Commun. Comput. Phys.
doi: 10.4208/cicp.OA-2020-0187

Vol. **28**, No. 5, pp. 1886-1906
November 2020

A Multi-Scale DNN Algorithm for Nonlinear Elliptic Equations with Multiple Scales

Xi-An Li[1], Zhi-Qin John Xu[1,2,3,*] and Lei Zhang[1,2,3]

[1] *School of Mathematical Sciences, Shanghai Jiao Tong University, Shanghai 200240, China.*
[2] *Institute of Natural Sciences, Shanghai Jiao Tong University, Shanghai 200240, China.*
[3] *MOE-LSC, Shanghai Jiao Tong University, Shanghai 200240, China.*

Received 23 September 2020; Accepted (in revised version) 21 October 2020

Abstract. Algorithms based on deep neural networks (DNNs) have attracted increasing attention from the scientific computing community. DNN based algorithms are easy to implement, natural for nonlinear problems, and have shown great potential to overcome the curse of dimensionality. In this work, we utilize the multi-scale DNN-based algorithm (MscaleDNN) proposed by Liu, Cai and Xu (2020) to solve multi-scale elliptic problems with possible nonlinearity, for example, the p-Laplacian problem. We improve the MscaleDNN algorithm by a smooth and localized activation function. Several numerical examples of multi-scale elliptic problems with separable or non-separable scales in low-dimensional and high-dimensional Euclidean spaces are used to demonstrate the effectiveness and accuracy of the MscaleDNN numerical scheme.

AMS subject classifications: 65N30, 35J66, 41A46, 68T07

Key words: Multi-scale elliptic problem, p-Laplacian equation, deep neural network (DNN), variational formulation, activation function.

1 Introduction

In this paper, we will introduce a DNN based algorithm for the following elliptic equation with multiple scales and possible nonlinearity

$$\begin{cases} -\mathrm{div}\Big(a(\boldsymbol{x},\nabla u(\boldsymbol{x}))\Big) = f(\boldsymbol{x}), & \text{in } \Omega, \\ u(\boldsymbol{x}) = g(\boldsymbol{x}), & \text{on } \partial\Omega, \end{cases} \tag{1.1}$$

*Corresponding author. *Email addresses:* lixian9131@163.com, lixa0415@sjtu.edu.cn (X.-A. Li), xuzhiqin@sjtu.edu.cn (Z.-Q. J. Xu), Lzhang2012@sjtu.edu.cn (L. Zhang)

where $\Omega\subset\mathbb{R}^d$, $d\geq 2$, is a polygonal (polyhedral) domain (open, bounded and connected), $a(\boldsymbol{x},\nabla u(\boldsymbol{x})):\Omega\times\mathbb{R}^d\to\mathbb{R}^d$ is the flux function, and $f:\Omega\to\mathbb{R}$ is the source term.

Deep neural networks (DNNs) has not only achieved great successes in computer vision, natural language processing and other machine learning tasks [19, 28], but also captured great attention in the scientific computing community due to its universal approximating power, especially in high dimensional spaces [46]. It has found applications in the context of numerical solution of ordinary/partial differential equations, integral-differential equations and dynamical systems [16, 20, 26, 36, 41, 47].

Recent theoretical studies on DNNs have shed some light on the design of DNN-based algorithms for scientific computing tasks, in particular, for multi-scale problems. For example, the frequency principle (F-Principle) [15, 37, 44, 45], shows that, DNNs often fit target functions from low frequency components to high frequency ones, as opposed to the behavior of many conventional iterative numerical schemes (e.g., Gauss-Seidel method), which exhibit faster convergence for higher frequencies. To improve the convergence for high-frequency or multi-scale problems, a series of algorithms are developed to accelerate the learning of high-frequency components based on F-Principle [5, 6, 27, 30]. In particular, a multi-scale DNN algorithm(MscaleDNN) has achieved favourable performance boost for high-frequency problems [30]. The idea of the MscaleDNN to convert high-frequency contents into low-frequency ones as follows. The Fourier space is partitioned with respect to the radial direction. Since scaling input can shift the frequency distribution along the radial direction, a scaling down operation is used to scale the high-frequency components to low-frequency ones. Such radial scaling is independent of dimensionality, hence MscaleDNN is applicable for high-dimensional problems. Also, borrowing the multi-resolution concept of wavelet approximation theory using compact scaling and wavelet functions, an localized activation function (i.e., sReLU) was designed in previous work [30], which is a product of two ReLU functions. By setting multiple scalings in a MscaleDNN, numerical results in previous study [30] show that MscaleDNN is effective for linear elliptic partial differential equations with high frequencies.

We focus our exposition on the numerical method, and therefore restrict the flux function in (1.1) to the following Leray-Lions form [13] since it admits a natural variational form. Namely, for $(\boldsymbol{x},\boldsymbol{\xi})\in\Omega\times\mathbb{R}^d$, $a(\boldsymbol{x},\boldsymbol{\xi})=\kappa(\boldsymbol{x})\boldsymbol{\phi}'(|\boldsymbol{\xi}|)\frac{\boldsymbol{\xi}}{|\boldsymbol{\xi}|}$, where $\boldsymbol{\phi}\in C^2$ is the so-called $N-$function (an extension for the convex function with $\boldsymbol{\phi}'(0)=0$, see [13] for the precise definition). For p-Laplacian problem, $\boldsymbol{\phi}(t)=\frac{1}{p}t^p$, and when $p=2$ then $a(\boldsymbol{x},\boldsymbol{\xi})=\kappa(\boldsymbol{x})\boldsymbol{\xi}$ becomes linear. $\kappa(\boldsymbol{x})\in L^\infty(\Omega)$ is symmetric, uniformly elliptic on Ω, and may contain (non-separable) multiple scales. More general nonlinear flux will be treated in future work. With the above setup, the elliptic problem (1.1) is monotone and coercive, therefore it admits a unique solution. Those models have applications in many areas such as heterogeneous (nonlinear) materials [18], non-Newtonian fluids, surgical simulation, image processing, machine learning [40], *etc*.

In the last decades, much effort has been made for the numerical solution of the (1.1). In particular, for p-Laplacian equation with $\kappa(\boldsymbol{x})=1$, Some degrees of effectiveness can

be achieved by mesh based methods such as finite element method (FEM) [2,3,25], finite difference method (FDM) [31], discontinuous Galerkin method [12], and meshless methods [8, 29] *etc*.. In addition to those discretization methods, iterative methods such as preconditioned steepest descent, quasi-Newton or Newton method are employed to deal with the nonlinearity. The fully nonlinear problem (1.1) may become singular and/or degenerate, some regularization of the nonlinear operator needs to be added to ensure the convergence of the nonlinear iteration [25].

However, conventional methods cannot deal with the multiple scales, which is of great interest in applications for composite materials, geophysics, machine learning *etc* [17]. Homogenization method [11, 42] relies on the assumption of scale separation and periodicity. In addition, for nonlinear problems, one need to resort to a series of cell problems with the cell size going to infinity, which limits the practical utility of the method. In comparison, numerical homogenization methods, can solve linear multi-scale problems [14, 24, 34] and nonlinear multi-scale problems [1, 9] on the coarse scale, without resolving all the fine scales. Nonetheless, the aforementioned numerical methods are easy to implement in low-dimensional space $\mathbb{R}^d (d=1,2,3)$, however, they will encounter great difficulty in high-dimensions.

In this paper, based on the Deep Ritz method [16], we proposed an improved version of the MscaleDNN algorithm to solve elliptic problems (1.1) with multiple scales and/or nonlinearity. We improve the MscaleDNN by designing a new activation function due to the following intuition. The original activation function, i.e., sReLU, is localized only in the spatial domain due to the first-order discontinuity. However, the MscaleDNN requires the localization in the Fourier domain, which is equivalent to the smoothness in the spatial domain. Therefore, we design a smooth and localized activation function, which is a production of sReLU and the sine function, i.e., s2ReLU. In addition, our DNN structures also employ the residual connection technique, which was first proposed in [23] for image analysis and has become very popular due to its effectiveness. We employ this improved MscaleDNN to solve multi-scale elliptic problems, such as the multi-scale p-Laplacian equation, in various dimensions. Numerical experiments demonstrate that the algorithm is effective to obtain the oscillatory solution for multi-scale problems with or without nonlinearity, even in relative high dimensions. And the performance of s2ReLU activation function is much better than that of sReLU in the MscaleDNN framework.

The paper is structured as follows. In Section 2, we briefly introduce the framework of deep neural network approximation. Section 3 provides a variational formulation to solve the nonlinear multi-scale problem by MscaleDNN. In Section 4, some numerical experiments are carried out to demonstrate the effectiveness of our method. Concluding remarks are made in Section 5.

2 Deep neural networks and ResNet architecture

In recent years, the DNN has achieved great success in a wide range of applications, such as classification in complex systems and construction of response surfaces for high-

dimensional models. Mathematically, the DNN is a nested composition of sequential linear functions and nonlinear activation functions. A standard single layer network, e.g., the neural unit of a DNN with a d-dimensional vector $\boldsymbol{x}\in\mathbb{R}^d$ as its input and a m-dimensional vector as its output, is in the form of

$$\boldsymbol{y}=\sigma(\boldsymbol{W}\boldsymbol{x}+\boldsymbol{b}), \tag{2.1}$$

where $\boldsymbol{W}=(w_{ij})\in\mathbb{R}^{m\times d}$, $\boldsymbol{b}\in\mathbb{R}^m$ are denoted by weights and biases, respectively. $\sigma(\cdot)$ is an element-wise non-linear function, commonly known as the activation function. Various activation functions are proposed in machine learning literature, such as sigmoid, tanh, ReLU, Leaky-ReLU *etc* [21]. In DNN, the single layer (2.1) is also denoted as the hidden layer. Its output can be transformed through new weights, biases, and activation functions in the next layer.

