation of the loss function	Runtime in second
2000	0.8762	0.1941	1.2294	1.5767	1649
4000	1.2456	0.4419	0.8682	1.1733	3104
6000	1.6858	0.7241	0.6715	0.9980	4527

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 April 2023 doi:10.20944/preprints202304.0369.v1

8000	1.9266	0.7531	0.5606	0.8855	5948

It can be seen from Tables 5 and 6 that with the iteration, the loss function value of the NAdam optimizer is smaller than that of the Adam optimizer at the same iteration steps, and the final loss function value is smaller, but the operation time is longer.

Figures 6 and 7 show the changes of the initial value estimate and loss function with the number of iteration steps when Adam and NAdam optimizers are used for solving. It can be seen that the convergence speed of the latter is obviously faster than that of the former, whether it is the initial value estimate or the loss function. Among them, the former tends to converge after about 7000 iterations, while the latter tends to converge after about 5500 iterations. After 8000 iterations, the numerical solution obtained by Adam algorithm is 2.73136686, while the numerical solution obtained by NAdam algorithm is 2.6456454, with little difference between the two.

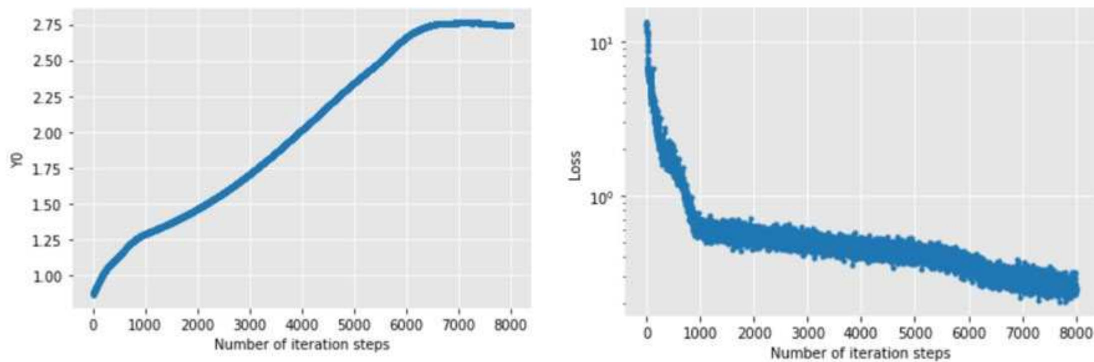

Figure 6. Changes of initial value estimation and loss function with iteration steps under Adam optimizer.

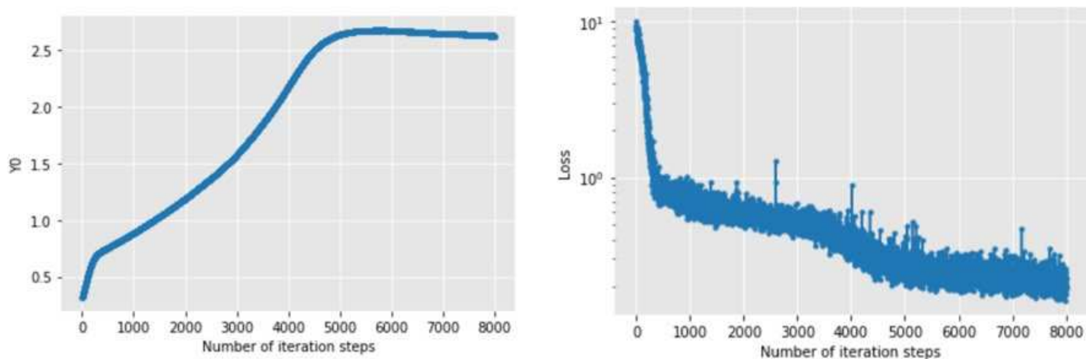

Figure 7. Changes of initial value estimation and loss function with iteration steps under NAdam optimizer.

5. Conclusions

In this paper, we propose an algorithm that can be used to solve a class of partial differential equations with jump terms and their corresponding backward stochastic differential equations. Through the nonlinear Feynman-Kac formula, the above-mentioned high-dimensional nonlinear PDE with jumps and its corresponding BSDE can be expressed equivalently, and the numerical solution problem can be regarded as a stochastic control problem. Next, we treat the gradient and jump process of the unknown solution separately as policy functions, and use two neural networks at each time division to approximate the gradient and jump process of the unknown solution respectively. In this way, we can use deep learning to solve the " curse of dimensionality " caused by high-dimensional PDE with jumps and obtain numerical solutions.

