Journal of Computational Physics 472 (2023) 111690

Contents lists available at ScienceDirect

Journal of Computational Physics

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

A non-gradient method for solving elliptic partial differential equations with deep neural networks

Yifan Peng [b], Dan Hu [a,*], Zin-Qin John Xu [a]

[a] School of Mathematical Sciences, Institute of Natural Sciences, and MOE-LSC, Shanghai Jiao Tong University, Shanghai, 200240, China
[b] Zhiyuan College, Shanghai Jiao Tong University, Shanghai, 200240, China

ARTICLE INFO

Article history:
Received 4 March 2022
Received in revised form 31 August 2022
Accepted 9 October 2022
Available online 21 October 2022

Keywords:
Elliptic partial differential equations
Deep neural networks
High dimension
Nongradient method

ABSTRACT

Deep learning has achieved wide success in solving Partial Differential Equations (PDEs), with particular strength in handling high dimensional problems and parametric problems. Nevertheless, there is still a lack of a clear picture on the designing of network architecture and the training of network parameters. In this work, we developed a non-gradient framework for solving elliptic PDEs based on Neural Tangent Kernel (NTK): 1. ReLU activation function is used to control the compactness of the NTK so that solutions with relatively high frequency components can be well expressed; 2. Numerical discretization is used for differential operators to reduce computational cost; 3. A dissipative evolution dynamics corresponding to the elliptic PDE is used for parameter training instead of the gradient-type descent of a loss function. The dissipative dynamics can guarantee the convergence of the training process while avoiding employment of loss functions with high order derivatives. It is also helpful for both controlling of kernel property and reduction of computational cost. Numerical tests have shown excellent performance of the non-gradient method.

1. Introduction

Partial differential equations (PDEs) have prevalent applications in many fields including physics, chemistry, engineering, and finance [1]. Standard numerical methods to compute solutions of PDEs are based on discretization of solutions, such as finite difference method [2], finite element method [3], and finite volume method [4]. Although these methods have achieved widespread success in the past decades, they suffer from the curse of dimensionality when the dimension of the PDE is high (e.g., greater than four) [5].

Neural networks are powerful in handling high dimensional problems. In recent years, deep learning has achieved remarkable success in many artificial intelligence tasks, such as computer vision and natural language processing [1]. Deep learning has also been well used in solving high dimensional PDEs [6–11], including optimal control problems [12–14], variational problems [15], and inverse problems [16,17]. Traditional technologies and insights have also been combined with deep learning for solving particular PDE problems: Neural parameters are set in Fourier domain to deal with long-range interaction among particles [18]; Finite element method is combined with neural network to estimate the required depth and width for solution representation [19]; Weak formulations are used to solve high-dimensional PDEs defined on arbitrary do-

* Corresponding author.
E-mail address: hudan80@sjtu.edu.cn (D. Hu).

https://doi.org/10.1016/j.jcp.2022.111690
0021-9991/

mains [20]; Operator splitting is used for parameter training [21]; Viscosity solutions of Hamilton-Jacobi partial differential equations can be obtained with specific neural network architecture [22,23].

The standard framework of deep learning is to construct a loss function and use gradient type method to optimize the loss function. For solving PDEs, widely-used loss functions include the mean square formulation [24–26] and the Ritz formulation for PDEs in a variational form [15]. In the mean square formulation, the loss is simply the mean square error of the PDE and its boundary conditions. The corresponding neural network is also called physics-informed neural network (PINN) [25,27–30]. Under this formulation, balance of the boundary conditions or initial conditions [31,32] and the convergence behavior of the solution [33,34] are further explored in detail. It has also been shown by the frequency principle that it is very difficult for deep neural networks to learn high-frequency components of the solution [35–37].

In recent years, studies on Neural Tangent Kernel (NTK) [38–42] have provided theoretical understandings on the evolution of the function represented by deep neural networks with a particular parameter initialization (NTK parametrization). In this case, the function represented by the neural network can be effectively approximated by its first-order Taylor expansion at its initial parameters [42] and the evolution dynamics of the output function is approximately linear. NTK theories have also been applied in understanding deep learning for PDEs [32]. This study provided useful understanding on the training dynamics and spectral bias based on the behavior of the kernel. Although feature learning can also be important for deep learning under other initialization settings, it is still far from a clear understanding on the performance and training dynamics [43].

In this work, we develop an efficient non-gradient framework to solve elliptic PDEs using deep neural networks with NTK initialization:

1. Instead of introducing loss functions, the residual of the PDEs are directly used in training the deep neural networks. The non-gradient training dynamics can be effectively described by corresponding integral-differential equations, which are similar to the effective training dynamics obtained from the gradient decent dynamics based on the mean-square or Ritz loss. In particular, the kernel of the integral-differential equation is simply the neural tangent kernel for the non-gradient training dynamics. The dissipation of the non-gradient training dynamics leads to the convergence of the training dynamics.
2. In order to control the spectral bias, sufficient locality of the neural tangent kernel is required for function approximation and fast convergence of the training dynamics. By studying the property of the training kernels, we select ReLU as the activation function in the deep neural networks so as to control the locality of the support of the training kernel.
3. Numerical discretizations for differential operators are used to compute the residuals of the PDEs. This approach has twofold benefits: It avoids problems introduced by the low smoothness of ReLU function and reduces the computational cost for each training step.

In our numerical tests, the non-gradient method has been successfully employed in solving high-dimensional elliptic equations including Poisson equation, advection-diffusion equation, and a nonlinear elliptic equation obtained from the steady state Fokker-Planck equation.

The paper is organized as below: In Section 2, we discuss the basic frameworks to solve partial differential equations and the corresponding training dynamics. Then we introduce our non-gradient method and discuss the detailed techniques of the method. In Section 3, we show basic properties of the Neural Tangent Kernel (NTK). By visualizing the NTKs and the mean square kernels (MSKs), we show the advantage of our non-gradient method. In Section 4, we use numerical examples to show the potential of the non-gradient method. Finally, conclusions and discussions are included in Section 5.

2. Basic framework

In this work, we use the following general form to represent elliptic PDEs

$$\mathcal{A}u(x) = g(x), x \in \Omega \tag{1}$$

where $\mathcal{A}$ is a linear or non-linear elliptic operator. Without loss of generality, we assume that the real parts of the eigenvalues of $\mathcal{A}$ are non-negative. In particular, for second-order linear elliptic PDEs, we have

$$\mathcal{A}u(x) = -\nabla \cdot (D(x) \cdot \nabla u(x)) + \mathbf{b}(x) \cdot \nabla u(x) + c(x)u(x) = g(x), x \in \Omega, \tag{2}$$

where the coefficients satisfy $D(x) \geq 0$ and $c(x) \geq 0$.

2.1. Loss functional for elliptic PDEs

The mean square loss functional [25] of the equation is simply

$$\mathcal{L}(\theta) = \int_{\Omega} |\mathcal{A}f_\theta(x) - g(x)|^2 \mu(dx), \tag{3}$$

where θ represents the parameters of the neural network and μ represents the sampling distribution. For linear elliptic PDEs with $\mathbf{b}(x) = \mathbf{0}$, the Ritz loss [15]

$$\mathcal{L}(\theta) = \int_{\Omega} \frac{1}{2}\nabla_x f_\theta(x)^T D(x)\nabla_x f_\theta(x) + \frac{1}{2}c(x) f_\theta(x)^2 - g(x) f_\theta(x)\mu(dx) \tag{4}$$

can also be used in training deep neural networks. Boundary conditions can also be satisfied by introducing mean square loss of boundary conditions. In the following, we neglect the effects of boundary conditions and discuss the effective training dynamics.

2.2. Evolution dynamics of the output function for GD training

The gradient descent dynamics based on the corresponding loss functionals is

$$\theta^{t+1} - \theta^t = -\eta\nabla_\theta \mathcal{L}, \tag{5}$$

where η is the learning rate. By regarding η as the time step, the continuum limit dynamics for $\eta \to 0$ is

$$\frac{\partial \theta^t}{\partial t} = -\nabla_\theta \mathcal{L}, \tag{6}$$

and the evolution of the output function f_θ satisfies

$$\frac{\partial f_{\theta^t}(x)}{\partial t} = -\nabla_\theta f_{\theta^t}(x)^T \nabla_\theta \mathcal{L}. \tag{7}$$

Suppose the sample points are uniformly distributed in Ω. For different types of loss functionals of linear elliptic PDEs, Eq. (7) satisfies the general form

$$\frac{\partial f_{\theta_t}(x)}{\partial t} = -\int_{\Omega} K(x, y, t)(\mathcal{A} f_{\theta_t}(y) - g(y))\mu(dy), \tag{8}$$

where the kernel function $K(x, y, t)$ is given as follows:

1. For the deep Ritz method,
$$K(x, y, t) = \nabla_\theta f_{\theta^t}(x)^T \nabla_\theta f_{\theta^t}(y) = \Theta_t(x, y), \tag{9}$$
where $\Theta_t(x, y)$ is exactly the Neural Tangent Kernel (NTK) [38].
2. For the mean square formulation,
$$K(x, y, t) = \nabla_\theta f_{\theta^t}(x)^T \nabla_\theta f_{\theta^t}(y)\mathcal{A}^T = \mathcal{A}\Theta_t(x, y), \tag{10}$$
where $\mathcal{A}^T$ is the adjoint operator of $\mathcal{A}$ and the differential operator $\mathcal{A}$ acts on variable y. This kernel will be referenced as the MSK in this work.