Given an input datum $\boldsymbol{x}\in\mathbb{R}^d$, the output of a DNN, denoted by $\boldsymbol{y}(\boldsymbol{x};\boldsymbol{\theta})$, can be written as

$$\boldsymbol{y}(\boldsymbol{x};\boldsymbol{\theta})=\boldsymbol{W}^{[L]}\circ\sigma(\boldsymbol{W}^{[L-1]}\circ\sigma(\cdots\circ\sigma(\boldsymbol{W}^{[1]}\circ\sigma(\boldsymbol{W}^{[0]}\boldsymbol{x}+\boldsymbol{b}^{[0]})+\boldsymbol{b}^{[1]}))+\boldsymbol{b}^{[L-1]})+\boldsymbol{b}^{[L]}, \tag{2.2}$$

where $\boldsymbol{W}^{[l]}\in\mathbb{R}^{n_{l+1}\times n_l}$, $\boldsymbol{b}^{[l]}\in\mathbb{R}^{n_{l+1}}$ are the weights and biases of the l-th hidden layer, respectively. $n_0=d$, and "$\circ$" stands for the elementary-wise operation. $\boldsymbol{\theta}$ represents the set of parameters $\boldsymbol{W}^{[L]},\cdots\boldsymbol{W}^{[1]},\boldsymbol{W}^{[0]},\boldsymbol{b}^{[L]},\cdots\boldsymbol{b}^{[1]},\boldsymbol{b}^{[0]}$.

Many experiments have shown that the approximating capability of the DNN will became better and more robust with increasing depth. However, the problem of gradient explosion or vanishing might occur when the depth of DNNs increases, which will have a negative effect on the performance of the DNNs. ResNet (Residual Neural Network) [23] skillfully overcomes the vanishing (exploding) gradient problem in backpropagation by introducing a skip connection between input layer and output layer or some hidden layers. It makes the network easier to train, and also improves the performance of DNN. For example, it outperforms the VGG models and obtain excellent results by using extremely deep residual nets on the ImageNet classification data set. Mathematically, a Resnet unit with L layers produces a filtered version y_N for the input $\boldsymbol{y}^{[0]}$ is as follows

$$\boldsymbol{y}^{[\ell+1]}=\boldsymbol{y}^{[\ell]}+\sigma(\boldsymbol{W}^{[\ell+1]}\boldsymbol{y}^{[\ell]}+\boldsymbol{b}^{[\ell+1]}),\quad \text{for } \ell=0,1,2,\cdots,N-1.$$

In this work, we also employ the strategy of one-step skip connection for two consecutive layers in DNNs if they have the same number of neurons. For those consecutive layers with different neuron numbers, the skip connection step is omitted.

3 Unified DNN model for multi-scale problems with scale transformation

The multi-scale elliptic problem (1.1) with N-function $\boldsymbol{\phi}$ admits a natural variational form [10]. Define the energy functional as

$$\mathcal{J}(v):=\int_\Omega\kappa(\boldsymbol{x})\boldsymbol{\phi}(|\nabla v|)d\boldsymbol{x}-\int_\Omega fvd\boldsymbol{x}, \tag{3.1}$$

where v is the trial function in the admissible Orlicz-Sobolev space $V := W_g^{1,\boldsymbol{\phi}}(\Omega)$ [13] where the subscript $_g$ means that the trace on $\partial\Omega$ is g. The solution of (1.1) can be obtained by minimizing $\mathcal{J}(v)$, i.e.,

$$u = \underset{v\in V}{\arg\min}\,\mathcal{J}(v). \tag{3.2}$$

Therefore, we can employ the Deep Ritz method to solve (3.2), which is an efficient approach to solve variational problems that stem from generally partial differential equations [16].

We consider an ansatz $y(\cdot;\boldsymbol{\theta})$ represented by a DNN with parameter θ as the trial function in the variational problem (3.2), where $\boldsymbol{\theta}\in\Theta$ denotes the parameters of the underlying DNN. Substituting $y(\boldsymbol{x};\boldsymbol{\theta})$ into (3.1) and (3.2), we can obtain the following equation

$$u = \underset{y\in V}{\arg\min}\left[\int_\Omega \kappa(\boldsymbol{x})\boldsymbol{\phi}(|\nabla y(\boldsymbol{x};\boldsymbol{\theta})|)d\boldsymbol{x} - \int_\Omega f(\boldsymbol{x})y(\boldsymbol{x};\boldsymbol{\theta})d\boldsymbol{x}\right]. \tag{3.3}$$

Similar to the general strategy of searching a solution which satisfying boundary conditions of (1.1) in the admissible space V [10, 16], we further approximate the integral by Monte Carlo method [38] and convert the minimization problem with respect to $y\in V$ to an equivalent one with respect to the parameters $\boldsymbol{\theta}$,

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}\in\Theta}{\arg\min}\frac{1}{n_{it}}\sum_{i=1}^{n_{it}}\left[\kappa(\boldsymbol{x}_I^i)\boldsymbol{\phi}(|\nabla y(\boldsymbol{x}_I^i;\boldsymbol{\theta})|) - f(\boldsymbol{x}_I^i)y(\boldsymbol{x}_I^i;\boldsymbol{\theta})\right] \quad \text{for } \boldsymbol{x}_I^i\in\Omega. \tag{3.4}$$

and $y(\boldsymbol{x},\theta) = g(\boldsymbol{x})$ for $\boldsymbol{x}\in\partial\Omega$.

Boundary conditions are indispensable constraints for numerical solution of PDEs. Analogously, imposing boundary conditions is also an important issue in DNN representation. We approximate the squared L^2 norm of the boundary discrepancy $\int_{\partial\Omega}(y(\boldsymbol{x},\theta) - g(\boldsymbol{x}))^2$ by a Monte Carlo approximation,

$$\frac{1}{n_{bd}}\sum_{j=1}^{n_{bd}}\left[y(\boldsymbol{x}_B^j;\boldsymbol{\theta}) - g(\boldsymbol{x}_B^j)\right]^2 \quad \text{for } \boldsymbol{x}_B^j\in\partial\Omega. \tag{3.5}$$

We define the following loss function where the boundary condition is treated as a penalty term with parameter β,

$$\mathcal{L}(\boldsymbol{\theta};\mathcal{X}_I,\mathcal{X}_B) = \underbrace{\frac{1}{n_{it}}\sum_{i=1}^{n_{it}}\left[\kappa(\boldsymbol{x}_I^i)\boldsymbol{\phi}(|\nabla y(\boldsymbol{x}_I^i;\boldsymbol{\theta})|) - f(\boldsymbol{x}_I^i)y(\boldsymbol{x}_I^i;\boldsymbol{\theta})\right]}_{\text{loss\_it}} + \underbrace{\frac{\beta}{n_{bd}}\sum_{j=1}^{n_{bd}}\left[y(\boldsymbol{x}_B^j;\boldsymbol{\theta}) - g(\boldsymbol{x}_B^j)\right]^2}_{\text{loss\_bd}} \tag{3.6}$$

where $\mathcal{X}_I = \{\boldsymbol{x}_I^i\}_{i=1}^{n_{it}}$ and $\mathcal{X}_B = \{\boldsymbol{x}_B^j\}_{j=1}^{n_{bd}}$ represent the training data in the iterior of Ω and on the boundary $\partial\Omega$, respectively. loss_it and loss_bd stand for the loss computed with the

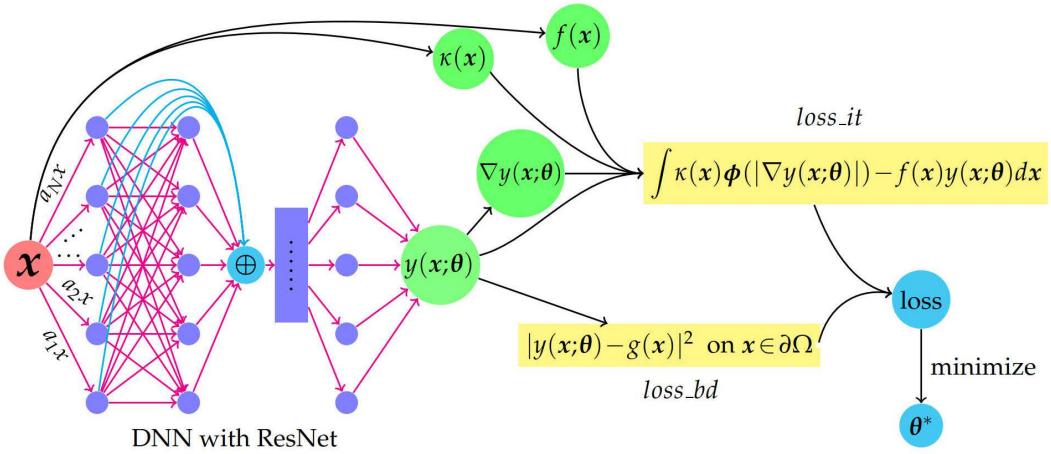


Figure 1: Schematic of a DNN for solving the nonlinear multi-scale problem.

