Journal of Computational Physics 400 (2020) 108963

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

A mesh-free method for interface problems using the deep learning approach

Zhongjian Wang, Zhiwen Zhang *

Department of Mathematics, The University of Hong Kong, Pokfulam Road, Hong Kong, China

ARTICLE INFO

Article history:
Received 11 February 2019
Received in revised form 17 August 2019
Accepted 17 September 2019
Available online 23 September 2019

Keywords:
Deep learning
Variational problems
Mesh-free method
Linear elasticity
High-contrast
Interface problems

ABSTRACT

In this paper, we propose a mesh-free method to solve interface problems using the deep learning approach. Two types of PDEs are considered. The first one is an elliptic PDE with a discontinuous and high-contrast coefficient. While the second one is a linear elasticity equation with discontinuous stress tensor. In both cases, we represent the solutions of the PDEs using the deep neural networks (DNNs) and formulate the PDEs into variational problems, which can be solved via the deep learning approach. To deal with inhomogeneous boundary conditions, we use a shallow neural network to approximate the boundary conditions. Instead of using an adaptive mesh refinement method or specially designed basis functions or numerical schemes to compute the PDE solutions, the proposed method has the advantages that it is easy to implement and is mesh-free. Finally, we present numerical results to demonstrate the accuracy and efficiency of the proposed method for interface problems.

1. Introduction

In recent years, deep learning methods have achieved unprecedented successes in various application fields, including computer vision, speech recognition, natural language processing, audio recognition, social network filtering, and bioinformatics, where they have produced results comparable to and in some cases superior to human experts [22,14]. Motivated by this exciting progress, there are increased new research interests in the literature for the application of deep learning methods for scientific computation, including approximating multivariate functions and solving differential equations using the DNNs; see e.g. [16,27,37,38,19,42,31,34,43,32,18] and references therein.

There are many classical works on the approximation power of neural networks (NNs); see e.g. [8,17,11,30]. We refer the reader to recent works on the expressive power (i.e., approximation power) of DNNs; see e.g. [7,33,40,27]. We also mention the recent work by [16], where the authors investigate the relationship between DNNs with rectified linear unit (ReLU) function as the activation function and continuous piecewise linear functions in the finite element method (FEM). They prove that a ReLU-DNN with enough hidden layers and enough neurons within each layer can include the continuous piecewise linear FEM space. Thus, one can represent a solution of PDE using the ReLU-DNN.

Solving ODEs or PDEs with a neural network as an approximation is a natural idea, which has been considered in various forms in the literature, e.g. [23,26,21]. The main idea is to train NNs to approximate the solution by minimizing the residual

* Corresponding author.
E-mail addresses: ariswang@connect.hku.hk (Z. Wang), zhangzw@hku.hk (Z. Zhang).

https://doi.org/10.1016/j.jcp.2019.108963
0021-9991/

of the ODEs or PDEs and also of the initial and boundary conditions. These papers estimate neural network solutions on an a priori fixed mesh.

Thanks to the widespread availability of cheap computing resources (e.g. TensorFlow and PyTorch) and theoretical advances in stochastic optimization (e.g. stochastic gradient descent), solving PDEs or stochastic PDEs using a DNN has become an emerging research topic. A deep Ritz method [38] was developed to solve Poisson problems and eigenvalue problems from variational principles using DNNs. Meanwhile, deep learning-based numerical methods [15] were proposed to solve high-dimensional parabolic PDEs and backward stochastic differential equations. Recently, a physics-informed neural network (PINN) method [32] and a deep Galerkin method (DGM) [34] were developed to solve PDEs efficiently. The main idea of PINN and DGM is to train DNNs to approximate the solution by minimizing the residual of the PDEs and also of the initial and boundary conditions. In the context of surrogate modeling and uncertainty quantification (UQ), several efficient methods based on the DNNs were developed recently, including the Bayesian deep convolutional encoder-decoder networks [42], deep multiscale model learning [36], and physics-constrained deep learning method [43]. We also refer the interested reader to [19,33,18] and references therein.

In this paper, we will use the deep learning method to solve interface problems, which have many applications in physics and engineering sciences. For example, to model the heterogeneous porous medium in the reservoir simulation, the permeability field is often assumed to be a multiscale function with high-contrast and discontinuous features. Another example is to study the evolution of the shape and location of fibroblast cells under stress [41]. The model is based on ideas of a continuum mechanical description of stress-induced phase transitions, where the cell is modeled as a transformed inclusion in a linear elastic matrix and the cell surface evolves according to a special kinetic relation. In this model, the stress tensor has discontinuity across the cell surface due to the transformation in the strain tensor caused by a contraction in the cell.

