

Deep Learning Algorithms for High-Dimensional Partial Differential Equations

Jiequn Han
Center of Computational Mathematics
Flatiron Institute, Simons Foundation

Tutorial lecture for Machine Learning and Its Applications
Oct 10, 2022

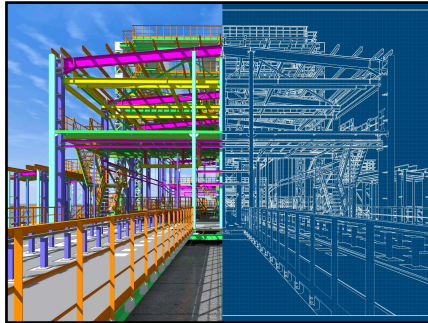
Lecture Overview

1. Background of PDE, curse of dimensionality
2. Example 1: Deep BSDE method
3. Example 2: Variational formula, Deep Ritz method
4. Summary

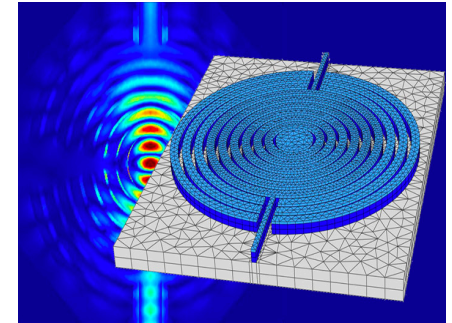
“Third Pillar” of Science

Together with theory and experimentation, computational science now constitutes the “third pillar” of scientific inquiry.

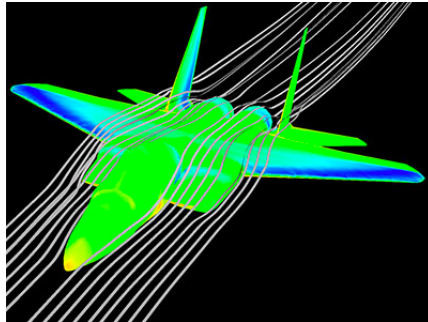
— President’s Information Technology Advisory Committee report (2005)



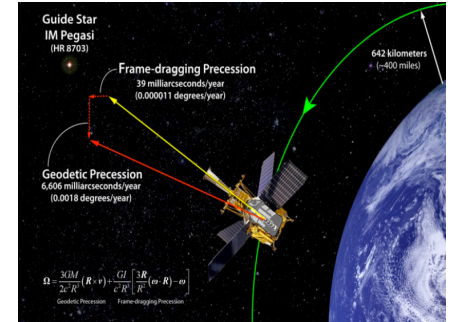
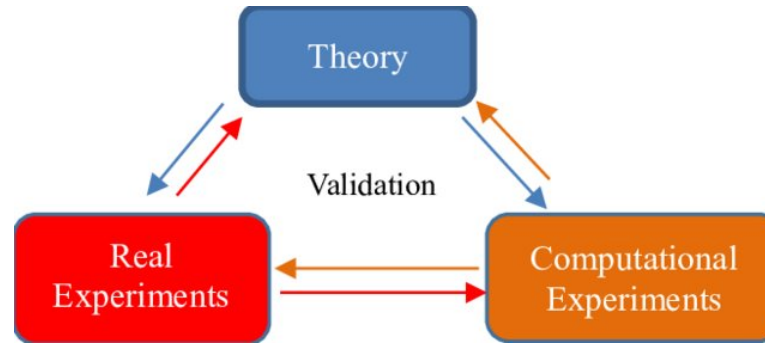
structural analysis



optics



fluid dynamics



control

Partial Differential Equation (PDE)

PDE: an equation involving an unknown function of *multiple variables* and certain of its *partial derivatives*.

Fix an integer $k \geq 1$ and let Ω be an open subset of \mathbb{R}^d ($d \geq 2$), a general k -th order PDE has the form

$$F(D^k u(x), D^{k-1} u(x), \dots, Du(x), u(x), x) = 0 \quad (x \in \Omega)$$

where $F : \mathbb{R}^{d^k} \times \mathbb{R}^{d^{k-1}} \times \dots \times \mathbb{R}^d \times \mathbb{R} \times \Omega \rightarrow \mathbb{R}$ is given, and $u : \Omega \rightarrow \mathbb{R}$ is unknown

We *solve* the PDE if we find all u verifying the above formula, usually only among those satisfying certain *auxiliary boundary functions* on some part of $\partial\Omega$.

Time evolution PDE: variables = (t, x) (time, space)

$$\partial_t u(t, x) = F(\partial_x^k u(t, x), \partial_x^{k-1} u(t, x), \dots, \partial_x u(x), u(x), x)$$

Examples of PDE

Poisson equation: $\Delta u(x) = f(x)$

(Laplace operator $\Delta u := \partial_{x_1}^2 u + \partial_{x_2}^2 u + \dots + \partial_{x_n}^2 u$)

Heat equation: $u_t(t, x) = \Delta_x u(t, x)$

Wave equation: $u_{tt}(t, x) = \Delta_x u(t, x)$

Schrödinger equation: $iu_t(t, x) + \Delta_x u(t, x) = 0$

Navier-Stokes equations (4 eqns, 4 vars, 3-dim velocity \mathbf{u} , 1-dim pressure p)

$$\begin{aligned} \nabla \cdot \mathbf{u} &= 0 \\ \frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} - \nu \nabla^2 \mathbf{u} + \frac{1}{\rho} \nabla p &= 0 \end{aligned}$$

Finite Difference for 1D Poisson (1)

1D Poisson equation for $x \in [0,1]$ (it is an ordinary differential equation, ODE):

$$\begin{aligned}u''(x) &= f(x), \\u(0) &= a, \quad u(1) = b\end{aligned}$$

Finite difference approximation: taking $h > 0$ sufficiently small

$$\begin{aligned}u'(x) &\approx \frac{u(x+h) - u(x)}{h}, & u'(x) &\approx \frac{u(x) - u(x-h)}{h}, \\u''(x) &\approx \frac{u'(x+h) - u'(x)}{h} \approx \frac{u(x+h) - 2u(x) + u(x-h)}{h^2}\end{aligned}$$

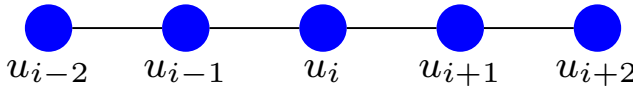
The approximation error $\sim O(h^2)$

Finite Difference for 1D Poisson (2)