Eq. (8) means that under the GD training dynamics, the output function is not modified point-to-point. Instead, if there is a residual of the PDE at a sample point y, the output function is modified by a multiple of the kernel function $K(x, y, t)$. In particular, if the kernel function $K(x, y, t) = \delta(x - y)$, Eq. (8) reduces to the corresponding parabolic equation

$$\frac{\partial f_{\theta_t}(x)}{\partial t} = -\mathcal{A} f_{\theta_t}(x) + g(x). \tag{11}$$

If the kernel function $K(x, y, t)$ is a function of $(x - y)$ (which is approximately true under NTK parametrization), the integral becomes a convolution. In this case, $\left|\hat{K}(\omega)\right|$ provides a bias for learning rate of components of different frequencies.

2.3. Non-gradient method for elliptic equations

As we will discuss in the following, the kernel function for the deep Ritz formulation can be easily controlled for efficient training. However, many elliptic equations do not have a variational form, such as the steady-state Fokker-Planck equation with non-gradient forces and the steady state advection-diffusion equations. An interesting observation for these equations is that the dissipative property of their corresponding time-dependent parabolic equations can guarantee the convergence of the corresponding parabolic equations to steady state as time goes to infinite. In other words, although the eigenvalues of these elliptic equations are not positive real numbers, they all have positive real parts.

Inspired by the above observation of dissipation, we make a brave attempt to develop the non-gradient method to train the parameters of deep neural networks with random samples

$$\theta^{t+1} - \theta^t = -\eta \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta f_\theta(x_i) \left(\mathcal{A} f_{\theta_t}(x_i) - g(x_i)\right), \tag{12}$$

where m is the sample size and x_i are the sample points. For uniformly distributed sample points, when $m \to \infty$, the corresponding continuum-limit dynamics of the output function satisfies

$$\frac{\partial f_{\theta_t}(x)}{\partial t} = -\int_\Omega \Theta_t(x, y)(\mathcal{A} f_{\theta_t}(y) - g(y))\mu(dy). \tag{13}$$

This dynamics (13) has a same form with that for deep Ritz method. However, we do not require the residual $\mathcal{A} f_{\theta_t}(y) - g(y)$ to be in a variational formulation. The relaxation of this requirement expands the application range of Eq. (12) greatly. The convergence of the training dynamics relies on the dissipation of Eq. (13). Furthermore, Eq. (12) can also be employed to solve nonlinear elliptic equations in case that there is a dissipation relation.

Similar to that in previous studies, boundary conditions can be satisfied by introducing a mean square loss of boundary residues. For Dirichlet boundary conditions, the dynamics of the output function becomes

$$\frac{\partial f_{\theta_t}(x)}{\partial t} = -\int_\Omega \Theta_t(x, y)(\mathcal{A} f_{\theta_t}(y) - g(y))\mu(dy) - \lambda_b \int_{\partial\Omega} \theta_t(x, y)(u(y) - \tilde{u}(y))\tilde{\mu}(dy), \tag{14}$$

where λ_b is a weight parameter and $\tilde{u}$ is the boundary value of the function.

In real applications, random samples are used to numerically evaluate the high-dimensional integrals in Eq. (14) in a Monte-Carlo fashion. A training step for the non-gradient method for elliptic equations is included in Algorithm 1.

Algorithm 1 Non-gradient method (with Dirichlet boundary condition).

Input: neural network parameters θ^t and uniformly distributed random samples $\{x_i^t\}_{i=1}^m \in \Omega$ and $\{\tilde{x}_i^t\}_{i=1}^{m_b} \in \partial\Omega$.
Output: neural network parameters θ^{t+1}.

1: Compute gradients:

$$\nabla_\theta f_{\theta^t} = (\nabla_\theta f_{\theta^t}(x_1^t), \cdots, \nabla_\theta f_{\theta^t}(x_m^t)) \in \mathcal{R}^{|\theta| \times m},$$
$$\tilde{\nabla}_\theta f_{\theta^t} = (\nabla_\theta f_{\theta^t}(\tilde{x}_1^t), \cdots, \nabla_\theta f_{\theta^t}(\tilde{x}_{m_b}^t)) \in \mathcal{R}^{|\theta| \times m_b};$$

2: Compute residues:

$$\mathbf{r}^t = (\mathcal{A} f_{\theta^t}(x_1) - g(x_1), \cdots, \mathcal{A} f_{\theta^t}(x_m) - g(x_m))^T \in \mathcal{R}^m,$$
$$\tilde{\mathbf{r}}^t = \left(f_\theta(\tilde{x}_1^t) - \tilde{u}(\tilde{x}_1^t), \cdots, f_\theta(\tilde{x}_{m_b}^t) - \tilde{u}(\tilde{x}_{m_b}^t)\right)^T \in \mathcal{R}^{m_b};$$

3: Update neural parameters $\theta^{t+1} = \theta^t - \frac{\eta}{m} \nabla_\theta f_{\theta^t} \cdot \mathbf{r}^t - \frac{\lambda_b \eta}{m_b} \tilde{\nabla}_\theta f_{\theta^t} \cdot \tilde{\mathbf{r}}^t$.

2.4. Discretization

In principle, the linear or nonlinear elliptic operator $\mathcal{A} f_\theta$ on the network output function can be evaluated analytically. Nevertheless, the higher order derivatives in the operator put forward requirements for the smoothness of activation functions used in the deep neural networks. As we will discuss in the following section, Relu activation function has its advantages in controlling the compactness of the kernel. In order to solve this dilemma, we use central difference to discretize the elliptic operators at all sample points. Namely, the residual r^t in Algorithm 1 is evaluated by

$$\mathbf{r}^t = (\mathcal{A}_h f_{\theta^t}(x_1) - g(x_1), \cdots, \mathcal{A}_h f_{\theta^t}(x_m) - g(x_m))^T,$$

where h is the adjustable step size for the discretization. In particular, the linear operators can be discretized as

$$\mathcal{A}_h f_\theta(\mathbf{x}) = \sum_{i=1}^{d} \frac{D(\mathbf{x} + \frac{h}{2}\mathbf{e}_i)\left(f_\theta(\mathbf{x} + h\mathbf{e}_i) - f_\theta(\mathbf{x})\right) - D(\mathbf{x} - \frac{h}{2}\mathbf{e}_i)\left(f_\theta(\mathbf{x}) - f_\theta(\mathbf{x} - h\mathbf{e}_i)\right)}{h^2}$$
$$+ \sum_{i=1}^{d} \frac{b_i(\mathbf{x})\left(f_\theta(\mathbf{x} + h\mathbf{e}_i) - f_\theta(\mathbf{x} - h\mathbf{e}_i)\right)}{2h} + c(\mathbf{x}) f(\mathbf{x}),$$

where $\mathbf{e}_i$ are the standard basis vectors.

2.5. Evaluation of computational cost

Here we compare the computational cost for each sample point of the non-gradient method with that of the loss-based methods deep learning methods.

For each sample point $\mathbf{x}$, the computational cost includes two parts – evaluation of the output function f at $2d+1$ nearby points (or analytically evaluate the residual at the sample point) and calculates the gradient $\nabla_\theta f_\theta(\mathbf{x})$; whereas in the stochastic gradient descent dynamics for the mean square formulation, the computational cost includes evaluation of the residual and computes the gradient with respect to θ of $2d+1$ functions (the output function $f_\theta(\mathbf{x})$, d terms in its gradient $\nabla_x f_\theta(\mathbf{x})$, and d terms in its Laplacian $\triangle_x f_\theta(\mathbf{x})$). In deep Ritz method, since there is no need to compute $\nabla_\theta \triangle_x f_\theta(\mathbf{x})$, the computational cost is approximately half that of the mean square formulation. Since the computational cost for function evaluation is much smaller than that for gradient calculation for deep neural networks, the non-gradient method significantly reduces the computational cost in comparison with the mean square formulation.

3. Properties of the training kernels

Clearly, the properties of the kernel function can influence the training dynamics deeply. Neural tangent kernels are usually continuous (or even smooth) functions. The convolution property implies that high frequency components of the solution are learned very slowly because the Fourier transform $\widehat{K(\omega)}$ (which is the eigenvalue of the convolution operator and ω is the spatial wavenumber) becomes very small for large ω. In order for a relatively fast learning rate for components of given frequencies (e.g., when we have some a priori knowledge on the spectral property of the solution), we need to control the compactness of the kernel so that $\widehat{K(\omega)}$ is sufficiently large for desired frequencies. In particular, the Fourier modes of widely spread kernels decay faster with the frequency than that of locally compacted kernels. Locality of the kernels are important in learning high frequency components of the solution. Meanwhile, the kernel should also not be too local for high dimensional problems. Otherwise, the number of sample points required for training the neural network grows exponentially with the dimension and it becomes impossible to train a network for high dimensional problems. Therefore, it is important to carefully control the locality of the training kernels for efficient training.

In the following, we first briefly review the important theories on NTKs; then we show the properties (with an emphasize on the locality) of the kernels under different activation functions and discuss their effects.

