CQF Final Project

Yibo Wang

January 2025

Abstract

This paper serves as the report for the CQF final project, the theme of which is *Deep Neural Networks for Solving High Dimensional PDEs (DN)*.

This paper leverages the Deep Backward Stochastic Differential Equation (DeepBSDE) method to overcome the challenges facing high-dimensional problems. The DeepBSDE framework reformulates high-dimensional parabolic PDEs as backward stochastic differential equations, allowing the gradient of the solution to be approximated using neural networks. Specifically, we apply the DeepBSDE method to solve high-dimensional nonlinear Black-Scholes and HJB equations, demonstrating its efficacy in terms of accuracy and computational cost.

Our results indicate that the DeepBSDE method provides a scalable and robust approach for tackling complex high-dimensional PDEs, offering new possibilities for practical applications in finance and beyond.

---

1 Introduction

Traditional methods often fail to efficiently compute solutions for high-dimensional partial differential equations (PDE) as the dimensionality increases. **Deep neural networks (DNNs)** have emerged as powerful tools for addressing these problems across various fields.

The theme of this project, "Deep Neural Networks for Solving High Dimensional PDEs," is quite broad, as deep neural networks encompass various architectures, and high-dimensional partial differential equations arise in numerous forms across a wide range of critical fields. Among the most commonly studied are **nonlinear parabolic PDEs**. Given our focus on quantitative finance, it is only natural to explore this theme within the context of financial mathematics.

To this end, we have chosen two fundamental and widely used models in financial mathematics: the **Black-Scholes equation** and the **Hamilton-Jacobi-Bellman (HJB) equation**, which are central to applications in finance, control theory, and dynamic optimization but are notoriously difficult to solve. Our aim is to address the challenges of solving these models in high-dimensional settings using advanced deep neural network approaches. Specifically, we have opted for the **Deep Backward Stochastic Differential Equations (DeepBSDE)** framework, a method particularly suited for tackling nonlinear parabolic PDEs in the domain of quantitative finance.

The following sections will provide a detailed explanation of the methods and approaches used in this project, as well as the rationale behind their selection.

2 Background Knowledge

In this section, we will fully present the necessary background knowledge used in this paper. Because when facing a difficult task, we have to ask ourselves many questions to figure out what our main target is, and where the real problems are. For example, what is a deep neural network? Why nonlinear parabolic PDEs? How are Black-Scholes equation and HJB equation like in higher dimension? Why can we use DeepBSDE to solve them? Well, we have to work with the mathmatics first in order to get the answers.

2.1 Deep Neural Network

Deep neural network is actually an extension of Artificial Neural Network (ANN) that features multiple hidden layers, enabling it to model complex nonlinear relationships. Therefore, we have to take a look at the ANNs.

2.1.1 Artificial Neural Networks

The architecture of the Artificial Neural Network (ANN) is depicted in Figure 1, which shows $m+2$ layers, each containing n neurons.

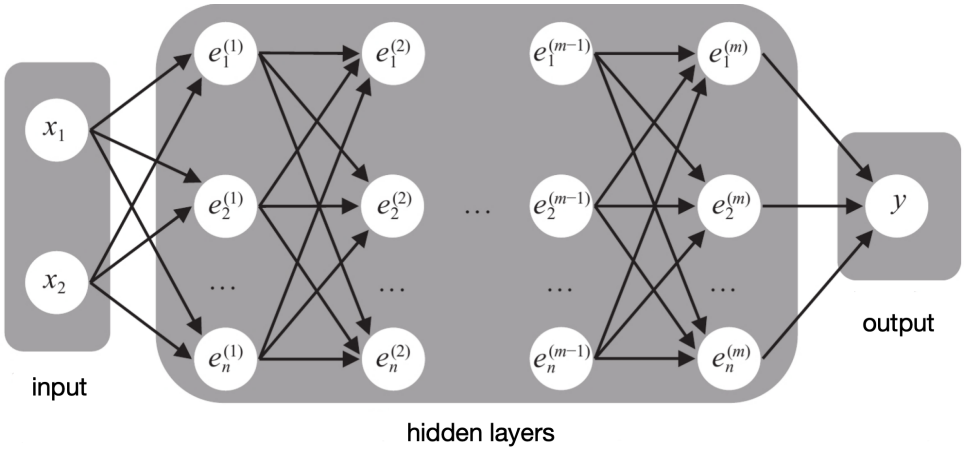

Figure 1: ANN structure

The values of the nodes are calculated as follows:

$$e_j^{(l)} = \max\left(\sum_i w_{ij}^{(l)} e_i^{(l-1)} + b_j^{(l)}, 0\right), \quad \forall l \in \{1, 2, \ldots, L\},$$

where:

