ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Seminar for Applied Mathematics SAM

Overcoming the curse of dimensionality: Solving high-dimensional partial differential equations using deep learning

J. Han and A. Jentzen and W. E

Research Report No. 2017-44
September 2017

Seminar für Angewandte Mathematik
Eidgenössische Technische Hochschule
CH-8092 Zürich
Switzerland

Overcoming the curse of dimensionality: Solving high-dimensional partial differential equations using deep learning

Jiequn Han[1], Arnulf Jentzen[2], and Weinan E[*4,3,1]

[1]Program in Applied and Computational Mathematics,
Princeton University, Princeton, NJ 08544, USA
[2]Department of Mathematics, ETH Zürich, Rämistrasse 101, 8092 Zürich, Switzerland
[3]Department of Mathematics, Princeton University, Princeton, NJ 08544, USA
[4]Center for Data Science and Beijing International Center for Mathematical Research,
Peking University and Beijing Institute of Big Data Research, Beijing, 100871, China

Abstract

Developing algorithms for solving high-dimensional partial differential equations (PDEs) has been an exceedingly difficult task for a long time, due to the notoriously difficult problem known as "the curse of dimensionality". This paper presents a deep learning-based approach that can handle general high-dimensional parabolic PDEs. To this end, the PDEs are reformulated as a control theory problem and the gradient of the unknown solution is approximated by neural networks, very much in the spirit of deep reinforcement learning with the gradient acting as the policy function. Numerical results on examples including the nonlinear Black-Scholes equation, the Hamilton-Jacobi-Bellman equation, and the Allen-Cahn equation suggest that the proposed algorithm is quite effective in high dimensions, in terms of both accuracy and speed. This opens up new possibilities in economics, finance, operational research, and physics, by considering all participating agents, assets, resources, or particles together at the same time, instead of making ad hoc assumptions on their inter-relationships.

1 Introduction

Partial differential equations (PDEs) are among the most ubiquitous tools used in modeling problems in nature. Some of the most important ones are naturally formulated as PDEs in high dimensions. Well-known examples include:

1. The Schrödinger equation in quantum many-body problem. In this case the dimensionality of the PDE is roughly three times the number of electrons or quantum particles in the system.

2. The nonlinear Black-Scholes equation for pricing financial derivatives, in which the dimensionality of the PDE is the number of underlying financial assets under consideration.

[*]weinan@math.princeton.edu

3. The Hamilton-Jacobi-Bellman equation in dynamic programming. In a game theory setting with multiple agents, the dimensionality goes up linearly with the number of agents. Similarly, in a resource allocation problem, the dimensionality goes up linearly with the number of devices and resources.

As elegant as these PDE models are, their practical use has proven to be very limited due to the curse of dimensionality [1]: The computational cost for solving them goes up exponentially with the dimensionality. Due to this reason, there are only a very limited number of cases where practical high-dimensional algorithms have been developed (cf. e.g., [2, 3, 4] and the references mentioned therein).

Another area where the curse of dimensionality has been an essential obstacle is machine learning and data analysis, where the complexity of nonlinear regression models, for example, goes up exponentially with the dimensionality. In both cases the essential problem we face is how to represent or approximate a nonlinear function in high dimensions. The traditional approach, by building functions using polynomials, piecewise polynomials, wavelets, or other basis functions, is bound to run into the curse of dimensionality problem.

In recent years a new class of techniques, the deep neural network model, has shown remarkable success (see, e.g., [5, 6, 7, 8, 9]). Neural network is an old idea but recent experience has shown that deep networks with many layers seem to do a surprisingly good job in modeling complicated data sets. In terms of representing functions, the neural network model is compositional: it uses compositions of simple functions to approximate complicated ones. In contrast, the approach of classical approximation theory is usually additive. Although we still lack a theoretical framework for understanding deep neural networks, their practical success has been very encouraging.