3.1. Theories of neural tangent kernel

In the widely used initialization strategy of neural network parameters, such as LeCun's initialization, the weights and bias parameters are given by Gaussian distributions with mean zero and variance inversely proportional to the number of input units. Such initialization strategy is very similar to that used in NTK parametrization [38]. For a network with L layers given by

$$\begin{aligned}\tilde{\alpha}^{(l+1)}(x) &= \frac{\sqrt{1-\beta^2}}{\sqrt{n_l}} W^{(l)} \alpha^{(l)}(x) + \beta b^{(l)},\\ \alpha^{(l+1)}(x) &= \sigma(\tilde{\alpha}^{(l+1)}(x)),\end{aligned} \tag{15}$$

where $\alpha^l(x)$ $(l = 1, 2, \cdots, L)$ are the outputs of layer l and $\alpha^0(x) = x$ is the input of the network, $W^{(l)}$ and $b^{(l)}$ are the weight matrices and bias vectors, respectively, n_l is the network width of layer l, β is an adjustable coefficient for the bias vector $b^{(l)}$, and $\sigma()$ is the activation function, the NTK parametrization simply initializes all weights, $W^{(l)}$ and $b^{(l)}$, using independent standard Gaussian distribution $\mathcal{N}(0,1)$. Note that the variance of the effective weights $\frac{\sqrt{1-\beta^2}}{\sqrt{n_l}} W^{(l)}$ is also inversely proportional to the number of input units n_l due to the prefactor. In general, we assume that the activation function $\sigma()$ has been normalized, namely, we have $\mathbb{E}_{x\sim\mathcal{N}(0,1)}[\sigma^2(x)] = 1$. Otherwise, we can normalize σ by

$$\tilde{\sigma}(x) = \frac{\sigma(x) - \mathbb{E}_{x\sim\mathcal{N}(0,1)}\sigma(x)}{\sqrt{\mathbb{E}_{x\sim\mathcal{N}(0,1)}(\sigma(x) - \mathbb{E}_{x\sim\mathcal{N}(0,1)}\sigma(x))^2}}. \tag{16}$$

By this normalization, the output vector $\tilde{\alpha}^l(x)$ of each layer satisfies the standard Gaussian distribution.

For simplicity of description and notations, we use θ to represent all the parameters $W^{(l)}$ and $b^{(l)}$. As mentioned above, the Neural Tangent Kernel (NTK) is defined as $\Theta_t(x,y) = \nabla_\theta f_{\theta^t}(x)^T \nabla_\theta f_{\theta^t}(y)$. In particular, $\Theta_0(x,y) = \nabla_\theta f_{\theta^0}(x)^T \nabla_\theta f_{\theta^0}(y)$ is the initial NTK, where θ^0 is the initial value of parameters.

We briefly collect the important theories of NTKs in below [38]. In the rigorous proof of these theories, one needs to assume that the input x lies on a sphere $\|x\|_2 = c$. In real applications, this requirement is not crucial.

Theorem 1. *When the width n_l of hidden layers approaches to infinity, the NTK approaches to a deterministic matrix:*

$$\Theta_t(x,y) \to \Theta_\infty(x,y) \tag{17}$$

for any x, y.

Table 1
Value of r for different activation function and different value of β.

β	0.0	0.1	0.2	0.3	0.4	1.0
ReLU	1.47	1.45	1.41	1.33	1.23	0.00
ELU	1.08	1.07	1.06	0.98	0.91	0.00
Swish	1.21	1.20	1.16	1.10	1.02	0.00

The correctness of this theorem relies on two facts: First, as $n_l \to \infty$, we have $\Theta_0(x, y) \to \Theta_\infty(x, y)$ as a consequence of the central limit theorem; Second, under NTK parametrization, the change of parameters during training is small for large network width n_l. In real applications, n_l is usually not so big. As a result, there is usually a slight but visible change of parameters during training. In this case, the NTK can also be slightly changed. Nevertheless, the property of the NTK is largely determined by the initial value. Therefore, we can pay our main attention on the initial NTK, which is determined by the network structure (e.g., the depth of the network) and the activation functions.

When the input x lies on a sphere $\|x\|_2 = c$, the kernel $\Theta_\infty(x, y)$ can be written in the form of $\Theta_\infty(\|x - y\|_2)$. This provides the convolution property of Eq. (13). As we have discussed above, the locality and compactness of the NTK is crucial in determining the spectral bias effect of the training dynamics (13). The next theorem provides deep insight in understanding the properties of NTKs $\Theta_\infty(x, y)$ [40].

Theorem 2. *Denote $r = (1 - \beta^2)\mathbb{E}_{x\sim\mathcal{N}(0,1)}[\dot\sigma^2(x)]$, where $\dot\sigma(x)$ is the derivative of σ with respect to x, and denote the normalized NTK*

$$\vartheta^L(x, y) = \frac{\Theta_\infty(x, y)}{\sqrt{\Theta_\infty(x, x)\Theta_\infty(y, y)}}.$$

where L emphasises the depth of neural network. If the NTK parametrization is used and $r > 1$ (the chaos region), we have

$$|\vartheta^L(x, y)| \le Ch_0^L \tag{18}$$

for $x \neq y$, where C is a constant and $h_0 = h_0(x, y) < 1$ is a function dependent only on x and y for given $\sigma(\cdot)$ and β.

From numerical tests one can estimate that $h_0(x, y)$ is generally a decreasing function of $\|x-y\|_2$ (at least when $\|x-y\|_2$ is not too big). Therefore for $r > 1$, Theorem 2 means that the kernel Θ_t is locally compacted for sufficiently deep neural networks, since the upper bound $Ch_0^L(x, y)$ ($\|x - y\|_2 > R_0$) becomes very small for sufficiently large L, where R_0 is the effective radius for the compact kernel.

It is worth noting that $r = (1 - \beta^2)\mathbb{E}_{x\sim\mathcal{N}(0,1)}[\dot\sigma^2(x)]$ depends on both β and the activation function. For each activation function, r is an decreasing function of β ($\beta > 0$) and reaches the maximum at $\beta = 0$. In Table 1, we show r for different β under the above three activation functions. In the following, we will show that r is critical in determining the function $h_0(x, y)$, thus plays a dominate role in the locality and compactness of the NTK.

3.2. Compactness of kernels

Next, we discuss the properties, in particular, the compactness of the kernels under different activation functions, different network depth, different dimension, and different choices of the parameter β. Based on these properties, we select a good choice for general applications in solving elliptic equations.

Three activation functions with different smoothness are discussed in the following:

1. $\text{Swish}(x) = \frac{x}{1+e^{-x}}$;
2. $\text{Elu}(x) = \begin{cases} x, & \text{if } x > 0, \\ e^x - 1, & \text{else;} \end{cases}$
3. $\text{Relu}(x) = \max(x, 0)$.

Normalization of the activation functions are performed based on Eq. (16).

Without loss of generality, the domain for kernel testings are set to be $[-\frac{\pi}{2}, \frac{\pi}{2}]^d$. Since the NTKs are approximately translation invariant, the numerically normalized kernels $\vartheta(x, y)$ are averaged for 20 equally spaced values of y to visualize the kernels. Meanwhile, since the kernels are approximately axi-symmetric, one-dimensional curves are used to illustrate the kernels. The width of each layer is fixed to be 2000 in the tests to efficiently remove numerical fluctuations.

3.2.1. Compactness of NTKs

As discussed above, the training kernels are exactly the NTKs under the deep Ritz formulation and our non-gradient formulation. In other words, the training effect under these formulations is largely determined by the characteristics of the NTKs. In the following, we study the compactness of the NTKs numerically. As we will see that the Relu activation function can be easily used in controlling the compactness of the NTKs by carefully selected bias parameter β and network depth L.

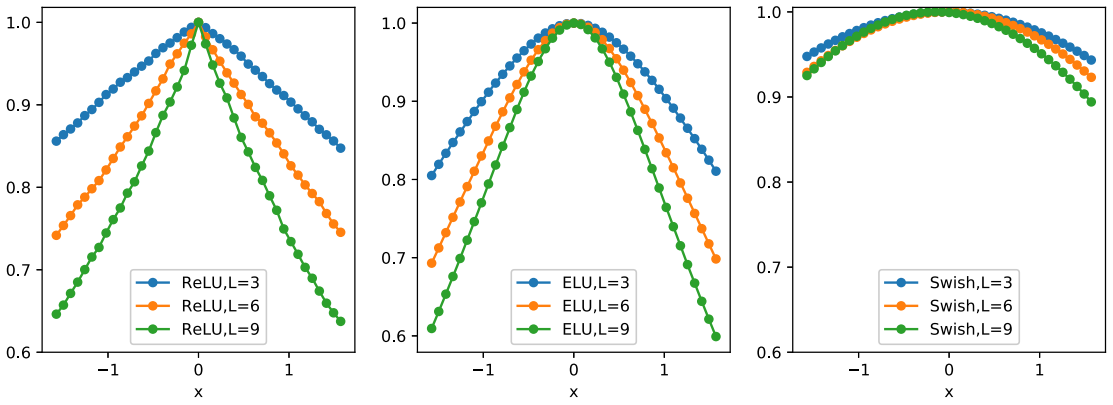

Fig. 1. Kernels for $r = 1.07$ and network depth $L = 3$, 6, and 9 under different activation functions. The input dimension $d = 16$.