- $e_j^{(l)}$: value of the j-th node in the l-th layer,
- $e_i^{(l-1)}$: output value of the i-th node in the $(l-1)$-th layer,

- $w_{ij}^{(l)}$: weight connecting the i-th node in the $(l-1)$-th layer to the j-th node in the l-th layer,
- $b_j^{(l)}$: bias of the j-th node in the l-th layer,
- $\max(\cdot, 0)$ is one of the activation functions, ReLU (Rectified Linear Unit) function.

For an input vector $x \in R^d$, the prediction function of the network reads:

$$y(x) = F(x, (W, b)).$$

2.1.2 The Role of DNN

DNNs play a crucial role in modern AI by enabling the modeling of complex, nonlinear relationships in data. Their key roles include:

- Feature Extraction : Automatically learning hierarchical representations, from basic patterns to high-level abstractions, eliminating the need for manual feature engineering.
- Nonlinear Modeling : Capturing intricate relationships in data using activation functions like ReLU, making them effective for tasks like image recognition and financial forecasting.
- Handling High-Dimensional Data : Excelling in domains with large, complex datasets (e.g., images, natural language) by avoiding the "curse of dimensionality."
- Function Approximation : Acting as universal approximators to solve tasks like differential equations or mapping inputs to outputs with high precision.
- Generalization Across Domains : Adapting to diverse fields such as computer vision, natural language processing, finance, and healthcare.

2.2 Nonlinear Parabolic PDEs

As elegant as PDE models are, their practical use is limited by the curse of dimensionality: the computational cost of solving them grows exponentially with dimensionality. Similarly, in machine learning and data analysis, nonlinear regression models face the same obstacle. Traditional approaches using polynomials, wavelets, or other basis functions inevitably encounter this problem.

Neural networks, though an old idea, have recently shown surprising effectiveness in modeling complex data. Unlike classical additive approximation methods, neural networks use compositions of simple functions, supported by universal approximation theorems for single-layer networks. While a theoretical framework for multilayer networks remains elusive, their success in artificial intelligence has inspired applications to problems plagued by the curse of dimensionality.

In this paper, we extend deep neural networks to solve high-dimensional nonlinear PDEs, particularly parabolic PDEs like the Black-Scholes and Hamilton-Jacobi-Bellman equations. By reformulating these PDEs as backward stochastic differential equations (BSDEs) and approximating gradients with deep neural networks, our approach resembles deep reinforcement learning, where the BSDE acts as the model and the solution gradient as the policy. Numerical results demonstrate the method's effectiveness in both accuracy and computational cost.

2.2.1 Why Nonlinear Parabolic PDEs in Finance Are Our Goal ?

Due to the "curse of dimensionality", there is only a very limited number of cases where practical high-dimensional algorithms have been developed in the literature. For linear parabolic PDEs, one can use the Feynman-Kac formula and Monte Carlo methods to develop efficient algorithms to evaluate solutions at any given space-time locations.

Feynman–Kac Formula :

Let $u : R_+ \times R^d \longrightarrow R$ be a function of class $C^{1,2}$ on $]0,\infty[\times R^d$, bounded on $[0,t] \times R^d$ for every $t > 0$, such that

$$\begin{cases} \dfrac{\partial u}{\partial t} = \frac{1}{2}\,\Delta u \;-\; v\,u, \quad (t,x) \in]0,\infty[\times R^d, \\ u(0,\cdot) \;=\; f. \end{cases}$$

Then we have

$$u(t,x) \;=\; E\Big\{\, f\big(B_t + x\big)\, \exp\Big(-\int_0^t v\big(B_s + x\big)\,\mathrm{d}s\Big)\Big\}, \quad (t,x) \in R_+ \times R^d.$$

In particular, such a solution is unique.

According to this formula and those well-known numerical methods, it will not be that hard for us to deal with the linear parabolic PDEs. However, when it comes to nonlinear cases, it's totally different things mathematically. But that dosen't mean that we don't have any method dealing with them.

