Solving High-dimensional PDEs Using Deep Learning

Jiequn Han

The Program in Applied & Computational Mathematics,
Princeton University
Joint work with Weinan E and Arnulf Jentzen

Inverse Problems and Machine Learning,
Caltech, February 9, 2018

Outline

1. Introduction
2. Mathematical Formulation
3. Neural Network Approximation
4. Numerical Examples
5. Summary

Table of Contents

Well-known Examples of PDEs

- The Schrödinger equation in quantum many-body problem,

$$i\hbar\frac{\partial}{\partial t}\Psi(t,x) = (-\frac{1}{2}\Delta + V)\Psi(t,x).$$

- The Black-Scholes equation for pricing financial derivatives,

$$v_t + \frac{1}{2}\operatorname{Tr}\Big(\sigma\sigma^{\mathrm{T}}(\mathsf{Hess}_x v)\Big) + r\nabla v\cdot x - rv = 0.$$

- The Hamilton-Jacobi-Bellman equation in stochastic control (dynamic programming),

$$v_t + \max_u\left\{\frac{1}{2}\operatorname{Tr}\Big(\sigma\sigma^{\mathrm{T}}(\mathsf{Hess}_x v)\Big) + \nabla v\cdot b + f\right\} = 0.$$

Curse of Dimensionality

- The dimension of PDEs can be easily large in practice.

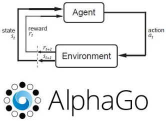
Equation	Dimension (roughly)
Schrödinger equation	$\#$ of electrons $\times 3$
Black-Scholes equation	$\#$ of underlying financial assets
HJB equation	the same as the state space

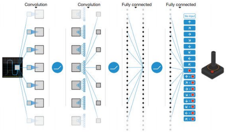
- A key computational challenge is the curse of dimensionality: the complexity is exponential in dimension d for finite difference/element method – usually unavailable for $d \geq 4$.

- There is a huge gap between PDE modelings and computational algorithms.

Remarkable Success of Deep Learning

- Machine learning/data analysis also face the same curse of dimensionality
- In recent years, deep learning has achieved remarkable success
- An old but essential idea: represent functions in a compositional form rather than additive





Related Work in High-dimensional Case

- Linear parabolic PDEs: Monte Carlo methods based on the Feynman-Kac formula
- Semilinear parabolic PDEs:
  1. branching diffusion approach (Henry-Labordère 2012, Henry-Labordère et al. 2014)
  2. multilevel Picard approximation (E et al. 2016)
- Hamilton-Jacobi PDEs: using Hopf formula and fast convex/nonconvex optimization methods (Darbon & Osher 2016, Chow et al. 2017)

Table of Contents

Semilinear Parabolic PDE

We consider a general semilinear parabolic PDE in $[0, T] \times \mathbb{R}^d$:

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

- Terminal condition is given: $u(T, x) = g(x)$.
- To fix ideas, we are interested in the solution at $t = 0$, $x = \xi$ for some vector $\xi \in \mathbb{R}^d$.

Connection between PDE and BSDE

- The link between parabolic PDEs and backward stochastic differential equations (BSDEs) has been extensively investigated (Pardoux & Peng 1992, El Karoui et al. 1997, etc).
- In particular, Markovian BSDEs give a nonlinear Feynman-Kac representation of some nonlinear parabolic PDEs.
- Consider the following BSDE

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

The solution is an adapted process $\{(X_t, Y_t, Z_t)\}_{t\in[0,T]}$ with values in $\mathbb{R}^d \times \mathbb{R} \times \mathbb{R}^d$.