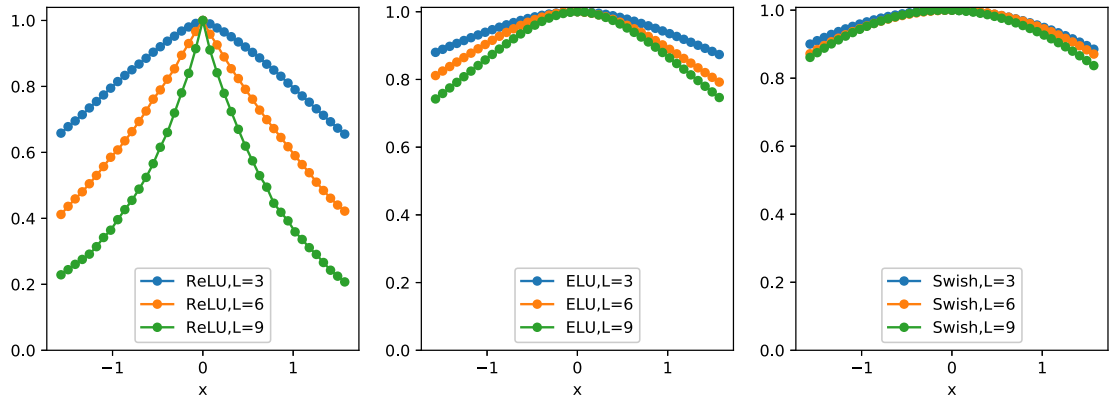

Fig. 2. Kernels for $\beta = 0.2$ under different activation functions and network depths. The input dimension is 16.

The quantity r defined in Theorem 2 is crucial in determining the compactness of the kernels – as we can see for all tested activation functions, the greater the parameter r is, the more locally compacted the kernel is.

In Fig. 1, the normalized NTKs are shown for the case that $r = 1.07$ with the input dimension $d = 16$. Note that β is different for the three activation functions (see Table 1). In particular, r almost reaches its maximum for the Elu activation function. Since r is not relatively small in these cases, the kernels are not well compacted locally — they all maintain a relatively large value on the domain boundary $|x| = \pi/2$.

In all following tests, we fix $\beta = 0.2$ for a fair comparison of the compactness of the NTKs under different activation functions.

The NTKs for $\beta = 0.2$ are shown in Fig. 2. Since r for the Relu activation function is much greater than that for the other two activation functions, the NTKs under Relu activation function are much more compacted than the others. Furthermore, we can see from both Fig. 1 and 2 that as the depth of the network increases, the kernels become more locally compacted. This is consistent with the power law bound in the inequality (18). In fact, this power law inequality can help us to estimate the changes of the NTKs as the depth of the network increases.

In the original NTK theory [38–42], the input x is defined on the unit sphere. In this case, the NTK $\Theta_\infty(x, y)$ depends only on the angle between x and y. Namely, the NTK is independent on the dimension of the input x. However, in our applications, the input x is defined in the entire cubic domain. In this case, the input dimension of x also plays an important role in determining the compactness of the NTKs. In general, the NTKs become less compacted for larger dimension d. One may also map x onto the unit sphere by introducing a virtual dimension. Under this mapping, the angle between the images of X and Y is also dependent on the spatial dimension.

A comparison for the case of $d = 2$ and $d = 16$ are shown in Fig. 3. From Fig. 2 and 3, we can see that deeper networks are required to maintain sufficient compactness of kernels in higher dimensional spaces. In particular, networks with Relu activation function have much better performance in controlling the compactness than networks with the other two activation functions. In other words, using Relu activation function allows us to obtain compacted kernels with relatively shallow networks.

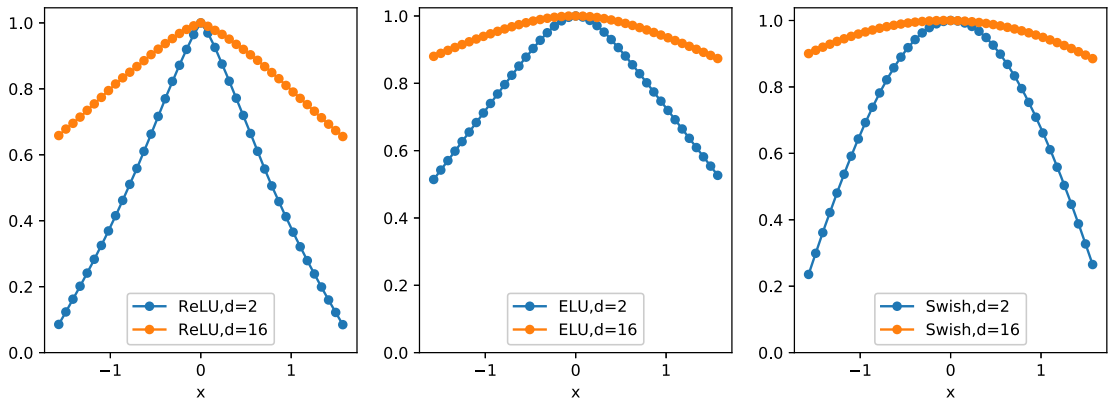

Fig. 3. Kernels for different input dimensions under different activation functions. The depth $L = 3$ and the bias parameter $\beta = 0.2$.

3.2.2. Compactness of the MSKs

As discussed above, the training kernel for the mean square formulation is $\mathcal{A}\Theta(x, y)$. The properties of the MSKs also determine the performance of the training process under the mean square formulation. In the following, we will compare NTKs and MSKs to show part of the benefits of non-gradient method. Without loss of generality, we only consider the Poisson equation in which $\mathcal{A} = -\Delta$. Further more, due to the nonsmoothness of the Relu activation function, we use the discretized operator Δ_h to calculate MSKs. Similarly, the MSKs are normalized in the following numerical tests.

A comparison of the NTKs and MSks are shown in Fig. 4. We can see that the MSKs for the Relu activation function are too local due to the sharp peak at the origin. Thus very large amount of sample points are required to train networks in high dimensional spaces when Relu activation function is used under the mean square formulation (PINN). For the Swish activation functions, the compactness of the MSKs are even worse than the corresponding NTKs.

The MSKs for Elu activation function maintain relatively good compactness. This suggests that using networks with Elu activation function to solve high dimensional elliptic equations can be efficient under the mean square formulation. However, we can also see that as the network depth increases, the compactness of the MSKs do not change significantly. This means that it can be hard to control the compactness of MSKs by tuning the hyper-parameters and the network depth.

3.3. Kernel control

In principle, the convergence speed and training efficiency is determined by the eigenvalue of the training dynamics (e.g., the integro-differential operator on the right hand side of Eq. (13)). For data fitting problems, this eigenvalue is directly $\widehat{\Theta_\infty(\omega)}$, whereas for the training dynamics of our non-gradient method (13), the eigenvalue is $\widehat{\Theta_\infty(\omega)}\widehat{A(\omega)}$, where $\widehat{A(\omega)}$ is the eigenvalue of the elliptic operator (e.g., if the operator is $A = -\Delta$, $\widehat{A(\omega)} = |\omega|^2$.) Therefore, if there are relatively high frequencies of interest in the solution, we need to use a relatively local kernel. As discussed above, such a task can be achieved by carefully choosing the activation function, the parameter β, and the network depth L. In particular, the Relu activation function is welcome in our non-gradient method. For high-dimensional equations with important high frequency components in the solution, sufficiently deep networks are helpful for the training efficiency.

4. Numerical results

Next, we show the numerical results obtained with our non-gradient method. In our simulations, the ReLU activation function is used so that the training kernels are well compacted. Three types of elliptic equations, including a symmetric equation, an asymmetric equation, and a nonlinear equation, are used for numerical tests.

In our numerical tests, Adam optimizer [44] and the default decay strategy for learning rate in Tensor Flow is used for all problems. The initial learning rate is 0.01 and the decay step is 1000. The width of each hidden layer is fixed to be 2000. In order to analyze the convergence of the non-gradient method, the L^2 relative error

$$e_{\mathtt{rel}} = \sqrt{\frac{\sum_{i=1}^{N_{\mathtt{test}}} (f_\theta(x_i) - u_\infty(x_i))^2}{\sum_{i=1}^{N_{\mathtt{test}}} f_\theta(x_i)^2}}$$

is calculated on $N_{\mathtt{test}} = 10000$ random samples, where u_∞ is the exact solution.