2.2.2 General Approach to Solving Nonlinear Parabolic PDEs

For a class of inviscid Hamilton-Jacobi equations, Darbon and Osher have developed an effective algorithm in the high-dimensional case, based on the Hopf formula for the Hamilton-Jacobi equations. A general algorithm for nonlinear parabolic PDEs based on the multilevel decomposition of Picard iteration has been shown to be quite efficient on a number of examples in finance and physics. The branching diffusion method exploits the fact that solutions of semilinear PDEs with polynomial nonlinearity can be represented as an expectation of a functional of branching diffusion processes. This method does not suffer from the curse of dimensionality, but still has limited applicability due to the blow-up of approximated solutions in finite time.

Thus, we can introduce our efficient and simple deep-learning method naturally.

2.3 Methodology

The starting point of the present paper is deep learning. It should be stressed that even though deep learning has been a very successful tool for a number of applications, adapting it to the current setting with practical success is still a highly non-trivial task. Here by using the reformulation of BSDEs, we are able to cast the problem of solving PDEs as a learning problem and we design a deep learning framework that fits naturally to that setting. This has proven to be quite successful in practice.

We consider a general class of PDEs known as semilinear parabolic PDEs. These PDEs can be represented as follows:

$$\frac{\partial u}{\partial t}(t,x)+\frac{1}{2}\mathrm{Tr}\big(\sigma\sigma^T(t,x)(\mathrm{Hess}_x u)(t,x)\big)+\nabla u(t,x)\cdot\mu(t,x)+f\big(t,x,u(t,x),\sigma^T(t,x)\nabla u(t,x)\big)=0, \tag{1}$$

with some specified terminal condition $u(T,x)=g(x)$. Here t and x represent the time and d-dimensional space variable respectively, μ is a known vector-valued function, σ is a known $d\times d$ matrix-valued function, σ^T denotes the transpose associated to σ, ∇u and $\mathrm{Hess}_x u$ denote the gradient and the Hessian of function u respect to x, Tr denotes the trace of a matrix, and f is a known nonlinear function. To fix ideas, we are interested in the solution at $t=0, x=\xi$ for some vector $\xi\in R^d$.

Let $\{W_t\}_{t\in[0,T]}$ be a d-dimensional Brownian motion and $\{X_t\}_{t\in[0,T]}$ be a d-dimensional stochastic process which satisfies

$$X_t=\xi+\int_0^t \mu(s,X_s)\,ds+\int_0^t \sigma(s,X_s)\,dW_s. \tag{2}$$

Then the solution of (1) satisfies the following BSDE :

$$u(t,X_t)=u(0,X_0)-\int_0^t f\big(s,X_s,u(s,X_s),\sigma^T(s,X_s)\nabla u(s,X_s)\big)\,ds+\int_0^t \big[\nabla u(s,X_s)\big]^T\sigma(s,X_s)\,dW_s. \tag{3}$$

To derive a numerical algorithm to compute $u(0,X_0)$, we treat $u(0,X_0)\approx\theta_{u_0}$, $\nabla u(0,X_0)\approx\theta_{\nabla u_0}$ as parameters in the model and view the BSDE as a way of computing the values of u at the terminal time T, given $u(0,X_0)$ and $\nabla u(t,X_t)$. We apply a temporal discretization to the stochastic differential equations. Given a partition of the time interval $[0,T]$: $0=t_0<t_1<\cdots<t_N=T$,

we consider the Euler scheme for $n=0,\ldots,N-1$:

$$X_{t_{n+1}}-X_{t_n}\approx\mu(t_n,X_{t_n})\Delta t_n+\sigma(t_n,X_{t_n})\Delta W_n, \tag{4}$$

$$u(t_{n+1},X_{t_{n+1}})\approx\nu(t_n,X_{t_n})-f\big(t_n,X_{t_n},\nu(t_n,X_{t_n}),\sigma^T(t_n,X_{t_n})\nabla\nu(t_n,X_{t_n})\big)\Delta t_n \tag{5}$$

$$+\big[\nabla\nu(t_n,X_{t_n})\big]^T\sigma(t_n,X_{t_n})\Delta W_n, \tag{6}$$

where $\Delta t_n=t_{n+1}-t_n$ and $\Delta W_n=W_{t_{n+1}}-W_{t_n}$.

