# Deep Learning-Based Numerical Methods for High-Dimensional Stochastic Control Problems and Parabolic PDEs

Jiequn Han

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

Stochastic Control, Computational Methods, and Applications,
IMA, May 9, 2018

# Outline

# Table of Contents

# Well-known Examples of Controls/PDEs

- Stochastic controls: portfolio optimization, robotics, resource allocation, mean field games, etc.
- PDEs:
  - The Hamilton-Jacobi-Bellman equation in stochastic control (dynamic programming),

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

  - The Black-Scholes equation for pricing financial derivatives,

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

  - The Schrödinger equation in quantum many-body problem,

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

# Curse of Dimensionality

- The dimension can be easily large in practice.

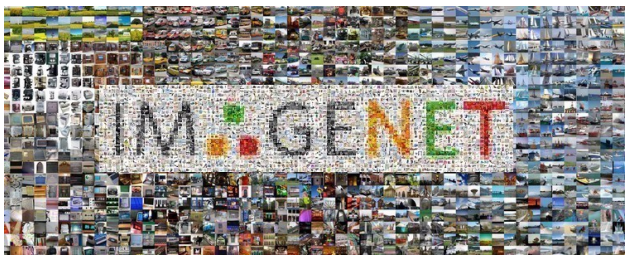
| Equation | Dimension (roughly) |
|---|---|
| HJB equation | the same as the state space |
| Black-Scholes equation | # of underlying financial assets |
| Schrödinger equation | # of electrons $\times$ 3 |

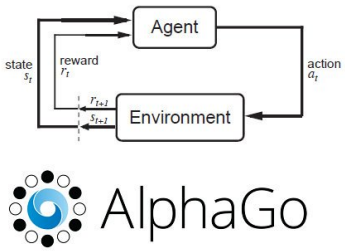
- A key computational challenge is the curse of dimensionality: the complexity is exponential in dimension $d$ for finite difference/element method – usually unavailable for $d \geq 4$.

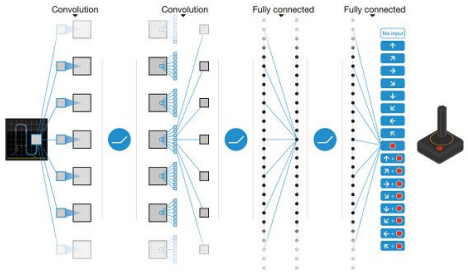
- There is a huge gap between stochastic control/PDE modelings and computational algorithms.

# 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



# Deep Learning 101

- Representation: in a compositional form rather than additive,

$$f(x) = \mathcal{L}^{out} \circ \mathcal{L}^{N_h} \circ \mathcal{L}^{N_h-1} \circ \cdots \circ \mathcal{L}^{1}(x),$$

$$h_p = \mathcal{L}^{p}(h_{p-1}) = \sigma\big(W_p h_{p-1} + b_p\big),$$

  $\sigma$ : element-wise nonlinear activation function: $\max(0, x)$, hyperbolic tangent, sigmoid, etc.

- Optimization:

$$\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} L(f(x_i; \theta)).$$

  Algorithm: stochastic gradient descent (SGD) and its variants.

# Table of Contents

# Formulation

Model dynamics:

$$s_{t+1} = s_t + b_t(s_t, a_t) + \xi_{t+1}.$$

State: $s_t$, control: $a_t$, randomness: $\xi_t$. Objective:

$$\min_{\{a_t\}_{t=0}^{T-1}} \mathbb{E}\{ \sum_{t=0}^{T-1} c_t(s_t, a_t(s_t)) + c_T(s_T) \mid s_0\},$$

Define cumulative cost for later use:

$$C_t = \sum_{\tau=0}^{t} c_\tau(s_\tau, a_\tau), \quad t = 0, 1, \cdots, T-1,$$
$$C_T = C_{T-1} + c_T(s_T).$$

# Neural Network Approximation

We look for a feedback control:

$$a_t = a_t(s_t).$$

- Traditional methods in operation research: discretize state and/or control into finite spaces + approximate dynamic programming.
- Neural network approximation:

$$a_t(s_t) \approx a_t(s_t|\theta_t),$$

Solve directly the approximate optimization problem

$$\min_{\{\theta_t\}_{t=0}^{T-1}} \mathbb{E}\{ \sum_{t=0}^{T-1} c_t(s_t, a_t(s_t|\theta_t)) + c_T(s_T)\},$$

rather than dynamic programming principle.