interior points and the boundary points, respectively. The penalty parameter β controls the relative contribution of loss_bd in the loss function. It increases gradually within the training process in order to better enforce the boundary condition. Our goal is to find a set of parameters $\boldsymbol{\theta}$ such that the approximate function $y(\boldsymbol{x};\boldsymbol{\theta})$ minimizes the loss function $\mathcal{L}(\boldsymbol{\theta})$. If $\mathcal{L}(\boldsymbol{\theta})$ is small enough, then $y(\boldsymbol{x};\boldsymbol{\theta})$ will be a "good" approximate solution of (1.1), i.e.,

$$\boldsymbol{\theta}^* = \arg\min \mathcal{L}(\boldsymbol{\theta}) \iff u(\boldsymbol{x}) \approx y(\boldsymbol{x};\boldsymbol{\theta}^*).$$

In order to obtain $\boldsymbol{\theta}^*$, one can update the parameter $\boldsymbol{\theta}$ by the gradient descent method over the all training examples at each iteration. The objective function decreases along a descent direction w^k after an iteration, i.e., $\mathcal{L}(\boldsymbol{\theta}^{k+1}) < \mathcal{L}(\boldsymbol{\theta}^k)$, where $\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^k + \eta w^k$ with some properly chosen learning rate or step size η. Since DNNs are highly non-convex, $\boldsymbol{\theta}^*$ may only converge to a local minimum. Stochastic gradient descent (SGD), as the common optimization technique of deep learning, has been proven to be very effective in practice (can avoid the problem of local minimum) and is a fundamental building block of nearly all deep learning models. In the implementation, SGD method chooses a "mini-batch" of data $\mathcal{X}_m^k$ (a subset of interior points and boundary points in our case) at each step. In this context, the SGD is given by:

$$\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^k - \eta^k \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}^k; \mathcal{X}_m^k),$$

where the "learning rate" η^k decreases with k increasing.

Remark 3.1. The $\boldsymbol{\theta}$ consist of weights and biases in our model are initialized by using the normal distribution $\mathcal{N}\big(0, (\frac{2}{n_{in}+n_{out}})^2\big)$, where n_{in} and n_{out} are the input and output dimensions of the corresponding layer, respectively.

3.1 MscaleDNN

A conventional DNN model can achieve a satisfactory solution for PDE problems when the coefficient $\kappa(\boldsymbol{x})$ is homogeneous (e.g., smooth or possessing few scales) [4,22,39,43]. However, it is difficult to solve PDEs (1.1) with multi-scale $\kappa(\boldsymbol{x})$ due to the complex interaction of nonlinearity and multiple scales. The MscaleDNN architecture has been proposed to approximate the solution with high frequency and multiple scales by converting original data to a low frequency space [30], as described in the following:

- Divide the neurons in the first hidden-layer into N groups, and generate the scale vector
$$K=(a_1,\cdots,a_1,a_2,\cdots,a_2,\cdots,a_N,\cdots,a_N)^T,\quad a_i\geqslant 1. \tag{3.7}$$
Note that the scale parameters a_i's are hyper-parameters and can be set by several methods. Numerical results in previous work [7,30] show that the effectiveness of the MscaleDNN is not sensitive to the selection of scale parameters if it covers the scales of the target function.

- Convert the input data $\boldsymbol{x}$ to $\tilde{\boldsymbol{x}}=K\odot\boldsymbol{x}$, then feed $\tilde{\boldsymbol{x}}$ to the first hidden-layer of DNN, where $\odot$ is the Hadamard product. From the viewpoint of Fourier transformation and decomposition, the Fourier transform $\hat{f}(k)$ of a given band-limited function $f(\boldsymbol{x})$ has a compact support $B(K_{\max})=\{k\in\mathbb{R}^d,\ |k|\leqslant K_{\max}\}$ which can be partitioned as the union of M concentric annulus with uniform or nonuniform width, e.g.,
$$P_i=\Big\{k\in\mathbb{R}^d,\ (i-1)K_0\leqslant|k|\leqslant iK_0\Big\},\quad K_0=K_{\max}/M,\quad i\leqslant i\leqslant M,$$
then $\hat{f}(k)$ can be expressed as follows
$$\hat{f}(k)=\sum_i^M\hat{f}_i(k),\quad \text{with}\quad \text{supp}\hat{f}_i(k)\subset P_i.$$
By the down-scaling operation which shift the high frequency region P_i into a low frequency ones, the scale transform reads,
$$\hat{f}_i^{(scale)}(k)=\hat{f}(a_ik),\quad a_i>1,$$
and
$$f_i^{(scale)}(\boldsymbol{x})=f_i\left(\frac{1}{a_i}\boldsymbol{x}\right)$$
or
$$f_i(\boldsymbol{x})=f_i^{(scale)}(a_i\boldsymbol{x}).$$
Then, instead of finding a function $\hat{f}_i(k)$ in the support set of P_i, the transformed function $\hat{f}_i^{(scale)}(k)$ will be explored in
$$\text{supp}\hat{f}_i^{(scale)}(k)\subset\left\{k\in\mathbb{R}^d,\ \frac{(i-1)K_0}{\alpha_i}\leqslant|k|\leqslant\frac{iK_0}{\alpha_i}\right\}.$$

When $\frac{iK_0}{\alpha_i}$ is small, a DNN $\ell_i(\theta)$ can be used to learn $f_i^{(scale)}(x)$ quickly, which further means that $\ell_i(\theta)$ can approximate $f_i(\boldsymbol{x})$ immediately, i.e.,

$$f_i(\boldsymbol{x}) \approx \ell_i(a_i\theta).$$

Finally, $f(\boldsymbol{x})$ can be expressed by

$$f(\boldsymbol{x}) \approx \sum_i^M \ell_i(a_i\theta). \tag{3.8}$$

In general, (3.8) suggests an ansatz to approximate the solution more quickly with DNN representations, hence, converting the original data $\boldsymbol{x}$ into $\tilde{\boldsymbol{x}}$ is a nice trick when dealing with multi-scale problems.

- Output the result of DNN.

3.2 Activation function

The role of activation functions is to decide whether particular neuron should fire or not. When the activation function is absent, the neural network will simply be a linear transformation involving weights and biases, which in turn becomes a linear regression model. Although linear model is simple to solve, but its expressive power for complex problems is limited. A nonlinear activation function performs the nonlinear transformation of the input data, making it capable to learn and perform more complex tasks. Thus, choosing the right activation function is essential for the efficiency and accuracy of the DNN. The significance of activation functions for different models have been investigated in, e.g., [27, 35] *etc*.

In the previous work [30], the activation function $\text{sReLU}(x) = \text{ReLU}(x) \times \text{ReLU}(1-x)$ smoother than $\text{ReLU}(x) = \max\{0, x\}$ is used in the MscaleDNN algorithm to solve PDEs. It is localized in the spatial domain due to the first-order discontinuity, but it lacks of adequate smoothness. To improve the efficiency of scale separation, we propose a smoother activation function which is a production of sReLU and the sine function, and is referred to sin-sReLU,

$$\text{sin-sReLU}(x) = \sin(2\pi x) * \text{ReLU}(x) * \text{ReLU}(1-x). \tag{3.9}$$

For convenience, we abbreviate it by s2ReLU here and thereafter.

As shown in Figs. 2(a) and 2(b), sReLU and s2ReLU are localized in the spatial domain. For sReLU, since its first-order derivative is discontinuous, its Fourier transform decays slowly. As shown in Fig. 2(c), sReLU functions with different scales overlap with each other. However, since s2ReLU is a smooth function, it decays faster and has better localization property in the frequency domain, compared with sReLU, as shown in