1D Poisson equation for $x \in [0,1]$ (it is an ordinary differential equation, ODE):

$$\begin{aligned}u''(x) &= f(x), \\u(0) &= a, \quad u(1) = b\end{aligned}$$

Let N be an integer and $h = 1/N$, denote $x_i = ih, i = 0, 1, \dots, N, N + 1$. Denote $u_i \approx u(x_i), f_i = f(x_i)$, by finite difference approximation, we have


$$\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} = f_i, \quad i = 1, 2, \dots, N - 1$$

Together with the boundary conditions $u_0 = a, u_N = b$, we have a linear system consisting of $(N + 1)$ equations and $(N + 1)$ variables. We solve it to obtain an approximating solution, with error $\sim O(h^2)$

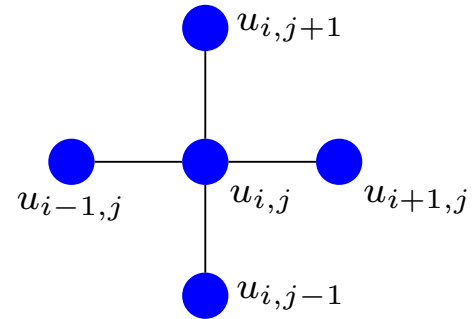
Finite Difference for 2D Poisson (1)

2D Poisson equation for $(x, y) \in \Omega := [0,1] \times [0,1]$ (we write two spacial variables x, y explicitly):

$$\begin{aligned}\Delta u(x, y) &= f(x, y), \quad (x, y) \in \Omega \\ u(x, y) &= g(x, y), \quad (x, y) \in \partial\Omega\end{aligned}$$

We still have the finite difference approximation:

$$\begin{aligned}\Delta u(x, y) &= u_{xx}(x, y) + u_{yy}(x, y) \\ &\approx \frac{u(x-h, y) - 2u(x, y) + u(x+h, y)}{h^2} + \frac{u(x, y-h) - 2u(x, y) + u(x, y+h)}{h^2} \\ &= \frac{u(x-h, y) + u(x+h, y) - 4u(x, y) + u(x, y-h) + u(x, y+h)}{h^2},\end{aligned}$$

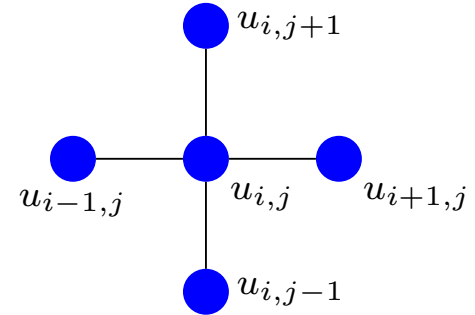


Finite Difference for 2D Poisson (2)

2D Poisson equation for $(x, y) \in \Omega := [0,1] \times [0,1]$ (we write two spacial variables x, y explicitly):

$$\begin{aligned}\Delta u(x, y) &= f(x, y), & (x, y) \in \Omega \\ u(x, y) &= g(x, y), & (x, y) \in \partial\Omega\end{aligned}$$

If we similarly define $h = 1/N$, $x_i = ih$, $y_j = jh$,
 $i, j = 0, 1, \dots, N, N + 1$, denote $u_{i,j} \approx u(x_i, y_j)$, we can have a
linear system consisting of $(N + 1)^2$ variables and $(N + 1)^2$
equations. We solve it to obtain an approximating solution, with
error $\sim O(h^2)$



Curse of Dimensionality (1)

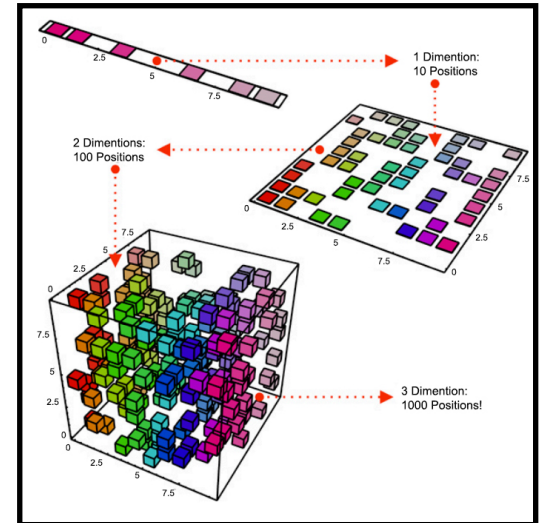
To summarize, with $h = 1/N$, to have a solution with error $O(h^2)$:

- 1D problems \rightarrow solving a system of size $O(N)$
- 2D problems \rightarrow solving a system of size $O(N^2)$
- 3D problems \rightarrow solving a system of size $O(N^3)$
- ...
- d-dim problems \rightarrow solving a system of size $O(N^d)$

Curse of dimensionality:

of nodes per dimension = N \rightarrow solution with error $O(h^2)$

As the dimension of variables d grows, the computational cost $O(N^d)$ grows exponentially with respect to d



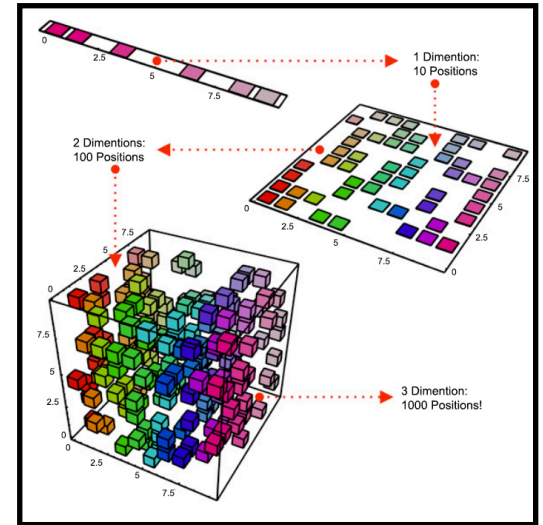
To reduce the error by a factor 4, the system size grows with a factor of 2^d

Curse of Dimensionality (2)

The **curse of dimensionality** exists not only in the finite difference method, but also in other numerical methods for PDEs, such as finite element method, spectral method, etc. In a lot of problems, we can only handle $d = 3 \sim 5$

Example of **high-dimensional PDEs**: Black-Scholes equation, Schrödinger equation, Hamilton-Jacobi-Bellman equation, etc.

The phenomenon also exists in other scientific computing problems whenever they involve **many state variables**



The deep-rooted reason is related to the fact that, to approximate high-dimensional functions with traditional methods, we need **exponentially many basis functions**.