# Network Architecture

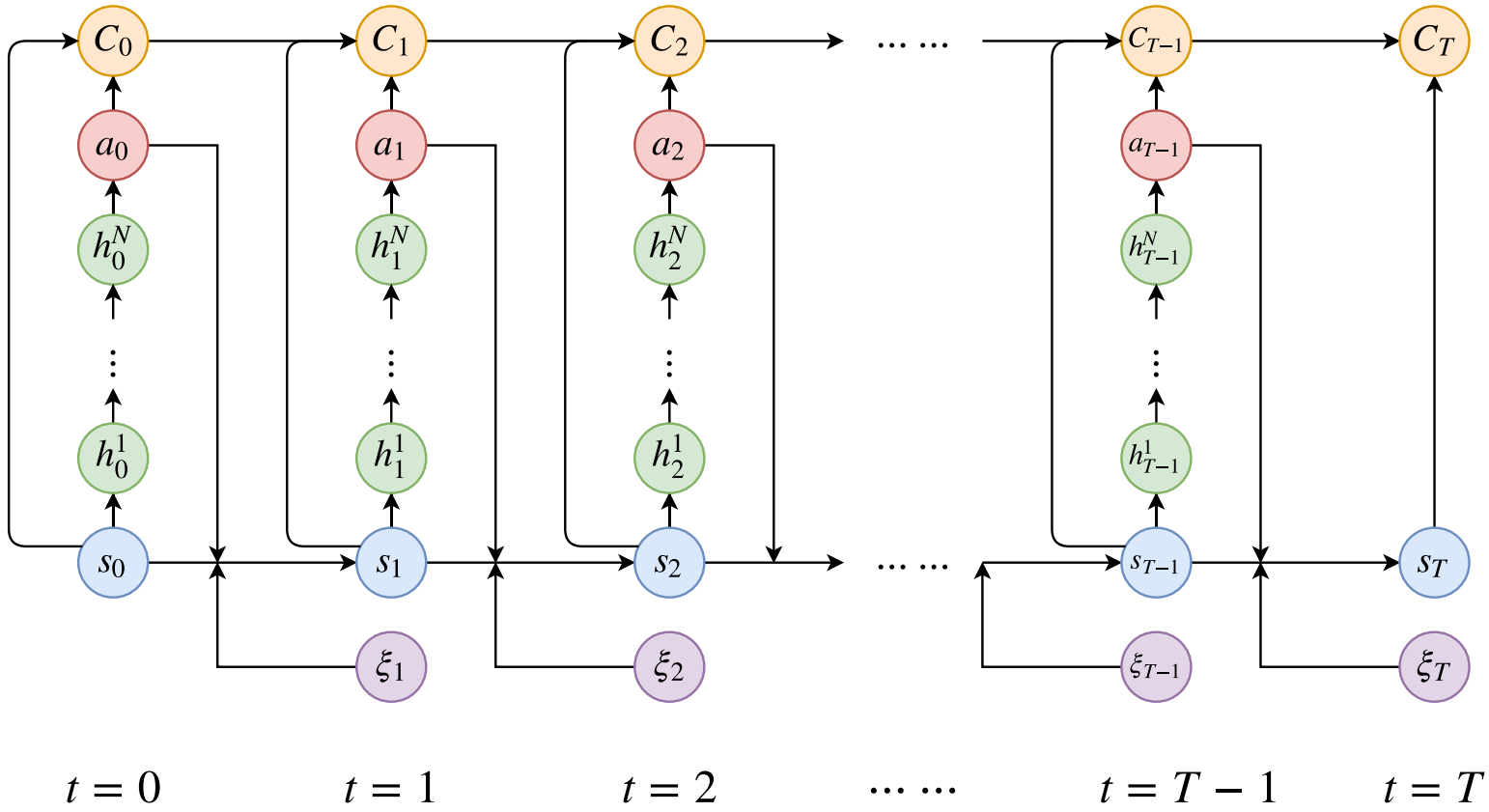



**Figure:** Network architecture for solving stochastic control in discrete time. The whole network has $(N+1)T$ layers in total that involve free parameters to be optimized simultaneously. Each column (except $\xi_t$) corresponds to a sub-network at $t$.