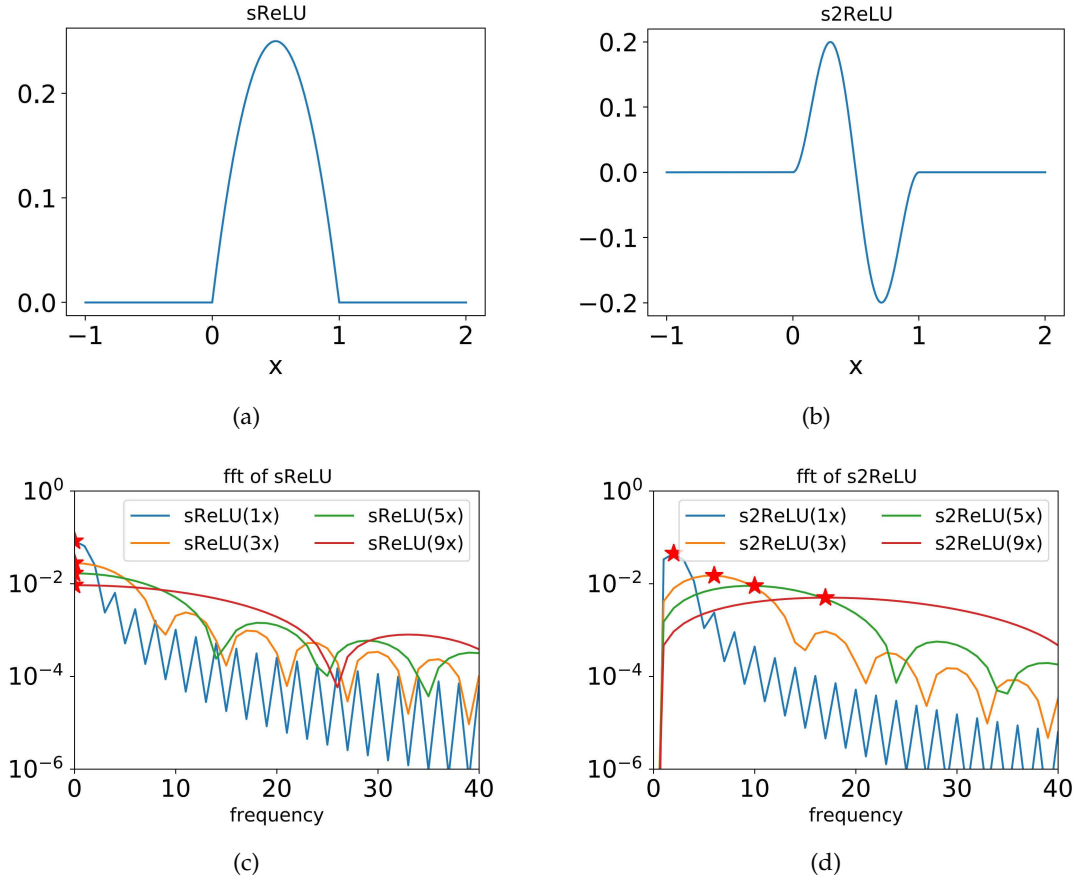


Figure 2: sReLU function (left) and s2ReLU function (right) in the spatial (upper) and frequency (lower, the peak of each line is indicated by a star) domains.

Fig. 2(d). The localization in the frequency domain, leads to the fact, that the peak-amplitude frequencies of different scaled s2ReLU functions (stars in Fig. 2(d)) are separated and increase as the scales increase. Therefore, s2ReLU potentially could be more efficient to represent multi-scale functions. In the numerical experiments, we will show that s2ReLU has more superior performance compared to sReLU with MscaleDNN or ReLU with conventional DNN.

4 Numerical experiments

In this section, several test problems are presented to illustrate the effectiveness of our method to solve multi-scale nonlinear problems. In our numerical experiments, all training and testing data are generated with a uniform distribution over corresponding domains, and all networks are trained by the Adam optimizer. In addition, the initial learn-

ing rate is set as 2×10^{-4} with a decay rate 5×10^{-5} for each training step. For the first hidden-layer in MscaleDNN, we divide the neurons into $N=100$ groups to generate the scale vector $K=\{1,1,2,3,\cdots,99\}$ as in (3.7). Here, we provide two criteria to evaluate our model:

$$MSE=\frac{1}{n_s}||\tilde{u}-u^*||_2^2, \quad \text{and} \quad REL=\frac{1}{n_s}\frac{||\tilde{u}-u^*||_2^2}{||u^*||_2^2},$$

where $\tilde{u}$ and u^* are approximate solutions of deep neural network and the exact solution (or the reference solution computed on a very fine mesh), respectively, at n_s testing sample points. To evaluate the effectiveness, we test our model for every 1000 epochs in the training process. In our work, the penalty parameter β is set as follows,

$$\beta=\begin{cases}\beta_0, & \text{if } i<M_{\max}*0.1,\\ 10\beta_0, & \text{if } M_{\max}*0.1<=i<M_{\max}*0.2,\\ 50\beta_0, & \text{if } M_{\max}*0.2<=i<M_{\max}*0.25,\\ 100\beta_0, & \text{if } M_{\max}*0.25<=i<M_{\max}*0.5,\\ 200\beta_0, & \text{if } M_{\max}*0.5<=i<M_{\max}*0.75,\\ 500\beta_0, & \text{otherwise,}\end{cases} \tag{4.1}$$

where the $\beta_0=1000$ in all our tests and $M_{\max}$ represents the total number of epoches. We perform neural network training and testing in TensorFlow (version 1.4.0) on a work station with 256GB RAM and a single NVIDIA GeForce GTX 2080Ti 12GB.

For the numerical examples, we use the following p-Laplacian problem as an prototypical example of the more general form (1.1),

$$\begin{cases}-\text{div}\Big(\kappa(\boldsymbol{x})|\nabla u(\boldsymbol{x})|^{p-2}\nabla u(\boldsymbol{x})\Big)=f(\boldsymbol{x}), & \boldsymbol{x}\in\Omega,\\ u(\boldsymbol{x})=g(\boldsymbol{x}), & \boldsymbol{x}\in\partial\Omega,\end{cases} \tag{4.2}$$

then the energy in (3.1) can be rewritten as

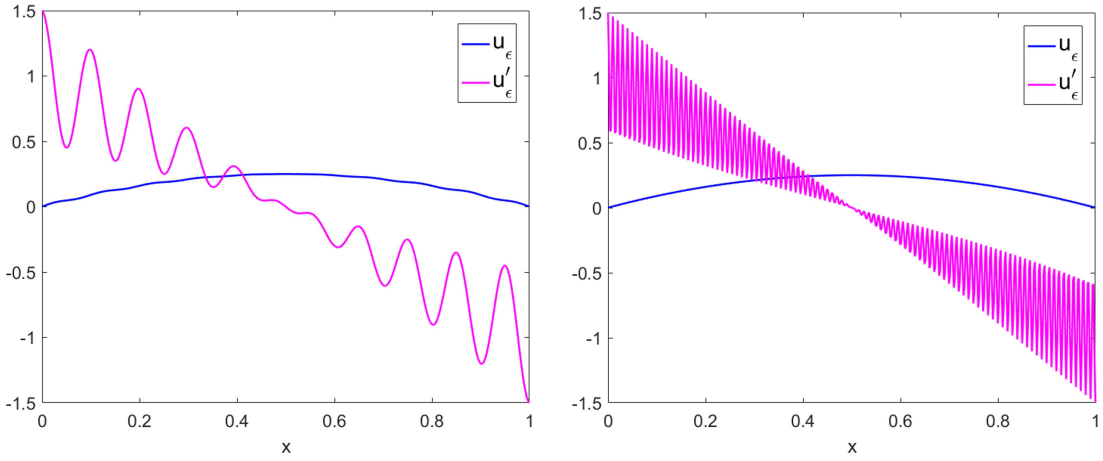
$$\mathcal{J}(v):=\frac{1}{p}\int_\Omega\kappa(\boldsymbol{x})|\nabla v|^p d\boldsymbol{x}-\int_\Omega fv d\boldsymbol{x}, \quad v\in V,$$

with $V:=W_g^{1,p}$, namely, the Sobolev space $W^{1,p}$ with trace g on $\partial\Omega$.

4.1 One dimensional examples

We consider the following one-dimensional highly oscillatory elliptic problem in $\Omega=[0,1]$.

$$\begin{cases}-\dfrac{d}{dx}\left(\kappa(x)\left|\dfrac{d}{dx}u_\epsilon(x)\right|^{p-2}\dfrac{d}{dx}u_\epsilon(x)\right)=f(x),\\ u_\epsilon(0)=u_\epsilon(1)=0.\end{cases} \tag{4.3}$$



(a) $\epsilon=0.1$ (b) $\epsilon=0.01$

Figure 3: The graphs for the original function and the derivative of u_ϵ.

For Eq. (4.3), we consider $\epsilon=0.1$ and $\epsilon=0.01$. We use the MscaleDNN models with activation functions sReLU and s2ReLU to solve this problem, respectively. In addition, a DNN model with ReLU is used as a baseline for comparison. At each training step, we uniformly sample $n_{it}=3000$ interior points in Ω and $n_{bd}=500$ boundary points on $\partial\Omega$ as the training dataset, and uniformly sample $n_s=1000$ points in Ω as the testing dataset.

Example 4.1. We consider the case of $p=2$ for the linear diffusion problem with highly oscillatory coefficients (4.3). $f\equiv 1$ and

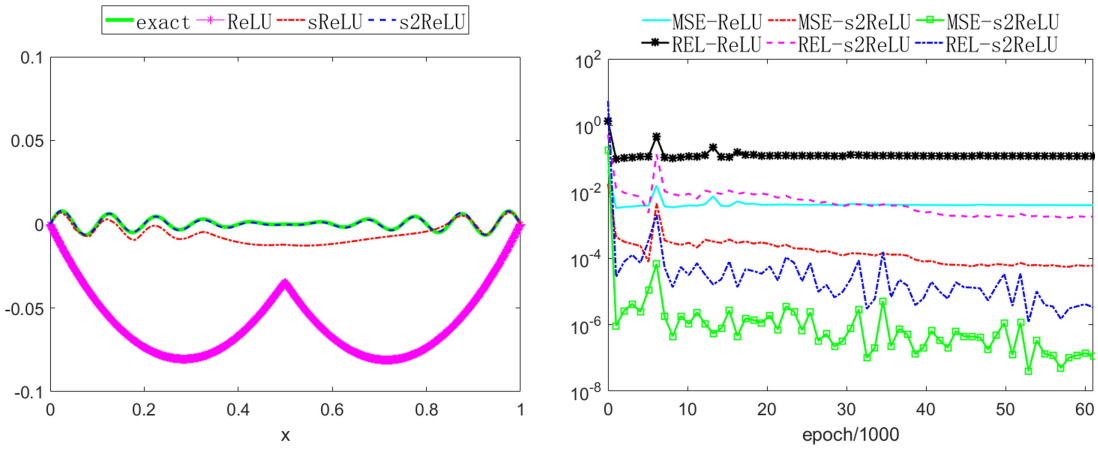
$$\kappa(x)=\left(2+\cos\left(2\pi\frac{x}{\epsilon}\right)\right)^{-1}, \tag{4.4}$$

with a small parameter $\epsilon>0$ such that $\epsilon^{-1}\in\mathbb{N}^+$. In one-dimensional setting, the corresponding unique solution is given by

$$u_\epsilon(x)=x-x^2+\epsilon\left(\frac{1}{4\pi}\sin\left(2\pi\frac{x}{\epsilon}\right)-\frac{1}{2\pi}x\sin\left(2\pi\frac{x}{\epsilon}\right)-\frac{\epsilon}{4\pi^2}\cos\left(2\pi\frac{x}{\epsilon}\right)+\frac{\epsilon}{4\pi^2}\right). \tag{4.5}$$