Given this temporal discretization, the path $\{X_{t_n}\}_{0\le n\le N}$ can be easily sampled. Our key step next is to approximate the function $x\mapsto\sigma^T(t,x)\,\nabla u(t,x)$ at each time step $t=t_n$

by a multilayer feedforward neural network

$$\sigma^T(t_n, X_{t_n})\,\nabla u(t_n, X_{t_n}) \;=\; \big(\sigma^T \nabla u\big)(t_n, X_{t_n}) \;\approx\; \big(\sigma^T \nabla u\big)(t_n, X_{t_n} \mid \theta_n), \tag{7}$$

for $n = 1, \ldots, N-1$, where θ_n denotes parameters of the neural network approximating

$$x \;\mapsto\; \sigma^T(t, x)\,\nabla u(t, x) \quad \text{at } t = t_n.$$

Thereafter, we stack all the sub-networks in (7) together to form a deep neural network as a whole, based on the summation of (5) over $n = 1, \ldots, N-1$. Specifically, this network takes the paths $\{X_{t_n}\}_{0\le n\le N}$ and $\{W_{t_n}\}_{0\le n\le N}$ as the input data and gives the final output, denoted by $\hat{u}\big(\{X_{t_n}\}_{0\le n\le N}, \{W_{t_n}\}_{0\le n\le N}\big)$, as an approximation of $u\big(t_N, X_{t_N}\big)$. The difference in the matching of a given terminal condition can be used to define the expected loss function

$$l(\theta) \;=\; E\Big[\|(g(X_{t_N}) \;-\; \hat{u}\big(\{X_{t_n}\}, \{W_{t_n}\}\big)\|^2\Big]. \tag{8}$$

The total set of parameters are: $\theta \;=\; \{\,\theta_{u_0},\, \theta_{\nabla u_0},\, \theta_1, \ldots, \theta_{N-1}\}$.

We can now use a stochastic gradient descent-type (SGD) algorithm to optimize the parameter θ, just as in the standard training of deep neural networks. In our numerical examples, we use the Adam optimizer. Since the BSDE is used as an essential tool, we call the methodology introduced above deep BSDE method.

BSDE Reformulation

The link between (nonlinear) parabolic PDEs and backward stochastic differential equations (BSDEs) has been extensively investigated in the literature. In particular, Markovian BSDEs give a nonlinear Feynman-Kac representation of some nonlinear parabolic PDEs

Let $(\Omega, \mathcal{F}, P)$ be a probability space, $W : [0,T] \times \Omega \to R^d$ be a d-dimensional standard Brownian motion, $\{\mathcal{F}_t\}_{t\in[0,T]}$ be the normal filtration generated by $\{W_t\}_{t\in[0,T]}$. Consider the following BSDE

$$X_t = \xi \;+\; \int_0^t \mu\big(s, X_s\big)\,\mathrm{d}s \;+\; \int_0^t \sigma\big(s, X_s\big)\,\mathrm{d}W_s, \tag{9}$$

$$Y_t = g\big(X_T\big) \;+\; \int_t^T f\big(s, X_s, Y_s, Z_s\big)\,\mathrm{d}s \;-\; \int_t^T \big(Z_s\big)^T\,\mathrm{d}W_s, \tag{10}$$

for which we are seeking a $\{\mathcal{F}_t\}_{t\in[0,T]}$-adapted solution process $\{(X_t, Y_t, Z_t)\}_{t\in[0,T]}$ with values in $R^d \times R \times R^d$. Under suitable regularity assumptions on the coefficient functions μ, σ, f, one can prove existence and up-to-indistinguishability uniqueness of solutions.

Furthermore, we have that the nonlinear parabolic partial differential equation (1) is related to the BSDE in the sense that for all $t \in [0,T]$ it holds P-a.s. that

$$Y_t \;=\; u\big(t, X_t\big) \quad \text{and} \quad Z_t \;=\; \sigma^T\big(t, X_t\big)\,\nabla u\big(t, X_t\big). \tag{11}$$

Therefore, we can compute the quantity $u(0, X_0)$ associated to (1) through Y_0 by solving the BSDE. Then we discretize the equation temporally and use neural networks to approximate the spacial gradients and finally the unknown function, as introduced in the former section of the paper.