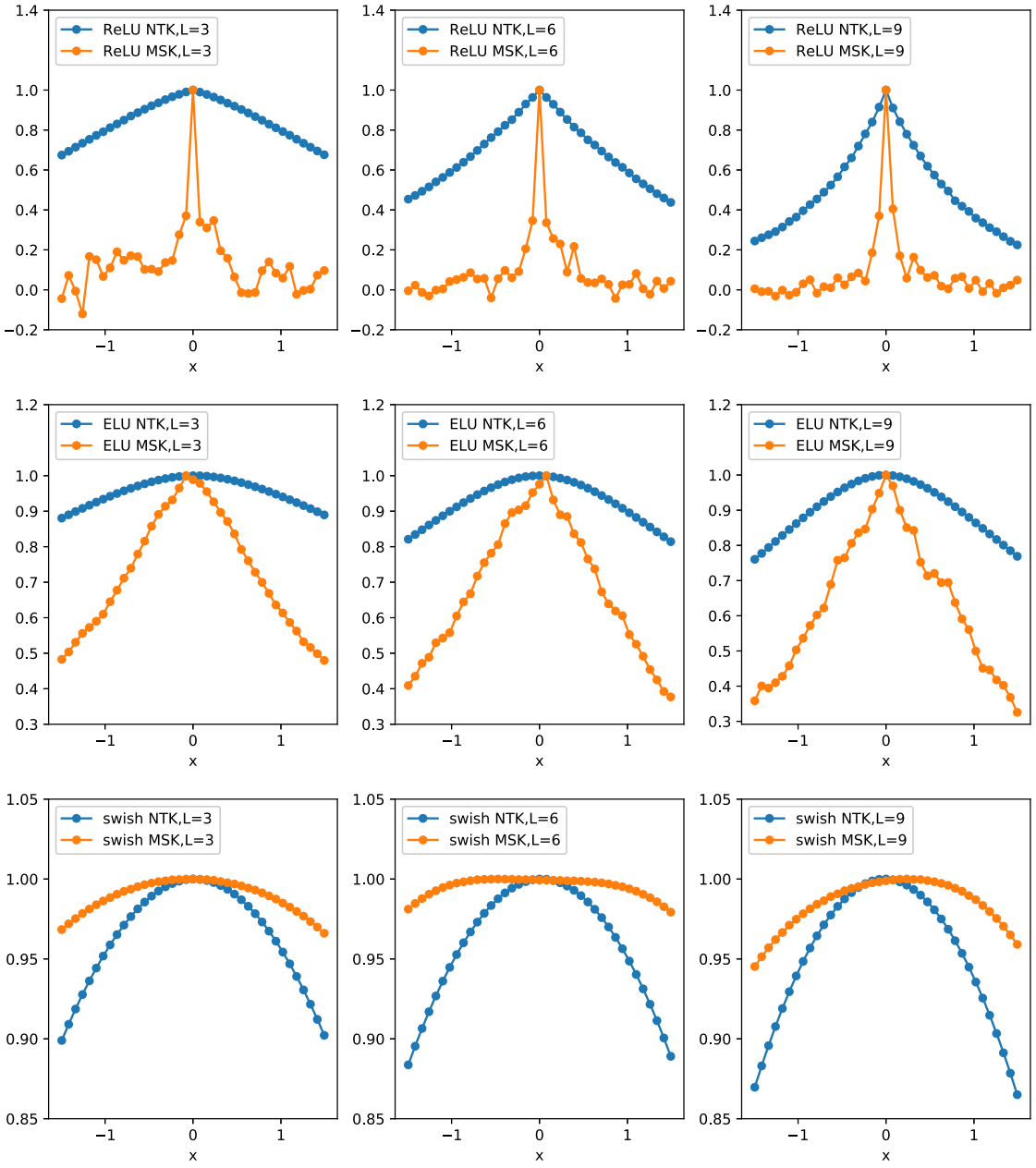

Fig. 4. The NTKs (blue) and MSKs (orange) for different activation functions and network depth. The activation functions for the upper, the middle, and the bottom rows are ReLU, Elu, and swish, respectively. The network depth L for the left, the middle, and the right columns are 3, 6, and 9, respectively. The dimension $d = 16$ and bias parameter $\beta = 0.2$.

4.1. Symmetric elliptic equation

The first equation we considered is a symmetric equation,

$$\begin{cases} -\Delta u(x) + cu(x) = g(x), & \text{in } \Omega = [-\frac{\pi}{2}, \frac{\pi}{2}]^d, \\ u(x) = u_\infty(x), & \text{on } \partial\Omega, \end{cases} \tag{19}$$

where $c = 2$ and $g(x)$ is given by $u(x) = u_\infty(x)$ in each following case.

First, we consider a two-dimensional case with relatively high frequency components, $u_\infty(x) = \sum_{i=1}^{2} (\cos(x_i) + \cos(15x_i))$. Thus a network with layers is used. For the two-dimensional case, $L = 3$ is sufficient to make the NTK sufficiently compact. In each iteration, we use 1000 random samples in the domain and 200 samples on the boundary. The solution of the equation converges well after 20000 epochs. The solution and L_2 error are shown in Fig. 5. Note that the analytic solutions

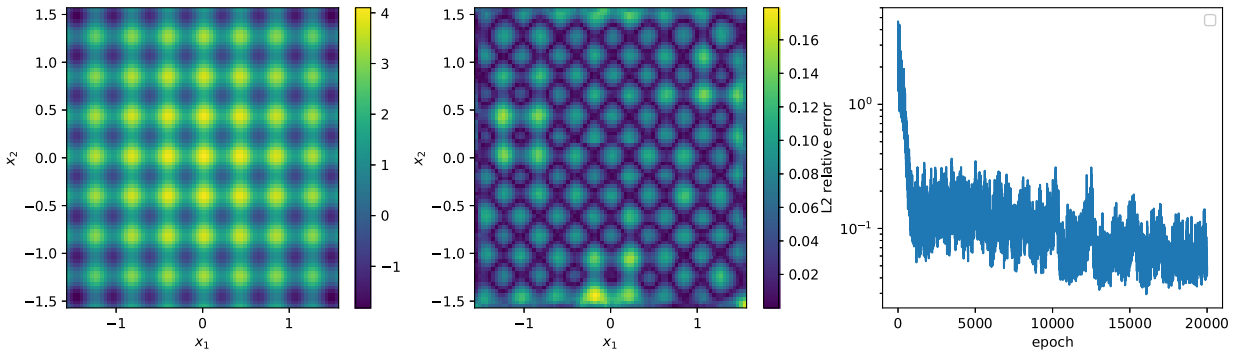

Fig. 5. Left: Numerical solution of the two dimensional case. Middle: The corresponding absolute error. Right: Convergence of the L_2 relative error.

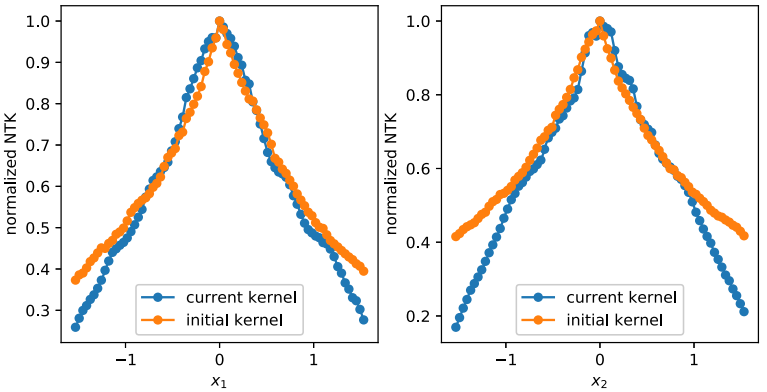

Fig. 6. The normalized NTKs before and after training for two dimensional example. Left: the first dimension; Right: the second dimension.

of the equations are all known in our examples, which allows us to evaluate the L_2 error. Nevertheless, the L_2 error and information of the solution is not used in our non-gradient method. We can see that the high frequency components are resolved very well.

As shown in Fig. 6, the change of NTKs is insignificant during the training. This implies that a well-behaved initial NTK is sufficient to ensure fast convergence and high accuracy approximation of the solution.

Next, we consider a ten-dimensional example with $u_\infty(x) = \cos(10x_1) + \cos(10x_2) + \sum_{i=1}^{10} \cos(x_i)$. Note that there are high frequency components in the first two dimensions. In order to make the NTK sufficiently compact, the depth of network is set to be $L = 6$. The sample size is $m = 10000$ and $m_b = 5000$. The solution of the equation converges very well after 30000 epochs of training. The solution on three cross sections are shown in Fig. 7. Similarly, the high frequency components are well resolved.

4.2. Advection diffusion equation

When the elliptic operator is asymmetric, the deep Ritz formulation can not be applied. However, our non-gradient method is still efficient in handling this kind of elliptic equations. As an example, we consider the advection-diffusion equation

$$-\Delta u(x) + \vec{b}(x) \cdot \nabla u(x) + 2u(x) = g(x), \tag{20}$$

where $x = (x^1, \cdots, x^d) \in \Omega = [-\frac{\pi}{2}, \frac{\pi}{2}]^d$ and $\vec{b}(x) = (\sin(x_1), \cdots, \sin(x_d))$. $g(x)$ and the Dirichlet boundary value are given by the exact solution $u_\infty(x) = \cos(10x_1) + \sum_{i=1}^{d} \cos(x_i)$.

The network structure and sample size are the same as the above ten dimensional case for the symmetric equations. Two cross-sectional illustrations of the solution and the convergence of the numerical error are shown in Fig. 8. The good approximation of the solution shows the effectiveness of our non-gradient method for asymmetric elliptic equations.

