Journal of Computational Physics 541 (2025) 114327

Contents lists available at ScienceDirect

Journal of Computational Physics

journal homepage: www.elsevier.com/locate/jcp

Deep learning for high-dimensional PDEs with fat-tailed Lévy measure

Kamran Arif[a], Guojiang Xi[b], Heng Wang[a], Weihua Deng[a,*]

[a] School of Mathematics and Statistics, State Key Laboratory of Natural Product Chemistry, Lanzhou University, Lanzhou, 730000, China
[b] Agile and Intelligent Computing Key Laboratory of Sichuan Province, China

A R T I C L E I N F O

Keywords:
Jump process
Fat-tailed Lévy measure
Fractional Laplacian
BSDE
Deep learning

A B S T R A C T

The partial differential equations (PDEs) for jump process with Lévy measure have wide applications. When the measure has fat tails, it will bring big challenges for both computational cost and accuracy. In this work, we develop a deep learning method for high-dimensional PDEs related to fat-tailed Lévy measure, which can be naturally extended to the general case. Building on the theory of backward stochastic differential equations for Lévy processes, our deep learning method avoids the need for neural network differentiation and introduces a novel technique to address the singularity of fat-tailed Lévy measures. The developed method is used to solve four kinds of high-dimensional PDEs: the diffusion equation with fractional Laplacian; the advective diffusion equation with fractional Laplacian; the advective diffusion reaction equation with fractional Laplacian; and the nonlinear reaction diffusion equation with fractional Laplacian. The parameter β in fractional Laplacian is an indicator of the strength of the singularity of Lévy measure. Specifically, for $\beta \in (0,1)$, the model describes super-ballistic diffusion; while for $\beta \in (1,2)$, it characterizes super-diffusion. In addition, we experimentally verify that the developed algorithm can be easily extended to solve fractional PDEs with finite general Lévy measures. Our method achieves a relative error of $\mathcal{O}(10^{-3})$ for low-dimensional problems and $\mathcal{O}(10^{-2})$ for high-dimensional ones. We also investigate three factors that influence the algorithm's performance: the number of hidden layers; the number of Monte Carlo samples; and the choice of activation functions. Furthermore, we test the efficiency of the algorithm in solving problems in 3D, 10D, 20D, 50D, and 100D. Our numerical results demonstrate that the algorithm achieves excellent performance with deeper hidden layers, a larger number of Monte Carlo samples, and the Softsign activation function.

1. Introduction

With the rapid development of science and technology, an increasing number of novel phenomena are being observed. The motion of particles is no exception, e.g., Brownian yet non-Gaussian, strong anomalous diffusion, etc, especially in biological field [1–4]. Currently, it is widely recognized that non-Brownian motion is much more popular than Brownian one. Roughly speaking, according to the relationship between the variance and time t of a stochastic process, anomalous diffusion can be classified as sub-diffusion, normal diffusion, and super-diffusion [5]. Additional types of diffusion include ballistic diffusion, super-ballistic diffusion, polymer diffusion, turbulent diffusion, localization, etc.

* Corresponding author.
E-mail addresses: kamranarif850@gmail.com (K. Arif), 994679291@qq.com (G. Xi), 220220934161@lzu.edu.cn (H. Wang), dengwh@lzu.edu.cn (W. Deng).

https://doi.org/10.1016/j.jcp.2025.114327
Received 17 March 2025; Received in revised form 29 July 2025; Accepted 23 August 2025
Available online 29 August 2025

Jump process is a class of important microscopic models to describe diffusion. Compound Poisson process belongs to Lévy process, and is also a jump process [6]. Generally, the governing equations for the probability density functions of the statistical observables of the jump process involve nonlocal operators, and if the equation is discussed in the bounded domain Ω, the boundary conditions should be specified in $\mathbb{R}^d\backslash\Omega$ [7]. The compound Poisson process can describe both normal diffusion and super-diffusion, in fact, it can even characterize the ballistic diffusion and super-ballistic diffusion, which depend on the choice of the probability measure of the jump length. When the second moment of the jump length is bounded, macroscopiclly, the compound Poisson process behaves like Brownian motion (its scaling limit is Brownian motion [8]). In this case, the operators of the corresponding macroscopic equations are nonlocal but non-singular. However, when the probability measure of jump length has fat tails, like $|x|^{-\beta-d}$, the nonlocal operators have singular kernel; and the compound Poisson process respectively describes super-diffusion for $\beta \in (1,2)$, ballistic diffusion for $\beta = 1$, and super-ballistic diffusion for $\beta \in (0,1)$. This paper focuses on the jump length with distribution $|x|^{-\beta-d}$.

There have been extensive studies of traditional numerical methods for solving the governing macroscopic equation of the compound Poisson process with a fat-tailed Lévy measure, including the finite difference method [9], the finite element method [10], etc. The main challenges of the traditional numerical methods come from the fractional Laplacian operator of the macroscopic equation, which affects the regularity of the solution of the equation, hugely increases both the memory and computational costs, and makes it hard to program even for three-dimensional case. The main objective of this paper is to treat high-dimensional cases, like one hundred dimension. So, we turn to deep learning method.

When neural network is used to solve partial differential equations (PDEs), it is expected to conquer the curse of dimensionality occuring in the traditional methods [11]. Several frameworks have been proposed for solving PDEs using neural networks. One approach is Physics-Informed Neural Networks (PINNs), which takes the sampling points in the solving domain and its boundary, then constructs the loss function based on the residual of the PDE [12–15]. The second one is to use the energy functional of the PDEs as the loss function [16]. The third one is for the PDEs which don't have the energy functional, like Petrov-Galerkin framework, which uses the adversarial network [17]. In the above three frameworks, neural network is taken as the approximation function of the solution of the PDE. The idea of the fourth one is much different from the first three, which has the stronger sense of machine learning; first, one needs to find a backward stochastic differential equation (BSDE) driven by a stochastic process, which has the "same" solution as the PDE, then uses the sample trajectories generated by the stochastic process to train the BSDE [18]. Based on BSDE theory, this framework can obtain solutions at arbitrary target positions (or regions) in high-dimensional Cauchy problems without imposing boundary constraints. The major advantages of BSDE over PINN on these unbounded problems are that BSDE is more theoretically grounded and that BSDE requires lower-order derivatives making it computational efficient.

The theoretical results on BSDE provide the foundation of the deep learning method under the framework of BSDE [19], which is developed to solve semilinear parabolic differential equations [18]. More recently, the idea is used to solve the equation with nonlocal operator [20]; and the operator is the generator of Lévy process describing normal diffusion, implying that the used Lévy measure rapidly tends to zero when $|x| \to \infty$. The process related to the equation discussed in this paper is Lévy process characterizing super-diffusion, ballistic diffusion, and super-ballistic diffusion, which means the corresponding Lévy measure has fat tails. This paper aims to develop a deep learning method for high-dimensional PDEs related to fat-tailed Lévy measure, which can be naturally extended to the general case. Based on the theory of BSDE for Lévy process, in developing the deep learning method, the differentiation of neural network is circumvented, and the technique is introduced to treat the singularity of the fat-tailed Lévy measure. The developed method is used to solve four kinds of high-dimensional PDEs: the diffusion equation with fractional Laplacian; the advective diffusion equation with fractional Laplacian; the advective diffusion reaction equation with fractional Laplacian; the nonlinear reaction diffusion equation with fractional Laplacian. The parameter β in fractional Laplacian is an indicator of the strength of the singularity of Lévy measure. Specifically, for $\beta \in (0,1)$, the model describes super-ballistic diffusion, whereas for $\beta \in (1,2)$, it characterizes super-diffusion. The developed deep learning method reaches the relative error of $\mathcal{O}(10^{-3})$ in low-dimensional problems, and $\mathcal{O}(10^{-2})$ in high-dimensional problems. Our deep learning method is influenced by three factors: the number of hidden layers; the number of Monte Carlo samples; and activation functions. The numerical results demonstrate that the algorithm achieves optimal stability with deeper hidden layers, a larger number of Monte Carlo samples, and the Softsign activation function. More importantly, the developed algorithm is verified to effectively solve diffusion equations with fractional Laplacians in various dimensions and can be easily extended to solve equations with finite general Lévy measures. We have open-sourced the code for readers to test: https://github.com/WANGH950/Fat-tailedDeepLearning.

The rest of this paper is organized as follows. In Section 2, we introduce the linear and nonlinear PDEs with fractional Laplacian and present the overall methodology of deep learning method. Section 3 reports the numerical results for the 3-dimensional and 100-dimensional equations and shows the performance of the developed method. We conclude the paper with some discussions in the last section.

2. Methodology

The Lévy process $X(t)$ is a stochastic process with stationary and independent increments, which has an exponential characteristic function

$$\mathbb{E}\left[e^{i(\xi,X(t))}\right] = e^{t\phi(\xi)}$$

with

$$\phi(\xi) = i(\mu,\xi) - \frac{1}{2}(\xi,a\xi) + \int_{\mathbb{R}^d\backslash\{0\}} \left[e^{i(\xi,y)} - 1 - i(\xi,y)\chi_{0<|y|<r}(y)\right]\nu(dy).$$

Here, $\mu \in \mathbb{R}^d$, $a \in \mathbb{R}^d \times \mathbb{R}^d$, $r \in \mathbb{R}^+$, and ν is a Lévy measure on $\mathbb{R}^d \setminus \{0\}$. In the case that ν is finite, one can take $r = 0$. By doing the derivative of time t and Fourier inverse transform of ξ, one can get the corresponding advective diffusion equation

$$\frac{\partial u(x,t)}{\partial t} + \mu \cdot \nabla u(x,t) = \frac{1}{2}\mathrm{Tr}\left(a^T \left(\mathrm{Hess}_x\right) u(x,t)\right) + \int_{\mathbb{R}\setminus\{0\}} \left[u(x-y,t) - u(x,t) + y^T \nabla u(x,t)\chi_{0<|y|<r}(y)\right]\nu(dy)$$

with initial condition $u(x,0) = g(x)$. This paper focuses on the jump process with fat-tailed Lévy measure $\nu \sim |x|^{-\beta-d}$ and the corresponding equations (hence let $a = 0$), and the method considered can be naturally extended to the general form ($a \neq 0$ and arbitrary Lévy measures). Next, we will present the specific form of the equation, deal with the singularity of the Lévy measure with fat tails, derive the corresponding BSDE, and develop the deep learning algorithm.