There has been a lot of effort in developing accurate and efficient finite element methods (FEMs) for interface problems; see e.g. [2,5] and references therein for some early works. An immersed-interface finite element method [25,13] was developed to solve elliptic interface problems with non-homogeneous jump conditions. The method considered uniform triangular grids and approximated the interface by a straight line segment when it intersects a coarse grid element. By matching the jump condition, a special basis function for elements that were cut through by the interface was created and proved to have a second-order convergence rate in the L_2 norm and a first-order convergence rate in the H_1 semi-norm. However, the constants in the error estimate depend on the contrast of the coefficient. Later, a new multiscale finite element method [6] was developed that was able to accurately capture solutions of elliptic interface problems with high-contrast coefficients by using only coarse quasi-uniform meshes, and without resolving the interfaces. Moreover, an optimal error estimate was obtained in the sense that the hidden constants in the estimates were independent of the contrast of the PDE coefficients.

Alternatively, some efficient finite difference methods (FDMs) were proposed to solve interface problems. Such results include, among others, an immersed boundary method (IBM) [28] was developed to study the motion of one or more massless, elastic surfaces immersed in an incompressible, viscous fluid, particularly in bio-fluid dynamics problems where complex geometries and immersed elastic membranes are present. We refer to [29] for an extensive review of the IBM. Another related work is the immersed interface method (IIM) for elliptic interface problems developed in [24]. By incorporating the jump condition across the interface to modify the finite difference schemes near the interface, a second order accuracy was maintained. An important development of interface capturing methods is the ghost fluid method (GFM) [12], which incorporated the interface jump condition into the finite difference discretization by tracking the interface with a level set function. The GFM can capture discontinuities in multi-medium compressible multiphase flows.

In this paper, we are interested in developing deep learning methods to solve interface problems. Our work is inspired by the deep Ritz method proposed in [38], where the Poisson problems and eigenvalue problems were studied. We intend to investigate the expressive power of the DNNs in representing solutions of interface problems. In addition, we will study the performance of the stochastic gradient descent (SDG) method [4] in solving optimization problems associated with the interface problems. Two types of PDEs will be considered. The first one is an elliptic PDE with a discontinuous and high-contrast coefficient, which is a challenging problem and has been intensively studied; see [3,25,6,9]. The second one is a linear elasticity equation with discontinuous stress tensor [41].

In both problems, we first represent the solutions of the PDEs using the DNNs and formulate the PDEs into variational problems, which can be solved using the deep learning approach. Then, we use the SGD method to solve the variational problem. To impose inhomogeneous boundary conditions, we propose to use a shallow neural network to approximate the boundary conditions. Since the boundary conditions are defined in a lower-dimensional space (compared with the solution itself), a shallow neural network can approximate the boundary conditions well and reduce computational costs (e.g. computing the automatic differentiation) in the training process.

We find that the proposed DNN method is easy to implement and is mesh-free since we do not need to choose an adaptive mesh or a specially designed numerical scheme to discretize the PDEs. Our numerical results show that the DNN method can efficiently solve the two types of PDEs considered here. The accuracy of the DNN method is mainly determined by the expressive power of the DNNs and the stochastic optimization methods. Moreover, we observe that the convergence time of the SGD method is random, which is due to the fact that the iteration process of the SGD method can be get stuck into some local minimums. Especially, we find that it takes a longer time to get out of local minima in a 'harder' case of the high-contrast problem; see Section 5.1 for more details.

The rest of the paper is organized as follows. In Section 2, we review the basic idea of the DNN and the deep Ritz method. In Section 3, we introduce the idea of using a shallow neural network to deal with inhomogeneous boundary conditions. The derivation of the methodology for both the elliptic PDEs and linear elasticity PDEs will be presented in Section 4. In Section 5, we present numerical results to demonstrate the performance of our method. Concluding remarks will be made in Section 6.

2. Some preliminaries

In this section, we briefly discuss the definition and approximation properties of the DNNs and the formulation of the deep Ritz method [38].

2.1. The DNN and its approximation property

There are two ingredients in defining a DNN. The first one is a (vector) linear function of the form $T : R^n \to R^m$, defined as $T(x) = Ax + b$, where $A = (a_{ij}) \in R^{m\times n}$, $x \in R^n$ and $b \in R^m$. The second one is a nonlinear activation function $\sigma : R \to R$. A frequently used activation function, known as the rectified linear unit (ReLU), is defined as $\sigma(x) = \max(0, x)$ [22]. In the artificial neural network literature, the sigmoid function is another frequently used activation function, which is defined as $\sigma(x) = (1 + e^{-x})^{-1}$. By applying the activation function in an element-wise manner, one can define (vector) activation function $\sigma : R^m \to R^m$.