3 Pratical Applications & Numerical Results

3.1 Nonlinear Black–Scholes Equation with Default Risk

A key issue in the trading of financial derivatives is to determine an appropriate fair price. Black & Scholes illustrated that the price u of a financial derivative satisfies a parabolic PDE, nowadays known as the Black–Scholes equation. The Black–Scholes model can be augmented to take into account several important factors in real markets, including defaultable securities, higher interest rates for borrowing than for lending, transaction costs, uncertainties in the model parameters, etc. Each of these effects results in a nonlinear contribution in the pricing model. In particular, the credit crisis and the ongoing European sovereign debt crisis have highlighted the most basic risk that has been neglected in the original Black-Scholes model, the default risk.

Ideally the pricing models should take into account the whole basket of underlyings that the financial derivatives depend on, resulting in high-dimensional nonlinear PDEs. However, existing pricing algorithms are unable to tackle these problems generally due to the curse of dimensionality. To demonstrate the effectiveness of the deep BSDE method, we study a special case of the recursive valuation model with default risk. We consider the fair price of a European claim based on 100 underlying assets conditional on no default having occurred yet. When default of the claim's issuer occurs, the claim's holder only receives a fraction $\delta \in [0,1]$ of the current value. The possible default is modeled by the first jump time of a Poisson process with intensity Q, a decreasing function of the current value, i.e., the default becomes more likely when the claim's value is low. The value process can then be modeled by (1) with the generator

$$f\big(t,x,u(t,x),\sigma^T(t,x)\,\nabla u(t,x)\big) \;=\; \big(1-\delta\big)\,Q\big(u(t,x)\big)\,u(t,x) \;-\; R\,u(t,x), \tag{12}$$

where R is the interest rate of the risk-free asset. We assume that the underlying asset price moves as a geometric Brownian motion and choose the intensity function Q as a piecewise-linear function of the current value with three regions ($v^h < v^\ell < v^\gamma$) :

$$Q(y) = \mathbf{1}_{(-\infty,\,v^h)}(y)\,\gamma^h \;+\; \mathbf{1}_{[v^h,\,v^\ell)}(y)\left[\,\frac{\gamma^h-\gamma^\ell}{v^h-v^\ell}\,\big(y-v^h\big) \;+\; \gamma^h\right]. \tag{13}$$

The associated nonlinear Black–Scholes equation in $[0,T]\times R^{100}$ becomes

$$\frac{\partial u}{\partial t}(t,x) + \bar{\mu}\cdot\nabla u(t,x) + \frac{\sigma^2}{2}\,\sum_{i=1}^{d}|x_i|^2\,\frac{\partial^2 u}{\partial x_i^2}(t,x) - \big(1-\delta\big)\,Q\big(u(t,x)\big)\,u(t,x) - R\,u(t,x) \;=\; 0. \tag{14}$$

We choose $T=1$, $\delta=2/3$, $R=0.02$, $\bar{\mu}=0.02$, $\sigma=0.2$, $v^h=50$, $v^\ell=70$, $\gamma^h=0.2$, $\gamma^\ell=0.02$, and the terminal condition $g(z)=\min\{z_1,\ldots,z_{100}\}$ for $z=(x_1,\ldots,x_{100})\in R^{100}$. Fig. 1 shows the mean and the standard deviation of θ_{u_0} as an approximation of $u(t=0,x=(100,\ldots,100))$, with the final relative error being 0.46%. The not explicitly known "exact" solution of (14) at $t=0, x=(100,\ldots,100)$ has been approximately computed by means of the multilevel Picard method :

$$u(t=0,x=(100,\ldots,100)) \approx 57.300.$$

In comparison, if we do not consider the default risk, we get

$$\hat{u}(t = 0, x = (100, \ldots, 100)) \approx 60.781.$$

In this case, the model becomes linear and can be solved using straightforward Monte Carlo methods. However, neglecting default risks results in a considerable error in the pricing, as illustrated above. The deep BSDE method allows us to rigorously incorporate default risks into pricing models. This in turn makes it possible to evaluate financial derivatives with substantially lower risks for the involved parties and the societies.