2.1. PDEs with fat-tailed Lévy measure

Consider the general form of the fractional Laplacian equation

$$\frac{\partial u(x,t)}{\partial t} + (\mu \cdot \nabla u)(x,t) - (-\Delta)^{\beta/2} u(x,t) + f(t,x,u(x,t)) = 0 \tag{1}$$

with the terminal condition $u(x,T) = g(x)$. Here $u : \mathbb{R}^d \times [0,T] \to \mathbb{R}$ is the unknown function, $\mu : \mathbb{R}^d \times [0,T] \to \mathbb{R}^d$ and $f : [0,T] \times \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}$ are given functions, and $-(-\Delta)^{\beta/2}$ is fractional Laplacian operator defined as

$$-(-\Delta)^{\beta/2} u(x,t) = \int_{\mathbb{R}^d\setminus\{0\}} \left[u(x-y,t) - u(x,t) + y^T \nabla u(x,t)\chi_{0<|y|<r}(y)\right]\nu_\beta(dy), \tag{2}$$

where $\beta \in (0,2)$ and $\nu_\beta(dy) = c_{\beta,d}\frac{1}{|y|^{\beta+d}}dy$ is the considered fat-tailed Lévy measure with

$$c_{\beta,d} = \frac{2^\beta \Gamma\left(\frac{d+\beta}{2}\right)}{\pi^{d/2}\left|\Gamma\left(-\frac{\beta}{2}\right)\right|}.$$

In order to address the singularity of $\nu_\beta(dy)$ as $|y| \to 0$, we need to do some subtle processing on the fractional Laplacian operator. Specifically, we decompose (2) into two parts

$$-(-\Delta)^{\beta/2} u(x,t) = \int_{0<|y|<r} \left[u(x-y,t) - u(x,t) + y^T \nabla u(x,t)\right]\nu_\beta(dy) + \int_{|y|\geq r} [u(x-y,t) - u(x,t)]\nu_\beta(dy).$$

Then, one can do Taylor's expansion for $u(x-y,t)$ in the first term

$$\begin{aligned}
&\int_{0<|y|<r} \left[u(x-y,t) - u(x,t) + y^T \nabla u(x,t)\right]\nu_\beta(dy) \\
&= \frac{1}{2}\int_{0<|y|<r} y^T \mathrm{Hess}_x u(x-\theta y,t) y \nu_\beta(dy) \\
&\approx \frac{1}{2}\int_{0<|y|<r} y^T \mathrm{Hess}_x u(x,t) y \nu_\beta(dy) \\
&= \frac{1}{2}\int_{0<|y|<r} \mathrm{Tr}\left[y^T \mathrm{Hess}_x u(x,t) y\right]\nu_\beta(dy) \\
&= \frac{1}{2}\int_{0<|y|<r} \mathrm{Tr}\left[\mathrm{Hess}_x u(x,t) y y^T\right]\nu_\beta(dy) \\
&= \frac{1}{2}\mathrm{Tr}\left[\mathrm{Hess}_x u(x,t) \int_{0<|y|<r} c_{\beta,d}\frac{yy^T}{|y|^{\beta+d}}dy\right] \\
&= \frac{1}{2}c_{\beta,d}k_{\beta,d,r}\Delta u(x,t)
\end{aligned}$$

with some $\theta \in (0,1)$, where

$$k_{\beta,d,r} = \frac{\pi^{d/2}}{\Gamma\left(\frac{d}{2}+1\right)}\frac{r^{2-\beta}}{2-\beta}.$$

Finally, we can approximate fractional Laplacian operator as

$$-(-\Delta)^{\beta/2} u(x,t) \approx \frac{1}{2}c_{\beta,d}k_{\beta,d,r}\Delta u(x,t) + \int_{|y|\geq r} [u(x-y,t) - u(x,t)]\nu_\beta(dy),$$

and (1) can be approximately written as

$$\frac{\partial u(x,t)}{\partial t} + (\mu \cdot \nabla u)(x,t) + \frac{1}{2}c_{\beta,d}k_{\beta,d,r}\Delta u(x,t) + \int_{|y|\geq r} [u(x-y,t) - u(x,t)]\nu_\beta(dy) + f(t,x,u(x,t)) = 0. \tag{3}$$

The difference between the solutions of (1) and (3) is shown in Appendix C; see the L^2 estimate (C.1) and L^∞ one (C.2).

To go further, we first introduce a class of stochastic process driven by the Brownian motion $B(t)$ and compound Poisson process $L(t)$ with the intensity $\frac{c_{\beta,d}}{\widetilde{c}_{\beta,d,r}}$ and the jump length distribution $\widetilde{c}_{\beta,d,r}\frac{1}{|y|^{\beta+d}}$ for $|y| \geq r$ (for the finite measure in the case $\nu(dy) = \lambda\nu(y)dy$, we take the intensity as λ and the jump length distribution as $\nu(-y)$). The considered process satisfies

$$dX(t) = \mu(X(t),t)dt + \sqrt{c_{\beta,d}k_{\beta,d,r}}dB(t) + dL(t). \tag{4}$$

Driven by the stochastic process $X(t)$, the following result holds (The proof is provided in Appendix A).

Lemma 1. *If $X(t)$ is a stochastic process satisfying (4), then for any $u(x,t) \in \mathbb{C}^2(\mathbb{R}^d) \times \mathbb{C}^1([0,\infty])$, Itô's formula is given by*

$$\begin{aligned} &u(X(t),t) - u(X(0),0) \\ &= \int_0^t \left(\frac{\partial u(X(s),s)}{\partial t} + \frac{1}{2}c_{\beta,d}k_{\beta,d,r}\Delta u(X(s),s) + (\mu\cdot\nabla u)(X(s),s)\right)ds + \sqrt{c_{\beta,d}k_{\beta,d,r}}\int_0^t \nabla u(X(s),s)\cdot dB(s) \\ &\quad + \int_0^t\int_{|y|\geq r}[u(X(s-)+y,s) - u(X(s-),s)]J(dy\times ds), \end{aligned}$$

where J is the Poisson random measure of $L(t)$.

By combining Lemma 1 and (3), one can immediately get the BSDE that the solution of (3) satisfies (See the proof in Appendix B).

Theorem 1. *Let $u(x,t)$ be the solution of the Eq. (3) and $X(t)$ be a stochastic process satisfying (4). Then $(u(X(t),t), \sqrt{c_{\beta,d}k_{\beta,d,r}}\nabla u(X(t),t), u(X(t-)+y) - u(X(t-)))$ is the solution of the BSDE*

$$Y(t) - g(X(T)) = \int_t^T f(s,X(s),Y(s))ds - \int_t^T Z(s)\cdot dB(s) - \int_t^T\int_{|y|\geq r} U(s-,y)\widetilde{J}(dy\times ds), \tag{5}$$

where $\widetilde{J}(dy\times dt) = J(dy\times dt) - \lambda\nu(-y)dydt$ is the compensated Poisson random measure.

This theorem can be used to get the exact expression of the solution for the linear case, i.e., $u(x,t) = \mathbb{E}\left[e^{c(T-t)}g(X(T))|X(t)=x\right]$ when $f(t,x,y) = cy$ with constant c. Consequently, this expectation can be approximated by Monte Carlo simulation to obtain the "exact" solution of the equation. The solution of BSDE (5) is a triplet $(Y(t), Z(t), U(t,y))$, and its existence and uniqueness has been discussed in [21]. The mathematical theory of BSDE ensures that one can get the solution of the PDE (3) directly by solving the BSDE (5), and $(Y(t), Z(t), U(t,y))$ satisfies the relationship between the solution of the PDE and its difference. In the next step, we will present a suitable neural network architecture to solve the BSDE (5).

2.2. Neural network architecture and approximation

Our interest is to get $u(x,t)$ at fixed time t and position x, which can be approximated by a parameter θ_u. Then, one can regard the BSDE (5) with a triplet of the solution $(u(X(s),s), \sqrt{c_{\beta,d}k_{\beta,d,r}}\nabla u(X(s),s), u(X(s-)+y,s) - u(X(s-),s))$ $(t \leq s \leq T)$ as the way to obtain the approximation of the terminal value $u(X(T),T)$, which can be achieved when $\sqrt{c_{\beta,d}k_{\beta,d,r}}\nabla u(X(s),s)$ and $u(X(s-)+z) - u(X(s-))$ are known. Specifically, we first use a neural network to approximate $\nabla u(x,s) \approx \psi(x,s|\theta_{\nabla u})$ with parameters $\theta_{\nabla u}$, and another neural network to approximate $u(x+y,s) - u(x,s) \approx \psi(x,y,s|\theta_{Ju})$ with parameters θ_{Ju}. Then one can use the simple Euler scheme for the partition of the time interval $[t,T]: t = t_0 < t_1 < \ldots < t_{N-1} < t_N = T$ to discrete the stochastic Eqs. (4) and (5) as

$$X(t_{k+1}) = X(t_k) + \mu(X(t_k),t_k)\Delta t_k + \sqrt{c_{\beta,d}k_{\beta,d,r}}\Delta B(t_k) + \Delta L(t_k) \tag{6}$$

and

$$\begin{aligned} u(X(t_{k+1}),t_{k+1}) &= u(X(t_k),t_k) - f(t_k,X(t_k),u(X(t_k),t_k))\Delta t_k + \sqrt{c_{\beta,d}k_{\beta,d,r}}\nabla u(X(t_k),t_k)\cdot\Delta B(t_k) \\ &\quad + u(X(t_k)+\Delta L(t_k),t_k) - u(X(t_k),t_k) - \int_{|y|\geq r}\left[u(X(t_k)-y,t_k) - u(X(t_k),t_k)\right]\nu_\beta(dy)\Delta t_k, \end{aligned} \tag{7}$$

where $\Delta t_k = t_{k+1} - t_k$, $\Delta B(t_k) = B(t_{k+1}) - B(t_k)$, and $\Delta L(t_k) = L(t_{k+1}) - L(t_k)$. Finally, we take the discrete time $\{t_k\}_k$ and the randomly generated paths $\{B(t_k)\}_k$, $\{L(t_k)\}_k$, and $\{X(t_k)\}_k$ as the input data of the neural network, and use the scheme (7) to get the approximation of the terminal value

$$\hat{u}\left(\left\{t_k, X(t_k), B(t_k), L(t_k)\right\}_k|\theta = \left\{\theta_u, \theta_{\nabla u}, \theta_{Ju}\right\}\right).$$

The difference from the given terminal condition can be used to construct the loss function

$$Loss(\theta) = \mathbb{E}\left[\left|g(X(T)) - \hat{u}\left(\left\{t_k, X(t_k), B(t_k), L(t_k)\right\}_k|\theta\right)\right|^2\right]. \tag{8}$$

The largest computational cost of the aforementioned algorithm arises from the last integral term in the numerical scheme (7), which can be efficiently approximated by using Monte Carlo integration

$$\int_{|y|\geq r} f(y)\nu_\beta(dy) \approx \frac{c_{\beta,d}}{\widetilde{c}_{\beta,d,r}}\frac{1}{M}\sum_{i=1}^M f(y_i) \tag{9}$$