Wonder of DL: Approximating High-Dim Functions

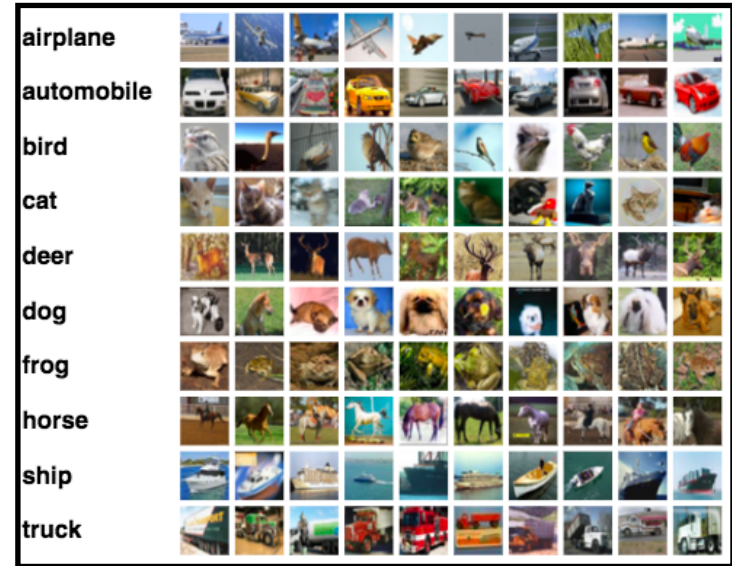
Deep learning: general function approximation

Given $S = \{(x_j, y_j = f^*(x_j)), j = 1, \dots, N\}$

learn (i.e., approximate) f^*

which is a high-dimensional mapping

CIFAR 10 - input: each image is $32 * 32 * 3 = 3072$ dimensional; output: 10 categories



Imagine we have something like
“polynomials” but works in high dimensions!

Deep Learning 101

Representation: in a compositional form rather than additive

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

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

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

Optimization: solve (near-) optimal parameter $\theta = \{W_1, b_1, W_2, b_2, \dots\}$

$$\min_{\theta} \frac{1}{N} \sum_{i=1}^N (f(x_i; \theta) - y_i)^2 \quad \text{or} \quad \min_{\theta} \mathbb{E}_{(X,Y)} (f(X; \theta) - Y)^2$$

Algorithm: compute the gradient efficiently through backpropagation and optimize through stochastic gradient descent (SGD) and its variants.

Deep Learning for PDE 101

Step 1. Choose a **formulation** equivalent to the PDE and write down the corresponding functional optimization problem.

Step 2. Utilize deep neural network based approximation to design candidate/trial function space (**use DL representation**)

Step 3. Approximate the objective functional with some numerical quadrature (**traditional numerical analysis**)

Step 4. Solve the final optimization problem (**use DL optimization**)

Deep Learning for PDE 101

Step 1. Choose a **formulation** equivalent to the PDE and write down the corresponding functional optimization problem.

Step 2. Utilize deep neural network based approximation to design candidate/trial function space (**use DL representation**)

Step 3. Approximate the objective functional with some numerical quadrature (**traditional numerical analysis**)

Step 4. Solve the final optimization problem (**use DL optimization**)

Physics-Informed Neural Network (1)

$$F(D^2u(x), Du(x), x) = 0 \quad x \in \Omega$$

$$u(x) = u_0(x) \quad x \in \partial\Omega$$

Step 1. Choose a **formulation**. Consider the **strong formulation/least-squares formulation**: u is the PDE solution is equivalent to

$$\int_{\Omega} |F(D^2u(x), Du(x), x)|^2 dx = 0, \quad \int_{\partial\Omega} |u(x) - u_0(x)|^2 dx = 0.$$

u minimizes the functional
$$\int_{\Omega} |F(D^2u(x), Du(x), x)|^2 dx + \int_{\partial\Omega} |u(x) - u_0(x)|^2 dx.$$

Step 2. **DL representation**.

$$\min_{u_{\theta}} \int_{\Omega} |F(D^2u_{\theta}(x), Du_{\theta}(x), x)|^2 dx + \int_{\partial\Omega} |u_{\theta}(x) - u_0(x)|^2 dx,$$

where u_{θ} is a neural network with parameters θ .

Physics-Informed Neural Network (2)

$$F(D^2u(x), Du(x), x) = 0 \quad x \in \Omega$$

$$u(x) = u_0(x) \quad x \in \partial\Omega$$

Step 3. Use **Monte Carlo method** to approximate the integral functional

$$\min_{u_\theta} \frac{1}{N_1} \sum_{i=1}^{N_1} |F(D^2u_\theta(x_e^i), Du_\theta(x_e^i), x_e^i)|^2 + \frac{1}{N_2} \sum_{j=1}^{N_2} |u_\theta(x_b^j) - u_0(x_b^j)|^2,$$

where x_e^i ($i = 1, \dots, N_1$) and x_b^j ($j = 1, \dots, N_2$) are points sample in Ω and $\partial\Omega$ to evaluate the equation loss and boundary loss, respectively.

Step 4. **DL optimization**. Compute the gradient through backpropagation and use stochastic (the data points x_e^i, x_b^j are resampled in different steps) gradient descent to optimize parameters θ .

Physics-Informed Neural Network (3)

Burger's equation:

$$u_t + uu_x - (0.01/\pi)u_{xx} = 0, \quad x \in [-1, 1], \quad t \in [0, 1],$$

$$u(0, x) = -\sin(\pi x),$$

$$u(t, -1) = u(t, 1) = 0.$$

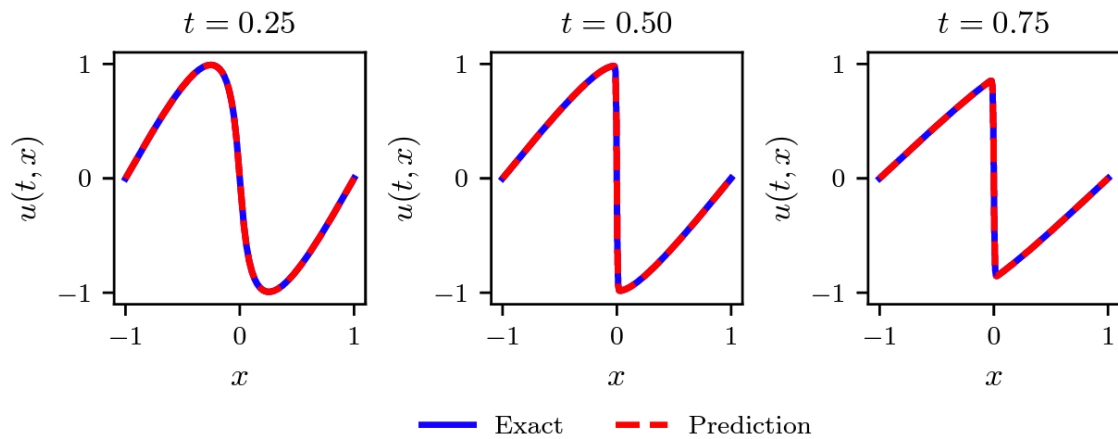


Table of Contents

1. Background of PDE, curse of dimensionality
- 2. Example 1: Deep BSDE method**
3. Example 2: Variational formula, Deep Ritz method
4. Summary