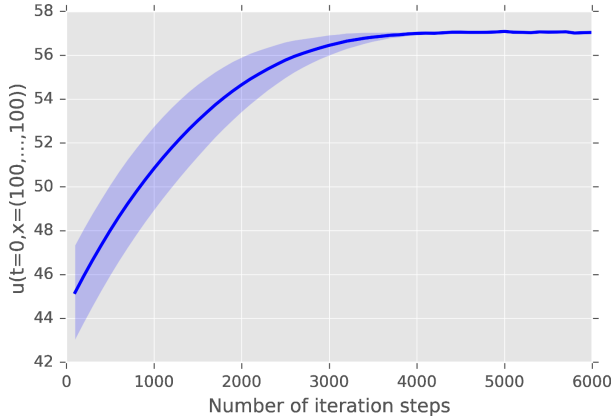

Figure 2: Plot of θ_{u_0} as an approximation of $u(t = 0, x = (100, \ldots, 100))$ against the number of iteration steps in the case of the 100-dimensional nonlinear Black–Scholes equation (14) with 40 equidistant time steps ($N = 40$) and learning rate 0.008. The shaded area depicts the mean $\pm$ the standard deviation of θ_{u_0} for 5 independent runs. The deep BSDE method achieves a relative error of size 0.46% in a runtime of 1607 seconds.

3.2 Hamilton–Jacobi–Bellman (HJB) Equation

The term "curse of dimensionality" was first used explicitly by Richard Bellman in the context of dynamic programming, which has now become the cornerstone in many areas such as economics, behavioral science, computer science, and even biology, where intelligent decision making is the main issue. In the context of game theory where there are multiple players, each player has to solve a high-dimensional HJB type equation in order to find his/her optimal strategy. In a dynamic resource allocation problem involving multiple entities with uncertainty, the dynamic programming principle also leads to a high-dimensional HJB equation for the value function. Until recently these high-dimensional PDEs have basically remained intractable. We now demonstrate below that the deep BSDE method is an effective tool for dealing with these high-dimensional problems.

We consider a classical linear–quadratic–Gaussian (LQG) control problem in 100 dimension:

$$\mathrm{d}X_t \;=\; 2\sqrt{\lambda}\, m_t\, \mathrm{d}t \;+\; \sqrt{2}\, \mathrm{d}W_t, \tag{15}$$

with $t \in [0, T]$, $X_0 = x$, and with the cost functional $J(\{m_t\}_{0 \le t \le T}) \;=\; E\Big[\int_0^T \|m_t\|^2 \, \mathrm{d}t + g(X_T)\Big]$.

Here $\{X_t\}_{t\in[0,T]}$ is the state process, $\{m_t\}_{t\in[0,T]}$ is the control process, λ is a positive constant representing the "strength" of the control, and $\{W_t\}_{t\in[0,T]}$ is a standard Brownian motion. Our goal is to minimize the cost functional through the control process. The HJB equation for this problem is given by

$$\frac{\partial u}{\partial t}(t,x) \;+\; \Delta u(t,x) \;-\; \lambda\,\|\nabla u(t,x)\|^2 \;=\; 0. \tag{16}$$

The value of the solution $u(t,x)$ of (16) at $t=0$ represents the optimal cost when the state starts from x. Applying Itô's formula, one can show that the exact solution of (16) with the terminal condition $u(T,x)=g(x)$ admits the explicit formula

$$u(t,x) \;=\; -\,\frac{1}{\lambda}\,\ln\Big(\,E\Big[\exp\big(-\lambda\,g\big(x+\sqrt{2}\,W_{T-t}\big)\big)\Big]\Big). \tag{17}$$

This can be used to test the accuracy of the proposed algorithm.

We solve the Hamilton–Jacobi–Bellman equation (16) in the 100-dimensional case with $g(x)=\ln\big((1+\|x\|^2)/2\big)$ for $x\in R^{100}$. Figure 3(a) shows the mean and the standard deviation of the relative error for $u(t=0,x=(0,\dots,0))$ in the case $\lambda=1$. The deep BSDE method achieves a relative error of 0.17% in a runtime of 330 seconds on a Macbook Pro. We also use the BSDE method to approximatively calculate the optimal cost $u(t=0,x=(0,\dots,0))$ against different values of λ; see Figure 3(b). The curve in Figure 3(b) clearly confirms the intuition that the optimal cost decreases as the control strength increases.