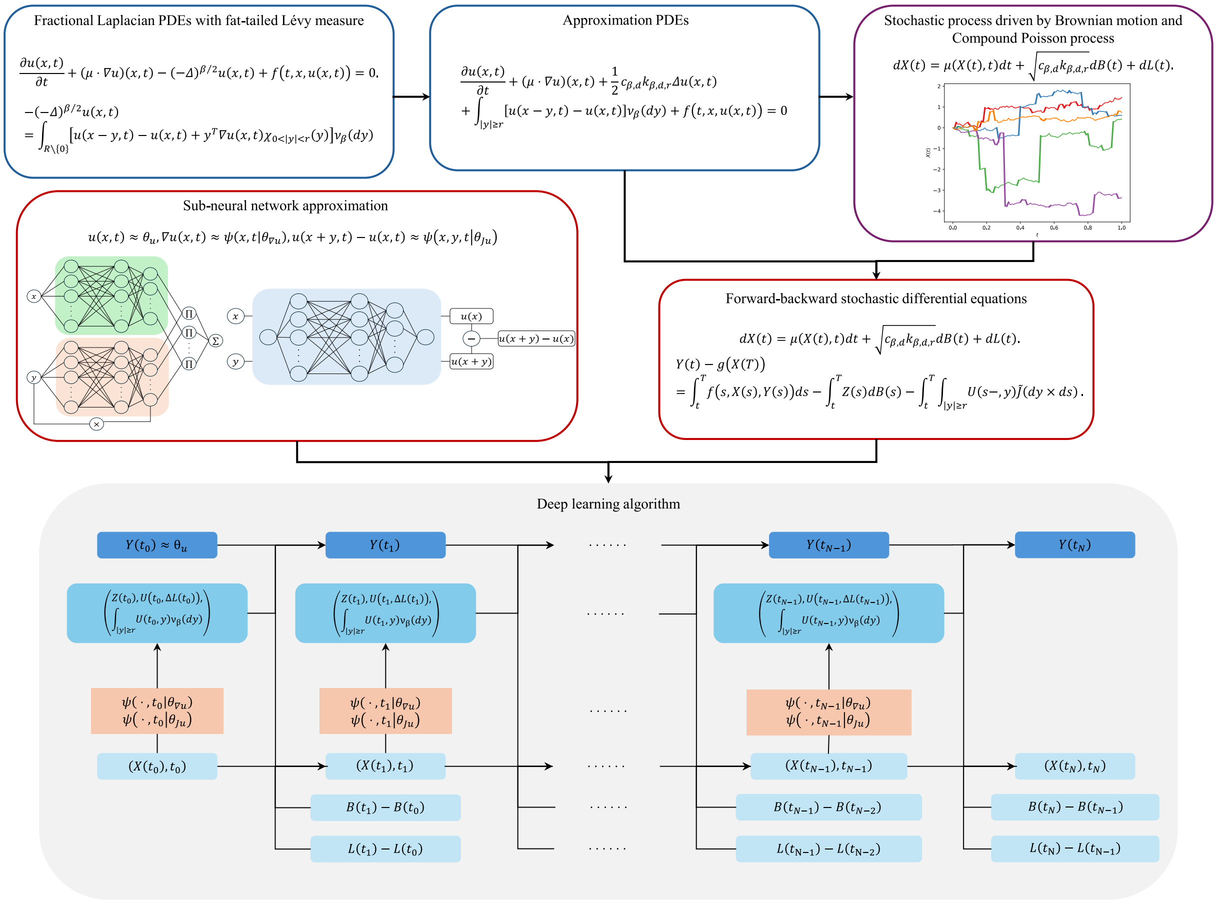
Fractional Laplacian PDEs with fat-tailed Lévy measure
Approximation PDEs
Stochastic process driven by Brownian motion and Compound Poisson process
Sub-neural network approximation
Forward-backward stochastic differential equations
Deep learning algorithm

Fig. 1. The processing flow of the developed deep learning algorithm.

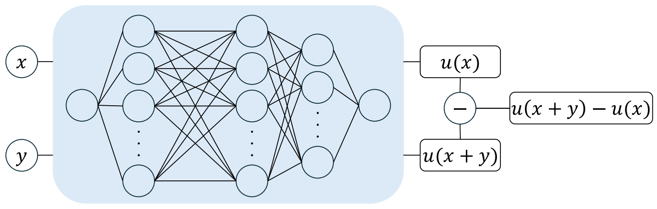

Fig. 2. The first structure for approximating $u(x+y)-u(x)$.

with independent and identically distributed random samples $y_i \sim \tilde{c}_{\beta,d,r}\frac{1}{|y|^{\beta+d}}$, $i = 1, 2, \dots, M$ combining with the following two neural network structures for approximating $U(s-, y) = u(X(s-) + y) - u(X(s-))$.

The first structure (see Fig. 2) is inspired by the relationship $U(s, y) = u(X(s) + y, s) - u(X(s), s)$, thus one can use a neural network to approximate $u(x, s) \approx \tilde{\psi}(x, s|\theta_{Ju})$ and calculate the residual directly

$$\begin{aligned} u(x+y,s)-u(x,s) &\approx \psi(x,y,s|\theta_{Ju}) \\ &= \tilde{\psi}(x+y,s|\theta_{Ju}) - \tilde{\psi}(x,s|\theta_{Ju}). \end{aligned} \tag{10}$$

The second one (see Fig. 3) is a simplified version of the tensor neural network [22], and we approximate $u(x+y)-u(x)$ by the structure

$$\psi(x,y,t|\theta_{Ju}) = \sum_{i=1}^{P} \psi_i(x,t|\theta^x_{Ju})\psi_i(y|\theta^y_{Ju})\tanh(|y|) \tag{11}$$

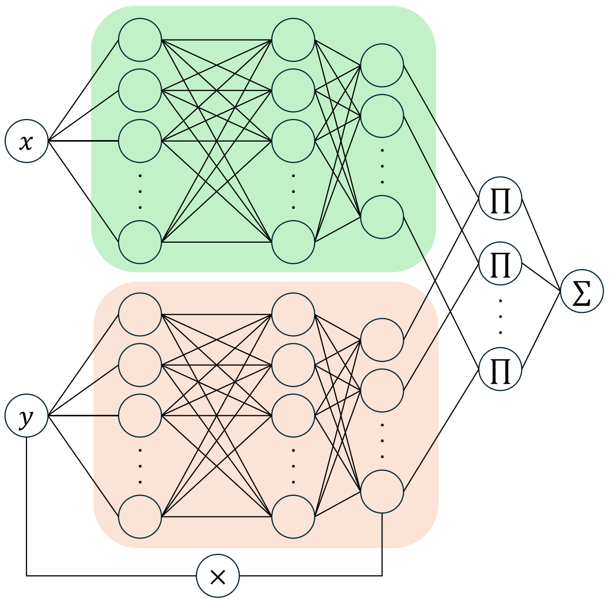

Fig. 3. The second structure (tensor decomposition) for approximating $u(x+y)-u(x)$.

for given $P \in \mathbb{N}$. Here, we multiply the final result by $\tanh(|y|)$ to ensure the structure of the approximated function $u(x+0,t)-u(x,t) = 0$ at $y=0$. This tensor decomposition structure facilitates the high-dimensional numerical integration, such that

$$\begin{aligned}
&\int_{|y|\geq r} \left[u(X(t_k)-y,t_k)-u(X(t_k),t_k)\right]\nu_\beta(dy) \\
&\approx \frac{c_{\beta,d}}{\widetilde{c}_{\beta,d,r}} \frac{1}{M}\sum_{i=1}^{M}\left[u(X(t_k)-y_i,t_k)-u(X(t_k),t_k)\right] \\
&\approx \frac{c_{\beta,d}}{\widetilde{c}_{\beta,d,r}} \frac{1}{M}\sum_{i=1}^{M}\sum_{j=1}^{P}\psi_j(X(t_k),t_k|\theta^x_{Ju})\psi_j(y_i|\theta^y_{Ju})\tanh(|y_i|) \\
&= \frac{c_{\beta,d}}{\widetilde{c}_{\beta,d,r}} \sum_{j=1}^{P}\psi_j(X(t_k),t_k|\theta^x_{Ju})\frac{1}{M}\sum_{i=1}^{M}\psi_j(y_i|\theta^y_{Ju})\tanh(|y_i|),
\end{aligned} \tag{12}$$

which makes the last term of the scheme (7) be computed only once in all iterations. Compared with the method in (10), which requires calculations at each iteration step, this method significantly reduces computational and memory costs. After all, one can use stochastic gradient descent (SGD) algorithm to optimize the parameters θ. The Adam optimizer [23] is used in the numerical experiments conducted in this paper. The processing flow of the algorithm discussed above is shown in Fig. 1, and the pseudocode is shown in Algorithm 1.

3. Numerical results

In this section, we will present some numerical results to validate the efficiency of the algorithm introduced in the previous section. Specifically, we solve four kinds of high-dimensional PDEs: the diffusion equation with fractional Laplacian; the advective diffusion equation with fractional Laplacian; the advective diffusion reaction equation with fractional Laplacian; and the nonlinear reaction diffusion equation with fractional Laplacian. In addition, we also test the performance of the developed algorithm in solving fractional equations with generalized Lévy measures, and investigate the factors influencing the algorithm's accuracy and its performance through extensive numerical experiments.

3.1. Diffusion equations

We begin with a representative type of equation, namely, diffusion equations with fractional Laplacian. The corresponding approximation version has the form

$$\frac{\partial u(x,t)}{\partial t} + \frac{1}{2}c_{\beta,d}k_{\beta,d,r}\Delta u(x,t) + \int_{|y|\geq r}[u(x-y,t)-u(x,t)]\nu_\beta(dy) = 0 \tag{13}$$

with a given terminal condition $u(x,T)=g(x)$.

Algorithm 1 The deep learning algorithm for solving fractional Laplacian equations with fat-tailed Lévy measure.

```
Input: Parameter $\theta_u$, neural networks $\psi(\cdot|\theta_{\nabla u})$ and $\psi(\cdot|\theta_{Ju})$, maximum number of iterations $\mathcal{N}$, time $t$, position $x$, the partition of the
       time interval $[t,T] : t = t_0 < t_1 < \dots < t_{N-1} < t_N = T$, Monte Carlo sampling number $M$, and learning rate $\alpha$;
Output: $\theta_u$;
 1: Initialize $step = 1$;
 2: while $step < \mathcal{N}$ do
 3:     Initialize $X(t_0) = x$, $u(X(t_0), t_0) = \theta_u$;
 4:     Generate Monte Carlo sample points $\{y_i\}_{i=1}^M$;
 5:     for $k$ from 0 to $N-1$ do
 6:         Generate $\Delta B(t_k)$, $\Delta L(t_k)$;
 7:         $\nabla u(X(t_k), t_k) \leftarrow \psi(X(t_k), t_k|\theta_{\nabla u})$;
 8:         $u(X(t_k) + \Delta L(t_k), t_k) - u(X(t_k), t_k) \leftarrow \psi(X(t_k), \Delta L(t_k), t_k|\theta_{Ju})$;
 9:         $\int_{|y|>r} \left[u(X(t_k) - y, t_k) - u(X(t_k), t_k)\right] \nu_\beta(dy) \xleftarrow{\text{Eqs. (10) or (12)}} \{\psi(X(t_k), y_i, t_k|\theta_{Ju})\}_{i=1}^M$;
10:         $u(X(t_{k+1}), t_k + 1) \xleftarrow{\text{Eq. (7)}} \Delta t_k, \Delta B(t_k), u(X(t_k), t_k), \nabla u(X(t_k), t_k),$
                $u(X(t_k) + \Delta L(t_k), t_k) - u(X(t_k), t_k),$
                $\int_{|y|>r} \left[u(X(t_k) - y, t_k) - u(X(t_k), t_k)\right] \nu_\beta(dy)$;
11:         $X(t_{k+1}) \xleftarrow{\text{Eq. (6)}} \Delta t_k, \Delta B(t_k), \Delta L(t_k), X(t_k)$;
12:     end for
13:     $Loss(\theta) \xleftarrow{\text{Eq. (8)}} g(X(T)), \hat{u}\left(\left\{t_k, X(t_k), B(t_k), L(t_k)\right\}_k|\theta\right)$;
14:     $\theta \xleftarrow{\text{SGD algorithm}} \theta, \alpha$
15:     $step \leftarrow step + 1$;
16: end while
17: return $\theta_u$.
```

To evaluate the accuracy of the algorithm, we first consider a 1-dimensional case with an exact solution $u(x,t) = x$, where $t \in [0,T]$. We choose the following parameters: $x = 1$, $t = 0$, $\beta = 1.5$, $r = 0.1$, $T = 1$, $N = 100$, $M = 10^2$ (for approximation method (10)), $M = 10^4$ (for approximation method (11)), $P = 128$, and subnetworks with a width 128 and 3 hidden layers. The final result is obtained after 2×10^4 iterations, with an initial learning rate of 5×10^{-4} that decays by a factor of 0.5 every 5×10^3 steps. Fig. 4 illustrates the deep learning approximation results, the relative errors with respect to the exact solution $u(1,0) = 1$ (using two different jump approximation methods (10) and (11)) over 2×10^4 iterations, and the comparison between the 5 BSDE trajectories $Y(t)$ and the true path $u(X(t),t)$ after training. It can be observed that both methods achieve sufficient accuracy rapidly. Our algorithm produces a relative error of $\mathcal{O}(10^{-3})$ and effectively simulates long-distance jumps. To reduce memory and computational costs, we employ method (11) as the residual approximation in the subsequent numerical examples.