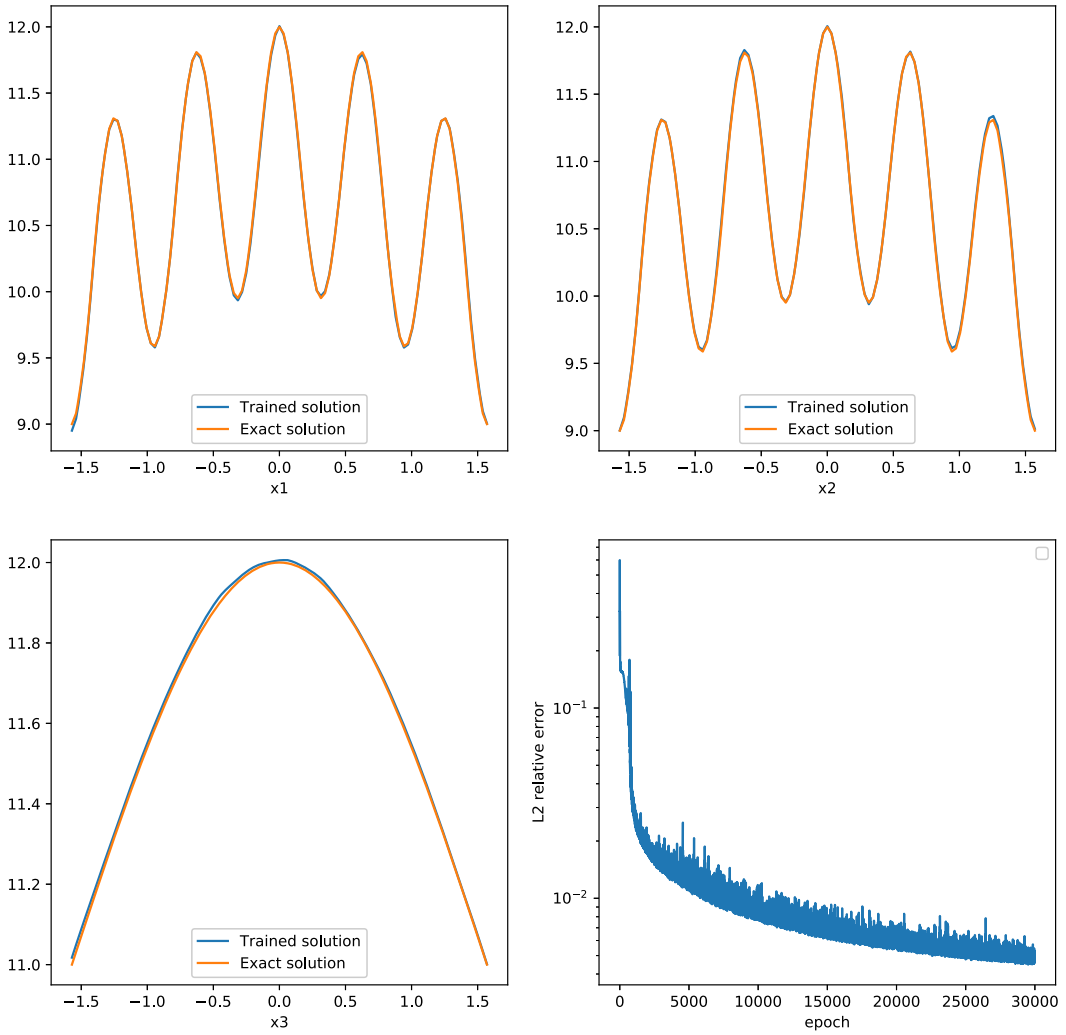

Fig. 7. The solution in three cross sections and the convergence of the L_2 relative error.

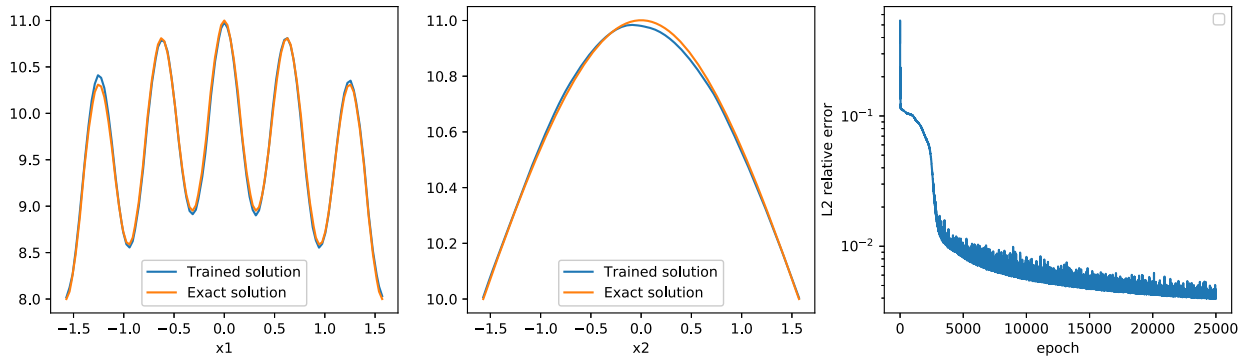

Fig. 8. Cross-sectional illustrations of the solution and convergence of the L_2 relative error of the ten dimensional advection-diffusion equation.

4.3. Fokker-Planck equation

The Fokker-Planck equation

$$\frac{\partial P(x,t)}{\partial t} = \nabla \cdot (-\mathbf{b}(x)P(x,t) + \epsilon \nabla P(x,t)), \tag{21}$$

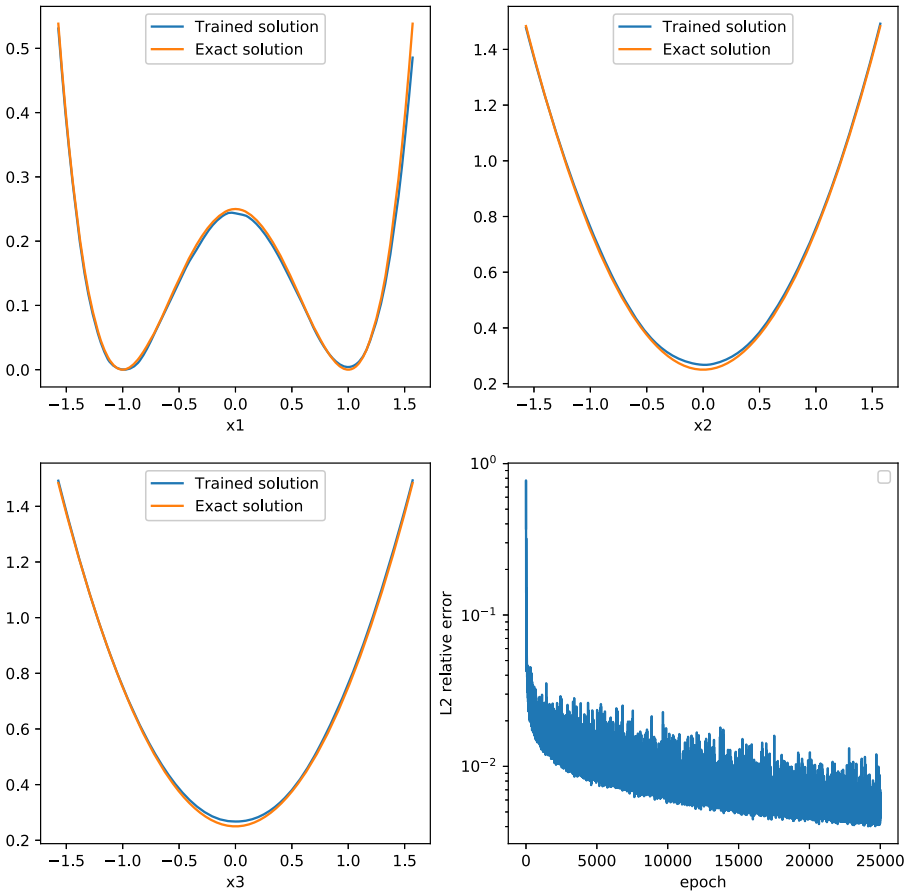

Fig. 9. Cross-sectional illustration and convergence of the L_2 relative error for the 10-dimensional Fokker-Planck equation.

describes the evolution of the probability density function (PDF) of Brownian particles (or systems) under the driven force $\mathbf{b}(x)$. The steady state solution $P(x)$ of the Fokker-Planck equation is important in many applications such as many rare event dynamics. When ϵ is small, the PDF P can change drastically, which brings difficulty in numerical methods. Another difficulty is that the PDF must be kept positive. Therefore, people turn to solve the equation of $u(x) = -\epsilon \log P(x)$,

$$\epsilon(\Delta u(x,t) + \nabla \cdot \mathbf{b}(x)) - \nabla u(x,t) \cdot \mathbf{b}(x) - \|\nabla u(x,t)\|^2 = 0. \tag{22}$$

Note that Eq. (22) can be obtained by substituting $P(x) = \exp(-\frac{u(x)}{\epsilon})$ into the steady state equation of Eq. (21). Eq. (22) is a nonlinear elliptic equation. The corresponding parabolic equation of Eq. (22) also maintains a natural energy dissipation relation.

In our simulation, the exact solution is selected to be

$$u_\infty(x) = \frac{(1 - {x_1}^2)^2}{4} + x_1 x_2 + \frac{\sum_{i=2}^{d} {x_i}^2}{2}$$

and $\mathbf{b}(x)$ is given by

$$\mathbf{b}(x) = \nabla \times F - \frac{1}{\epsilon} \nabla u(x) \times F - \nabla u(x), \tag{23}$$

where the cross product "$\times$" acts only on the first three dimensions and the other components are all zero. A generalization of the cross product can be defined by exterior differentiation. It is direct to verify that u_∞ is the solution of Eq. (22) once $\mathbf{b}(x)$ is given by Eq. (23). Furthermore, when $F \neq 0$, u cannot be trivially read from $\mathbf{b}(x)$ (e.g., by solving the equation $\triangle u = \nabla \cdot \mathbf{b}$).