In this paper, we extend the power of deep neural networks to another dimension by developing a strategy for solving a large class of high-dimensional nonlinear PDEs using deep learning. The class of PDEs that we deal with are (nonlinear) parabolic PDEs. Special cases include the Black-Scholes equation and the Hamilton-Jacobi-Bellman equation. To do so, we make use of the reformulation of these PDEs as backward stochastic differential equations (BSDEs) (see, e.g., [10, 11]) and approximate the gradient of the solution using deep neural networks. The methodology bears some resemblence to deep reinforcement learning as the BSDE plays the role of model-based reinforcement learning (or control theory models) and the gradient of the solution plays the role of policy function. Numerical examples manifest that the proposed algorithm is quite satisfactory in both accuracy and computational cost.

To keep our demonstrations as accessible as possible, we neglect several mathematical and technical issues in below. See our supplementary materials for more details.

2 Methodology

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

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

with some specified terminal condition $u(T,x) = g(x)$. Here t and x represent the time and d-dimensional space variables 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 $\xi\in\mathbb{R}^d$.

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 (cf., e.g., [10, 11]):

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

We refer to the supplementary materials for further explanation of (3).

To derive a numerical algorithm to compute $u(0,X_0)$, we treat $u(0,X_0)\approx\theta_{u_0}$ and $\nabla u(0,X_0)\approx\theta_{\nabla u_0}$ as parameters in the model and view (3) as a way of computing the values of u at the terminal time T, knowing $u(0,X_0)$ and $\nabla u(0,X_0)$. We apply a temporal discretization to (2) and (3). Given a partition of the time interval $[0,T]$: $0=t_0<t_1<\ldots<t_N=T$, we consider the simple Euler scheme:

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

and

$$\begin{aligned} u(t_{n+1},X_{t_{n+1}}) \approx\ & u(t_n,X_{t_n}) - f\big(t_n,X_{t_n},u(t_n,X_{t_n}),\sigma^{\mathrm{T}}(t_n,X_{t_n})\,\nabla u(t_n,X_{t_n})\big)\,(t_{n+1}-t_n) \\ & + [\nabla u(t_n,X_{t_n})]^{\mathrm{T}}\,\sigma(t_n,X_{t_n})\,(W_{t_{n+1}}-W_{t_n}). \end{aligned} \tag{5}$$

Given this temporal discretization, the path $\{X_{t_n}\}_{0\leqslant n\leqslant N}$ can be easily sampled using (4). Our key step next is to approximate the function $x\mapsto\sigma^{\mathrm{T}}(t,x)\,\nabla u(t,x)$ at each time step $t=t_n$ by a multilayer feedforward neural network

$$\sigma^{\mathrm{T}}(t_n,X_{t_n})\,\nabla u(t_n,X_{t_n}) = (\sigma^{\mathrm{T}}\nabla u)(t_n,X_{t_n}) \approx (\sigma^{\mathrm{T}}\nabla u)(t_n,X_{t_n}|\theta_n), \qquad n=1,\ldots,N-1, \tag{6}$$

where θ_n denotes parameters of the neural network approximating $x\mapsto\sigma^{\mathrm{T}}(t,x)\,\nabla u(t,x)$ at $t=t_n$.

Thereafter, we stack all the subnetworks in (6) together to form a deep neural network as a whole, based on (5). Specifically, this network takes the path $\{X_{t_n}\}_{0\leqslant n\leqslant N}$ and $\{W_{t_n}\}_{0\leqslant n\leqslant N}$ as the input data and gives the final output, denoted by $\hat{u}(\{X_{t_n}\}_{0\leqslant n\leqslant N}, \{W_{t_n}\}_{0\leqslant n\leqslant N})$, as an approximation of $u(t_N, X_{t_N})$. We refer to the supplementary materials for more details on the architecture of the neural network. The error in the matching of given terminal condition can be used to define the expected loss function

$$l(\theta) = \mathbb{E}\Big[\big|g(X_{t_N}) - \hat{u}\big(\{X_{t_n}\}_{0\leqslant n\leqslant N}, \{W_{t_n}\}_{0\leqslant n\leqslant N}\big)\big|^2\Big]. \tag{7}$$

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 training of deep neural networks. In our numerical examples, we use the Adam optimizer [12]. We refer to the supplementary materials for more details on the training of the deep neural networks. Since the BSDE is used as an essential tool, we call the methodology introduced above *deep BSDE solver*.