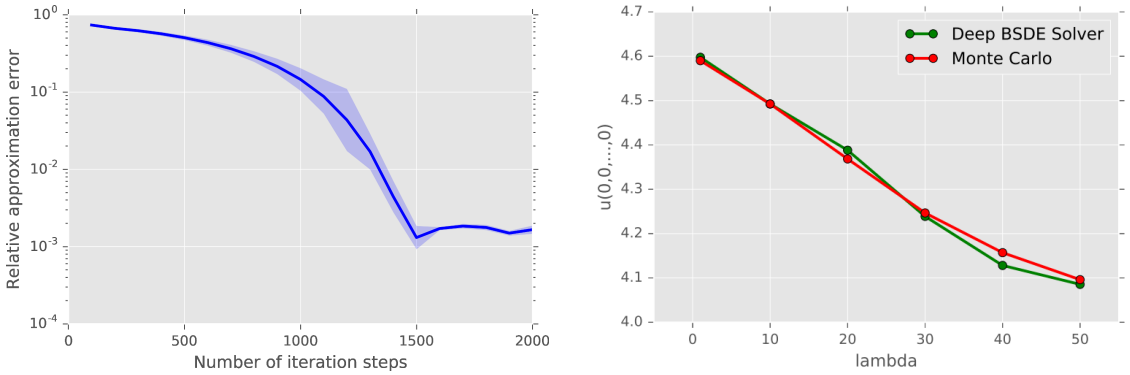

Figure 3: **Top:** Relative error of the deep BSDE method for $u(t=0,x=(0,\dots,0))$ when $\lambda=1$ against the number of iteration steps in the case of the 100-dimensional Hamilton–Jacobi–Bellman equation (16) with 20 equidistant time steps ($N=20$) and learning rate 0.01. The shaded area depicts the mean $\pm$ the standard deviation of the relative error for 5 different runs. The deep BSDE method achieves a relative error of size 0.17% in a runtime of 330 seconds. **Bottom:** Optimal cost $u(t=0,x=(0,\dots,0))$ against different values of λ in the case of the 100-dimensional Hamilton–Jacobi–Bellman equation (16), obtained by the deep BSDE method and classical Monte Carlo simulations of (17).

3.3 DNN Structure

Each sub-network is fully connected and consists of 4 layers, with 1 input layer (d-dimensional), 2 hidden layers (both (d+10)-dimensional), and 1 output layer (d-dimensional). We choose the rectifier function (ReLU) as our activation function. We also adopted the technique of batch normalization in the sub-networks, right after each linear transformation and before activation. This technique accelerates the training by allowing a larger step size and easier parameter initialization. All the parameters are initialized through a normal or a uniform distribution without any pre-training.

We use TensorFlow to implement our algorithm with the Adam optimizer to optimize parameters. Adam is a variant of the SGD algorithm, based on adaptive estimates of lower-order moments. We set the default values for corresponding hyper-parameters as recommended and choose the batch size as 64. In each of the presented numerical examples the means and the standard deviations of the relative L^1-approximation errors are computed approximatively by means of 5 independent runs of the algorithm with different random seeds.

The accuracy of the deep BSDE method certainly depends on the number of hidden layers in the sub-network approximation. To test this effect, we solve a reaction-diffusion type PDE with different numbers of hidden layers in the sub-network. The PDE is a high-dimensional version ($d = 100$) of the example analyzed numerically in Gobet & Turkedjiev ($d = 2$):

$$\frac{\partial u}{\partial t}(t,x) \;+\; \frac{1}{2}\,\Delta u(t,x) \;+\; \min\Big\{\,1,\;\big(u(t,x) \;-\; u^*(t,x)\big)^2\Big\} \;=\; 0, \tag{18}$$

in which $u^*(t,x)$ is the explicit oscillating solution

$$u^*(t,x) \;=\; \kappa \;+\; \sin\Big(\sum_{i=1}^{d} x_i\Big)\;\exp\Big(\frac{\lambda^2\,(d-T)}{2}\Big). \tag{19}$$

Parameters are chosen as follows : $\kappa = 1.6$, $\lambda = 0.1$, $T = 1$. A residual structure with skip connection is used in each sub-network with each hidden layer having d neurons. We increase the number of hidden layers in each sub-network from 0 to 4 and report the relative error in Table 1. It is evident that the approximation accuracy increases as the number of hidden layers in the sub-network increases.

number of layers[‡]	**29**	**58**	**87**	**116**	**145**
mean of relative error	2.29%	0.90%	0.60%	0.56%	0.53%
std. of relative error	0.0026	0.0016	0.0017	0.0017	0.0014

Table 1: The mean and standard deviation (std.) of the relative error for the PDE in (18), obtained by the deep BSDE method 1 with different number of hidden layers.

Method 1 : The PDE is solved until convergence with 30 equidistant time steps ($N = 30$) and 40000 iteration steps. Learning rate is 0.01 for the first half of iterations and 0.001 for the second half.
Method 2 : We only count the layers that have free parameters to be optimized.