# Table of Contents

# Semilinear Parabolic PDE

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

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

- 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

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

# 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 $\theta_n$ 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

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

where

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

# Network Architecture

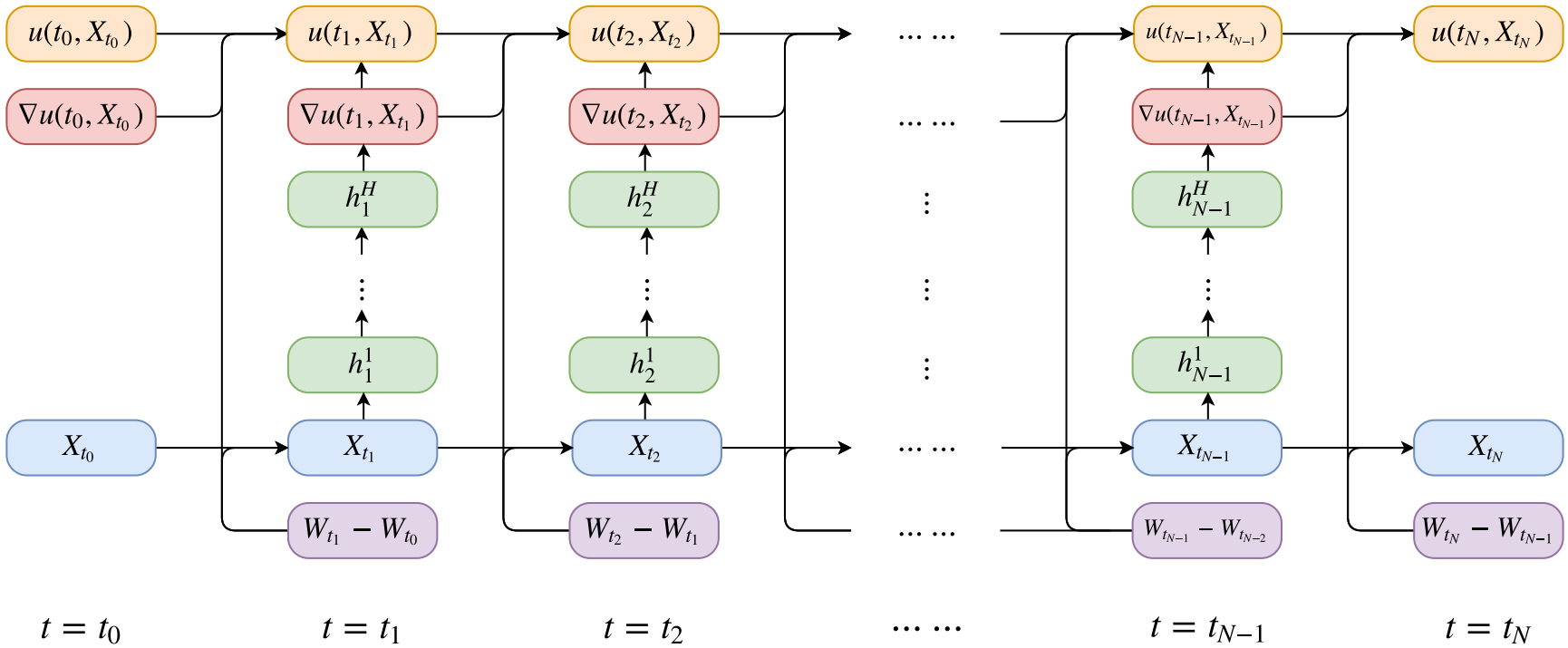



**Figure:** Network architecture for solving parabolic PDEs. Each column corresponds to a subnetwork at time $t = t_n$. The whole network has $(H + 1)(N - 1)$ layers in total that involve free parameters to be optimized simultaneously.

# Optimization

- This network takes the paths $\{X_{t_n}\}_{0\leq n\leq N}$ and $\{W_{t_n}\}_{0\leq n\leq N}$ as the input data and gives the final output, denoted by $\hat{u}(\{X_{t_n}\}_{0\leq n\leq N}, \{W_{t_n}\}_{0\leq n\leq N})$, as an approximation to $u(t_N, X_{t_N})$.
- The error in the matching of given terminal condition defines the expected loss function

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

- 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.