3 Examples

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 [13]. The Black-Scholes model can be augmented to take into several important factors in real markets, including defaultable securities, higher interest rates for borrowing than for lending, transactions costs, uncertainties in the model parameters, etc. (see, e.g., [14, 15, 16, 17, 18]). Each of these effects results in a nonlinear contribution in the pricing model (see, e.g., [15, 19, 20]). In particular, the credit crisis and the ongoing European sovereign debt crisis have hightlighted the most basic risk that has been neglected in the original Black-Scholes model, the default risk [19].

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 solver, we study a special case of the recursive valuation model with default risk [14, 15]. We consider the fair price of an 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

$$f\big(t, x, u(t,x), \sigma^{\mathrm{T}}(t,x)\nabla u(t,x)\big) = -(1-\delta)\, Q(u(t,x))\, u(t,x) - R\, u(t,x) \tag{8}$$

(see [14]), where R is the interest rate of the riskless 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 ($v^h < v^l$, $\gamma^h > \gamma^l$):

$$Q(y) = \mathbb{1}_{(-\infty, v^h)}(y)\,\gamma^h + \mathbb{1}_{[v^l, \infty)}(y)\,\gamma^l + \mathbb{1}_{[v^h, v^l)}(y)\left[\tfrac{(\gamma^h - \gamma^l)}{(v^h - v^l)}\left(y - v^h\right) + \gamma^h\right] \tag{9}$$

(see [15]). The associated nonlinear Black-Scholes equation in $[0, T] \times \mathbb{R}^{100}$ becomes

$$\begin{aligned}\frac{\partial u}{\partial t}(t,x) + \bar{\mu} x \cdot \nabla u(t,x) + \frac{\bar{\sigma}^2}{2} \sum_{i=1}^{d} |x_i|^2 \frac{\partial^2 u}{\partial x_i^2}(t,x) \\ - (1 - \delta + R) \min\left\{\gamma^h, \max\left\{\gamma^l, \frac{(\gamma^h - \gamma^l)}{(v^h - v^l)}\left(u(t,x) - v^h\right) + \gamma^h\right\}\right\} u(t,x) = 0.\end{aligned} \tag{10}$$

We choose $T = 1, \delta = 2/3$, $R = 0.02$, $\bar{\mu} = 0.02$, $\bar{\sigma} = 0.2$, $v^h = 50$, $v^l = 70$, $\gamma^h = 0.2$, $\gamma^l = 0.02$ and terminal condition $g(x) = \min\{x_1, \ldots, x_{100}\}$ for $x = (x_1, \ldots, x_{100}) \in \mathbb{R}^{100}$. Figure 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 know "exact" solution of (10) at $t = 0, x = (100, \ldots, 100)$ has been approximately computed by means of the multilevel Picard method [4]: $u(t{=}0, x{=}(100, \ldots, 100)) \approx 57.300$. In comparison, if we do not consider the default risk, we get $\tilde{u}(t{=}0, x{=}(100, \ldots, 100)) \approx 60.781$. 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 solver allows us to rigorously incorporate default risks into pricing models. This in turn makes it possible to evaluate financial derivatives with substantial lower risks for the involved parties and the societies.

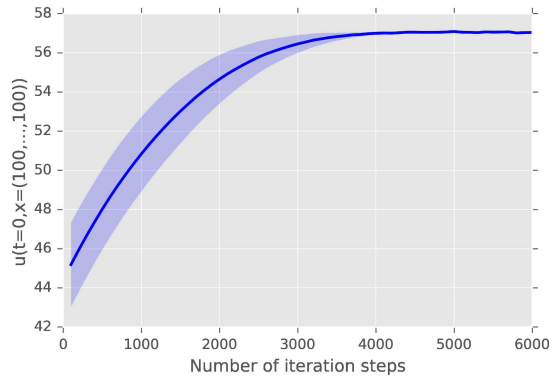


Figure 1: Plot of θ_{u_0} as an approximation of $u(t{=}0, x{=}(100, \ldots, 100))$ against number of iteration steps in the case of the 100-dimensional nonlinear Black-Scholes equation (10) with 40 equidistant time steps ($N = 40$) and learning rate 0.008. The shaded area depicts the mean $\pm$ the standard deviation of θ_{u_0} as an approximation of $u(t{=}0, x{=}(100, \ldots, 100))$ for 5 independent runs. The deep BSDE solver achieves a relative error of size 0.46% in a runtime of 617 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 [1], which has now become the cornerstone in many areas such as economics, behaviorial 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 (and high degrees of uncertainty), the dynamic programming principle also leads to a high-dimensional HJB equation [21] for the value function.