4 Conclusion

In this paper, we introduced the Deep Backward Stochastic Differential Equation (DeepBSDE) method as a novel deep learning approach to solving high-dimensional nonlinear partial differential equations (PDEs). By leveraging deep neural networks, we effectively tackled the curse of dimensionality, which has traditionally hampered the computation of solutions for complex high-dimensional PDEs. Our approach, which reformulates PDEs as backward stochastic differential equations (BSDEs), offers a scalable and robust solution to problems in fields such as finance, economics, and control theory.

Through numerical experiments, including the nonlinear Black-Scholes equation with default risk and the Hamilton-Jacobi-Bellman (HJB) equation, we demonstrated that the DeepBSDE method can solve high-dimensional PDEs with remarkable accuracy and efficiency. Specifically, the method was able to achieve very low relative errors (e.g., 0.46% for the Black-Scholes equation and 0.17% for the HJB equation) while significantly reducing computational costs compared to traditional methods.

The success of the DeepBSDE framework opens up exciting possibilities for practical applications. In finance, for example, it enables more realistic modeling of complex derivative pricing models that account for multiple underlying assets and default risks. In operational research and dynamic optimization, it offers a promising tool for solving large-scale problems involving multiple agents or entities, such as resource allocation or multi-agent game theory scenarios. The methodology's ability to handle high-dimensional settings without relying on ad hoc assumptions or approximations marks a significant advancement in the computational methods used in these fields.

Despite the impressive results achieved, there remain challenges to be addressed. While the DeepBSDE method has proven effective for the class of nonlinear parabolic PDEs considered in this paper, its applicability to other types of PDEs, such as elliptic or hyperbolic equations, remains an area for future research. Moreover, the theoretical foundations of this method, particularly the convergence properties of neural network-based solvers for high-dimensional PDEs, require further exploration.

Looking forward, we anticipate that further refinements to the DeepBSDE method—such as the development of more sophisticated neural network architectures, optimization techniques, and hardware acceleration strategies—could enhance its performance and enable even broader applications in fields ranging from physics to artificial intelligence.

References

[1] Bellman, R. E., *Dynamic Programming*, Princeton University Press, 1957.

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

[3] LeCun, Y., Bengio, Y., and Hinton, G., Deep learning, *Nature*, vol. 521, pp. 436–444, 2015.

[4] Krizhevsky, A., Sutskever, I., and Hinton, G. E., Imagenet classification with deep convolutional neural networks, *Advances in Neural Information Processing Systems*, vol. 25, pp. 1097–1105, 2012.

[5] Hinton, G., et al., Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups, *IEEE Signal Processing Magazine*, vol. 29, pp. 82–97, 2012.

[6] Silver, D., Huang, A., Maddison, C. J., Guez, A., et al., Mastering the game of Go with deep neural networks and tree search, *Nature*, vol. 529, pp. 484–489, 2016.

[7] Pinkus, A., Approximation theory of the MLP model in neural networks, *Acta Numerica*, vol. 8, pp. 143–195, 1999.

[8] Pardoux, É., and Peng, S., Backward stochastic differential equations and quasilinear parabolic partial differential equations, *Lecture Notes in Control and Inform. Sci.*, vol. 176, pp. 200–217, Springer, 1992.

[9] Pardoux, É., and Tang, S., Forward-backward stochastic differential equations and quasilinear parabolic PDEs, *Probability Theory and Related Fields*, vol. 114, pp. 123–150, 1999.

[10] Darbon, J., and Osher, S., Algorithms for overcoming the curse of dimensionality for certain Hamilton–Jacobi equations arising in control theory and elsewhere, *Research in the Mathematical Sciences*, vol. 3, no. 19, 2016.

[11] E, W., Hutzenthaler, M., Jentzen, A., and Kruse, T., On multilevel Picard numerical approximations for high-dimensional nonlinear parabolic partial differential equations and high-dimensional nonlinear backward stochastic differential equations, *arXiv preprint arXiv:1607.03295*, 2016.

[12] Henry-Labordère, P., Counterparty risk valuation: A marked branching diffusion approach, *arXiv preprint arXiv:1203.2369*, 2012.

[13] Henry-Labordère, P., Tan, X., and Touzi, N., A numerical algorithm for a class of BSDEs via the branching process, *Stochastic Processes and their Applications*, vol. 124, pp. 1112–1140, 2014.