Equipped with those definitions, we are able to define a continuous function $F(x)$ by a composition of linear transforms and activation functions, i.e.,

$$F(x) = T^k \circ \sigma \circ T^{k-1} \circ \sigma \cdots \circ T^1 \circ \sigma \circ T^0(x), \tag{1}$$

where $T^i(x) = A_i x + b_i$ with A_i be undetermined matrices and b_i be undetermined vectors, and $\sigma(\cdot)$ is the element-wisely defined activation function. Dimensions of A_i and b_i are chosen to make (1) meaningful. Such a DNN is called a $(k+1)$-layer DNN, which has k hidden layers. Denoting all the undetermined coefficients (e.g., A_i and b_i) in (1) as $\theta \in \Theta$, where θ is a high-dimensional vector and Θ is the space of θ. The DNN representation of a continuous function can be viewed as

$$F = F(x; \theta). \tag{2}$$

Let $\mathbb{F} = \{F(\cdot, \theta) | \theta \in \Theta\}$ denote the set of all expressible functions by the DNN parameterized by $\theta \in \Theta$. Then, $\mathbb{F}$ provides an efficient way to represent unknown continuous functions, comparing with a linear solution space used in classic numerical methods, e.g., a trial space spaced by linear nodal basis functions in the FEM.

One of the fundamental questions in the DNN is to study its approximation property, also known as its expressive power [7,33]. Early studies of approximation properties of a neural network can be found in [8,17], where approximation properties for the function classes given by a feed-forward neural network with a single hidden layer were studied. Later, error estimates for such neural networks in terms of a number of neurons, layers of the network, and activation functions were obtained; see e.g. [11,30] for a good review of relevant works.

In recent years, significant efforts have been devoted to study the benefits on the expressive power of NNs afforded by NN depth. For example, [7] proved that convolutional DNNs were able to express multivariate functions given in so-called Hierarchic Tensor (HT) formats. The expressive power of shallow and deep neural networks with piecewise linear activation functions was studied in [40] and rigorous upper and lower bounds for the network complexity in approximating Sobolev spaces was established. [27] obtained a new error bound for the approximation of multivariate functions using deep ReLU networks, which shows that the curse of dimensionality is lessened by establishing a connection between the deep networks and sparse grids. [16] studied the relationship between ReLU-DNN and continuous piecewise linear functions from the linear FEM and built a connection for the DNN method with the FEM method. Specifically, [16] proved the following statement.

Proposition 2.1. *Given a locally convex finite element grid $\mathcal{T}_h$, any linear finite element function with N degrees of freedom, can be written as a ReLU-DNN with at most $k = \lceil \log_2 k_h \rceil + 1$ hidden layers and at most $\mathcal{O}(k_h N)$ number of the neurons, where k_h denotes the maximum number of neighboring elements of one node.*

The Proposition 2.1 provides upper bounds in setting the number of hidden layers and the number of neurons within each layer, when one uses the DNN to approximate the solution space spanned by the FEM basis. In our numerical results, we find that choosing a relatively small number of hidden layers and neurons within each layer are good enough to obtain accurate results for the interface problems studied in this paper.

2.2. Formulation of the deep Ritz method

The deep Ritz method is a deep learning based numerical method for solving variational problems [38]. Therefore, it naturally can be used to solve PDEs. For example, we consider a Poisson equation defined on a compact domain $D \subsetneq R^d$,

$$\begin{cases} -\Delta u(x) = f(x), & x \in D, \\ u(x) = 0, & x \in \partial D. \end{cases} \tag{3}$$

Given the Poisson equation (3), we can derive the corresponding variational problem as

$$J(v) = \frac{1}{2}\int_D \nabla v(x) \cdot \nabla v(x) dx - \int_D v(x) f(x) dx, \quad v \in \mathbb{H}_0^1(D). \tag{4}$$

Then, the solution of (3) can be obtained by,

$$u = \arg\min_{v \in \mathbb{H}_0^1(D)} J(v). \tag{5}$$

From the perspective of scientific computing, the Poisson equation (3) can be solved using numerical methods, such as FDMs and FEMs. From the perspective of machine learning however, the numerical solution of $u(x)$ is interpreted as a function with $x \in R^d$ as its input and R^1 as its output, where d denotes the dimension the physical domain D. Thus, it can be approximated by $F(x)$ in (1).