Until recently these high-dimensional PDEs have basically remained intractable. We now demonstrate below that the deep BSDE solver is an effective tool for dealing with these high-dimensional problems. Note that Darbon & Osher have recently developed an algorithm for a class of inviscid Hamilton-Jacobi equations, which performs numerically well in the case of high dimensions, based on results from compressed sensing and on the Hopf formulas for the Hamilton-Jacobi equations (see [3]).

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

$$dX_t = 2\sqrt{\lambda}\, m_t\, dt + \sqrt{2}\, dW_t \tag{11}$$

with $t \in [0,T]$ and $X_0 = x$ and with the cost functional $J(\{m_t\}_{0\leqslant t\leqslant T}) = \mathbb{E}\big[\int_0^T (m_t)^2\, dt + g(X_T)\big]$. 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

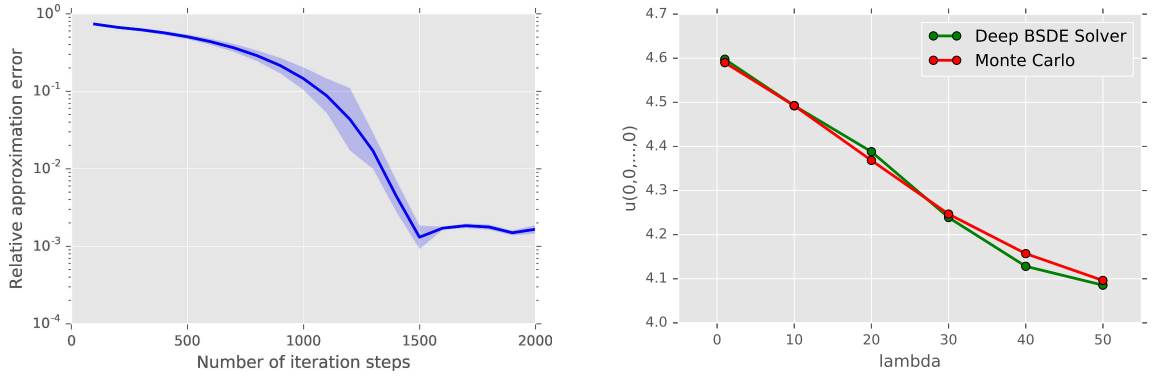
$$\frac{\partial u}{\partial t}(t,x) + \Delta u(t,x) - \lambda \|\nabla u(t,x)\|^2 = 0 \tag{12}$$

(cf., e.g., Yong & Zhou [22, Chapter 3]). The value of the solution $u(t,x)$ of (12) 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 (12) with the terminal condition $u(T,x) = g(x)$ admits the explicit formula

$$u(t,x) = -\frac{1}{\lambda} \ln\Big(\mathbb{E}\Big[\exp\Big(-\lambda g(x + \sqrt{2}W_{T-t})\Big)\Big]\Big). \tag{13}$$

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

We solve the PDE (12) in the 100-dimensional case with $g(x) = 2/(1+\|x\|^2)$ for $x \in \mathbb{R}^{100}$. Figure 2 (a) shows the mean and the standard deviation of the relative error for $u(t{=}0, x{=}(0,\dots,0))$ in the case where $\lambda = 1$: the deep BSDE solver achieves a relative error of 0.17% in a runtime of 330 seconds on a Macbook Pro. We also use the BSDE solver to approximatively calculate the optimal cost $u(t{=}0, x{=}(0,\dots,0))$ against different values of λ; see Figure 2 (b). The curve in Figure 2 (b) clearly confirms the intuition that the optimal cost decreases as the control strength increases.