Linear Parabolic PDEs (1)

Linear PDE:
$$\frac{\partial u}{\partial t}(t, x) + \frac{1}{2}\Delta u(t, x) + \nabla u(t, x) \cdot \mu(t, x) + f(t, x) = 0, \quad u(T, x) = g(x)$$
$$(\Omega = [0, T] \times \mathbb{R}^d)$$

Examples: Black-Scholes, diffusion equation, ... interested in the high-dim cases

Feynman-Kac formula:

$$\begin{aligned} dX_t &= \mu(t, X_t)dt + dW_t \\ \implies du(t, X_t) &= (u_t(t, X_t) + \nabla u(t, X_t) \cdot \mu(t, X_t) + \frac{1}{2}\Delta u(t, X_t))dt + [\nabla u(t, X_t)]^T dW_t \quad (\text{It\^o's lemma}) \\ \implies du(t, X_t) &= -f(t, X_t)dt + [\nabla u(t, X_t)]^T dW_t \\ \implies \mathbb{E}[u(T, X_T) - u(t, X_t)] &= -\mathbb{E} \int_t^T f(s, X_s)ds \\ \implies u(t, x) &= \mathbb{E} \left[g(X_T) + \int_t^T f(s, X_s)ds \mid X_t = x \right] \end{aligned}$$

$u(t, X_t)$ satisfies a **backward stochastic differential equation (BSDE)** with a given terminal condition

Linear Parabolic PDEs (2)

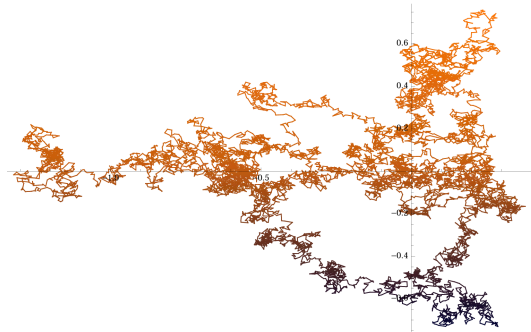
Linear PDE:
$$\frac{\partial u}{\partial t}(t, x) + \frac{1}{2}\Delta u(t, x) + \nabla u(t, x) \cdot \mu(t, x) + f(t, x) = 0, \quad u(T, x) = g(x)$$

Feynman-Kac formula

$$dX_t = \mu(t, X_t)dt + dW_t$$

Monte Carlo Simulation

$$\Rightarrow u(t, x) = \mathbb{E}\left[g(X_T) + \int_t^T f(s, X_s)ds \mid X_t = x \right]$$



In order to solve $u(t, x)$, we only need to simulate a lot of paths of diffusion process starting from x at time t and take the average.

The convergence rate is $1/\sqrt{N}$ ($N = \#$ of paths), which is independent of dimensions!

Semilinear Parabolic PDEs

$$\frac{\partial u}{\partial t}(t, x) + \frac{1}{2}\Delta u(t, x) + \nabla u(t, x) \cdot \mu(t, x) + f(t, x, u(t, x), \nabla u(t, x)) = 0, \quad u(T, x) = g(x)$$

With the idea of Feynman-Kac formula

$$dX_t = \mu(t, X_t)dt + dW_t$$

$$\implies du(t, X_t) = -f(t, X_t, u(t, X_t), \nabla u(t, X_t))dt + [\nabla u(t, X_t)]^T dW_t$$

$$\implies u(t, x) = \mathbb{E}[g(X_T) + \int_t^T f(s, X_s, u(s, X_s), \nabla u(s, X_s))ds \mid X_t = x]$$

But we do not know the value of $u(s, X_s)$, $\nabla u(s, X_s)$ along the paths :(

Variational Formulation (1)

$$\frac{\partial u}{\partial t}(t, x) + \frac{1}{2}\Delta u(t, x) + \nabla u(t, x) \cdot \mu(t, x) + f(t, x, u(t, x), \nabla u(t, x)) = 0, \quad u(T, x) = g(x)$$

Rename $Y_t = u(t, X_t)$

Suppose $Y_0 = u(0, X_0), Z_t = \nabla u(t, X_t)$

$$dX_t = \mu(t, X_t)dt + dW_t$$

$$dX_t = \mu(t, X_t)dt + dW_t$$

$$du(t, X_t) = -f(t, X_t, u(t, X_t), \nabla u(t, X_t))dt + [\nabla u(t, X_t)]^T dW_t$$

$$dY_t = -f(t, X_t, Y_t, Z_t)dt + (Z_t)^T dW_t$$

$$u(T, x) = g(x)$$

$$Y_T = g(X_T)$$

Formulation

$$\inf_{Y_0, \{Z_t\}_{0 \leq t \leq T}} \mathbb{E} |g(X_T) - Y_T|^2$$

s.t.

$$dX_t = \mu(t, X_t)dt + dW_t$$

$$dY_t = -f(t, X_t, Y_t, Z_t)dt + (Z_t)^T dW_t$$

A control perspective to match terminal condition

Variational Formulation (2)

Rename $Y_t = u(t, X_t)$

Suppose $Y_0 = u(0, X_0), Z_t = \nabla u(t, X_t)$

$$dX_t = \mu(t, X_t)dt + dW_t \longrightarrow \inf_{Y_0, \{Z_t\}_{0 \leq t \leq T}} \mathbb{E} |g(X_T) - Y_T|^2$$

$$du(t, X_t) = -f(t, X_t, u(t, X_t), \nabla u(t, X_t))dt + [\nabla u(t, X_t)]^T dW_t$$

$$u(T, x) = g(x)$$

$$s.t. \quad dX_t = \mu(t, X_t)dt + dW_t$$

$$dY_t = -f(t, X_t, Y_t, Z_t)dt + (Z_t)^T dW_t$$

1. The PDE solution is the minimizer
2. The minimizer is unique (BSDE theory)
3. 1+2: The minimizer is the PDE solution
4. An approximate minimizer is an approximate PDE solution (to be shown later)