We further evaluate the performance of our algorithm in solving 3-dimensional and 100-dimensional problems. For the 3-dimensional case, we choose $x = [0,0,0]^T$, $P = 128$, and subnetworks with a width of 128 and 4 hidden layers. For the 100-dimensional case, we choose $x = [0,0,\dots,0]^T$, $P = 256$, and subnetworks with a width of 512 and 4 hidden layers. The common parameters are set as $t = 0$, $\beta = 0.7$, $r = 0.1$, $T = 1$, $N = 100$, $M = 10^4$, and the terminal condition $g(x) = 10e^{-|x|}$. The final results are obtained after 10^5 iterations, with an initial learning rate of 5×10^{-4} that decays by a factor of 0.5 every 2×10^4 steps. Fig. 5 displays the approximate solutions and the relative errors for our deep learning method, where the "exact" solutions 2.0133 $(d = 3)$ and 0.1558 $(d = 100)$ can be obtained by approximating the stochastic representation

$$u(x,0) = \mathbb{E}\left[10e^{-|X(T)|}|X(0) = x\right].$$

Our method achieves the relative error of 0.35% for the 3-dimensional problem and 6.10% for the 100-dimensional problem. It converges rapidly (within fewer than 2×10^4 iterations) and demonstrates strong robustness after convergence.

3.2. Advective diffusion equations

The advective diffusion equation describes the probability density of random particles diffusing in a non-stationary medium and has a wide range of applications. In this section, we evaluate the performance of our algorithm for this case. Specifically, we consider the diffusion in a constant velocity field, and the corresponding equations can be expressed as

$$\frac{\partial u(x,t)}{\partial t} + \nabla u(x,t) + \frac{1}{2} c_{\beta,d} k_{\beta,d,r} \Delta u(x,t) + \int_{|y|\geq r} [u(x-y,t) - u(x,t)] \nu_\beta(dy) = 0 \tag{14}$$

with a given terminal condition $u(x,T) = g(x)$.

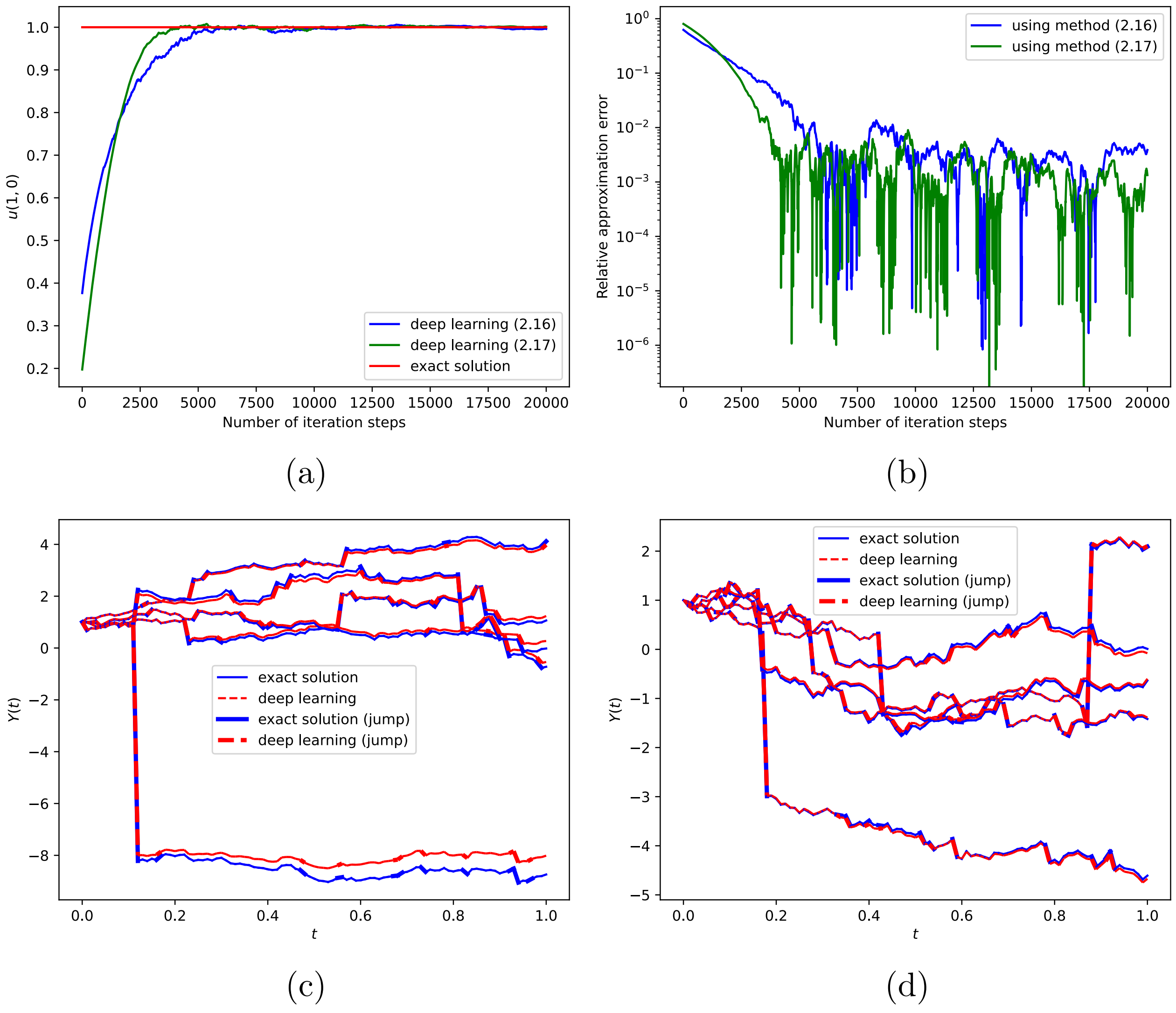

Fig. 4. Visualization of the effectiveness of deep learning methods for solving one-dimensional problem. (a) Plot of the exact solution $u(1,0)$ and the approximate solutions obtained by deep learning as a function of the number of iteration steps. The red line denotes the exact solution $u(1,0) = 1$, while the blue and green curves represent the approximate solutions using deep learning methods with approximation methods (10) and (11), respectively. (b) Plot of the relative error of the approximate solutions compared to the exact solution as a function of the number of iteration steps, using deep learning methods with approximation methods (10) and (11). (c) Depiction of five independent paths $Y(t) = u(X(t),t)$ obtained by the deep learning method with approximation method (10); (d) Depiction of five independent paths $Y(t) = u(X(t),t)$ obtained by the deep learning method with approximation method (11), where the blue curve indicates the exact solution, the red dashed curve represents the approximate solution by deep learning, and the bold lines denote the jumps. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

For the 3-dimensional case, we choose $x = [0,0,0]^T$, $\beta = 0.9$, $r = 0.1$, $P = 128$, and subnetworks with a width of 128 and 4 hidden layers. For the 100-dimensional case, we choose $x = [-1,-1,\dots,-1]^T$, $\beta = 1.5$, $r = 0.5$, $P = 256$, and subnetworks with a width of 512 and 4 hidden layers. The common parameters are set as $t = 0$, $T = 1$, $N = 100$, $M = 10^4$, and the terminal condition $g(x) = \frac{10(1+\sin(|x|))}{1+|x|^2}$. The final results are obtained after 10^5 iterations, with an initial learning rate of 5×10^{-4} that decays by a factor of 0.5 every 2.5×10^4 steps. Fig. 6 displays the approximate solutions and the relative errors as a function of the number of iterations, where the "exact" solutions $2.3560\,(d = 3)$ and $0.0909\,(d = 100)$ can be computed by approximating the stochastic representation

$$u(x,0) = \mathbb{E}\left[\frac{10(1+\sin(|X(T)|))}{1+|X(T)|^2}|X(0) = x\right].$$

The developed algorithm achieves the relative error of 0.66% for the 3-dimensional problem and 3.75% for the 100-dimensional problem. It converges rapidly (within fewer than 2×10^4 iterations for the 3-dimensional problem and fewer than 6×10^4 iterations for the 100-dimensional problem) and demonstrates strong robustness after convergence.

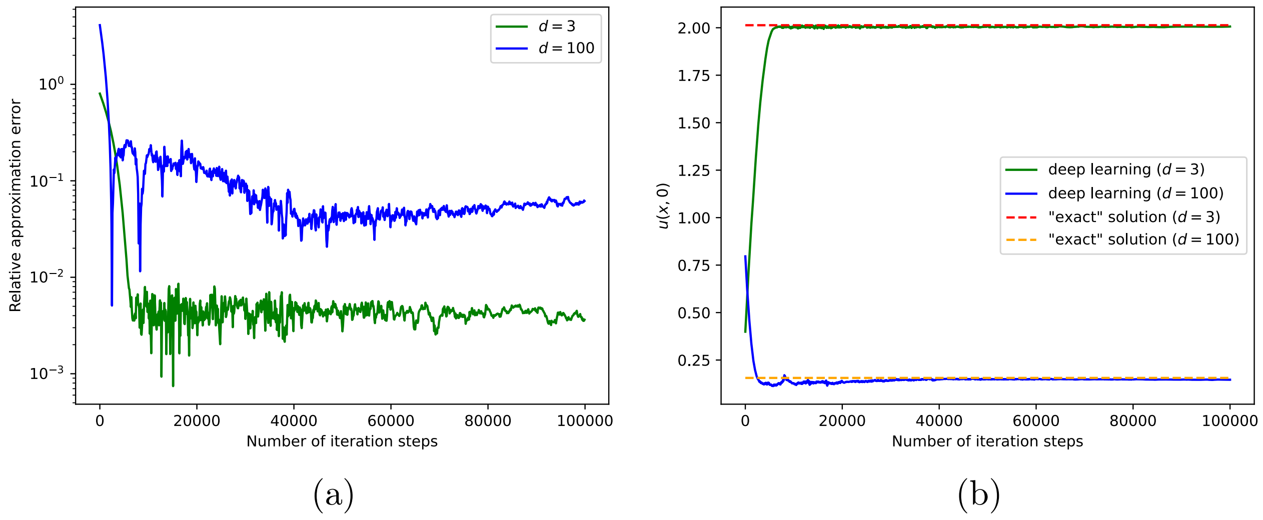

Fig. 5. (a) Relative error for the 3-dimensional (green) and 100-dimensional (blue) nonlinear problems (13) for 10^5 iterations of the developed algorithm. (b) Convergence behavior of the developed algorithm for the 3-dimensional (green) and 100-dimensional (blue) nonlinear problems (13), where the “exact” solutions (red and orange) are computed using the Monte Carlo method.(For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