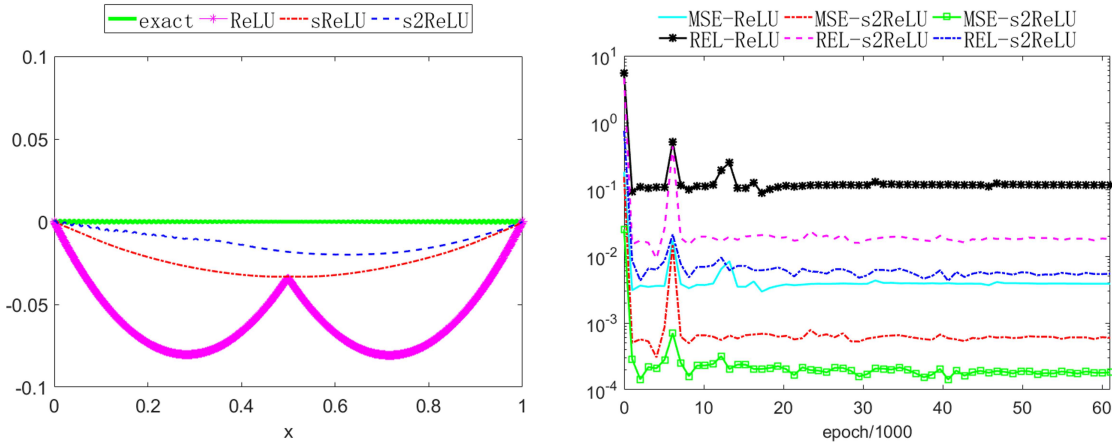
Since the oscillation amplitude is small, to show the highly oscillation, we display the first-order derivative of the target functions for $\epsilon=0.1$ and $\epsilon=0.01$ in Fig. 3, respectively.

Although the p-Laplacian equation is reduced to a linear one, the problem is still difficult to deal with by DNN due to the highly oscillatory coefficients with small ϵ [45]. Since the solution is a smooth $\mathcal{O}(1)$ function with a oscillating perturbation of $\mathcal{O}(\epsilon)$ for our one-dimensional problems, in the following, we then only illustrate the $\mathcal{O}(\epsilon)$ parts of the solutions by subtracting $u(x)-x(1-x)$. For $\epsilon=0.1$ as shown in Fig. 4(a), the solution of the MscaleDNN with activation function s2ReLU overlaps with the exact solution,



(a) solution (b) MSE and REL

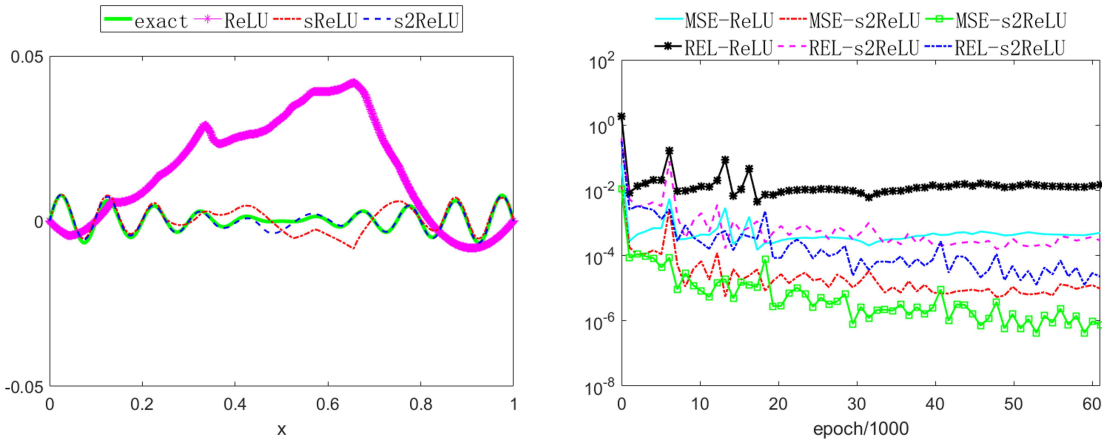
Figure 4: Testing results for $\epsilon=0.1$ when $p=2$. The network size is $(300,200,150,150,100,50,50)$.



(a) solution (b) MSE and REL

Figure 5: Testing results for $\epsilon=0.01$ when $p=2$. The network size is $(500,400,300,300,200,100,100)$.

while the one with sReLU deviates from the exact solution at the central part and the one with ReLU is completely different from the exact solution. As shown in Fig. 4(b), both the error and the relative error consistently show that MscaleDNN with s2ReLU can resolve the solution pretty well. For the case of $\epsilon=0.01$ in Fig. 5(a), the s2ReLU solution and the sReLU solution both deviate from the exact solution at the central part of $(0,1)$, but the s2ReLU solution still outperforms that of sReLU. The error curves in Fig. 5(b) enhance this conclusion. Figs. 4 and 5 clearly reveal that the performances of MscaleDNN model with s2ReLU and sReLU are superior to that of general DNN model with ReLU.



(a) solution (b) MSE and REL

Figure 6: Testing results for $\epsilon=0.1$ when $p=5$. The network size is $(300,200,150,150,100,50,50)$.

When p increases, the nonlinearity of the p-Laplacian problem (1.1) becomes more and more significant and has complex interaction with the highly oscillatory coefficients, hence the solution becomes increasingly more difficult. In following examples, we further consider the 1d example (4.3) with $p=5$, respectively.

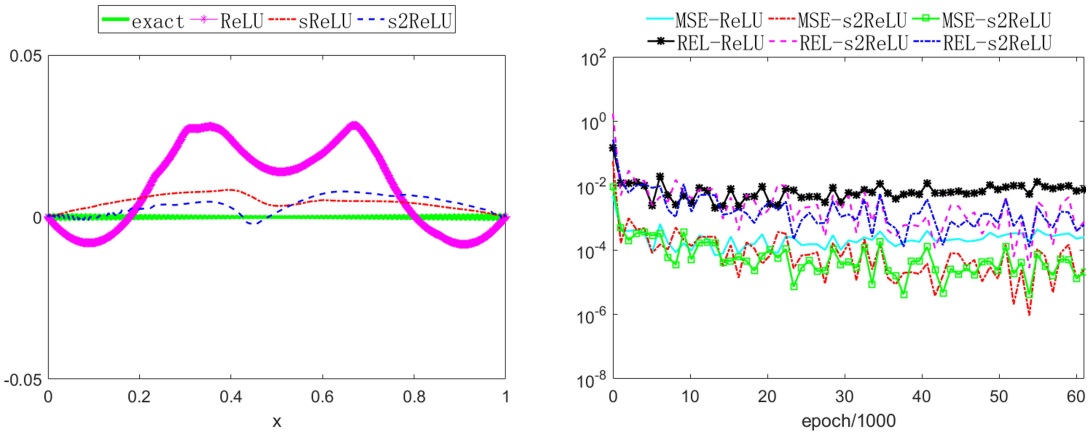
Example 4.2. For $p=5$, this p-Laplacian equation is a highly oscillatory diffusion problem. The exact solution $u_\epsilon(x)$ and $\kappa(x)$ are the same as that of Example 4.1. The force side f is given by

$$f(x)=\frac{-|2x-1|^3\left[2+\cos(2\pi\frac{x}{\epsilon})\right]^2\left[3\pi(2x-1)\sin\left(2\pi\frac{x}{\epsilon}\right)-4\epsilon\cos\left(2\pi\frac{x}{\epsilon}\right)-8\epsilon\right]}{8\epsilon}, \tag{4.6}$$

where $\epsilon>0$ and $\epsilon^{-1}\in\mathbb{N}^+$.

We show the testing results for $\epsilon=0.1$ and $\epsilon=0.01$ in Figs. 6 and 7, respectively. The MscaleDNN with activation function s2ReLU can well capture all the oscillation of the exact solution for $\epsilon=0.1$ in Fig. 6(a), which is much better than that of sReLU and ReLU, and the test error of s2ReLU is much lower as shown in Fig. 6(b). For $\epsilon=0.01$, MscaleDNNs still outperform activation function ReLU, while s2ReLU and sReLU are comparable, as shown in Fig. 7.

From the above results, we conclude that the MscaleDNN model with s2ReLU activation function can much better solve the p-Laplacian problem compared with the ones of sReLU and ReLU, even for a nonlinear case.



(a) solution (b) MSE and REL

Figure 7: Testing results for $\epsilon=0.01$ when $p=5$. The network size is $(500,400,300,300,200,100,100)$.