Connection between PDE and BSDE

- Under suitable regularity assumptions, the BSDE is well-posed and related to the PDE in the sense that for all $t \in [0, T]$ it holds a.s. that

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

- In other words, given the stochastic process satisfying

$$X_t = \xi + \int_0^t \mu(s, X_s)\, ds + \int_0^t \sigma(s, X_s)\, dW_s,$$

the solution of PDE satisfies the following SDE

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

BSDE and Control – A LQG Example

Consider a classical linear-quadratic-Gaussian (LQG) control problem in $\mathbb{R}^d$:

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

with cost functional $J(\{m_t\}_{0\le t\le T}) = \mathbb{E}\big[\int_0^T \|m_t\|_2^2\, dt + g(X_T)\big]$. The HJB equation for this problem is

$$\frac{\partial u}{\partial t}(t,x) + \Delta u(t,x) - \lambda \|\nabla u(t,x)\|_2^2 = 0.$$

The optimal control is given by

$$m_t^* = \frac{\nabla u(t,x)}{\sqrt{2\lambda}}, \qquad \text{(recall } Z_t = \sigma^{\mathrm{T}}(t,X_t)\nabla u(t,X_t)\text{)}.$$

In the context of BSDE for control, Y_t denotes the optimal value and Z_t denotes the optimal control (up to a constant scaling).

Table of Contents

Neural Network Approximation

- Key step: approximate the function $x \mapsto \sigma^{\mathrm{T}}(t,x)\,\nabla u(t,x)$ at each discretized time step $t = t_n$ by a feedforward neural network

$$\begin{aligned}\sigma^{\mathrm{T}}(t_n, X_{t_n})\,\nabla u(t_n, X_{t_n}) &= (\sigma^{\mathrm{T}}\nabla u)(t_n, X_{t_n}) \\ &\approx (\sigma^{\mathrm{T}}\nabla u)(t_n, X_{t_n}|\theta_n),\end{aligned}$$

  where θ_n denotes neural network parameters.

- Observation: we can stack all the subnetworks together to form a deep neural network (DNN) as a whole, based on the time discretization (see the next two slides).

Time Discretization

We consider the simple Euler scheme of the BSDE, with a partition of the time interval $[0, T]$, $0 = t_0 < t_1 < \ldots < t_N = T$:

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

and

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

where

$$\Delta t_n = t_{n+1} - t_n, \quad \Delta W_n = W_{t_{n+1}} - W_{t_n}.$$

Network Architecture

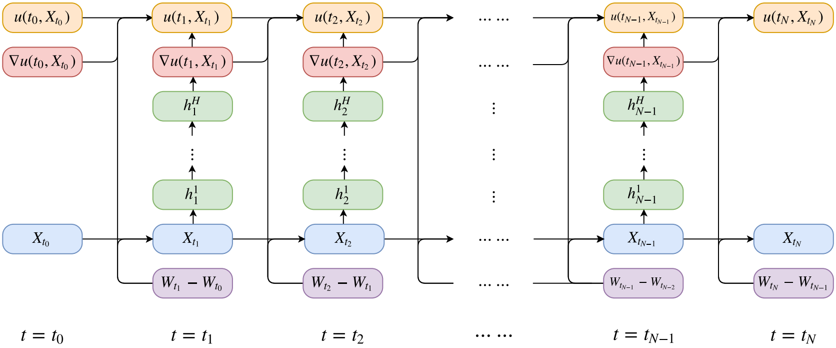


Figure: Network architecture for solving parabolic PDEs. Each column corresponds to a subnetwork at time $t = t_n$. The whole network has $(H + 2)(N - 1)$ layers in total.

Optimization

- 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 $\hat{u}(\{X_{t_n}\}_{0\le n\le N}, \{W_{t_n}\}_{0\le n\le N})$, as an approximation to $u(t_N, X_{t_N})$.

- The error in the matching of given terminal condition defines the expected loss function

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

- The paths can be simulated easily. Therefore the commonly used SGD algorithm fits this problem well.

- We call the introduced methodology deep BSDE method since we use the BSDE and DNN as essential tools.

Time Discretization as Skip Connection

Why such deep networks can be trained?

Intuition: there are skip connections between different subnetworks

$$\begin{aligned}
&u(t_{n+1}, X_{t_{n+1}}) - u(t_n, X_{t_n}) \\
\approx &- f\big(t_n, X_{t_n}, u(t_n, X_{t_n}), (\sigma^{\mathrm{T}}\nabla u)(t_n, X_{t_n}|\theta_n)\big)\, \Delta t_n \\
&+ (\sigma^{\mathrm{T}}\nabla u)(t_n, X_{t_n}|\theta_n)\, \Delta W_n,
\end{aligned}$$

Analogy to Deep Reinforcement Learning

- Deep Reinforcement Learning (DRL) has achieved great success in game domains and sophisticated control tasks. A common strategy is to represent policy function (control) through neural networks.
- Recall that in the example of LQG control problem, Z_t denotes the optimal control, which is approximated by neural networks.