Discretization and Representation

$$\inf_{Y_0, \{Z_t\}_{0 \leq t \leq T}} \mathbb{E} |g(X_T) - Y_T|^2$$

$$s.t. \quad dX_t = \mu(t, X_t)dt + dW_t$$

$$dY_t = -f(t, X_t, Y_t, Z_t)dt + (Z_t)^T dW_t$$

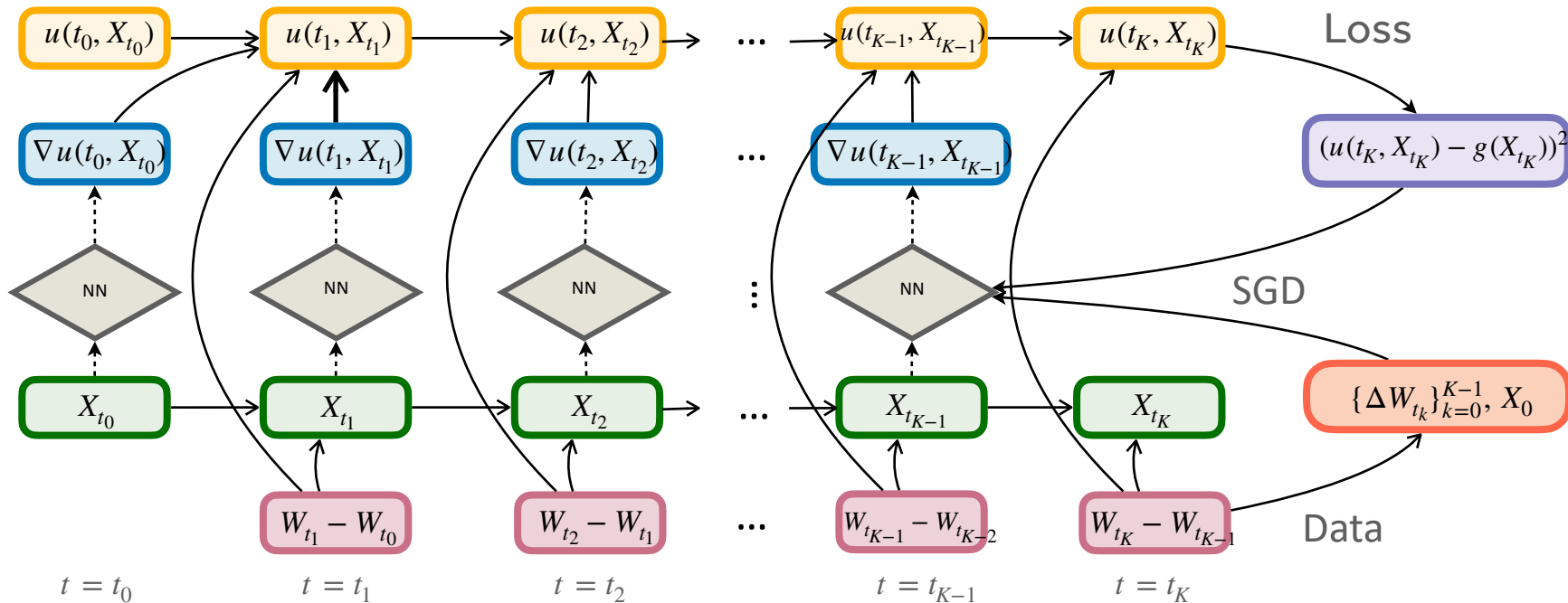
With temporal discretization $0 = t_0 < t_1 < \dots < t_K = T$, and $\Delta t_k = t_{k+1} - t_k$, $\Delta W_k = W_{t_{k+1}} - W_{t_k}$,

$$\min_{Y_0: \mathbb{R}^{d+1} \rightarrow \mathbb{R}^d, \mathcal{N}_k: \mathbb{R}^{d+1} \rightarrow \mathbb{R}^d, k=0, \dots, K-1} \mathbb{E} |g(X_{t_K}) - Y_{t_K}|^2$$
$$s.t. \quad X_{t_{k+1}} = X_{t_k} + \mu(t_k, X_{t_k})\Delta t_k + \Delta W_k$$
$$Y_{t_{k+1}} = Y_{t_k} - f(t_k, X_{t_k}, Y_{t_k}, \mathcal{N}_k(t_k, X_{t_k}))\Delta t_k + [\mathcal{N}_k(t_k, X_{t_k})]^T \Delta W_k$$

Represent high-dimensional functions Y_0, Z_{t_k} ($k = 0, 1, \dots, K - 1$) as neural networks

Deep BSDE Method

$$X_{t_{k+1}} = X_{t_k} + \mu(t_k, X_{t_k})\Delta t_k + \Delta W_k \quad Y_{t_{k+1}} = Y_{t_k} - f(t_k, X_{t_k}, Y_{t_k}, \mathcal{N}_k(t_k, X_{t_k}))\Delta t_k + [\mathcal{N}_k(t_k, X_{t_k})]^\top \Delta W_k$$



E, Han, and Jentzen, CMS (2017); Han, Jentzen, and E, PNAS (2018), code available at <https://github.com/frankhan91/DeepBSDE>

Numerical Analysis

4. An approximate minimizer is an approximate PDE solution

Theorem (A posteriori error estimate, Han-Long, 2020)

Denote by π the temporal discretization: $0 = t_0 < t_1 < \dots < t_K = T$, and $h = \max_{0 \leq k \leq K-1} \Delta t_k$.

Under some assumptions, there exists a constant C independent of d and h such that for sufficiently small h

$$\sup_{t \in [0, T]} \mathbb{E} |Y_t - \hat{Y}_t^\pi|^2 + \int_0^T \mathbb{E} |Z_t - \hat{Z}_t^\pi|^2 dt \leq C \left(h + \mathbb{E} |g(X_T) - Y_T^\pi|^2 \right),$$

Annotations: "gradient of solution u " points to the integral term; "loss function" points to the term $\mathbb{E} |g(X_T) - Y_T^\pi|^2$; "solution u " points to \hat{Y}_t^π ; "time mesh size" points to h .