(a) Relative error when $\lambda = 1$

(b) Optimal cost against different λ

Figure 2: (a) Relative error of the deep BSDE solver for $u(t{=}0, x{=}(0, \dots, 0))$ when $\lambda = 1$ against number of iteration steps in the case of the 100-dimensional Hamilton-Jacobi-Bellmann equation (12) 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 solver achieves a relative error of size 0.17% in a runtime of 330 seconds. (b) Optimal cost $u(t{=}0, x{=}(0, \dots, 0))$ against different values of λ in the case of the 100-dimensional Hamilton-Jacobi-Bellmann equation (12) obtained by the deep BSDE solver and classical Monte Carlo simulations for (13).

3.3 Allen-Cahn equation

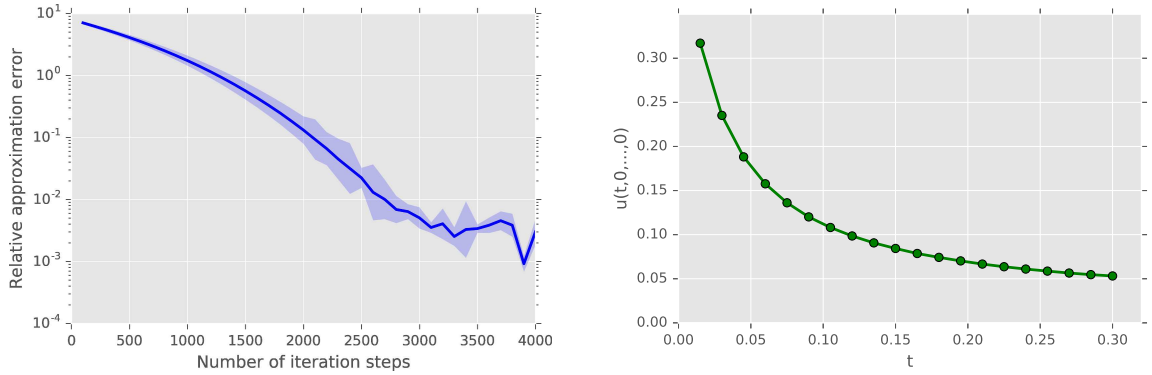
The Allen-Cahn equation is a reaction-diffusion equation that arises in physics, serving as a prototype for the modeling of phase separation and order-disorder transition (see, e.g., [23]). Here we consider a typical Allen-Cahn equation with the "double-well potential" in 100-dimensional space

$$\frac{\partial u}{\partial t}(t,x) = \Delta u(t,x) + u(t,x) - [u(t,x)]^3\,, \tag{14}$$

with the initial condition $u(0,x) = g(x)$, where $g(x) = 1/\left(2 + 0.4\,\|x\|^2\right)$ for $x \in \mathbb{R}^{100}$. By applying a transformation of the time variable $t \mapsto T - t$ ($T > 0$), we can turn (14) into the form of (1) such that the deep BSDE solver can be used. Figure 3 (a) shows the mean and the standard deviation of the relative error of $u(t{=}0.3, x{=}(0, \dots, 0))$. The not explicitly known "exact" solution of (14) at $t = 0.3$, $x = (0, \dots, 0)$ has been approximatively computed by means of the branching diffusion method (see, e.g., [2]): $u(t{=}0.3, x{=}(100, \dots, 100)) \approx 0.0528$. For this 100-dimensional example PDE, the deep BSDE solver achieves a relative error of 0.30% in a runtime of 647 seconds on a Macbook Pro. We also use the deep BSDE solver to approximatively compute the time evolution of $u(t, x{=}(0, \dots, 0))$ for $t \in [0, T]$; see Figure 3 (b).

4 Conclusions

The algorithm proposed in this paper opens up a host of new possibilities in several different areas. For example in economics one can consider many different interacting agents at the same time, instead of using the "representative agent" model. Similarly in finance, one can consider



(a) Relative error

(b) Time evolution of $u(t, x{=}(0, \ldots, 0))$

Figure 3: (a) Relative error of the deep BSDE solver for $u(t{=}0.3, x{=}(0, \ldots, 0))$ against the number of iteration steps in the case of the 100-dimensional Allen-Cahn equation (14) with 20 equidistant time steps ($N = 20$) and learning rate 0.0005. The shaded area depicts the mean $\pm$ the standard deviation of the relative error for 5 different runs. The deep BSDE solver achieves a relative error of size 0.30% in a runtime of 647 seconds. (b) Time evolution of $u(t, x{=}(0, \ldots, 0))$ for $t \in [0, 0.3]$ in the case of the 100-dimensional Allen-Cahn equation (14) computed by means of the deep BSDE solver.

all the participating instruments at the same time, instead of relying on ad hoc assumptions about their relationships. In operational research, one can handle the cases with hundreds and thousands of participating entities directly, without the need to make ad hoc approximations.