Table: Informal analogy

Deep BSDE method		DRL
BSDE	$\longleftrightarrow$	Markov decision model
gradient of the solution	$\longleftrightarrow$	optimal policy function

Table of Contents

Implementation

- Each subnetwork has 4 layers, with 1 input layer (d-dimensional), 2 hidden layers (both $d+10$-dimensional), and 1 output layer (d-dimensional).
- Choose the rectifier function (ReLU) as the activation function and optimize with Adam method.
- Implement in Tensorflow and reported examples are all run on a Macbook Pro.
- Github: https://github.com/frankhan91/DeepBSDE

LQG Example Revisited

We solve the introduced HJB equation in $[0,1]\times\mathbb{R}^{100}$. It admits an explicit formula, which allows accuracy test:

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

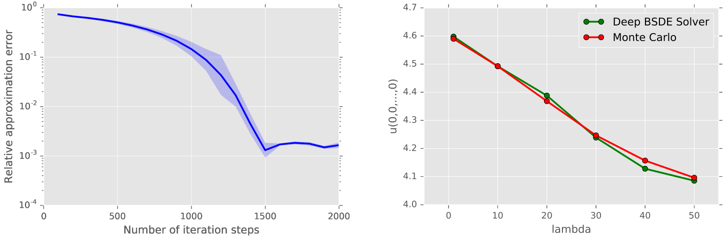


Figure: Left: Relative error of the deep BSDE method for $u(t{=}0, x{=}(0,\ldots,0))$ when $\lambda = 1$, which achieves 0.17% in a runtime of 330 seconds. Right: Optimal cost $u(t{=}0, x{=}(0,\ldots,0))$ against different λ.

Black-Scholes Equation with Default Risk

- The classical Black-Scholes model can and should be augmented by some important factors in real markets, including defaultable securities, transactions costs, uncertainties in the model parameters, etc.
- Ideally the pricing models should take into account the whole basket of financial derivative underlyings, resulting in high-dimensional nonlinear PDEs.
- To test the deep BSDE method, we study a special case of the recursive valuation model with default risk (Duffie et al. 1996, Bender et al. 2015).

Black-Scholes Equation with Default Risk

- Consider the fair price of a European claim based on 100 underlying assets conditional on no default having occurred yet.
- The underlying asset price moves as a geometric Brownian motion and the possible default is modeled by the first jump time of a Poisson process.
- The claim value is modeled by a parabolic PDE with the nonlinear function

$$\begin{aligned} & f\left(t, x, u(t,x), \sigma^{\mathrm{T}}(t,x) \nabla u(t,x)\right) \\ &= -\,(1-\delta)\, Q(u(t,x))\, u(t,x) - R\, u(t,x). \end{aligned}$$

Black-Scholes Equation with Default Risk

The not explicitly known "exact" solution at $t = 0$ $x = (100, \ldots, 100)$ is computed by the multilevel Picard method.

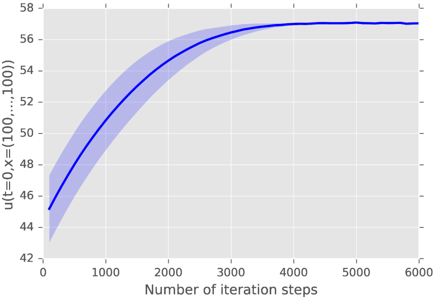


Figure: Approximation of $u(t{=}0, x{=}(100, \ldots, 100))$ against number of iteration steps. The deep BSDE method achieves a relative error of size 0.46% in a runtime of 617 seconds.

Allen-Cahn Equation

The Allen-Cahn equation is a reaction-diffusion equation for the modeling of phase separation and transition in physics. Here we consider a typical Allen-Cahn equation with the "double-well potential" in 100-dimensional space:

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

with initial condition $u(0,x) = g(x)$.