4.2 Two dimensional examples

We consider the following p-Laplacian problem in domain $\Omega=[-1,1]\times[-1,1]$,

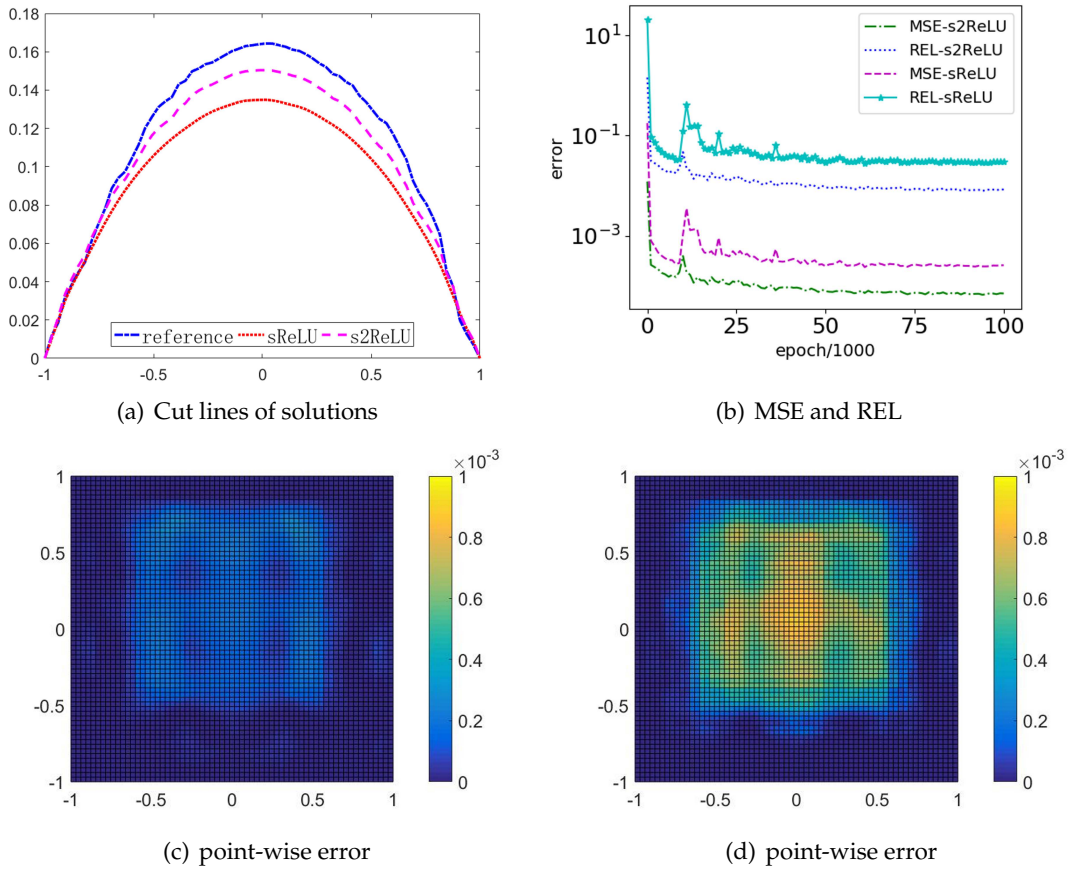
$$\begin{cases} -\mathrm{div}\left(\kappa(x_1,x_2)|\nabla u(x_1,x_2)|^{p-2}\nabla u(x,y)\right)=f(x_1,x_2), \\ u(-1,x_2)=u(1,x_2)=0, \quad u(x_1,-1)=u(x_1,1)=0. \end{cases} \tag{4.7}$$

In the following tests, we obtain the solution of (4.7) by employing two types of MscaleDNN with size $(1000,500,400,300,300,200,100,100)$ and activation functions sReLU and s2ReLU, respectively. Based on the conclusions of MscaleDNN for one-dimensional p-Laplacian problems and previous results for MscaleDNN in solving PDEs [30], a MscaleDNN with s2ReLU or sReLU outperforms DNN with ReLU, therefore, we will not show the results of DNN with ReLU in the following experiments.

Example 4.3. In this example, the forcing term $f(x_1,x_2)\equiv 1$ for $p=2$ and a multi-scale trigonometric coefficient $\kappa(x_1,x_2)$ is given by

$$\begin{aligned}\kappa(x_1,x_2)=\frac{1}{6}\Bigg(&\frac{1.1+\sin(2\pi x_1/\epsilon_1)}{1.1+\sin(2\pi x_2/\epsilon_1)}+\frac{1.1+\sin(2\pi x_1/\epsilon_2)}{1.1+\cos(2\pi x_2/\epsilon_2)}+\frac{1.1+\cos(2\pi x_1/\epsilon_3)}{1.1+\sin(2\pi x_2/\epsilon_3)}\\&+\frac{1.1+\sin(2\pi x_1/\epsilon_4)}{1.1+\cos(2\pi x_2/\epsilon_4)}+\frac{1.1+\cos(2\pi x_1/\epsilon_5)}{1.1+\sin(2\pi x_2/\epsilon_5)}+\sin(4x_1^2x_2^2)+1\Bigg),\end{aligned}$$

where $\epsilon_1=\frac{1}{5}$, $\epsilon_2=\frac{1}{13}$, $\epsilon_3=\frac{1}{17}$, $\epsilon_4=\frac{1}{31}$, $\epsilon_5=\frac{1}{65}$. For this example, the corresponding exact solution can not be expressed explicitly. Alternatively, a reference solution $u(x_1,x_2)$ is set as the finite element solution computed by numerical homogenization method [32–34] on a square grid $[-1,1]\times[-1,1]$ of mesh-size $h=(1+2^q)^{-1}$ with a positive integer $q=6$.



(a) Cut lines of solutions

(b) MSE and REL

(c) point-wise error

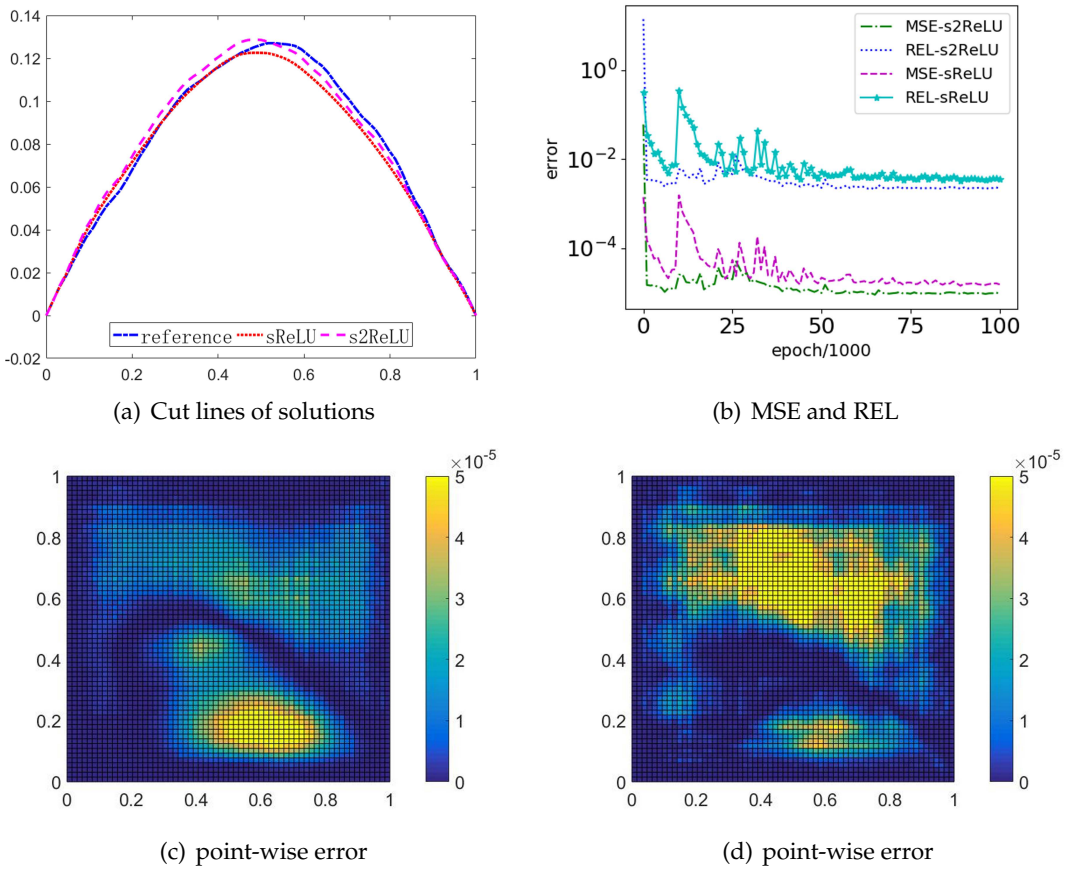
(d) point-wise error

Figure 8: Testing results for Example 4.3. 8(a): Cut lines along $x=0$ for reference solution, s2ReLU solution and sReLU solution, respectively. 8(b): Mean square error and relative error for s2ReLU and sReLU, respectively. 8(c): Point-wise square error for s2ReLU. 8(d): Point-wise square error for sReLU.

At each training step, we randomly sample $n_{it}=3000$ interior points and $n_{bd}=500$ boundary points as training dataset. The testing dataset are also sampled from a square grid $[-1,1]\times[-1,1]$ of mesh-size $h=(1+2^q)^{-1}$ with $q=6$.