Let $\tilde{u}$ denote the DNN representation of the solution of the Poisson equation. Substituting $\tilde{u}$ into the variational problem (4), we get the optimization problem

$$\tilde{u} = \arg\min_{F \in \mathbb{F}_0} J(F), \tag{6}$$

where $\mathbb{F}_0$ is a subspace of $\mathbb{F}$ that satisfies the boundary condition on ∂D. The justification of this assumption will be discussed later.

After parameterizing the expressible function space by $\theta \in \Theta$; see Eq. (2), we equivalently define the variational problem (4) as

$$\min_{\theta \in \Theta} J(\theta) = \frac{1}{2}\int_D |\nabla F(x, \theta)|^2 dx - \int_D F(x, \theta) f(x) dx. \tag{7}$$

In general the above variational problem (7) is non-convex, even the original variational problem (4) is so. In other words, the variational problem (4) is convex with respect to the solution $u(x)$, however, the variational problem (7) is non-convex with respect to the parameters in the DNN. Obviously, it is nontrivial to solve a non-convex variational problem due to the existence of local minima and saddle points, which poses a challenge to the DNN based methods for solving PDEs.

Since the dimension of the parameter space Θ is very large and the corresponding variational problem is non-convex, one usually uses the SGD method [4] to solve (7). There are plenty of optimization methods to search among the large parameter space. To accelerate the training of the DNN, we use the Adam optimizer version of SGD [20].

In practice, it is not straightforward to impose boundary conditions in the DNN representation. In the homogeneous Dirichlet problem (3), a relaxation approach was proposed to address this issue. Specifically, one adds a soft constraint (a boundary integral term) to the functional $J(\cdot)$ defined in (7) and obtains

$$\tilde{u}_\epsilon = \arg\min_{F \in \mathbb{F}} \left(J(F) + \frac{1}{\epsilon}\int_{\partial D} F(x, \theta)^2 dx \right). \tag{8}$$

Notice that the soft constraint term $\frac{1}{\epsilon}\int_{\partial D} F(x, \theta)^2 dx$ will approach zero when we decrease the parameter ϵ in the calculation. Therefore, the homogeneous boundary condition is satisfied in a certain weak scene.

3. Inhomogeneous boundary conditions

To demonstrate the idea, we consider an inhomogeneous Dirichlet problem as follows,

$$\begin{cases} \mathcal{L}u(x) = f(x), & x \in D, \\ u(x) = g(x), & x \in \partial D, \end{cases} \tag{9}$$

where $\mathcal{L}$ is a linear PDE operator, $f(x)$ is a source function, and $g(x)$ is a boundary condition. Let $J(v; f)$ denote the Lagrangian form associated with the homogeneous Dirichlet problem of (9), i.e., $g(x) = 0$; see (4) for instance.

To deal with the inhomogeneous boundary condition in (9), we first use a shallow neural network to approximate the boundary condition $g(x)$. Let $\tilde{g}(x)$ denote the approximation of $g(x)$ by using a shallow neural network, which is defined on whole domain D. However, we only require that the boundary value of $\tilde{g}$ satisfies the boundary condition $g(x)$, which can be obtained by solving the following optimization problem

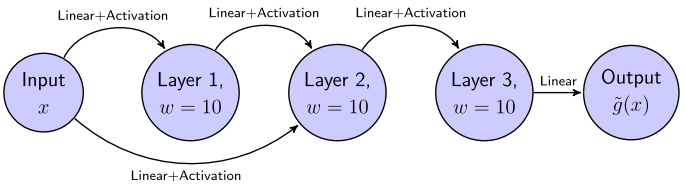


Fig. 1. Network Layout for $\tilde{g}$.

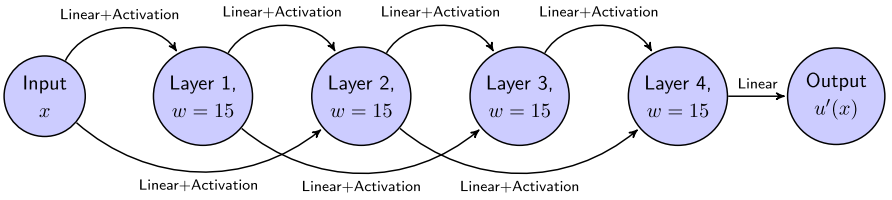


Fig. 2. Network Layout for u'.