It should be noted that although the methodology presented here is fairly general, we are so far not able to deal with the quantum many-body problem due to the difficulty in dealing with the Pauli exclusion principle.

Acknowledgement

The work of Han and E is supported in part by Major Program of NNSFC under grant 91130005, DOE grant DE-SC0009248 and ONR grant N00014-13-1-0338.

References

[1] Richard Ernest Bellman. *Dynamic Programming*. Rand Corporation research study. Princeton University Press, 1957.

[2] Pierre Henry-Labordère, Xiaolu Tan, and Nizar Touzi. A numerical algorithm for a class of BSDEs via the branching process. *Stochastic Processes and their Applications*, 124(2):1112–1140, 2014.

[3] Jérôme Darbon and Stanley Osher. Algorithms for overcoming the curse of dimensionality for certain Hamilton–Jacobi equations arising in control theory and elsewhere. *Research in the Mathematical Sciences*, 3(1):19, 2016.

[4] Weinan E, Martin Hutzenthaler, Arnulf Jentzen, and Thomas Kruse. On multilevel Picard numerical approximations for high-dimensional nonlinear parabolic partial differential equations and high-dimensional nonlinear backward stochastic differential equations. *arXiv:1607.03295*, 46 pages, 2016.

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[6] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[8] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[9] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[10] Étienne Pardoux and Shige Peng. Backward stochastic differential equations and quasilinear parabolic partial differential equations. In *Stochastic partial differential equations and their applications (Charlotte, NC, 1991)*, volume 176 of *Lecture Notes in Control and Inform. Sci.*, pages 200–217. Springer, Berlin, 1992.

[11] Étienne Pardoux and Shanjian Tang. Forward-backward stochastic differential equations and quasilinear parabolic PDEs. *Probability Theory and Related Fields*, 114(2):123–150, 1999.

[12] Diederik Kingma and Jimmy Ba. Adam: a method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

[13] Fischer Black and Myron Scholes. The pricing of options and corporate liabilities [reprint of J. Polit. Econ. 81 (1973), no. 3, 637–654]. In *Financial risk measurement and management*, volume 267 of *Internat. Lib. Crit. Writ. Econ.*, pages 100–117. Edward Elgar, Cheltenham, 2012.

[14] Darrell Duffie, Mark Schroder, and Costis Skiadas. Recursive valuation of defaultable securities and the timing of resolution of uncertainty. *The Annals of Applied Probability*, 6(4):1075–1090, 1996.

[15] Christian Bender, Nikolaus Schweizer, and Jia Zhuo. A primal-dual algorithm for BSDEs. *Mathematical Finance*, 2015.

[16] Yaacov Z. Bergman. Option pricing with differential interest rates. *The Review of Financial Studies*, 8(2):475–500, 1995.

[17] Hayne Leland. Option pricing and replication with transaction costs. *The Journal of Finance*, 40(5):1283–1301, 1985.

[18] Marco Avellaneda, Arnon Levy, and Antonio Parás. Pricing and hedging derivative securities in markets with uncertain volatilities. *Applied Mathematical Finance*, 2(2):73–88, 1995.

[19] Stéphane Crépey, , Rémi Gerboud, Zorana Grbac, and Nathalie Ngor. Counterparty risk and funding: the four wings of the TVA. *International Journal of Theoretical and Applied Finance*, 16(02):1350006, 2013.

[20] Peter A. Forsyth and Ken R. Vetzal. Implicit solution of uncertain volatility/transaction cost option pricing models with discretely observed barriers. *Applied Numerical Mathematics*, 36(4):427–445, 2001.

[21] Warren B. Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. John Wiley & Sons, 2011.

[22] Jiongmin Yong and Xun Yu Zhou. *Stochastic Controls*. Springer New York, 1999.

[23] Heike Emmerich. *The diffuse interface approach in materials science: thermodynamic concepts and applications of phase-field models*, volume 73. Springer Science & Business Media, 2003.