As shown in Figs. 8(a) and 8(b), for the high-frequency oscillatory coefficient $\kappa(x_1,x_2)$ in this example, the performances of our model with s2ReLU and sReLU are still favorable to solve (4.7) and our s2ReLU performs better than sReLU in overall training process. Figs. 8(c) and 8(d) not only show that the point-wise errors for major points are closed to zero, but also reveal that the point-wise error of s2ReLU is smaller than that of sReLU. In short, our model with s2ReLU activation function can obtain a satisfactory solution for p-Laplacian problem and it outperforms the one of sReLU.

Example 4.4. In this example, we test the performance of MscaleDNN to p-Laplacian problem for $p=3$. The forcing term $f(x_1,x_2)$ and $\kappa(x_1,x_2)$ are similar to that in Example



(a) Cut lines of solutions (b) MSE and REL

(c) point-wise error (d) point-wise error

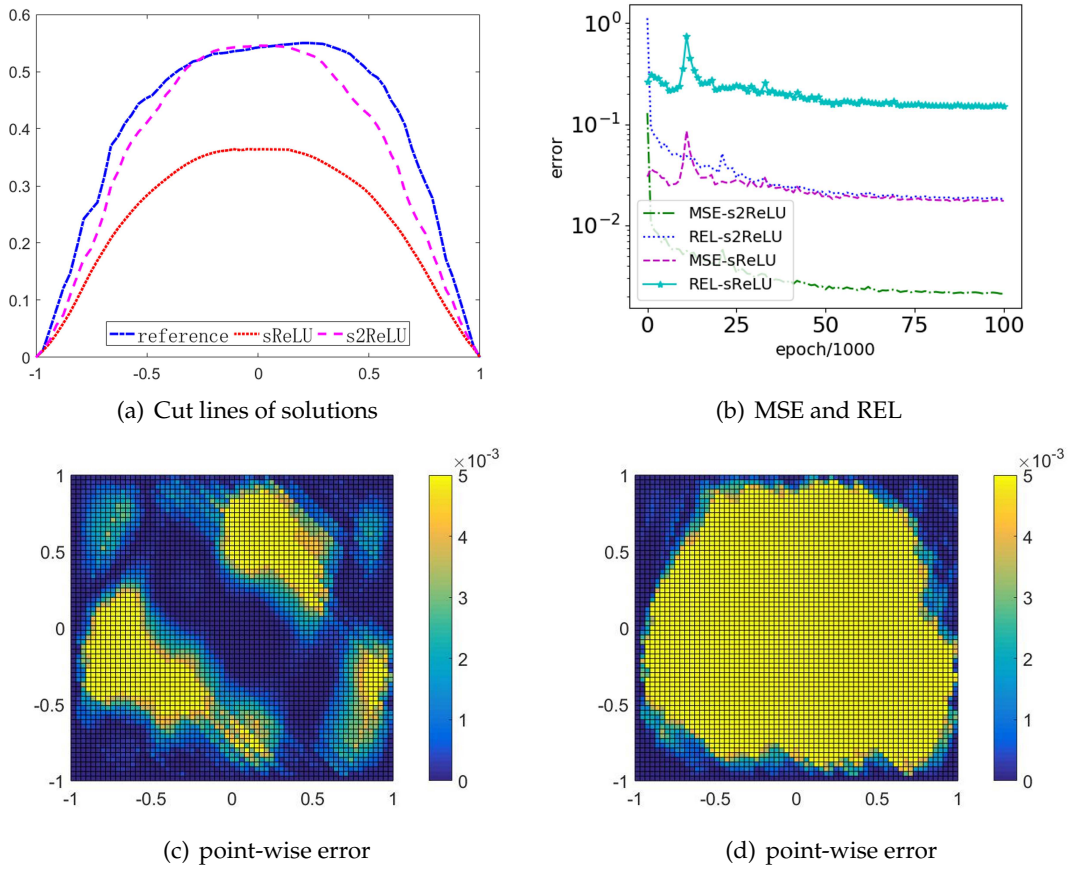
Figure 9: Testing results for Example 4.4. 9(a): Cut lines along $x=0.5$ for reference solution, s2ReLU solution and sReLU solution, respectively. 9(b): Mean square error and relative error for s2ReLU and sReLU, respectively. 9(c): Point-wise square error for s2ReLU. 9(d): Point-wise square error for sReLU.

4.3. Analogously, we still take the reference solution u as the finite element solution on a fine mesh over the square domain $[0,1]\times[0,1]$ of mesh-size $h=(1+2^q)^{-1}$ with a positive integer $q=6$. In addition, the training and testing datasets in this example are similarly constructed as Example 4.3.

From the results in Fig. 9, the performance of MscaleDNN with s2ReLU is also superior to the one of sReLU. The overall errors (including MSE and REL) of both activation functions are comparable, but the point-wise error of s2ReLu is smaller than that of sReLU.

Example 4.5. In this example, we take the forcing term $f=1$ for $p=2$, and

$$\kappa(x_1,x_2)=\Pi_{k=1}^{q}\Big(1+0.5\cos\Big(2^k\pi(x_1+x_2)\Big)\Big)\Big(1+0.5\sin\Big(2^k\pi(x_2-3x_1)\Big)\Big),$$



(a) Cut lines of solutions (b) MSE and REL

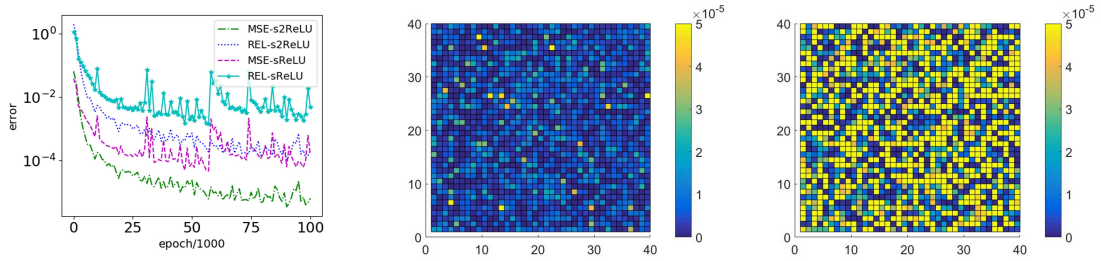
(c) point-wise error (d) point-wise error

Figure 10: Testing results for Example 4.5. 10(a): Cut lines along $x=0$ for reference solution, s2ReLU solution and sReLU solution, respectively. 10(b): Mean square error and relative error for s2ReLU and sReLU, respectively. 10(c): Point-wise square error for s2ReLU. 10(d): Point-wise square error for sReLU.

where q is a positive integer. The coefficient $\kappa(x_1,x_2)$ has non-separable scales. Similarly to Example 4.3, we take the reference solution u as the finite element solution on a fine mesh over the square domain $[-1,1]\times[-1,1]$ of mesh-size $h=(1+2^q)^{-1}$ with a positive integer $q=6$.

In this example, the training and testing datasets are similarly constructed as Example 4.3. In Figs. 10(a) and 10(b), the s2ReLU solution approximates the exact solution much better than that of sReLU solution. This can be clearly seen from the point-wise error in Figs. 10(c) and 10(d).

Based on the results of the two dimensional examples 4.3, 4.4 and 4.5, it is clear that the MscaleDNN model with s2ReLU activation function can approximate the solution of multiscale elliptic problems with oscillating coefficients with possible nonlinearity, and its performance is superior to that of sReLU. It is important to examine the capability of



(a) MSE and REL (b) point-wise error (c) point-wise error

Figure 11: Testing results for Example 4.6. 11(a): Mean square error and relative error for s2ReLU and sReLU, respectively. 11(b): Point-wise square error for s2ReLU. 11(c): Point-wise square error for sReLU.

MscaleDNN for high-dimensional (multi-scale) elliptic problems, which will be shown in the following.

4.3 High dimensional examples

Example 4.6. We consider the following p-Laplacian problem in domain $\Omega=[0,1]^5$

$$\begin{cases} -\mathrm{div}\Big(\kappa(x_1,x_2,\cdots,x_5)|\nabla u(x_1,x_2,\cdots,x_5)|^{p-2}\nabla u(x_1,x_2,\cdots,x_5)\Big)=f(x_1,x_2,\cdots,x_5), \\ u(0,x_2,\cdots,x_5)=u(1,x_2,\cdots,x_5)=0, \\ \qquad\qquad \cdots\cdots \\ u(x_1,x_2,\cdots,0)=u(x_1,x_2,\cdots,1)=0. \end{cases} \tag{4.8}$$

In this example, we take $p=2$ and

$$\kappa(x_1,x_2,\cdots,x_5)=1+\cos(\pi x_1)\cos(2\pi x_2)\cos(3\pi x_3)\cos(2\pi x_4)\cos(\pi x_5).$$

We choose the forcing term f such that the exact solution is

$$u(x_1,x_2,\cdots,x_5)=\sin(\pi x_1)\sin(\pi x_2)\sin(\pi x_3)\sin(\pi x_4)\sin(\pi x_5).$$

For five-dimensional elliptic problems, we use two types of MscaleDNNs with size (1000, 800, 500, 500, 400, 200, 200, 100) and activation functions s2ReLU and sReLU, respectively. The training data set includes 7500 interior points and 1000 boundary points randomly sampled from Ω. The testing dataset includes 1600 random samples in Ω. We plot the testing results in Fig. 11. To visually illustrate these results, we map the point-wise errors of sReLU and s2ReLU solutions, evaluated on 1600 sample points in Ω, onto a 40×40 2d array, respectively. We note that the mapping is only for the purpose of visualization, and is independent of the actual coordinates of those points.