$$\tilde{g}(x) = \arg\min_{G\in\mathbb{G}} \Big(\int_{\partial D} \big(G - g(x)\big)^2 dx \Big), \tag{10}$$

where $\mathbb{G}$ denotes the set of all expressible functions by a shallow neural network. The optimization problem (10) can be approximated by,

$$\frac{vol(\partial D)}{N_0} \sum_{i=1}^{N_0} \big(G(y_i) - g(y_i)\big)^2, \tag{11}$$

where $y_i \overset{i.i.d.}{\sim} Unif(\partial D)$ and N_0 is the number of sample points on the boundary ∂D. In practice, uniform sampler on ∂D is not necessary. One can change the integrand of (10) by multiplying the Radon-Nikodym derivative of the sampler's distribution. Let $\rho(dx)$ denote the distribution of the grid points on the boundary. Notice that $\int_{\partial D} \big(G - g(x)\big)^2 \rho(dx) = 0$ implies $\int_{\partial D} \big(G - g(x)\big)^2 dx = 0$. Once we obtain a sampler according to the distribution $\rho(dx)$, which is absolutely continuous w.r.t. Lebesgue measure of ∂D, we can still minimize (11) to obtain $\tilde{g}(x)$. Thus, we do not need to specially treat the shape of boundaries.

In our proposed approach, reasons for choosing a shallow network to approximate $g(x)$ are twofold. First, $\tilde{g}(x)$ is used to approximate the inhomogeneous boundary condition $g(x)$. Since $g(x)$ is defined in a lower-dimensional physical space (i.e., ∂D instead of D), we can use a shallow neural network to approximate it well. Compared with the neural network used to represent a solution, the shallow neural network $\tilde{g}(x)$ has fewer parameters, which helps shorten the training process. For instance, the shallow neural network can reduce computational costs in computing the automatic differentiation. Second, due to the simple structure of $\tilde{g}$, the term $\mathcal{L}\tilde{g} \cdot v$ in $J(v; f - \mathcal{L}\tilde{g})$ will not oscillate in D (especially in the weak form), which leads to a faster convergence in solving optimization problems.

Fig. 1 and Fig. 2 show possible network layouts for approximating $\tilde{g}$ and u', respectively, where w denotes the width of each hidden layer. For example, Layer 2 in Fig. 1 is in $\mathbb{R}^{10}$. To be more precise, denote Layer 1 to be $l_1 \in \mathbb{R}^{10}$, Layer 2 to be $l_2 \in \mathbb{R}^{10}$, then,

$$l_2 = \sigma\big(A[l_1; x] + b\big), \tag{12}$$

where $x \in \mathbb{R}^d$, A is a $10 \times (10 + d)$ matrix and b is a vector in $\mathbb{R}^{10}$ to be determined. To increase the performance, we also add connections between non-adjacent layers (e.g. Layer 1 and Layer 3 in Fig. 2). A simple calculation can show the parameters in the network for $\tilde{g}$ is far fewer than the one for u'.

Since the neural network that is used to represent $\tilde{g}$ is shallow (i.e., $\tilde{g}$ is represented by a composition of a few levels of smooth functions), we can assume $\mathcal{L}\tilde{g}$ is smooth at least in the weak sense. Thus, $\mathcal{L}\tilde{g}$ can be easily computed by the automatic differentiation. Then, we solve an auxiliary PDE as follows,

$$\begin{cases} \mathcal{L}u'(x) = f(x) - \mathcal{L}\tilde{g}(x), & x \in D, \\ u'(x) = 0, & x \in \partial D. \end{cases} \tag{13}$$

Now the problem (13) becomes a homogeneous Dirichlet problem, which can be solved using the deep Ritz approach; see Section 2.2. Finally, the solution of the inhomogeneous Dirichlet problem (9) can be represented as $u(x) = u'(x) + \tilde{g}(x)$.

In our method, we first train the shallow neural network approximation $\tilde{g}(x)$. Then, we keep $\tilde{g}(x)$ fixed and train the deep neural network approximation $u'(x)$. Our method shares some similarity with the PINN method [32] and the DGM

[34] in the sense that these methods train DNNs to approximate the solutions of PDEs by solving optimization problems. Our method has several new features. We solve PDEs through their corresponding variational problems, which avoids the need to compute high-order derivatives of the solution. For example, if $\mathcal{L}$ in (9) is a second-order elliptic operator, we only need to compute the first-order derivative of the solution. In the PINN method and DGM method, one needs to compute second-order derivatives of the solution when one computes the residual of the PDEs. Moreover, using a shallow neural network to approximate boundary condition allows us to simply impose inhomogeneous boundary conditions and reduce computational costs in the training process.