# 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:

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

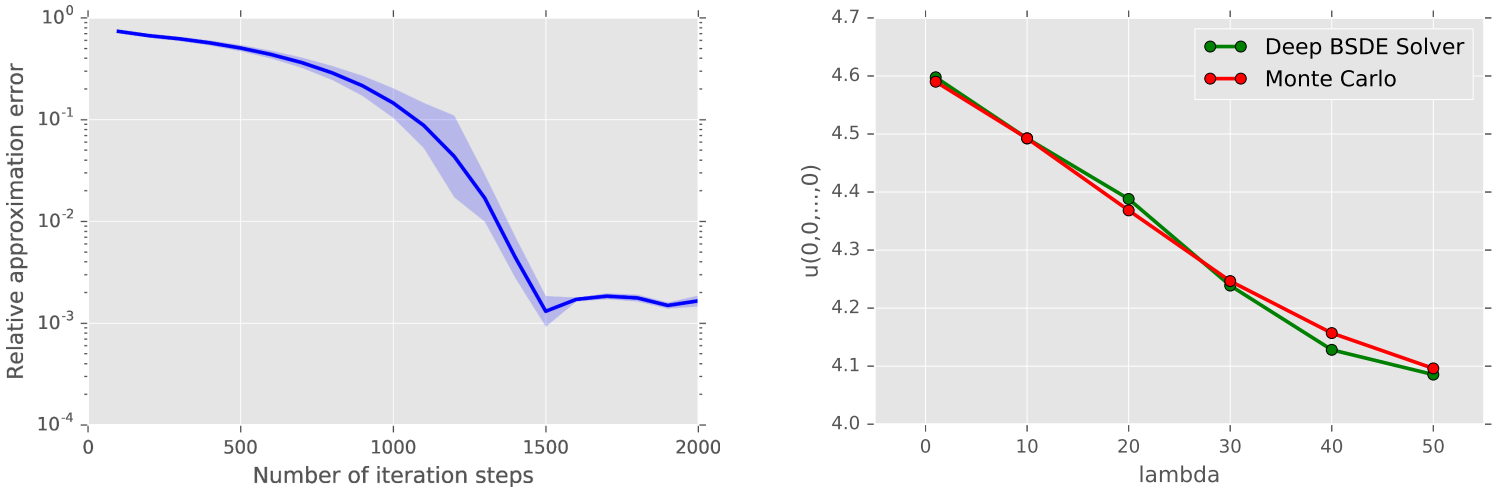



**Figure:** Left: Relative error of the deep BSDE method for $u(t{=}0, x{=}(0,\ldots,0))$ when $\lambda = 1$, which achieves $0.17\%$ in a runtime of $330$ seconds. Right: Optimal cost $u(t{=}0, x{=}(0,\ldots,0))$ against different $\lambda$.

# 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.

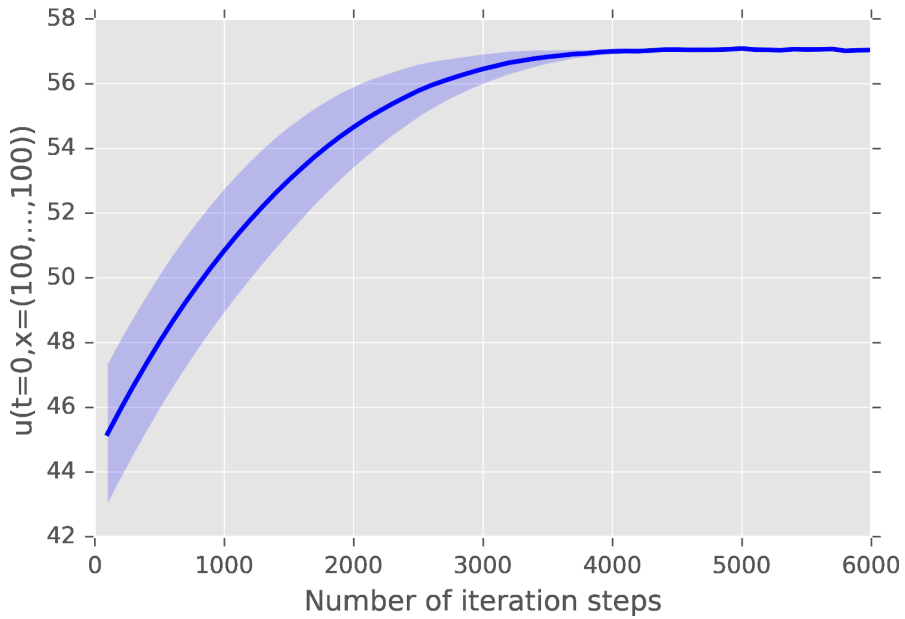



**Figure:** Approximation of $u(t{=}0, x{=}(100, \ldots, 100))$ against number of iteration steps. The deep BSDE method achieves a relative error of size $0.46\%$ in a runtime of $617$ seconds.