Allen-Cahn Equation

The not explicitly known “exact” solution at $t = 0.3$, $x = (0, \dots, 0)$ is computed by the branching diffusion method.

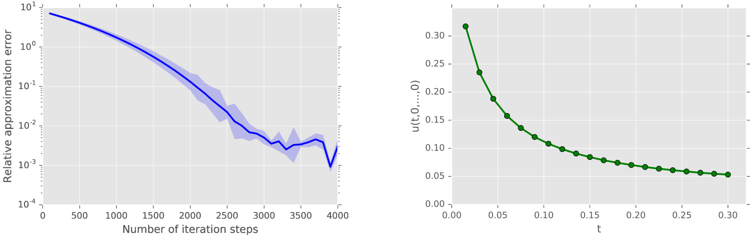


Figure: Left: relative error of the deep BSDE method for $u(t{=}0.3, x{=}(0, \dots, 0))$, which achieves 0.30% in a runtime of 647 seconds. Right: time evolution of $u(t, x{=}(0, \dots, 0))$ for $t \in [0, 0.3]$, computed by means of the deep BSDE method.

An Example with Quadratically Growing Derivatives

We consider an example studied for the numerical methods of PDE in literature (Gobet & Turkedjiev 2016).
The PDE is constructed artificially in a form

$$\begin{aligned}\frac{\partial u}{\partial t}(t,x) + \|(\nabla_x u)(t,x)\|_2^2 + \frac{1}{2}\,(\Delta_x u)(t,x) \\ = \frac{\partial \psi}{\partial t}(t,x) + \|(\nabla_x \psi)(t,x)\|_2^2 + \frac{1}{2}\,(\Delta_x \psi)(t,x),\end{aligned}$$

with the explicit solution

$$\psi(t,x) = \sin\Big(\big[T - t + \|x\|_2^2/d\,\big]^{0.4}\Big)\,.$$

An Example with Quadratically Growing Derivatives

Compared to the literature, we set $d = 100$ instead of $d \in \{3, 5, 7\}$ and $T = 1$ instead $T = 0.2$.

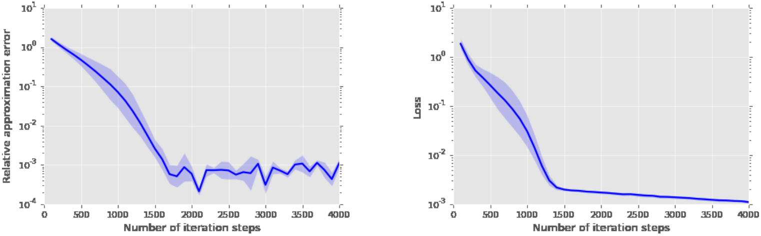


Figure: Left: relative error of the deep BSDE method for $u(t{=}0, x{=}(0, \ldots, 0))$, which achieves 0.09% in a runtime of 957 seconds. Right: learning curves of the loss function.

References and Follow-up Works

- References:
  - Han, Jentzen, and E, Solving high-dimensional partial differential equations using deep learning, *arXiv:1707.02568*
  - E, Han, and Jentzen, Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations, *Communications in Mathematics and Statistics* (2017)
- Follow-up works:
  - Beck et al. 2017: deep 2BSDE method – solve fully nonlinear PDEs and second-order BSDEs through their connections and approximate the gradient and Hessian by DNN.
  - Henry-Labordère 2017: deep primal-dual algorithm for BSDEs
  - Fujii et al. 2017: use asymptotic expansion as prior knowledge to reduce error and accelerate convergence.

Table of Contents

Summary

This work proposes the so-called deep BSDE method, which can solve general nonlinear high-dimensional parabolic PDEs.

1. We reformulate the parabolic PDEs as BSDEs and approximate the unknown gradient by deep neural networks.
2. Numerical results validate the proposed algorithm in high dimensions, in terms of both accuracy and speed.
3. This opens up new possibilities in various disciplines involving PDE modelings.

Thank you for your attention!