Among them, jump process can depict the sudden impact of the outside world in its process, so that it can accurately depict a class of uncertain things, and this situation is most common in financial

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 April 2023 doi:10.20944/preprints202304.0369.v1

markets. In this way, we extend the numerical solution problem of high-dimensional PDE to the case of jump diffusion. Furthermore, we can use this algorithm to solve a series of problems in finance, insurance, actuarial modeling and other fields, such as the pricing problem of some special path dependent financial derivatives, and the bond pricing problem under the jump diffusion model of interest rates.

In addition, we attempt to replace the traditional Adam algorithm with the new stochastic optimization approximation algorithm and apply it to our algorithm. Next, focusing on the financial field, we applied our algorithm to solve three common high-dimensional problems in finance-related fields and then compared the results. We concluded that our algorithm performs well in numerical simulation. It is also concluded that after applying the new optimizer to the deep learning algorithm, the convergence speed of the model is mostly faster, and the generalization ability is significantly improved, at the cost of the operation time may be longer.

Funding: This research received no external funding.

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

References

1. Kang W., Wilcox L. C. Mitigating the curse of dimensionality: sparse grid characteristics method for optimal feedback control and HJB equations [J]. Computational Optimization and Applications, 2017, 68(2): 289-315.
2. E. W., Han J., Jentzen A. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and Backward Stochastic Differential Equations[J]. Communications in Mathematics and Statistics, 2017, 5(4): 349-380.
3. Han J., Jentzen A., E W. Solving high-dimensional partial differential equations using deep learning [J]. Proceedings of the National Academy of Sciences of the United States of America, 2018, 115(34): 8505-8510.
4. Han J., Hu R. Deep fictitious play for finding Markovian Nash equilibrium in multi-agent games[J]. Proceedings of The First Mathematical and Scientific Machine Learning Conference, 2020, 107:221-245
5. Beck C., E W., Jentzen A. Machine learning approximation algorithms for high-dimensional fully nonlinear partial differential equations and second-order backward stochastic differential equations[J]. Journal of Nonlinear Science, 2019, 29(4): 1563-1619.
6. Raissi M, Perdikaris P., Karniadakis G. E. Machine learning of linear differential equations using Gaussian processes[J]. Journal of Computational Physics, 2017, 348: 683-693.
7. Sirignano J., Spiliopoulos K. DGM: A deep learning algorithm for solving partial differential equations[J]. Journal of Computational Physics, 2018, 375: 1339-1364.
8. S. Tang, X. Li, Necessary conditions for optimal control of stochastic systems with random jumps[J]. SIAM Journal on Control Optimization, 1994, 32 (5) :1447–1475.
9. Rong S., Theory of stochastic differential equations with jumps and applications[M]. London: Springer, 2005. 205-290.
10. Łukasz Delong, Backward stochastic differential equations with jumps and their actuarial and financial applications [M]. London: Springer, 2013. 85-88.
11. Henry-Labordere P, Oudjane N, Tan X, Touai N, Warin X, et al. Branching diffusion representation of semilinear PDEs and Monte Carlo approximation[J]. Annales de 1'Institut Henri Poincar"e, Probabilit"es et Statistiques, 2019, 55:184-210
12. Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton, On the importance of initialization and momentum in deep learning[C]. Proceedings of the 30th International Conference on Machine Learning ICML 2013. pages 1139–1147, 2013.
13. G. Barles, R. Buckdahn, E. Pardoux, Backward stochastic differential equations and integral-partial differential equations[J]. Stochastics Reports,1997, 60:57–83.
14. R. Buckdahn, E. Pardoux, BSDE's with jumps and associated integro-partial differential equations[M]. Preprint.
15. Merton R C., Option Pricing when Underlying Stock Returns are Discontinuous[J]. Journal of Financial Economics, 1976,3:125-144.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 April 2023
doi:10.20944/preprints202304.0369.v1

16. Li S., Zhao X., Yin C., et al. Stochastic interest model driven by compound Poisson process and Brownian motion with applications in life contingencies[J]. Quantitative Finance and Economic, 2018, 2 (1) :731-745.
17. Jiang Y., Li J., Convergence of the Deep BSDE method for FBSDEs with non-Lipschitz coefficients[J]. Probability, Uncertainty and Quantitative Risk, 2021, 6(4): 391-408.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.