A Supplementary Materials

A.1 BSDE reformulation

The link between (nonlinear) parabolic PDEs and backward stochastic differential equations (BSDEs) has been extensively investigated in the literature (see, e.g., [1, 2, 3]). In particular, Markovian BSDEs give a Feynman-Kac representation of some nonlinear parabolic PDEs. Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space, $W\colon [0,T] \times \Omega \to \mathbb{R}^d$ 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

$$\begin{cases} X_t = \xi + \displaystyle\int_0^t \mu(s, X_s)\, ds + \int_0^t \sigma(s, X_s)\, dW_s, & (15) \\ Y_t = g(X_T) + \displaystyle\int_t^T f(s, X_s, Y_s, Z_s)\, ds - \int_t^T (Z_s)^{\mathrm{T}}\, dW_s, & (16) \end{cases}$$

for which we are seeking for a $\{\mathcal{F}_t\}_{t\in[0,T]}$-adapted solution process $\{(X_t, Y_t, Z_t)\}_{t\in[0,T]}$ with values in $\mathbb{R}^d \times \mathbb{R} \times \mathbb{R}^d$. Under suitable regularity assumptions on the coefficient functions μ, σ, and f one can prove existence and up-to-indistinguishability uniqueness of solutions (cf., e.g., [1, 3]). Furthermore, we have that the nonlinear parabolic PDE (1) is related to the BSDE (15)–(16) in the sense that for all $t \in [0,T]$ it holds $\mathbb{P}$-a.s. that

$$Y_t = u(t, X_t) \qquad \text{and} \qquad Z_t = \sigma^{\mathrm{T}}(t, X_t)\, \nabla u(t, X_t), \tag{17}$$

(cf., e.g., [1, 2]). Therefore, we can compute the quantity $u(0, X_0)$ associated to the PDE (1) through Y_0 by solving the BSDE (15)–(16). More specifically, we plug the identities in (17) into (16) and rewrite the equation forwardly to obtain the formula (3). Then we discretize the equation temporally and use neural networks to approximate the spatial gradients and finally the unknown function, as introduced in Section 2 of the paper.

A.2 Neural network architecture

In this subsection we briefly illustrate the architecture of the deep BSDE solver. To simply the presentation we restrict ourself in these illustrations to the case where the diffusion coefficient σ in (1) satisfies $\forall\, x \in \mathbb{R}^d\colon \sigma(x) = \mathrm{Id}_{\mathbb{R}^d}$. Figure 4 illustrates the network architecture for the deep BSDE solver. Note that $\nabla u(t_n, X_{t_n})$ denotes the variable we approximate directly by subnetworks and $u(t_n, X_{t_n})$ denotes the variable we compute iteratively in the network. There are three types of connections in this network:

(i) $X_{t_n} \to h_n^1 \to h_n^2 \to \cdots \to h_n^N \to \nabla u(t_n, X_{t_n})$ is the multilayer feedforward neural network approximating the spatial gradients at time $t = t_n$. The weights θ_n of this subnetwork are the parameters we aim to optimize.

(ii) $(u(t_n, X_{t_n}), \nabla u(t_n, X_{t_n}), W_{t_{n+1}} - W_{t_n}) \to u(t_{n+1}, X_{t_{n+1}})$ is the forward iteration giving the final output of the network as an approximation of $u(t_N, X_{t_N})$, completely characterized by (5). There are no parameters to be optimized in this type of connection.

(iii) $(X_{t_n}, W_{t_{n+1}} - W_{t_n}) \to X_{t_{n+1}}$ is the shortcut connecting blocks at different time, which is characterized by (4). There are also no parameters to be optimized in this type of connection.

If we use H hidden layers in each subnetwork, as illustrated in Figure 4, then the whole network has $(H + 2)(N - 1)$ layers in total.