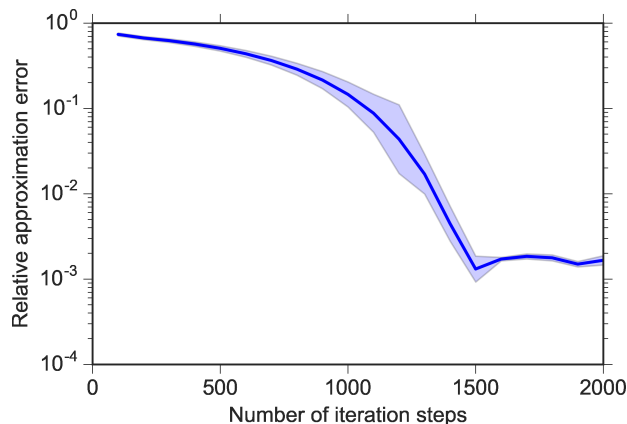
where $\hat{Y}_t^\pi = Y_{t_k}^\pi$, $\hat{Z}_t^\pi = Z_{t_k}^\pi$ for $t \in [t_k, t_{k+1})$.

A Hamilton-Jacobi-Bellman Example

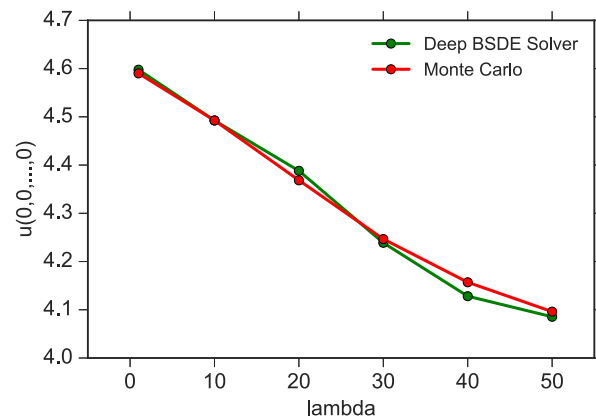
Consider a Hamilton-Jacobi-Bellman equation derived from a classical linear-quadratic-Gaussian (LQG) control problem in \mathbb{R}^{100}

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

Effectiveness: achieves 0.17% in a runtime of 5 minutes



Robust for different λ



Nonlinear Black-Scholes Equation

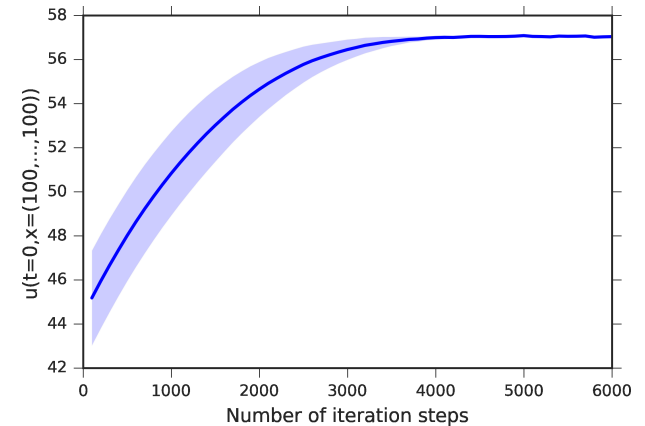
$$\frac{\partial u}{\partial t}(t, x) + \frac{1}{2}\sigma^2 \sum_{i=1}^d x_i^2 \frac{\partial^2 u}{\partial x_i^2} + r \nabla u(t, x) \cdot x - (1 - \delta)Q(u(t, x)) - \beta u(t, x) = 0.$$

It models the fair price of a European option based on 100 underlying assets with no default having occurred yet. (Duffie et al. 1996, Bender et al. 2015 (d=5))

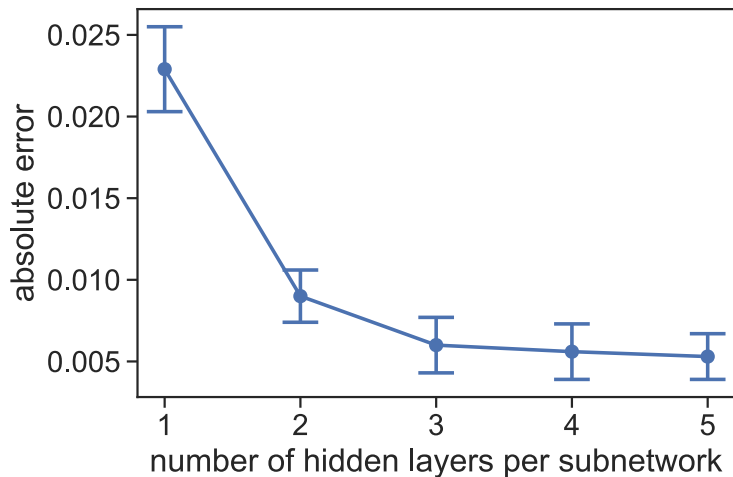
Applications: pricing basket options (Becker et al. 2019, 2020, Yu et al. 2019, Liang et al. 2020,), Libor market model (Want et al. 2018), risk value adjustment (Gnoatto et al. 2020), insurance (Kremsner et al. 2020), etc.

The Deep BSDE method has been tested on ~10 more PDEs.

E, Han, and Jentzen, CMS (2017); Han, Jentzen, and E, PNAS (2018)



Ablation Study



	LQG	Nonlinear BS	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)

Comparison with Strong Formulation

We can use Deep Neural Network to approximate high-dimensional functions

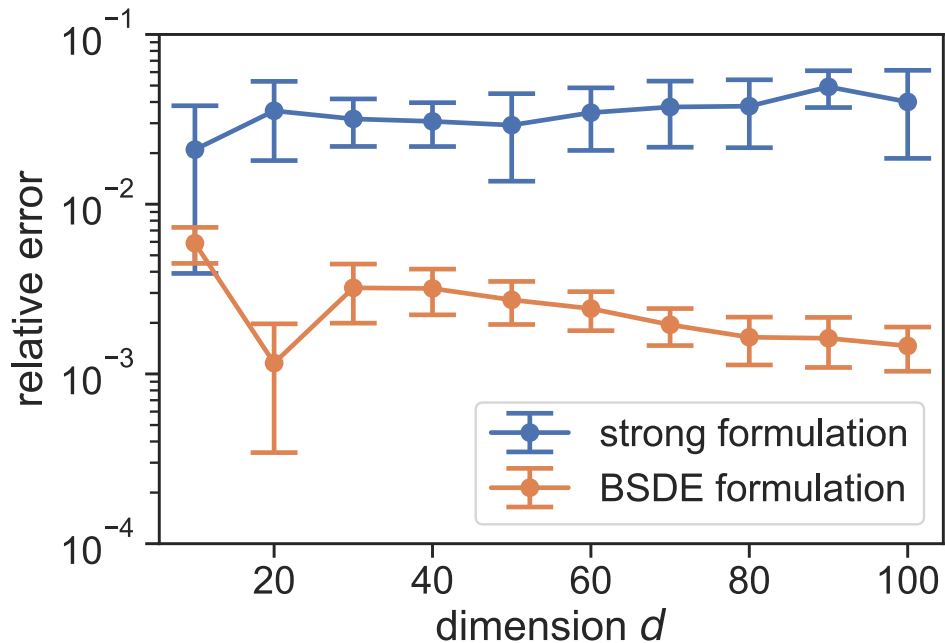
The PDE solutions themselves are high-dimensional functions

WHY DON'T WE USE DEEP NEURAL NETWORKS TO DIRECTLY APPROXIMATE THE PDE SOLUTION?