4. Derivation of the methodology

4.1. Elliptic PDEs with discontinuous and high-contrast coefficients

We first consider elliptic PDEs with discontinuous coefficients defined as follows,

$$\mathcal{L}(x)u(x) \equiv -\nabla \cdot (a(x)\nabla u(x)) = f(x), \quad x \in D, \tag{14}$$

$$u(x) = 0, \quad x \in \partial D, \tag{15}$$

where $D \subseteq R^d$ is a bounded domain and the boundary of D is a convex polygon. For notation simplicity, we study a homogeneous Dirichlet problem. The problems with inhomogeneous boundary conditions can be solved by using the approach studied in Section 3.

The coefficient $a(x)$ is assumed to be a scalar and has jumps across a number of smooth interior interfaces. Denoting the inclusions by D_1,...,D_m and setting $D_0 = D \setminus \bigcup_{i=1}^m D_i$, we assume that the coefficient $a(x)$ is piecewise constant with respect to the decomposition $\{D_i, i = 0, ..., m\}$. Setting $a_{min} = \min a(x)|_{D_i} : i = 0, ..., m$ and dividing (14) by a_{min}, we rescale the problem. Specifically, let $\alpha(x) = \frac{a(x)}{a_{min}}$ denote the re-scaled coefficient, which is piecewise constant with respect to the partition $\{D_i, i = 0, ..., m\}$ and $\alpha(x) \geq 1$ for all $x \in D$. Let α_i denote the restriction of $\alpha(x)$ to D_i. We are interested in studying two types of high-contrast cases,

$$\text{Case 1:} \quad \min_{i=1,...,m} \alpha_i \gg 1, \quad \alpha_0 = 1, \tag{16}$$

$$\text{Case 2:} \quad \alpha_0 \gg 1, \quad \max_{i=1,...,m} \alpha_i \leq K, \tag{17}$$

for some positive constant K. In Case 1, the inclusions are high permeability compared to the background, while the Case 2 contains a converse configuration.

Now, we are in the position to derive the formulation of deep learning approach to solve the elliptic PDEs (14)-(15) with high-contrast coefficients (16)-(17). We define the corresponding variational problem as

$$J(v) = \frac{1}{2}\int_D a(x)|\nabla v(x)|^2 dx - \int_D v(x) f(x) dx, \quad v \in \mathbb{H}_0^1(D). \tag{18}$$

Then, the solution of (14)-(15) can be obtained by solving $u(x) = \arg\min_{v \in \mathbb{H}_0^1(D)} J(v)$, where $J(\cdot)$ is defined in (18). Again, we denote the set of all expressible function by $\mathbb{F} = \{F(\cdot, \theta) | \theta \in \Theta\}$ and set $\mathbb{F}_0 = \{F \in \mathbb{F} \,|\, F|_{\partial D} = 0\}$. Moreover, let Θ_0 denote the parameter set satisfies the homogeneous boundary condition, i.e., $F(\cdot, \theta)|_{\partial D} = 0$, $\theta \in \Theta_0$. The approximation property of the DNN implies that $\mathbb{F}_0 \subsetneq \mathbb{C}_0^\infty(D) \subsetneq \mathbb{H}_0^1(D)$. Therefore, we represent the solution $u(x)$ to Eq. (14) using the DNN method.

Let $\tilde{u} = F(x; \theta)$ denote the DNN representation; see Eq. (1). Then, $\tilde{u}$ satisfies the following variational problem

$$\tilde{u} = \underset{F = F(\cdot,\theta) | \theta \in \Theta_0}{\arg\min} \frac{1}{2}\int_D a(x)|\nabla F(x, \theta)|^2 dx - \int_D F(x, \theta) f(x) dx. \tag{19}$$

Since the degree of freedom in the variational problem (19) is quite large and the parameter space Θ_0 may have complex geometry in Θ, we apply the Adam [20] with a soft constraint on the boundary of D to solve it. As such, we approximate gradient of each parameter θ_k by,

$$\begin{aligned} &\frac{\partial\left(J\left(F(\cdot,\theta)\right) + \frac{1}{\epsilon}\int_{\partial D} F^2(\cdot,\theta)\right)}{\partial \theta_k} \\ =&\frac{1}{2}\int_D \frac{\partial\left(a(x)|\nabla F(x,\theta)|^2\right)}{\partial \theta_k} dx - \int_D \frac{\partial\left(F(x,\theta) f(x)\right)}{\partial \theta_k} dx + \frac{1}{\epsilon}\int_{\partial D} \frac{\partial\left(F^2(x,\theta)\right)}{\partial \theta_k} dx \end{aligned}$$