In our simulation, we simply use $F = (1, 0, 0, \cdots, 0)$ to define $\mathbf{b}(x)$. The cross-sectional illustrations of solution are shown in Fig. 9. We can see that the solution is also approximated very well.

Note that $\epsilon = 2.0$ is used in our simulation. When ϵ is small (compared to the energy barrier of u_∞), the elliptic operator of the original Fokker-Plank equation (21) has an exponentially small eigenvalue corresponding to the rare event dynamics

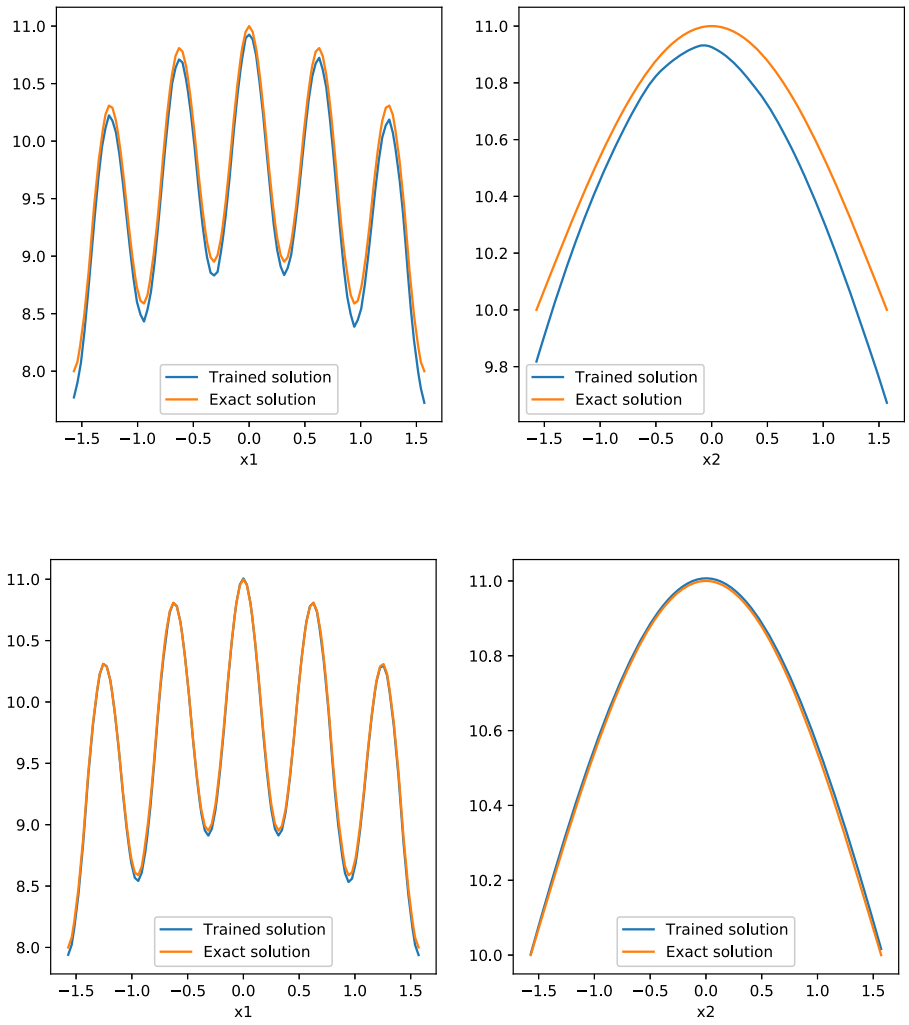

Fig. 10. Cross-sectional illustrations of the solutions learned by the PINN method with ReLU (the upper panel) and Swish (the bottom panel) activation functions.

due to the energy barrier in u_∞. This small eigenvalue results in a slow dissipation of both Eq. (21) and Eq. (22). As a result, the non-gradient method also converges very slowly. Note that this is not due to a drawback of our non-gradient method but an intrinsic difficulty of the original problem. Under the mean square formulation, the small eigenvalue also results in a large condition number of the optimization problem and the solution can not be directly found.

4.4. Comparison against PINN

Since the Deep Ritz method is effectively equivalent to our non-gradient method for elliptic equations with positive definite operators, we mainly compare our non-gradient method with the Physics-informed Neural Network (PINN) method based on the mean square loss of the residual.

The same example in subsection 4.2 is used for the comparison. The same network setup and learning rate as in subsection 4.2 are used for a relatively fair comparison, although the best learning rate for different methods may be different.

The cross-sectional illustrations of solutions obtained by the PINN method are shown in Fig. 10. We can see a relatively large numerical error in the solution obtained with the Relu activation function. The convergence analysis of the PINN method with different activation functions and the non-gradient method with Relu activation function is shown in Fig. 11. We can see that the convergence of the L^2 error is comparable for the non-gradient method and the PINN method with Elu and Swish activation functions, whereas the PINN method with Relu activation function converges slowly. Interestingly, the PINN loss also receives a decay in the training process of the non-gradient method, although the decay is less than that of the PINN method with Elu and Swish activation functions.

Although theoretical analysis has shown that the NTK (and MSK) can hardly change under the NTK parameterization if the network width goes to infinity, the MSKs obtained in our training process change a lot in fact, as shown in Fig. 12.

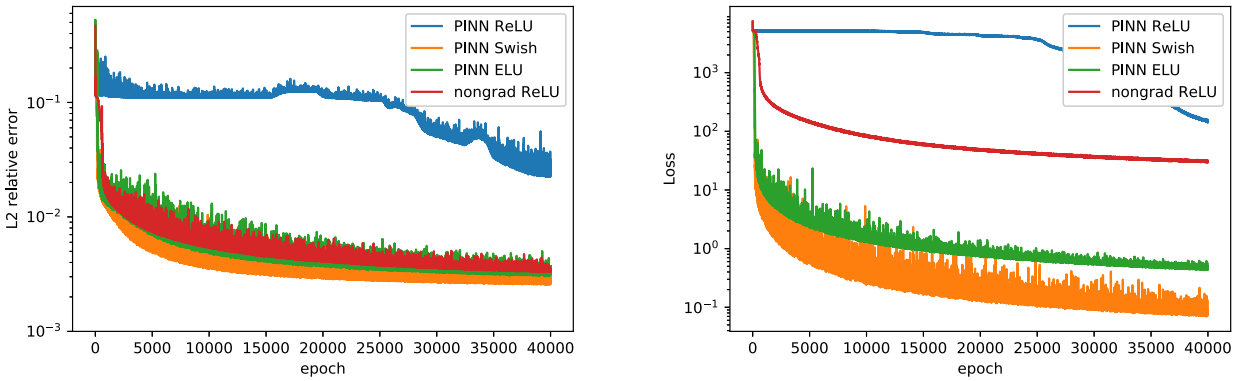

Fig. 11. Convergence of L2 relative error (Left) and the PINN Loss (Right).

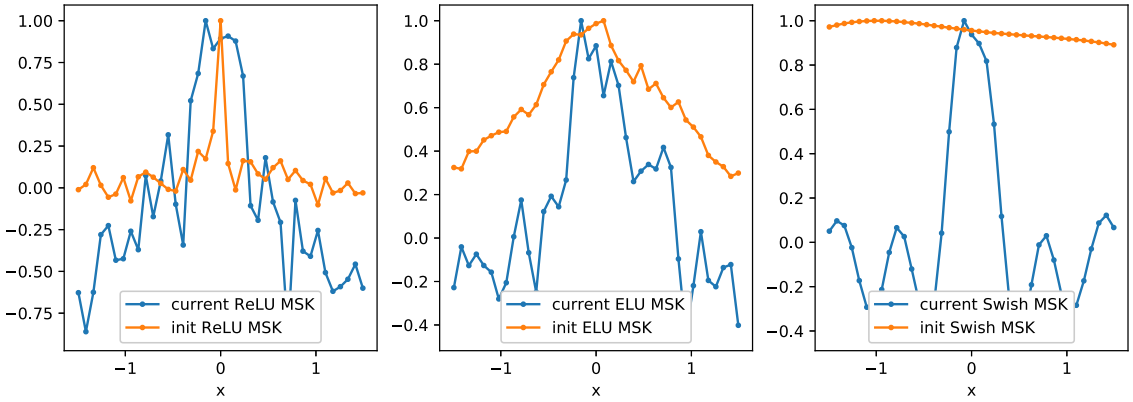

Fig. 12. The initial and final MSKs with different activation functions.

In other words, the training in the PINN method has learned certain features. It is of interest to see that all final MSKs maintain certain locality regardless of their initial shape.

5. Conclusion and discussion

In this work, we develop a non-gradient method to solve elliptic PDEs using deep neural networks. In our non-gradient method, the training kernel is exactly the neural tangent kernel (NTK).