YES! WE CAN

However, we can achieve better results, when we exploit the mathematical structure of the original problem

Comparison with Strong Formulation



Note: Hessian evaluation is needed in the strong formulation but not in the BSDE formulation.

Table of Contents

1. Background of PDE, curse of dimensionality
2. Example 1: Deep BSDE method
- 3. Example 2: Variational formula, Deep Ritz method**
4. Summary

Variational Formulation

Variational Principle/Variational Calculus: In many physical problems, the solution is related to **minimization/maximization**.

Example: light follows the path of shortest optical length connecting two points, which depends upon the material of the medium (Fermat's principle/principle of least action)

For (multi-dimensional) Poisson equation:
$$\begin{aligned} -\Delta u(x) &= f(x), & x \in \Omega \\ u(x) &= g(x), & x \in \partial\Omega \end{aligned}$$

Define the functional $J(v) = \int_{\Omega} \frac{1}{2} |\nabla v(x)|^2 - f(x)v(x) \, dx$, and admissible set

$K = \{v : v \in C(\bar{\Omega}), v(x) = g(x) \text{ for all } x \in \partial\Omega\}$

Formulation

We can prove, if u is the Poisson equations' solution $\iff u$ must be a minimizer of $\min_{v \in K} J(v)$

Proof of Variational Formulation

Poisson equation: $\Delta u(x) = f(x)$, $x \in \Omega$, $u(x) = g(x)$, $x \in \partial\Omega$

$$J(v) = \int_{\Omega} \frac{1}{2} |\nabla v(x)|^2 - f(x)v(x) \, dx, K = \{v : v \in C(\bar{\Omega}), v(x) = g(x) \text{ for all } x \in \partial\Omega\}$$

If u is the minimizer of $\min_{v \in K} J(v)$, u is the Poisson equations' solution

Proof: Let u be a minimizer and $\phi \in C_0^\infty(\Omega)$ be a perturbation. We consider the function

$$\begin{aligned} \tilde{J}(\lambda) &= J(u + \lambda\phi) = \int_{\Omega} \frac{1}{2} |\nabla(u + \lambda\phi)(x)|^2 - f(x)(u + \lambda\phi)(x) \, dx \\ &= J(u) + \lambda \int_{\Omega} \nabla u(x) \cdot \nabla \phi(x) - f(x)\phi(x) \, dx + \lambda^2 \int_{\Omega} \frac{1}{2} |\nabla \phi(x)|^2 \, dx \end{aligned}$$

By definition, 0 is a minimizer of $\tilde{J}(\lambda)$, we have

$$0 = \tilde{J}'(0) = \int_{\Omega} \nabla u(x) \cdot \nabla \phi(x) - f(x)\phi(x) \, dx = \int_{\Omega} (-\Delta u(x) - f(x))\phi(x) \, dx.$$

Since ϕ is arbitrarily chosen, we have $-\Delta u(x) - f(x) = 0$.

Deep Ritz Algorithm

DL Representation: use a neural network $u_\theta(x)$ as the the trial function

$$\min_{\theta} J(u_\theta(x)) = \int_{\Omega} \frac{1}{2} |\nabla u_\theta(x)|^2 - f(x)u_\theta(x) \, dx$$
$$s.t. \quad u_\theta(x) = g(x), x \in \partial\Omega$$

In practice, consider the relaxed version

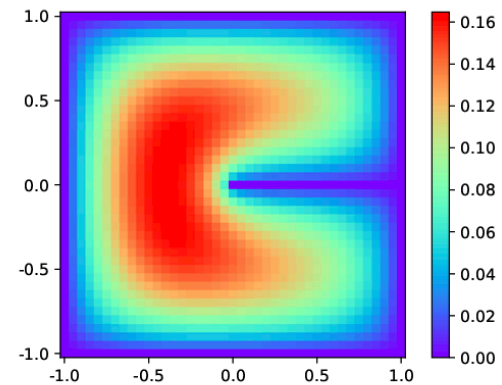
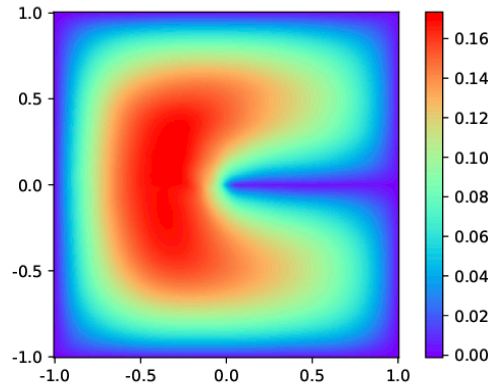
$$\min_{\theta} J(u_\theta(x)) = \int_{\Omega} \frac{1}{2} |\nabla u_\theta(x)|^2 - f(x)u_\theta(x) \, dx + \beta \int_{\partial\Omega} |u_\theta(x) - g(x)|^2 \, dx$$

Then use Monte Carlo method to approximate the objective and stochastic gradient descent to optimize parameters.