$$\approx \frac{vol(D)}{N_1}\sum_{i=1}^{N_1}\Big(\frac{1}{2}\frac{\partial\big(a(x_i)|\nabla F(x_i,\theta)|^2\big)}{\partial\theta_k}-\frac{\partial\big(F(x_i,\theta)f(x_i)\big)}{\partial\theta_k}\Big)+\frac{vol(\partial D)}{\epsilon N_2}\sum_{j=1}^{N_2}\frac{\partial\big(F^2(y_j,\theta)\big)}{\partial\theta_k}, \tag{20}$$

where $x_i \overset{i.i.d.}{\sim} Unif(D)$, $vol(D)$ is the volume of the domain, $y_j \overset{i.i.d.}{\sim} Unif(\partial D)$, $vol(\partial D)$ is the volume of the boundary, and $N = N_1 + N_2$ is called the batch number in the context of deep learning (meaning the number of collocation points used in one iteration). Notice that θ is a high-dimensional vector and θ_k is any component of θ. After we get the approximation of the gradient with respect to θ_k, we can update each component of θ as

$$\theta_k^{n+1} = \theta_k^n - \eta\frac{\partial\Big(J\big(F(\cdot,\theta)\big)+\frac{1}{\epsilon}\int_{\partial D}F^2(\cdot,\theta)\Big)}{\partial\theta_k}|_{\theta_k=\theta_k^n}, \tag{21}$$

where η is the learning rate.

Remark 4.1. From the derivation of the DNN formulation, one can see that the proposed method automatically deals with the interface condition (or discontinuous coefficients) without knowing locations of the interfaces a priori.

4.2. Linear elasticity with discontinuous stress tensors

In this subsection, we consider the DNN approach to solve linear elasticity interface problems. One application of the linear elasticity problem is to model the shape and location of fibroblast cells under stress [41]. The model is based on the idea of a continuum mechanical description of stress-induced phase transitions. To demonstrate the main idea, we consider a two-dimensional linear elasticity problem.

Suppose the matrix (meaning the material or tissue in cells) plus the cell together occupy a bounded domain $D \subseteq R^d$, $d = 2$ and D is composed of linear elastic homogeneous isotropic material. We assume the cell has small deformations, so the linearized theory of elasticity is used. Let $\mathbf{u} = (u_1, u_2)^T$ denote the displacement field. Then, the strain tensor is

$$\mathbf{E} = \frac{1}{2}(\nabla\mathbf{u} + \nabla\mathbf{u}^T), \quad \text{with} \quad E_{ij} = \frac{1}{2}\Big(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\Big). \tag{22}$$

In the matrix except the cell, the stress tensor is related to the strain tensor (gradient of the displacement) by $\mathbf{S} = \mathbb{C}\mathbf{E}$, where the elasticity tensor $\mathbb{C}$ is a linear transformation on the tensors. In the isotropic case, we have

$$\mathbb{C}\mathbf{A} = \lambda\,\mathrm{Tr}(\mathbf{A})\mathbf{1} + \mu(\mathbf{A} + \mathbf{A}^T), \tag{23}$$

for any two dimensional matrix $\mathbf{A}$. In Eq. (23), λ and μ are lamé constants, $\mathrm{Tr}(\cdot)$ is the trace operator, and $\mathbf{1}$ is the identity matrix. In components, the action of the elasticity tensor $\mathbb{C}$ reads

$$C_{ijkl}A_{kl} = \lambda A_{kk}\delta_{ij} + \mu(A_{ij} + A_{ji}), \tag{24}$$

where the Einstein summation convention is used.

The cell is modeled by a compact region Ω with smooth boundary; see Fig. 13. Let $\mathbf{E}_0$ denote a transformation strain, which is a constant symmetric matrix. We assume the stress tensor has a jump across the cell, i.e.,

$$\mathbf{S} = \begin{cases} \mathbb{C}\mathbf{E}, & \text{in } D\setminus\Omega, \\ \mathbb{C}(\mathbf{E}-\mathbf{E}_0), & \text{in } \Omega. \end{cases} \tag{25}$$

In the cell model, we set the transformation strain to be a contraction, which is represented by an isotropic compression $\mathbf{E}_0 = -\alpha\mathbf{1}$ with $\alpha > 0$. We suppose the cell model is in a quasi-static state. Therefore, the displacement field $\mathbf{u}$ satisfies the following linear elasticity PDE with a discontinuous stress tensor in a weak sense,

$$-\nabla\cdot\mathbf{S} = -\nabla\cdot\big(\mathbb{C}\mathbf{E} - \chi_\Omega\mathbb{C}\mathbf{E}_0\big) = 0, \quad x \in D, \tag{26}$$

where χ_Ω is the characteristic function of the cell domain Ω. We impose Dirichlet boundary conditions on ∂D. On the cell boundary $\partial\Omega$, the solution $\mathbf{u}$ satisfies the following jump conditions

$$[\mathbf{u}] = 0, \quad [\mathbf{S}]n = 0, \tag{27}$$

where n is the outward unit normal vector on $\partial\Omega$ and [ ] denotes the jump across the interface.