# 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 $t = 0.3$, $x = (0, \ldots, 0)$ is computed by the branching diffusion method.

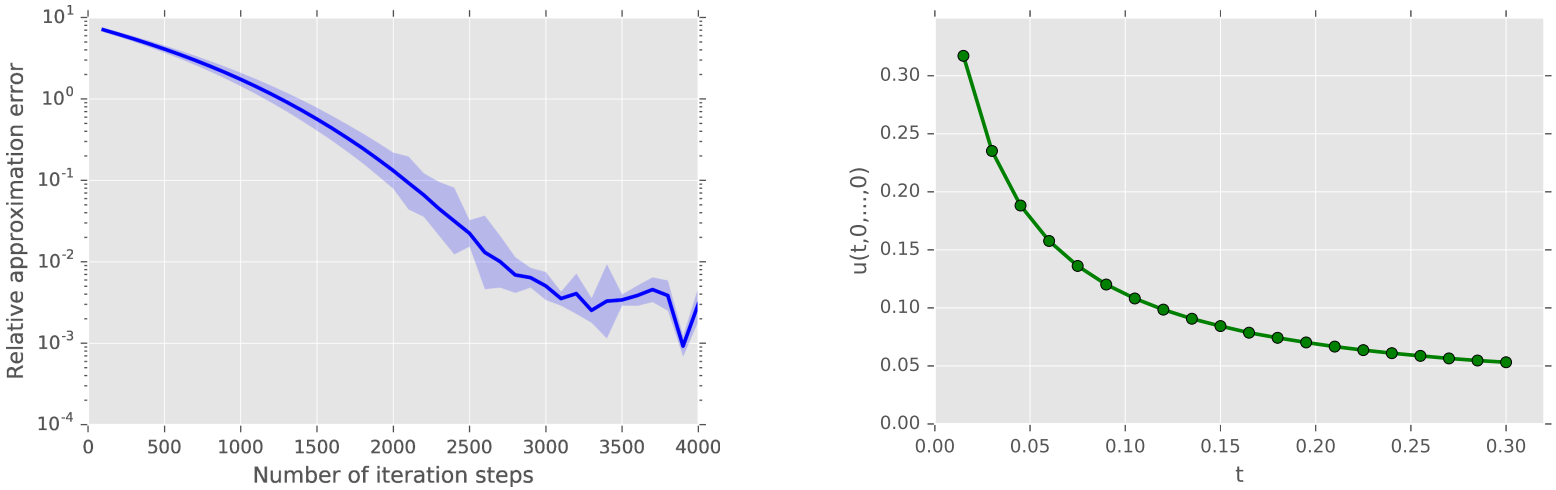



**Figure:** Left: relative error of the deep BSDE method for $u(t{=}0.3, x{=}(0, \ldots, 0))$, which achieves $0.30\%$ in a runtime of $647$ seconds. Right: time evolution of $u(t, x{=}(0, \ldots, 0))$ for $t \in [0, 0.3]$, computed by means of the deep BSDE method.

# An Example with Oscillating Explicit Solution

We consider an example studied for the numerical methods of PDE in literature (Gobet & Turkedjiev 2017). We set $d = 100$ instead of $d = 2$.

The PDE is constructed artificially in a form

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

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

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

# Effect of Number of Hidden Layers

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

| 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 |

† We only count the layers that have free parameters to be optimized.

# Effect of Activation Function

| | Nonlinear BS | LQG | Allen-Cahn |
|---|---|---|---|
| ReLU | 0.46% (0.0008) | 0.17% (0.0004) | 0.30% (0.0021) |
| Tanh | 0.44% (0.0006) | 0.17% (0.0005) | 0.28% (0.0024) |
| Sigmoid | 0.46% (0.0004) | 0.19% (0.0008) | 0.38% (0.0026) |
| Softplus | 0.45% (0.0007) | 0.17% (0.0004) | 0.18% (0.0017) |

**Table:** The mean and standard deviation (in parenthesis) of relative error obtained by the deep BSDE method with different activation functions, for the nonlinear Black-Scholes equation, the Hamilton-Jacobi-Bellman equation, and the Allen-Cahn equation.

# Table of Contents

# Summary

We 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!