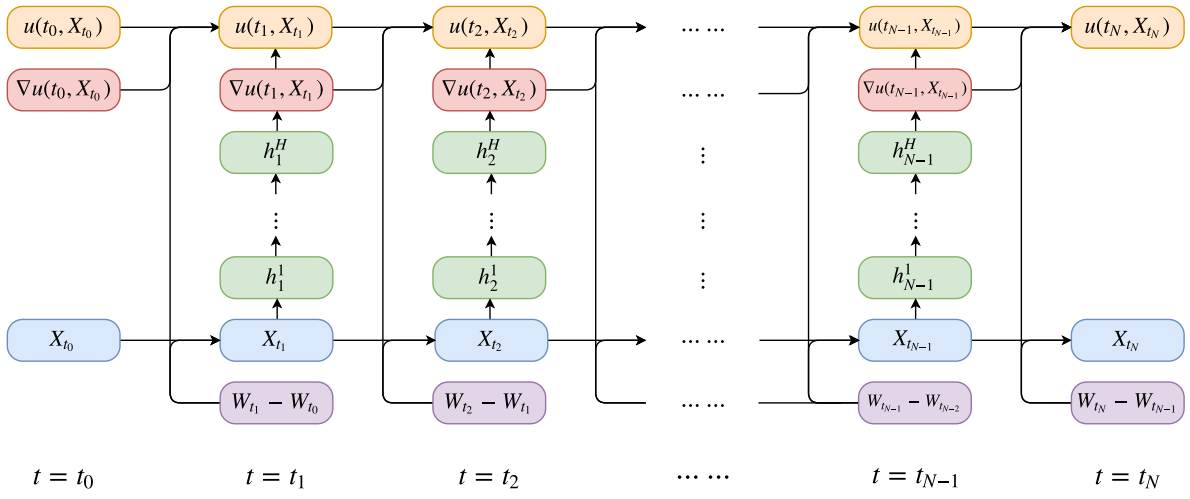


Figure 4: Illustration of the network architecture for solving semilinear parabolic PDEs with H hidden layers for each subnetwork and N time intervals. Each column for $t = t_1, t_2, \ldots, t_{N-1}$ corresponds to a subnetwork at time t. $h_n^1, \ldots, h_n^H$ are the hidden variables in the subnetwork at time $t = t_n$ for $n = 1, 2, \ldots, N - 1$.

Next we like to point out that the proposed deep BSDE solver can also be employed if we are interested in values of the PDE solution u in a region $D \subset \mathbb{R}^d$ at time $t = 0$ instead of at a single space-point $\xi \in \mathbb{R}^d$. In this case we choose $X_0 = \xi$ to be a non-degenerate D-valued random variable and we employ two additional neural networks parameterized by $\{\theta_{u_0}, \theta_{\nabla u_0}\}$ for approximating the functions $D \ni x \mapsto u(0, x) \in \mathbb{R}$ and $D \ni x \mapsto \nabla u(0, x) \in \mathbb{R}^d$.

A.3 Implementation

We briefly mention some details of the implementation for the numerical examples presented in the paper. Each subnetwork 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 for the hidden variables. We also adopted the technique of batch normalization [4] in the subnetworks, right after each linear transformation and before activation. This method 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 [5] to implement our algorithm with the Adam optimizer [6] to optimize parameters. Adam is an variant of the SGD algorithm, based on adaptive estimates of lower-order moments. We set the default values for corresponding hyper-parameters as recommended in [6] and choose the batch size as 64. In each of the numerical examples above 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. All the numerical examples reported are run on a Macbook Pro with a 2.9GHz Intel Core i5 processor and 16 GB memory.

Supplementary References

[1] Étienne Pardoux and Shige Peng. Backward stochastic differential equations and quasilinear parabolic partial differential equations. In *Stochastic partial differential equations and their applications (Charlotte, NC, 1991)*, volume 176 of *Lecture Notes in Control and Inform. Sci.*, pages 200–217. Springer, Berlin, 1992.

[2] Étienne Pardoux and Shanjian Tang. Forward-backward stochastic differential equations and quasilinear parabolic PDEs. *Probab. Theory Related Fields*, 114(2):123–150, 1999.

[3] Nicole El Karoui, Shige Peng, and Marie-Claire Quenez. Backward stochastic differential equations in finance. *Mathematical Finance*, 7(1):1–71, 1997.

[4] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, pages 448–456, 2015.

[5] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, et al. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation*, pages 265–283, 2016.

[6] Diederik Kingma and Jimmy Ba. Adam: a method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.