Numerical Results (1)

$$\begin{aligned} -\Delta u(x) &= 1, \quad x \in \Omega \\ u(x) &= 0, \quad x \in \partial\Omega \end{aligned}$$

$$\Omega = (0,1) \times (0,1) \setminus [0,1) \times \{0\}$$



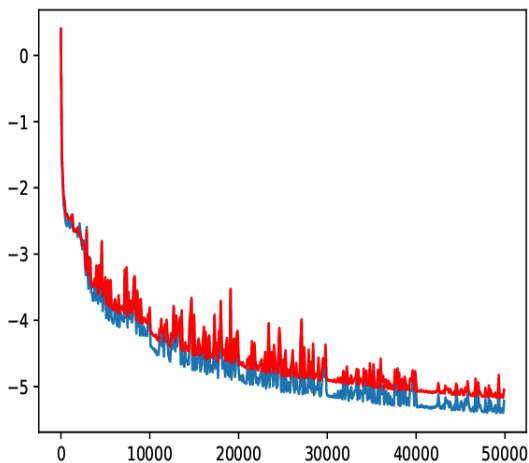
(a) Solution of Deep Ritz method, 811 parameters (b) Solution of finite difference method, 1,681 parameters

Numerical Results (2)

$$-\Delta u(x) = 0, \quad x \in (0,1)^{10}$$

$$u(x) = \sum_{k=1}^5 x_{2k-1} x_{2k}, \quad x \in \partial(0,1)^{10}$$

Relative L^2
error: 0.4%

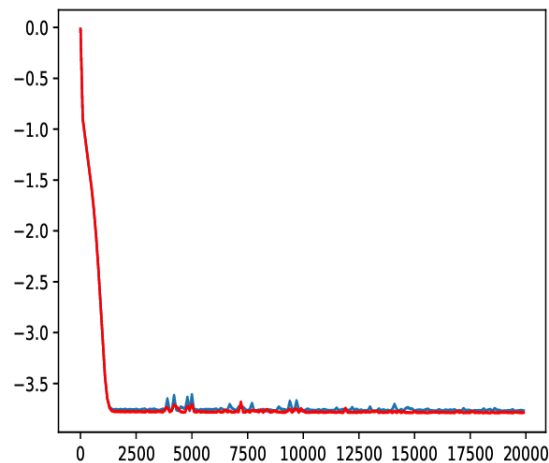


(a) $\ln e$ and $\ln \text{loss}_{\text{boundary}}$, $d=10$

$$-\Delta u(x) = -200, \quad x \in (0,1)^{100}$$

$$u(x) = \sum_{k=1}^d x_k^2, \quad x \in \partial(0,1)^{100}$$

Relative L^2
error: 2.2%



(b) $\ln e$ and $\ln \text{loss}_{\text{boundary}}$, $d=100$

Ground State of Schrödinger Equation

The **ground state** of a quantum-mechanical system is its stationary state of lowest energy

$$\min_{u(x)} \frac{\int_{\Omega} \frac{1}{2} |\nabla u(x)|^2 + V(x)u^2(x) \, dx}{\int_{\Omega} u^2(x) \, dx}$$

It is equivalent to the smallest eigenvalue problem $-\Delta u + Vu = \lambda u$

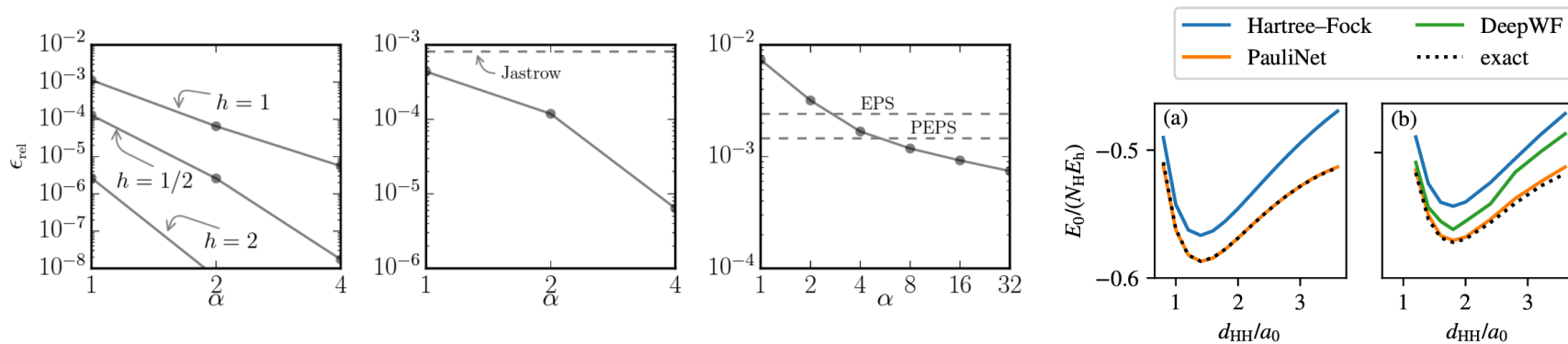
(analogous to the Rayleigh quotient in linear algebra: the eigenvector corresponding to the smallest eigenvalue of a p.s.d. matrix H is the minimizer of $x^T H x / x^T x$)

Solving the ground state is one of the central problems in quantum physics/chemistry. The dimension of x is proportional to the system's size.

People have used **Variational Monte Carlo (VMC)** method to solve the ground state. VMC's procedure is similar to the deep learning method for high-dimensional PDEs (more careful treatment of sampling and gradient computation) but with traditional trial function space.

Using Neural Networks in VMC

Carleo and Troyer (2017) introduce neural networks to represent quantum states for Ising system, which leads to tremendous research interest in applying ML in physical sciences.



Han, Zhang, and E (2018), Hermann, Schätzle, and Noé (2020), Pfau, Spencer, and Matthews (2020) study neural network representation for electron wavefunctions in quantum chemistry.

Table of Contents

1. Background of PDE, curse of dimensionality
2. Example 1: Deep BSDE method
3. Example 2: Variational formula, Deep Ritz method
4. Summary

Summary

For a long time, solving high-dimensional PDEs suffers from the curse of dimensionality. Deep learning provides us efficient tools for overcoming this difficulty.

See the review paper *Algorithms for solving high dimensional PDEs: from nonlinear Monte Carlo to machine learning*, E, Han, and Jentzen (2022)

Step 1. Choose a **formulation** equivalent to the PDE and write down the corresponding functional optimization problem.

Step 2. Utilize deep neural network based approximation to design candidate/trial function space (**use DL representation**)

Step 3. Approximate the objective functional with some numerical quadrature (**traditional numerical analysis**)

Step 4. Solve the final optimization problem (**use DL optimization**)

Not Covered

1. Use deep learning algorithms to solve high-dimensional optimal control problems/multi-agent games/Hamilton-Jacobi-Bellman equations.
2. Use machine learning to solve complex low-dimensional PDEs in fluid dynamics, climate modeling, etc.
3. Use deep learning algorithms to approximate operators corresponding to PDE solutions.
4. Combine data driven and model (PDE)-based methods.
5. Use dynamical system/control to study deep learning.