The first advantage of our non-gradient method is that we do not require a loss functional to find the solution. Instead, we directly use the residual of the equation to train the network. The second advantage of our method is that the training kernel can be well controlled by the activation function, the network depth, and the parameter β in NTK parameterization. By well controlled NTKs, the training dynamics can be efficient and the solution can be well approximated. Furthermore, using numerical discretization to evaluate the residual reduces the computational cost significantly. It also allows us to use the ReLU activation function for elliptic equations with high order derivatives. Comparing to the Deep Ritz method, our non-gradient method can be applied to asymmetric elliptic equations; Comparing PINN (the mean square formulation), the training kernel of the non-gradient method can be controlled more easily, which is important for learning of relatively high-frequency components in the solution.

Our numerical examples show the great potential of our non-gradient method in solving elliptic PDEs, where the elliptic operator can asymmetric or nonlinear. In this case, the training dynamic is no longer a gradient flow. Instead, the convergence of our non-gradient method relies on the dissipation of the training dynamics. Although we have assumed the equivalence between the dissipation properties of the corresponding parabolic equation and the training dynamics for an elliptic equation, there is no theoretical guarantee so far. Based on this assumption, we only require all eigenvalues of the (linearized) elliptic operator to maintain positive real parts instead of requiring the operator to be positive definite. For PDEs with one or a few negative eigenvalues, such as the Helmholtz equation, the non-gradient method cannot be applied yet.

CRediT authorship contribution statement

Yifan Peng: Methodology, Programming, and Writing. **Dan Hu:** Conceptualization, Methodology, and Writing. **Zhiqin Xu:** Programming.

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.

Data availability

No data was used for the research described in the article.

Acknowledgement

This work is supported by the National Key R&D Program of China (2019YFA0709503), the National Natural Science Foundation of China (Contract no. 11971312), and Student Innovation Center, Shanghai Jiao Tong University.

References

[1] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444.
[2] G.D. Smith, G.D. Smith, G.D.S. Smith, Numerical Solution of Partial Differential Equations: Finite Difference Methods, Oxford University Press, 1985.
[3] O.C. Zienkiewicz, R.L. Taylor, P. Nithiarasu, J. Zhu, The Finite Element Method, vol. 3, McGraw-Hill, London, 1977.
[4] R. Eymard, T. Gallouët, R. Herbin, Finite volume methods, Handb. Numer. Anal. 7 (2000) 713–1018.
[5] P. Indyk, R. Motwani, Approximate nearest neighbors: towards removing the curse of dimensionality, in: Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing, 1998, pp. 604–613.
[6] W. E, J. Han, A. Jentzen, Algorithms for solving high dimensional pdes: from nonlinear Monte Carlo to machine learning, arXiv:2008.13333, 2020.
[7] C. Beck, L. Gonon, A. Jentzen, Overcoming the curse of dimensionality in the numerical approximation of high-dimensional semilinear elliptic partial differential equations, arXiv preprint, arXiv:2003.00596, 2020.
[8] C. Beck, M. Hutzenthaler, A. Jentzen, B. Kuckuck, An overview on deep learning-based approximation methods for partial differential equations, arXiv preprint, arXiv:2012.12348, 2020.
[9] F. Hornung, A. Jentzen, D. Salimova, Space-time deep neural network approximations for high-dimensional partial differential equations, arXiv preprint, arXiv:2006.02199, 2020.
[10] T. Dockhorn, A discussion on solving partial differential equations using neural networks, arXiv preprint, arXiv:1904.07200, 2019.
[11] J. Berg, K. Nyström, A unified deep artificial neural network approach to partial differential equations in complex geometries, Neurocomputing 317 (2018) 28–41.
[12] W. E, J. Han, A. Jentzen, Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations, Commun. Math. Stat. (2017) 349–380.
[13] J. Han, W. E, Deep learning approximation for stochastic control problems, arXiv:1611.07422, 2016.
[14] T. Nakamura-Zimmerer, Q. Gong, W. Kang, Qrnet: optimal regulator design with lqr-augmented neural networks, IEEE Control Syst. Lett. 5 (4) (2020) 1303–1308.
[15] W. E, B. Yu, The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems, Commun. Math. Stat. (2018) 1–12.
[16] Y. Khoo, L. Ying, Switchnet: a neural network model for forward and inverse scattering problems, J. Sci. Comput. 41 (5) (2019) A3182–A3201.
[17] Y. Fan, L. Y, Solving inverse wave scattering with deep learning, arXiv:1911.13202, 2019.
[18] Y. Peng, L. Lin, L. Ying, L. Zepeda-Núñez, Efficient long-range convolutions for point clouds, arXiv preprint, arXiv:2010.05295, 2020.
[19] J. He, L. Li, J. Xu, C. Zheng, Relu deep neural networks and linear finite elements, arXiv preprint, arXiv:1807.03973, 2018.
[20] Y. Zang, G. Bao, X. Ye, H. Zhou, Weak adversarial networks for high-dimensional partial differential equations, J. Comput. Phys. 411 (2020) 109409.
[21] C. Beck, S. Becker, P. Cheridito, A. Jentzen, A. Neufeld, Deep splitting method for parabolic pdes, arXiv:1907.03452, 2019.
[22] J. Darbon, G.P. Langlois, T. Meng, Overcoming the curse of dimensionality for some Hamilton–Jacobi partial differential equations via neural network architectures, Res. Math. Sci. 7 (3) (2020) 1–50.
[23] J. Darbon, T. Meng, On some neural network architectures that can represent viscosity solutions of certain high dimensional Hamilton–Jacobi partial differential equations, J. Comput. Phys. 425 (2021) 109907.
[24] J. Sirignano, K. Spiliopoulos, Dgm: a deep learning algorithm for solving partial differential equations, J. Comput. Phys. 375 (2018) 1339–1364.
[25] 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.
[26] Z. Cai, J. Chen, M. Liu, X. Liu, Deep least-squares methods: an unsupervised learning-based numerical method for solving elliptic pdes, J. Comput. Phys. 420 (2020) 109707.
[27] E. Kharazmi, Z. Zhang, G.E. Karniadakis, Variational physics-informed neural networks for solving partial differential equations, arXiv preprint, arXiv: 1912.00873, 2019.
[28] S. Wang, H. Wang, P. Perdikaris, On the eigenvector bias of Fourier feature networks: from regression to solving multi-scale pdes with physics-informed neural networks, arXiv preprint, arXiv:2012.10047, 2020.
[29] A.D. Jagtap, K. Kawaguchi, G.E. Karniadakis, Adaptive activation functions accelerate convergence in deep and physics-informed neural networks, J. Comput. Phys. 404 (2020) 109136.
[30] M.A. Nabian, R.J. Gladstone, H. Meidani, Efficient training of physics-informed neural networks via importance sampling, Comput.-Aided Civ. Infrastruct. Eng. (2021).
[31] S. Wang, Y. Teng, P. Perdikaris, Understanding and mitigating gradient pathologies in physics-informed neural networks, arXiv preprint, arXiv:2001. 04536, 2020.
[32] S. Wang, X. Yu, P. Perdikaris, When and why pinns fail to train: a neural tangent kernel perspective, arXiv preprint, arXiv:2007.14527, 2020.
[33] Y. Shin, J. Darbon, G.E. Karniadakis, On the convergence of physics informed neural networks for linear second-order elliptic and parabolic type pdes, arXiv preprint, arXiv:2004.01806, 2020.
[34] S. Mishra, R. Molinaro, Estimates on the generalization error of physics informed neural networks (pinns) for approximating pdes, arXiv preprint, arXiv:2006.16144, 2020.
[35] Z.-Q.J. Xu, Frequency principle: Fourier analysis sheds light on deep neural networks, Commun. Comput. Phys. 28 (5) (2020) 1746–1767.
[36] Z.-Q.J. Xu, Y. Zhang, Y. Xiao, Training behavior of deep neural network in frequency domain, in: International Conference on Neural Information Processing, Springer, 2019, pp. 264–274.

[37] Y. Zhang, T. Luo, Z. Ma, Z.-Q.J. Xu, A linear frequency principle model to understand the absence of overfitting in neural networks, Chin. Phys. Lett. 38 (3) (2021) 038701.
[38] J. Arthur, G.G. Franck, H. Clement, Neural tangent kernel: convergence and generalization in neural networks, Adv. Neural Inf. Process. Syst. 31 (2018) 8571–8580.
[39] S. Arora, S. Du, W. Hu, Z. Li, R. Salakhutdinov, R. Wang, On exact computation with an infinitely wide neural net, Adv. Neural Inf. Process. Syst. 32 (2019) 8141–8150.
[40] J. Arthur, G.G. Franck, H. Clement, Freeze and chaos for dnns: an ntk view of batch normalization, checkerboard and boundary effects, arXiv:1907.05715, 2019.
[41] J. Huang, H. Yau, Dynamics of deep neural networks and neural tangent hierarchy, arXiv:1909.08156, 2019.
[42] J. Lee, L. Xiao, S. Schoenholz, Y. Bahri, R. Novak, J. Sohl-Dickstein, J. Pennington, Wide neural networks of any depth evolve as linear models under gradient descent, Adv. Neural Inf. Process. Syst. 32 (2019) 8572–8583.
[43] G. Yang, E.J. Hu, Feature learning in infinite-width neural networks, arXiv preprint, arXiv:2011.14522, 2020.
[44] D.P. Kingma, J. Ba Adam, A method for stochastic optimization, arXiv preprint, arXiv:1412.6980, 2014.