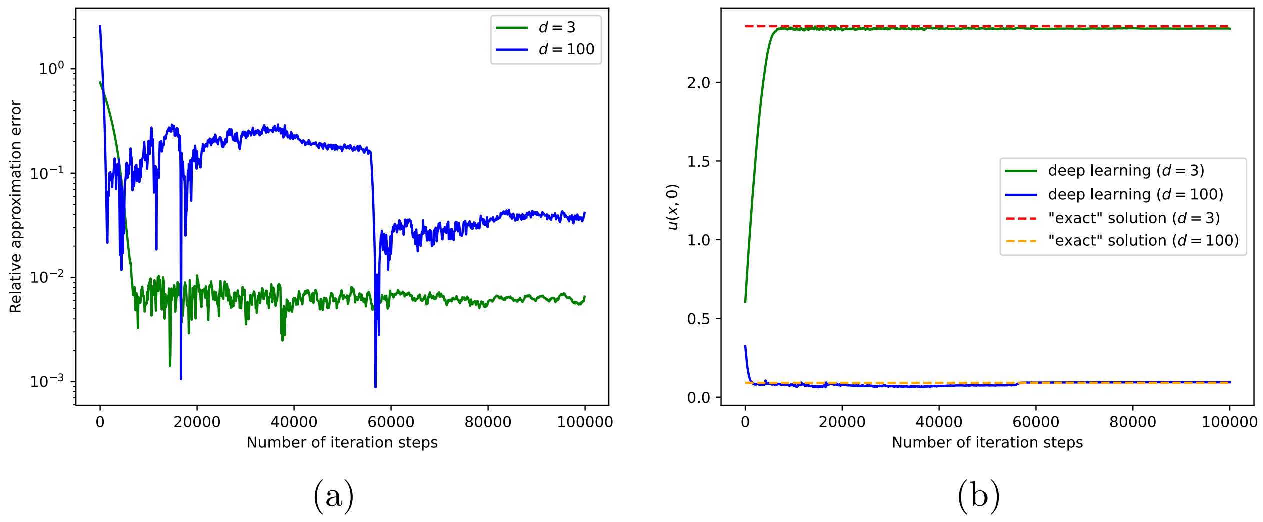

Fig. 6. (a) Relative error for the 3-dimensional (green) and 100-dimensional (blue) nonlinear problems (14) over 10^5 iterations of the proposed algorithm. (b) Convergence behavior of the proposed algorithm for the 3-dimensional (green) and 100-dimensional (blue) nonlinear problems (14), where the “exact” solutions (red and orange) are obtained using the Monte Carlo method. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

3.3. Advective reaction diffusion equations

The advective reaction-diffusion equation extends the advective diffusion equation by incorporating a reaction term (source term), which models a non-mass-conserved system. In such systems, random particles may be generated or consumed either by the system itself or due to external influences. We consider the linear equation

$$\frac{\partial u(x,t)}{\partial t} + \nabla u(x,t) + \frac{1}{2}c_{\beta,d}k_{\beta,d,r}\Delta u(x,t) + \int_{|y|\geq r}[u(x-y,t)-u(x,t)]\nu_\beta(dy) + \lambda u(x,t) = 0 \tag{15}$$

with a given terminal condition $u(x,T) = g(x)$.

For the 3-dimensional case, we choose $x = [0,0,0]^T$, $\beta = 1.1$, $r = 0.1$, $P = 128$, and subnetworks with a width of 128 and 4 hidden layers. For the 100-dimensional case, we choose $x = [0,0,\dots,0]^T$, $\beta = 1.6$, $r = 0.5$, $P = 256$, and subnetworks with a width of 512 and 4 hidden layers. The common parameters are set as $t = 0$, $\lambda = 1$, $T = 1$, $N = 100$, $M = 10^4$, and the terminal condition $g(x) = \frac{1+\sin(|x|)}{1+|x|^2}$. The final results are obtained after 10^5 iterations, with an initial learning rate of 5×10^{-4} and decays by a factor 0.5 every 2×10^4

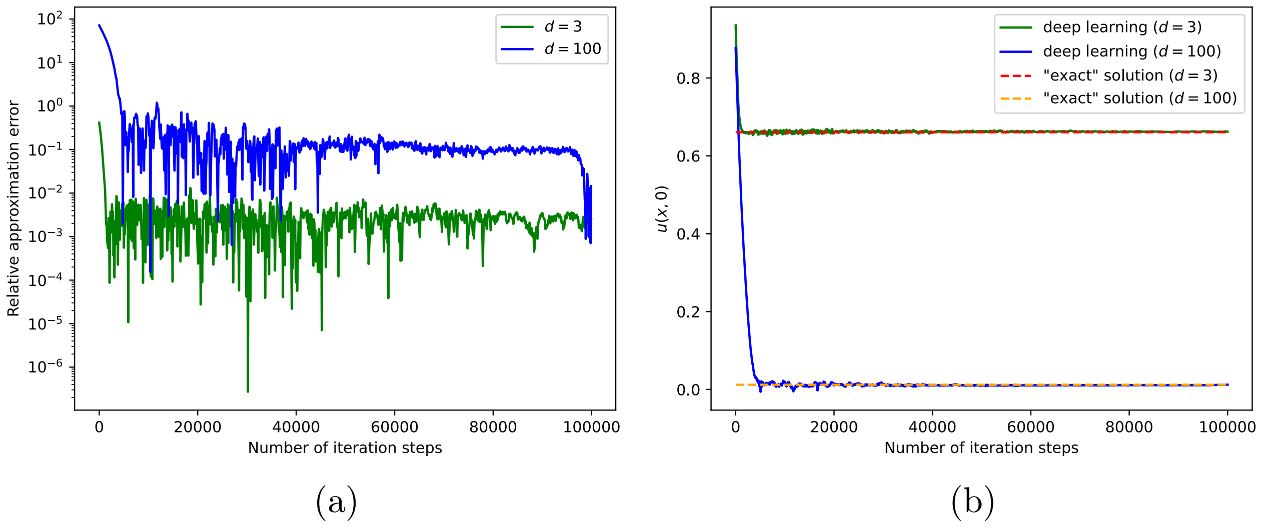

Fig. 7. (a) Relative error for the 3-dimensional (green) and 100-dimensional (blue) nonlinear problems (15) over 10^5 iterations of the proposed deep learning method. (b) Convergence behavior of the proposed algorithm for the 3-dimensional (green) and 100-dimensional (blue) nonlinear problems (15), where the "exact" solutions (red and orange) are computed using the Monte Carlo method. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

steps. Fig. 7 displays the approximate solutions and the relative errors for our deep learning method, where the "exact" solution 0.6606 ($d = 3$) and 0.0121 ($d = 100$) can be computed by approximating the stochastic representation

$$u(x, 0) = \mathbb{E}\left[e^{\lambda T}\frac{1 + \sin(|X(T)|)}{1 + |X(T)|^2}|X(0) = x\right].$$

The proposed deep learning method achieves the relative error of 0.21% for the 3-dimensional problem and 1.67% for the 100-dimensional problem. It converges rapidly (within fewer than 10^4 iterations) and demonstrates strong robustness after convergence.

3.4. Non-linear reaction diffusion equations

In this subsection, we consider the nonlinear reaction diffusion equation, which is described by

$$\frac{\partial u(x,t)}{\partial t} + \frac{1}{2}c_{\beta,d}k_{\beta,d,r}\Delta u(x,t) + \int_{|y|\geq r}[u(x-y,t) - u(x,t)]\nu_\beta(dy) + u(x,t) - u(x,t)^2 = 0 \tag{16}$$

with a given terminal condition $u(x, T) = g(x)$.

For the 3-dimensional case, we choose $x = [0, 0, 0]^T$, $\beta = 1.7$, $r = 0.2$, $P = 128$, and subnetworks with a width of 128 and 4 hidden layers. For the 100-dimensional case, we choose $x = [0, 0, \dots, 0]^T$, $\beta = 1.9$, $r = 0.5$, $P = 256$, and subnetworks with a width of 512 and 4 hidden layers. The common parameters are set as $t = 0$, $T = 0.5$, $N = 50$, $M = 10^4$, and the terminal condition $g(x) = 0.5 - 0.4\sin(|x|/10)$. The final results are obtained after 10^5 iterations, with an initial learning rate of 2×10^{-4} ($d = 3$) and 4×10^{-5} ($d = 100$), which decays by a factor of 0.5 every 2.5×10^4 steps. Fig. 8 displays the approximate solutions and the relative errors for our deep learning method, where the "exact" solution 0.5546 ($d = 3$) and 0.2628 ($d = 100$) can be obtained by the branching diffusion method [24]. Our method achieves the relative error of 0.39% for the 3-dimensional problem and 0.27% for the 100-dimensional problem. It converges rapidly (within fewer than 5×10^3 iterations for the 3-dimensional problem and fewer than 2×10^4 iterations for the 100-dimensional problem) and demonstrates strong robustness after convergence.

3.5. Finite Lévy measures

In this subsection, we consider the case of pure jump processes with several classes of finite Lévy measures. Specifically, we use the developed algorithm to solve fractional PDEs with Lévy measures being uniform distribution, normal distribution, exponential distribution, and Bernoulli distribution. Therefore, one needs to consider the equation

$$\frac{\partial u(x,t)}{\partial t} = \int_{\mathbb{R}^d}[u(x-y,t) - u(x,t)]\nu(dy) \tag{17}$$

with the initial condition $u(x, 0) = u_0(x)$. Here, u is the probability density function of a compound Poisson process with initial value $x_0 \sim u_0$, intensity λ, and jump length $\Delta X \sim \nu$, which can be statistically calculated by simulating particle trajectories using the Monte Carlo method.