The numerical results in Fig. 11(a) indicate that the MscaleDNN models with s2ReLU and sReLU can still well approximate the exact solution of elliptic equation in five-dimensional space. In particular, Figs. 11(b) and 11(c) show that the point-wise error of s2ReLU is much smaller than that of sReLU.

5 Conclusion

In this paper, we propose an improved version of MscaleDNN by designing an activation function localized in both spatial and Fourier domains, and use that to solve multi-scale elliptic problems. Numerical results show that this method is effective for the resolution of elliptic problems with multiple scales and possible nonlinearity, in low to median high dimensions. As a meshless method, DNN based method is more flexible for partial differential equations than traditional mesh-based and meshfree methods in regular or irregular region. In the future, we will optimize the MscaleDNN architecture and design DNN based algorithms for multi-scale nonlinear problems with more general nonlinearities.

Acknowledgments

X.L and L.Z are partially supported by the National Natural Science Foundation of China (NSFC 11871339, 11861131004). Z.X. is supported by National Key R&D Program of China (2019YFA0709503), Shanghai Sailing Program, and Natural Science Foundation of Shanghai (20ZR1429000), This work is also partially supported by HPC of School of Mathematical Sciences at Shanghai Jiao Tong University.

References

[1] A. Abdulle and G. Vilmart. Analysis of the finite element heterogeneous multiscale method for quasilinear elliptic homogenization problems. *Mathematics of Computation*, 83(286):513–536, 2013.

[2] J. W. Barrett and W. B. Liu. Finite element approximation of the parabolic p-Laplacian. *SIAM Journal on Numerical Analysis*, 31(2):413–428, 1994.

[3] L. Belenki, L. Diening, and C. Kreuzer. Optimality of an adaptive finite element method for the p-laplacian equation. *Ima Journal of Numerical Analysis*, 32(2):484–510, 2012.

[4] J. Berg and K. Nystrm. A unified deep artificial neural network approach to partial differential equations in complex geometries. *Neurocomputing*, 317:28–41, 2018.

[5] S. Biland, V. C. Azevedo, B. Kim, and B. Solenthaler. Frequency-aware reconstruction of fluid simulations with generative networks. arXiv preprint arXiv:1912.08776, 2019.

[6] W. Cai, X. Li, and L. Liu. A phase shift deep neural network for high frequency approximation and wave problems. Accepted by *SISC*, arXiv:1909.11759, 2019.

[7] W. Cai and Z.-Q. J. Xu. Multi-scale deep neural networks for solving high dimensional PDEs. arXiv preprint arXiv:1910.11710, 2019.

[8] S. Chaudhary, V. Srivastava, V. V. K. Srinivas Kumar, and B. Srinivasan. Web-spline-based mesh-free finite element approximation for p-Laplacian. *International Journal of Computer Mathematics*, 93(6):1022–1043, 2016.
[9] E. T. Chung, Y. Efendiev, K. Shi, and S. Ye. A multiscale model reduction method for nonlinear monotone elliptic equations in heterogeneous media. *Networks and Heterogeneous Media*, 12(4):619–642, 2017.
[10] P. G. Ciarlet and J. T. Oden. *The Finite Element Method for Elliptic Problems*. 1978.
[11] D. Cioranescu and P. Donato. *An Introduction to Homogenization*. 2000.
[12] B. Cockburn and J. Shen. A hybridizable discontinuous Galerkin method for the p-Laplacian. *SIAM Journal on Scientific Computing*, 38(1), 2016.
[13] L. Diening and F. Ettwein. Fractional estimates for non-differentiable elliptic systems with general growth. *Forum Mathematicum*, 20(3):523–556, 2008.
[14] W. E, B. Engquist, X. Li, W. Ren, and E. Vanden-Eijnden. Heterogeneous multiscale methods: A review. *Communications in Computational Physics*, 2(3):367–450, 2007.
[15] W. E, C. Ma, and L. Wu. Machine learning from a continuous viewpoint. arXiv preprint arXiv:1912.12777, 2019.
[16] W. E and B. Yu. The deep Ritz method: A deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6(1):1–12, 2018.
[17] F. Feyel. Multiscale FE2 elastoviscoplastic analysis of composite structures. *Computational Materials Science*, 16(1):344–354, 1999.
[18] M. G. D. Geers, V. G. Kouznetsova, K. Matouš, and J. Yvonnet. Homogenization methods and multiscale modeling: Nonlinear problems. *Encyclopedia of Computational Mechanics Second Edition*, pages 1–34, 2017.
[19] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT press, Cambridge, 2016.
[20] J. Han, C. Ma, Z. Ma, and W. E. Uniformly accurate machine learning-based hydrodynamic models for kinetic equations. *Proceedings of the National Academy of Sciences*, 116(44):21983–21991, 2019.
[21] S. O. Haykin. *Neural Networks: A Comprehensive Foundation*. 1998.
[22] C. He, X. Hu, and L. Mu. A mesh-free method using piecewise deep neural network for elliptic interface problems. arXiv preprint arXiv:2005.04847, 2020.
[23] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.
[24] T. Hou and Y. Efendiev. *Multiscale Finite Element Methods: Theory and Applications*. 2009.
[25] Y. Q. Huang, R. Li, and W. Liu. Preconditioned descent algorithms for p-Laplacian. *Journal of Scientific Computing*, 32(2):343–371, 2007.
[26] M. Hutzenthaler, A. Jentzen, T. Kruse, T. A. Nguyen, and P. von Wurstemberger. Overcoming the curse of dimensionality in the numerical approximation of semilinear parabolic partial differential equations. arXiv preprint arXiv:1807.01212, 2018.
[27] A. D. Jagtap, K. Kawaguchi, and G. E. Karniadakis. Adaptive activation functions accelerate convergence in deep and physics-informed neural networks. *Journal of Computational Physics*, 404:109136, 2020.
[28] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.
[29] X. Li and H. Dong. The element-free Galerkin method for the nonlinear p-Laplacian equation. *Computers and Mathematics With Applications*, 75(7):2549–2560, 2018.
[30] Z. Liu, W. Cai, and Z.-Q. J. Xu. Multi-scale deep neural network (MscaleDNN) for solving Poisson-Boltzmann equation in complex domains. Accepted by *Communications in Computational Physics*, arXiv:2007.11207, 2020.

[31] A. M. Oberman. Finite difference methods for the infinity Laplace and p-Laplace equations. *Journal of Computational and Applied Mathematics*, 254(1):65–80, 2013.
[32] H. Owhadi and L. Zhang. Homogenization of parabolic equations with a continuum of space and time scales. *SIAM Journal on Numerical Analysis*, 46(1):1–36, 2007.
[33] H. Owhadi and L. Zhang. Numerical homogenization of the acoustic wave equations with a continuum of scales. *Computer Methods in Applied Mechanics and Engineering*, 198:397–406, 2008.
[34] H. Owhadi, L. Zhang, and L. Berlyand. Polyharmonic homogenization, rough polyharmonic splines and sparse super-localization. *Mathematical Modelling and Numerical Analysis*, 48(2):517–552, 2014.
[35] S. Qian, H. Liu, C. Liu, S. Wu, and H. S. Wong. Adaptive activation functions in convolutional neural networks. *Neurocomputing*, 272:204–212, 2018.
[36] T. Qin, K. Wu, and D. Xiu. Data driven governing equations approximation using deep neural networks. *Journal of Computational Physics*, 395:620–635, 2019.
[37] N. Rahaman, D. Arpit, A. Baratin, F. Draxler, M. Lin, F. A. Hamprecht, Y. Bengio, and A. Courville. On the spectral bias of deep neural networks. *International Conference on Machine Learning*, 2019.
[38] C. P. Robert and G. Casella. *Monte Carlo Statistical Methods*. 1999.
[39] J. Sirignano and K. Spiliopoulos. DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339–1364, 2018.
[40] D. Slepčev and M. Thorpe. Analysis of p-Laplacian regularization in semi-supervised learning. arXiv preprint arXiv:1707.06213, 2017.
[41] C. M. Strofer, J.-L. Wu, H. Xiao, and E. Paterson. Data-driven, physics-based feature extraction from fluid flow fields using convolutional neural networks. *Communications in Computational Physics*, 25(3):625–650, 2019.
[42] L. Tartar. *The General Theory of Homogenization: A Personalized Introduction*. 2009.
[43] Z. Wang and Z. Zhang. A mesh-free method for interface problems using the deep learning approach. *Journal of Computational Physics*, 400:108963, 2020.
[44] Z.-Q. J. Xu, Y. Zhang, and Y. Xiao. Training behavior of deep neural network in frequency domain. *International Conference on Neural Information Processing*, pages 264–274, 2019.
[45] Z.-Q. J. Xu, Y. Zhang, T. Luo, Y. Xiao, and Z. Ma. Frequency principle: Fourier analysis sheds light on deep neural networks. Accepted by *Communications in Computational Physics*, arXiv:1901.06523, 2019.
[46] D. Yarotsky. Error bounds for approximations with deep ReLU networks. *Neural Networks*, 94:103–114, 2017.
[47] Y. Zhu, N. Zabaras, P.-S. Koutsourelakis, and P. Perdikaris. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *Journal of Computational Physics*, 394:56–81, 2019.