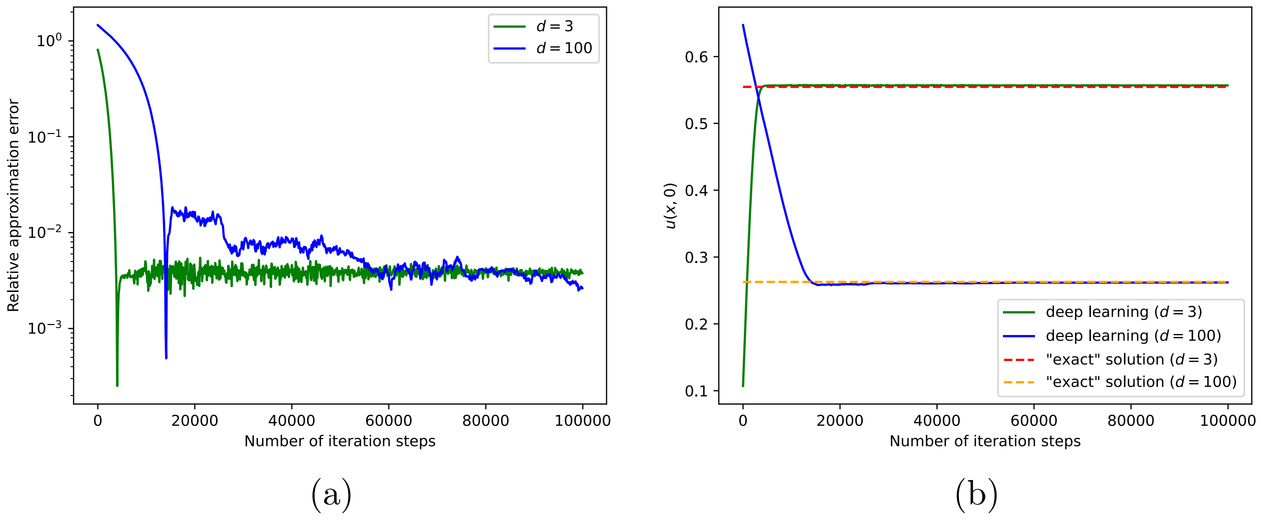

Fig. 8. (a) Relative error for the 3-dimensional (green) and 100-dimensional (blue) nonlinear problems (16) for 10^5 iterations of the developed algorithm. (b) Convergence behavior of the developed algorithm for the 3-dimensional (green) and 100-dimensional (blue) nonlinear problems (16), where the "exact" solutions (red and orange) are obtained via the branching diffusion method [24]. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 1
The relative errors for (13) (Linear) and (16) (Nonlinear) obtained by Algorithm 1 with subnetworks varying in the number of hidden layers.

No. of hidden layers		0	1	2	3	4	5	6
Rel. Error,%	Linear	6.94	24.74	24.82	9.74	6.10	5.20	5.86
	Nonlinear	Nan	Nan	Nan	0.92	0.04	0.19	0.22

Choosing $P = 128$, $T = 1$, $N = 100$, $M = 10^4$, and the initial distribution $u_0(x) = e^{-|x|^2/2}/\sqrt{2\pi}$, we test the performance of the proposed algorithm in solving Eq. (17) with Lévy measures $\nu(dy) = \lambda v(y)dy$ of uniform distribution $v(y) = 1$, $y \in [0, 1]$ with $\lambda = 5$, normal distribution $v(y) = \frac{e^{-|y-1|^2}}{\sqrt{2\pi}}$ with $\lambda = 3$, exponential distribution $v(y) = e^{-y}$, $y > 0$ with $\lambda = 1$, and Bernoulli distribution $v(y) = \begin{cases} 1/3, y = 1, \\ 2/3, y = -2 \end{cases}$ with $\lambda = 5$, respectively. Fig. 9 shows the particle trajectories of these types of pure jump Lévy processes with different jump length distributions simulated by the Monte Carlo method, the probability density obtained by solving the Eq. (17) with the proposed deep learning algorithm, the "exact" solution obtained by statistically calculating the particle positions, and the absolute error of the algorithm. The results of the deep learning algorithm are obtained through 10^4 iterations with a learning rate of 5×10^{-4}, and the "exact" solutions are obtained through statistical calculations of 10^6 independent particle trajectories simulated by the Monte Carlo method.

3.6. *Approximation effect of neural network*

In this section, we investigate three factors that influence the performance of the algorithm: the number of hidden layers in the subnetworks, the number of samples in Monte Carlo integration, and the choice of activation function. Meanwhile, we also test the efficiency of our algorithm and present the test results of the algorithm under various dimensions, including the relative errors at different training steps and computational time costs. Method (10) requires substantial memory (exceeding 64 GB) and computational resources during execution, which exceeds our resource limitations and is impractial for real-world applications. Therefore, we focus on testing the performance of the algorithm corresponding to the tensor network version (11). For each factor, we evaluate both a linear equation and the non-linear Eq. (16) to assess their performance under different scenarios. Except for the specific factors being tested, the parameter settings for each experiment are consistent with those used in the numerical examples in Section 3.

The depth of the neural network is one of the key factors influencing its performance. To evaluate its impact on the proposed algorithm, we vary the number of hidden layers in the subnetworks from 0 to 6 and conduct tests on both the 100-dimensional linear problem (13) and the nonlinear problem (16) (see Table 1). Except for the number of hidden layers of the subnetworks, all other testing parameters remain consistent with those in Sections 3.1 and 3.4, and the network employs the Softsign activation function

$$\mathrm{Softsign}(x) = \frac{x}{1 + |x|}.$$

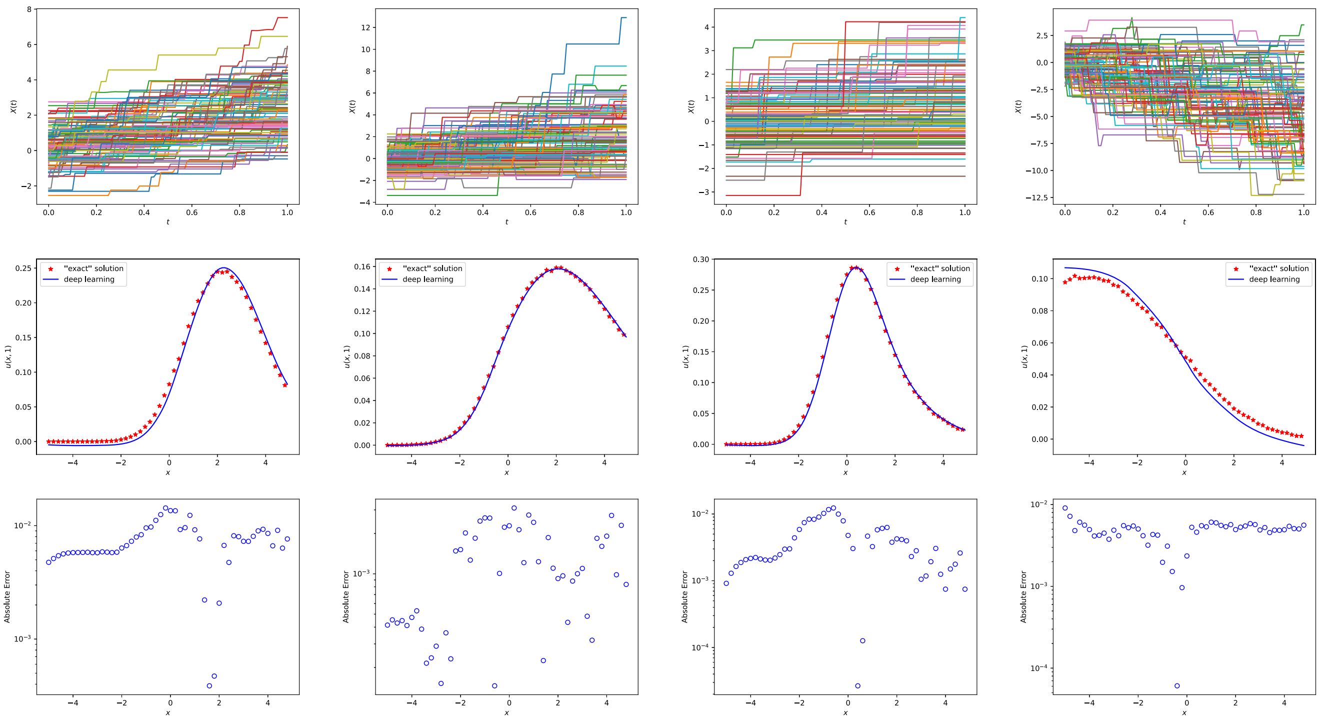

Fig. 9. Particle trajectories (1st row) of pure jump Lévy processes with four different finite measures (1st column: uniform distribution, 2nd column: normal distribution, 3rd column: exponential distribution, 4th column: Bernoulli distribution), plots of the solutions obtained by the proposed deep learning algorithm for solving (17) and the exact solutions (2nd row), and absolute errors (3rd row). The results of the proposed deep learning algorithm are obtained through 10^4 iterations with a learning rate of 5×10^{-4}, and the "exact" solutions are obtained through statistical calculations of 10^6 independent particle trajectories simulated by the Monte Carlo method.

Table 2
The relative errors for (15) (Linear) and (16) (Nonlinear) obtained by Algorithm 1, utilizing the non-local integral approximation method (12) with varying numbers of Monte Carlo samples.

No. of Monte Carlo samples		10^0	10^1	10^2	10^3
Rel. Error,%	Linear	7.58	7.51	9.99	1.22
	Nonlinear	0.83	0.20	0.08	0.34

For the linear problem (13), the relative error from the tests initially increase, then decrease, and finally shows a slight increase as the number of hidden layers increases from 0 (linear) to 2 (shallow) and then to 6 (deep). For the nonlinear problem (16), the training becomes unstable when the number of hidden layers is fewer than 3, which is attributed to the inherent instability of the problem itself. The relative error also demonstrates an initial decrease followed by a slight increase as the depth increases.

In this paper, we utilize the Monte Carlo method to approximate high-dimensional integrals (9) and design a simplified version of a tensor neural network (11) to reduce computational costs. To investigate the impact of the number of Monte Carlo samples on the algorithm's performance, we fix the number of hidden layers in the subnetwork as 4, select the Softsign activation function, and vary the number of Monte Carlo samples from 10^0 to 10^3 (see Table 2). We evaluate the algorithm's performance on the 100-dimensional linear problem (15) and the nonlinear problem (16). The results indicate that for the linear problem, increasing the number of Monte Carlo samples improves accuracy. However, the number of samples does not significantly affect the solution of the nonlinear problem.

The choice of activation function significantly influences the performance of the algorithm. To evaluate this impact, we select five common activation functions and test their performance on the 100-dimensional linear problem (14) and the nonlinear problem (16) (see Table 3). The results demonstrate that, in terms of both accuracy and stability, the Softsign activation function is the optimal choice. During testing, we observe that a large truncation of the jump length $\Delta L(t_k)$ (with a default truncation length of 100 in this paper) is necessary to prevent value overflow and ensure effective training.

To demonstrate the efficiency of the algorithm, we apply the proposed method to solve problems in various dimensions and present the convergence behaviour at different iteration steps along with the computational time. Specifically, we test both the linear problem (15) and the nonlinear problem (16) across five different dimensions ($d = 3, 10, 20, 50, 100$). The simulation results include the relative errors at different training steps compared to the "exact" solution obtained via the Monte Carlo method, as well as the

Table 3
The relative errors for (14) (Linear) and (16) (Nonlinear) obtained by Algorithm 1 with subnetworks employing different activation functions.

Activation functions		ReLU	SiLU	Sigmoid	Tanh	Softsign
Rel. Error,%	Linear	0.77	9.15	1.58	25.64	2.89
	Nonlinear	Nan	Nan	Nan	0.06	0.27

Table 4
The relative errors and GPU time consumption per iteration step for (15) (Linear) and (16) (Nonlinear) obtained by Algorithm 1 under different dimensions and different numbers of iterations.

d	Linear			Nonlinear		
	Step	Rel. Error,%	GPU,s	Step	Rel. Error,%	GPU,s
3	1×10^3	1.29	0.48	1×10^3	57.76	0.25
	1×10^4	0.37		1×10^4	4.32	
	5×10^4	0.34		5×10^4	0.59	
	1×10^5	0.24		1×10^5	0.57	
10	1×10^3	206.28	0.45	1×10^3	38.22	0.23
	1×10^4	3.37		1×10^4	0.21	
	5×10^4	0.08		5×10^4	0.19	
	1×10^5	0.01		1×10^5	0.22	
20	1×10^3	1188.16	0.46	1×10^3	113.81	0.24
	1×10^4	2.31		1×10^4	45.21	
	5×10^4	1.71		3×10^4	0.39	
	8×10^4	0.42		5×10^4	0.40	
	1×10^5	0.45		1×10^5	0.47	
50	1×10^3	63.04	0.48	1×10^3	23.23	0.24
	1×10^4	6.77		1×10^4	0.14	
	8×10^4	2.51		5×10^4	0.47	
	1×10^5	0.12		1×10^5	0.53	
100	1×10^3	5837.79	0.45	1×10^3	238.25	0.22
	1×10^4	57.58		1×10^4	121.12	
	7×10^4	4.25		5×10^4	0.23	
	1×10^5	2.01		1×10^5	0.32	

GPU (NVIDIA GeForce RTX 4090) time consumed per training step. For the problems with $d = 3$, we choose $P = 64$ and sub-networks with a width of 128 and 4 hidden layers. For the problems with $3 < d < 50$, we choose $P = 128$ and sub-networks with a width of 256 and 4 hidden layers. For the problems with $d \geq 50$, we choose $P = 256$ and sub-networks with a width of 512 and 4 hidden layers. For linear problem, we set $\beta = 1.3$, $r = 0.5$, with an initial learning rate of 5×10^{-4}, which decays by a factor 0.5 every 2×10^4 iterations. For nonlinear problems, we use $\beta = 1.9$, $r = 0.5$, with initial learning rates of 4×10^{-5} (for $d \geq 50$) and 2×10^{-4} (for $d < 50$), which decay by a factor 0.5 every 2.5×10^4 iterations. The results (see Table 4) show that the training accuracy of the algorithm varies with the increase of dimensions, and due to parallel training, the GPU time is roughly similar.

Remark 1. To theoretically ensure the effectiveness of the proposed algorithm, a good way is to get the posterior error estimate like [25]

$$\|X - \hat{X}\|_X + \|Y - \hat{Y}\|_Y + \|Z - \hat{Z}\|_Z + \|U - \hat{U}\|_U \leq C\big[\Delta t + \mathbb{E}\big[|g(\hat{X}(T)) - \hat{Y}(T)|^2\big]\big],$$

where $(X, Y, Z, U) = (X(t), Y(t), Z(t), U(t, y))$ is the exact solution of the FBSDE and $(\hat{X}, \hat{Y}, \hat{Z}, \hat{U}) = (\hat{X}(t), \hat{Y}(t), \hat{Z}(t), \hat{U}(t, y))$ is the numerical solution of the proposed deep learning algorithm. In other words, it is necessary to find suitable error norms $\|\cdot\|_X$, $\|\cdot\|_Y$, $\|\cdot\|_Z$, and $\|\cdot\|_U$ such that it is controlled by the step size Δt and the loss function $\mathbb{E}\big[|g(\hat{X}(T)) - \hat{Y}(T)|^2\big]$.

4. Conclusion

In this paper, we consider a class of PDEs associated with jump processes with fat-tailed Lévy measure, which involve nonlocal singular integral operator. To develop deep learning method for solving such equations, we first approximate the nonlocal singular integral operator and derive the BSDEs satisfied by the solutions of the approximated equations. These are then discretized to obtain a corresponding deep learning algorithm. To reduce the computational and memory costs associated with high-dimensional numerical integration, we propose a structure-preserving, simplified version of tensor neural networks.

The effectiveness of the algorithm is demonstrated by solving four types of PDEs: the diffusion equation with fractional Laplacian; the advective diffusion equation with fractional Laplacian; the advective reaction diffusion equation with fractional Laplacian; and the nonlinear reaction diffusion equation with fractional Laplacian. The proposed algorithm is verified to be applicable for solving fractional equations with finite generalized Lévy measures. The developed method reaches the relative error of $\mathcal{O}(10^{-3})$ for low-dimensional problems, and $\mathcal{O}(10^{-2})$ for high-dimensional problems. Through extensive numerical experiments, we explore the influence of hidden layer depth, Monte Carlo sampling size, and activation functions on the algorithm's performance. The results indicate that the algorithm exhibits the highest stability when deeper hidden layers, larger Monte Carlo sample size, and the $\mathrm{Softsign}$ activation function are employed. The efficiency of the algorithm in solving 3D, 10D, 20D, 50D, and 100D problems is tested, including the number of convergence steps and GPU time. The further error analysis of the algorithm will be addressed in future work. This work can be naturally generalized to problems with arbitrary Lévy measures, including those obtained from data, enabling the application of fractional-order PDEs to high-dimensional complex problems, especially those with long-range flight.

CRediT authorship contribution statement

Kamran Arif: Writing – original draft, Methodology, Investigation, Formal analysis, Conceptualization; **Guojiang Xi:** Funding acquisition, Data curation; **Heng Wang:** Writing – original draft, Methodology, Investigation, Formal analysis, Conceptualization; **Weihua Deng:** Writing – review & editing, Supervision, Project administration, Investigation, Funding acquisition, Conceptualization.

Data availability

Data will be made available on request.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was supported by the National Natural Science Foundation of China under Grant No. 12225107, the Major Science and Technology Projects in Gansu Province-Leading Talents in Science and Technology under Grant No. 23ZDKA0005, the Innovative Groups of Basic Research in Gansu Province under Grant No. 22JR5RA391, the Tianyuan Fund for Mathematics of the National Natural Science Foundation of China No. 12426105, the Gansu Province Postgraduate Innovation Star Program No. 2025CXZX-151, the Major State Research Development Program of Sichuan Province No. 2024YFG0004, the Transfer and Commercialization of Scientific and Technological Achievements Demonstration Program of Sichuan Province No. 2024ZHCG0169, the Opening Foundation of Agile and Intelligent Computing Key Laboratory of Sichuan Province, and Lanzhou Talent Work Special Fund.

Appendix A. Proof of Lemma 1

Proof. Define a sequence of stopping times recursively by $T_0 = 0$ and $T_n = \inf\{t > T_{n-1}; |X(t) - X(T_{n-1})| \neq 0\}$. For each $t > 0$, we have

$$\begin{aligned}
u(X(t),t) - u(X(0),0) &= \sum_{j=0}^{\infty} \left[u(X(t\wedge T_{j+1}), t\wedge T_{j+1}) - u(X(t\wedge T_j), t\wedge T_j)\right] \\
&= \sum_{j=0}^{\infty} \left[u(X(t\wedge T_{j+1}-), t\wedge T_{j+1}) - u(X(t\wedge T_j), t\wedge T_j)\right] + \sum_{j=0}^{\infty} \left[u(X(t\wedge T_{j+1}), t\wedge T_{j+1}) - u(X(t\wedge T_{j+1}-), t\wedge T_{j+1})\right] \\
&= \int_0^t \left(\frac{\partial u(X(s),s)}{\partial t} + \frac{1}{2} c_{\alpha,d} k_{\alpha,d,r} \Delta u(X(s),s) + (\mu\cdot\nabla u)(X(s),s)\right) ds + \sqrt{c_{\alpha,d} k_{\alpha,d,r}} \int_0^t \nabla u(X(s),s)\cdot dB(s) \\
&\quad + \int_0^t \int_{|y|\geq r} [u(X(s-)+y,s) - u(X(s-),s)] J(dy\times ds).
\end{aligned}$$

□

Appendix B. Proof of Theorem 1

Proof. Applying Lemma 1 to $u(X(s),s)$ between $s=t$ and $s=T$, one can get

$$\begin{aligned}
u(X(t),t)-g(X(T)) = &-\int_t^T\left(\frac{\partial u(X(s),s)}{\partial t}+\frac{1}{2}c_{\alpha,d}k_{\alpha,d,r}\Delta u(X(s),s)+(\mu\cdot\nabla u)(X(s),s)\right)ds-\sqrt{c_{\alpha,d}k_{\alpha,d,r}}\int_t^T\nabla u(X(s),s)\cdot dB(s)\\
&-\int_t^T\int_{|y|>r}[u(X(s-)-y,s)-u(X(s-),s)]J(dy\times ds)\\
=&\int_t^T f(s,X(s),u(X(s),s))ds-\sqrt{c_{\alpha,d}k_{\alpha,d,r}}\int_t^T\nabla u(X(s),s)\cdot dB(s)\\
&-\int_t^T\int_{|y|>r}[u(X(s-)+y,s)-u(X(s-),s)]\widetilde{J}(dy\times ds).
\end{aligned}$$

□

Appendix C. The difference between the solutions of (1) and (3)

We assume that the solutions to (1) and (3) are $u,\tilde{u}\in H^3(\mathbb{R}^d)\times C^1([0,T])$, and $\exists C\geq 0$ such that $|f(t,x,u(x,t))-f(t,x,\tilde{u}(x,t))|\leq C|e(x,t)|$ and $|\nabla\cdot\mu(x,t)|\leq C$, $\forall(t,x)\in[0,T]\times\mathbb{R}^d$, where $e(x,t)=u(x,t)-\tilde{u}(x,t)$.

First, we estimate the truncation error

$$\begin{aligned}
\left|R_r[u](x,t)\right| &= \left|\int_{|y|<r}\left[u(x-y,t)-u(x,t)+y^T\nabla u(x,t)\right]\nu_\beta(dy)-\frac{1}{2}c_{\beta,d}k_{\beta,d,r}\Delta u(x,t)\right|\\
&=\left|\int_{|y|<r}\frac{-\frac{1}{6}u^{(3)}(x,t)(y,y,y)+\mathcal{O}(|y|^4)}{|y|^{\beta+d}}dy\right|\\
&\leq\tilde{C}|u^{(3)}(x,t)|\int_{|y|<r}|y|^{3-\beta-d}dy\\
&=\tilde{C}|u^{(3)}(x,t)|S_{d-1}\int_0^r l^{2-\beta}dl\\
&=\bar{C}|u^{(3)}(x,t)|r^{3-\beta}.
\end{aligned}$$

Next, we estimate the L^2 error using the energy method. By doing time transformation $t\to T-t$, converting (1) and (3) into the initial value problems, then one can get the error equation

$$\frac{\partial e(x,t)}{\partial t}=(\mu\cdot\nabla e)(x,t)+\frac{1}{2}c_{\beta,d}k_{\beta,d,r}\Delta e(x,t)+\int_{|y|\geq r}[e(x-y,t)-e(x,t)]\nu_\beta(dy)+f(t,x,u(x,t))-f(t,x,\tilde{u}(x,t))+R_r[u](x,t).$$

Multiplying both sides of the error equation by e and integrating over $\mathbb{R}^d$, we obtain

$$\begin{aligned}
\frac{1}{2}\frac{d}{dt}\|e(\cdot,t)\|^2_{L^2(\mathbb{R}^d)}=&\int_{\mathbb{R}^d}e(x,t)(\mu\cdot\nabla e)(x,t)dx+\frac{1}{2}c_{\beta,d}k_{\beta,d,r}\int_{\mathbb{R}^d}e(x,t)\Delta e(x,t)dx+\int_{\mathbb{R}^d}e(x,t)\int_{|y|\geq r}[e(x-y,t)-e(x,t)]\nu_\beta(dy)dx\\
&+\int_{\mathbb{R}^d}e(x,t)[f(t,x,u(x,t))-f(t,x,\tilde{u}(x,t))]dx+\int_{\mathbb{R}^d}e(x,t)R_r[u](x,t)dx\\
=&\ \mathrm{I}+\mathrm{II}+\mathrm{III}+\mathrm{IV}+\mathrm{V}.
\end{aligned}$$

Then, one can estimate these five terms respectively as

$$\begin{aligned}
\mathrm{I}&=\int_{\mathbb{R}^d}e(x,t)(\mu\cdot\nabla e)(x,t)dx\\
&=\frac{1}{2}\int_{\mathbb{R}^d}\mu(x,t)\cdot\nabla[e(x,t)]^2dx\\
&=-\frac{1}{2}\int_{\mathbb{R}^d}\nabla\cdot\mu(x,t)[e(x,t)]^2dx\leq C\|e(\cdot,t)\|^2_{L^2(\mathbb{R}^d)},\\
\mathrm{II}&=\frac{1}{2}c_{\beta,d}k_{\beta,d,r}\int_{\mathbb{R}^d}e(x,t)\Delta e(x,t)dx=-\frac{1}{2}c_{\beta,d}k_{\beta,d,r}\int_{\mathbb{R}^d}|\nabla e(x,t)|^2dx\leq 0,\\
\mathrm{III}&=\int_{\mathbb{R}^d}e(x,t)\int_{|y|\geq r}[e(x-y,t)-e(x,t)]\nu_\beta(dy)dx\\
&=\frac{1}{2}\int_{\mathbb{R}^d}\int_{|y|\geq r}\left[-[e(x-y,t)]^2+2e(x-y,t)e(x,t)-[e(x,t)]^2\right]\nu_\beta(dy)dx\\
&=-\frac{1}{2}\int_{\mathbb{R}^d}\int_{|y|\geq r}[e(x-y,t)-e(x,t)]^2\nu_\beta(dy)dx\leq 0,
\end{aligned}$$

$$\begin{aligned}\mathrm{IV} &= \int_{\mathbb{R}^d} e(x,t)[f(t,x,u(x,t)) - f(t,x,\tilde{u}(x,t))]dx \\ &\le \int_{\mathbb{R}^d} |e(x,t)||f(t,x,u(x,t)) - f(t,x,\tilde{u}(x,t))|dx \\ &\le C\|e(\cdot,t)\|^2_{L^2(\mathbb{R}^d)},\end{aligned}$$

and

$$\mathrm{V} = \int_{\mathbb{R}^d} e(x,t)R_r[u](x,t)dx \le \|R_r[u](\cdot,t)\|_{L^2(\mathbb{R}^d)}\|e(\cdot,t)\|_{L^2(\mathbb{R}^d)}.$$

Combining all above terms result in

$$\frac{d}{dt}\|e(\cdot,t)\|^2_{L^2(\mathbb{R}^d)} \le 4C\|e(\cdot,t)\|^2_{L^2(\mathbb{R}^d)} + 2\|R_r[u](\cdot,t)\|_{L^2(\mathbb{R}^d)}\|e(\cdot,t)\|_{L^2(\mathbb{R}^d)}.$$

Assuming $\|e(\cdot,t)\|^2_{L^2(\mathbb{R}^d)} \neq 0$, $\forall t \in [0,T]$, one can obtain

$$\frac{d}{dt}\|e(\cdot,t)\|_{L^2(\mathbb{R}^d)} \le 2C\|e(\cdot,t)\|_{L^2(\mathbb{R}^d)} + \|R_r[u](\cdot,t)\|_{L^2(\mathbb{R}^d)}.$$

According to the Grönwall's theorem, one can get

$$\|e(\cdot,t)\|_{L^2(\mathbb{R}^d)} \le L_2(t)r^{3-\beta}, \tag{C.1}$$

where $L_2(t) = \frac{\bar{C}}{2C}\left(e^{2Ct}-1\right)\sup_{0\le s\le t}\|u(\cdot,s)\|_{H^3(\mathbb{R}^d)}$.

Finally, we assume $u, \tilde{u} \in C^3(\mathbb{R}^d)$, and estimate the L^∞ error. Assuming that $x^*(t)$ is the maximum point of e at time t and $e(x^*(t),t) \ge 0$, then

$$\begin{aligned}\frac{de(x^*(t),t)}{dt} &= \nabla e(x^*(t),t)\cdot dx^*(t) + \frac{\partial e(x^*(t),t)}{\partial t} \\ &\le f(t,x^*(t),u(x^*(t),t)) - f(t,x^*(t),\tilde{u}(x^*(t),t)) + R_r[u](x^*(t),t) \\ &\le Ce(x^*(t),t) + |R_r[u](x^*(t),t)|,\end{aligned}$$

since $\nabla e(x^*(t),t) = 0$, $\Delta e(x^*(t),t) \le 0$, and

$$\int_{|y|\ge r}\left[e(x^*(t)-y,t) - e(x^*(t),t)\right]\nu_\beta(dy) \le 0.$$

Similarly, one can obtain that when $x^*(t)$ is a minimum point of e and $e(x^*(t),t) < 0$, there exists

$$\frac{d(-e(x^*(t),t))}{dt} \le C(-e(x^*(t),t)) + |R_r[u](x^*(t),t)|.$$

That is

$$\frac{d\|e(\cdot,t)\|_{L^\infty(\mathbb{R}^d)}}{dt} \le C\|e(\cdot,t)\|_{L^\infty(\mathbb{R}^d)} + \|R_r[u](\cdot,t)\|_{L^\infty(\mathbb{R}^d)}.$$

According to the Grönwall theorem, one can get

$$\|e\|_{L^\infty(\mathbb{R}^d)}(t) \le L_\infty(t)r^{3-\beta}, \tag{C.2}$$

where $L_\infty(t) = \frac{\bar{C}}{C}\left(e^{Ct}-1\right)\sup_{0\le s\le t}\|u(\cdot,s)\|_{C^3(\mathbb{R}^d)}$.

References

[1] Y. Chen, X.D. Wang, W.H. Deng, Tempered fractional Langevin-Brownian motion with inverse β-stable subordinator, J. Phys. A 51 (49) (2018) 495001. https://doi.org/10.1088/1751-8121/aae8b3
[2] A. Cheung, P. Cramer, Structural basis of RNA polymerase II backtracking, arrest and reactivation, Nature 471 (2011) 249–253. https://doi.org/10.1038/nature09785
[3] B. Wang, S.M. Anthony, S.C. Bae, S. Granick, Anomalous yet Brownian, Proc. Natl. Acad. Sci. 106 (36) (2009) 15160–15164. https://doi.org/10.1073/pnas.0903554106
[4] D. Wang, D.A. Bushnell, X. Huang, K.D. Westover, M. Levitt, R.D. Kornberg, Structural basis of transcription: backtracked RNA polymerase II at 3.4 angstrom resolution, Science 324 (5931) (2009) 1203–1206. https://doi.org/10.1126/science.1168729
[5] R. Metzler, J. Klafter, The random walk's guide to anomalous diffusion: a fractional dynamics approach, Phys. Rep. 339 (2000). https://doi.org/10.1016/S0370-1573(00)00070-3
[6] D. Applebaum, Lévy Processes and Stochastic Calculus, Cambridge Studies in Advanced Mathematics, Cambridge University Press, 2 edition, 2009.
[7] W.H. Deng, B.Y. Li, W.Y. Tian, P.W. Zhang, Boundary problems for the fractional and tempered fractional operators, Multiscale Model. Sim. 16 (1) (2018) 125–149. https://doi.org/10.1137/17M1116222
[8] M.M. Meerschaert, A. Sikorskii, Stochastic Models for Fractional Calculus, De Gruyter, Berlin, Boston, Berlin, Boston, 2019. https://doi.org/10.1515/9783110560244
[9] W.Y. Tian, H. Zhou, W.H. Deng, A class of second order difference approximations for solving space fractional diffusion equations, Math. Comp. 84 (294) (2015) 1703–1727. https://doi.org/10.1090/S0025-5718-2015-02917-2
[10] G. Acosta, J.P. Borthagaray, A fractional Laplace equation: regularity of solutions and finite element approximations, SIAM J. Numer. Anal. 55 (2) (2017) 472–495. https://doi.org/10.1137/15M1033952

[11] W. E, C. Ma, L. Wu, S. Wojtowytsch, Towards a mathematical understanding of neural network-based machine learning: what we know and what we don't, CSIAM T. Appl. Math. 1 (4) (2020) 561–615. https://doi.org/10.4208/csiam-am.SO-2020-0002
[12] M. Raissi, Forward-backward stochastic neural networks: deep learning of high-dimensional partial differential equations, 2018, arXiv:1804.07010
[13] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707. https://doi.org/10.1016/j.jcp.2018.10.045
[14] G. Pang, L. Lu, G.E. Karniadakis, fPINNs: fractional physics-informed neural networks, SIAM J. Sci. Comput. 41 (4) (2019) A2603–A2626. https://doi.org/10.1137/18M1229845
[15] Z. Hu, K. Shukla, G.E. Karniadakis, K. Kawaguchi, Tackling the curse of dimensionality with physics-informed neural networks, Neural Netw. 176 (2024) 106369. https://doi.org/10.1016/j.neunet.2024.106369
[16] W. E, B. Yu, The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems, Commun. Math. Stat. 6 (1) (2018). https://doi.org/10.1007/s40304-018-0127-z
[17] Y. Zang, G. Bao, X. Ye, H. Zhou, Weak adversarial networks for high-dimensional partial differential equations, J. Comput. Phys. 411 (2020) 109409. https://doi.org/10.1016/j.jcp.2020.109409
[18] J. Han, A. Jentzen, W. E, Solving high-dimensional partial differential equations using deep learning, Proc. Natl. Acad. Sci. 115 (34) (2018) 8505–8510. https://doi.org/10.1073/pnas.1718942115
[19] E. Pardoux, S. Peng, Backward stochastic differential equations and quasilinear parabolic partial differential equations, in: B.L. Rozovskii, R.B. Sowers (Eds.), Stochastic Partial Differential Equations and Their Applications, Springer Berlin Heidelberg, Berlin, Heidelberg, 1992, pp. 200–217.
[20] L. Lu, H. Guo, X. Yang, Y. Zhu, Temporal difference learning for high-dimensional PIDEs with jumps, SIAM J. Sci. Comput. 46 (4) (2024) C349–C368. https://doi.org/10.1137/23M1584538
[21] Ł Delong, Backward Stochastic Differential Equations with Jumps and Their Actuarial and Financial Applications, Springer London, 2013. https://doi.org/10.1007/978-1-4471-5331-3
[22] W. Yifan, H. Xie, J. Pengzhan, Tensor neural network and its numerical integration, J. Comput. Math. 42 (6) (2024) 1714–1742. https://doi.org/10.4208/jcm.2307-m2022-0233
[23] D.P. Kingma, J. Ba, Adam: a method for stochastic optimization, 2017, arXiv:1412.6980
[24] E. Gobet, Monte-Carlo Methods and Stochastic Processes: From Linear to Non-Linear, Chapman & Hall/CRC, 2016, pp. 212–214.
[25] J. Han, J. Long, Convergence of the deep BSDE method for coupled FBSDEs, Probab. Uncertain. Quant. Risk 5 (1) (2020) 5. https://doi.org/10.1186/s41546-020-00047-w