Then, the linear elasticity interface problem (26)-(27) can be computed by numerical methods, such as the immersed interface method [39] or matched interface and boundary method [35]. However, the implementation of the numerical scheme is not simple due to the jump conditions on the interface, especially when the interface has a complicated geometry.

In the sequel, we shall develop the formulation for solving the linear elasticity interface problem (26)-(27) using the DNN method. In the isotropic case, let $\mathbf{e}(\mathbf{v}) \equiv (e_{ij}(\mathbf{v}))$, where $e_{ij}(\mathbf{v}) = \frac{1}{2}(\partial_j v_i + \partial_i v_j)$ and $\mathbf{v} = (v_1, v_2)^T$ is a vector valued function. Then, Eq. (26) is rewritten as,

$$-\nabla \cdot (\lambda \operatorname{Tr}(\mathbf{e}(\mathbf{u}))I_2 + 2\mu\mathbf{e}(\mathbf{u}) - \chi_\Omega \mathbb{C}\mathbf{E}_0) = 0. \tag{28}$$

Then, the variational problem associated with (28) is given by,

$$J(\mathbf{v}) = \int_D \Big(\frac{\lambda}{2}\operatorname{Tr}(\mathbf{e}(\mathbf{v}))^2 + \mu\mathbf{e}(\mathbf{v}) : \mathbf{e}(\mathbf{v}) + 2\chi_\Omega\alpha(\lambda+\mu)\operatorname{Tr}(\mathbf{e}(\mathbf{v}))\Big)dx, \tag{29}$$

where : denotes the inner product between matrices, i.e., $A : B = \operatorname{Tr}(A^T B) = \sum_{i,j} a_{ij}b_{ij}$. Finally, the solution of (28) can be obtained by $\mathbf{u}(x) = \arg\min_{\mathbf{v}\in(\mathbb{H}^1_0(D))^2} J(\mathbf{v})$, where $J(\cdot)$ is defined in (29). The remaining steps of the implementation of the DNN method for (29) is exactly the same as we discussed in Section 4.1, so we skip the details here.

5. Numerical example

In this section, we shall carry out numerical experiments to demonstrate the performance of the DNN method in solving interface problems. In addition, we are interested in understanding the SGD method in solving the non-convex optimization problem. The TensorFlow [1] provides an efficient tool to calculate the partial derivatives in (20), which will be used in our implementation.

5.1. 2D high-contrast elliptic problems

We consider 2D elliptic PDEs with high-contrast coefficients defined as follows,

$$-\nabla \cdot (a(x)\nabla u(x)) = f(x), \quad x \in D, \tag{30}$$

$$u(x) = g(x), \quad x \in \partial D, \tag{31}$$

where $x = (x_1, x_2)$, the domain is $D = [-1, 1] \times [-1, 1]$, and the coefficient $a(x)$ is a piecewise constant defined by

$$\alpha = \begin{cases} \alpha_1, & r < r_0, \\ \alpha_0, & r \geq r_0, \end{cases} \tag{32}$$

where $r = (x_1^2 + x_2^2)^{1/2}$ and $r_0 = \pi/6.28$. Moreover, the source term $f(x) = -9r$ and the boundary condition $g(x) = \frac{r^3}{\alpha_0} + (\frac{1}{\alpha_1} - \frac{1}{\alpha_0})r_0^3$. We choose the source term and boundary condition in such a way that the exact solution (in the polar coordinate) is

$$u(r, \theta) = \begin{cases} \frac{r^3}{\alpha_1}, & r < r_0, \\ \frac{r^3}{\alpha_0} + (\frac{1}{\alpha_1} - \frac{1}{\alpha_0})r_0^3, & r \geq r_0. \end{cases} \tag{33}$$

In our first experiment, we choose $\alpha_0 = 10^3$ and $\alpha_1 = 1$ in (32); see Fig. 3 for the profile of the coefficient. Notice that problem (30)-(31) is an inhomogeneous Dirichlet problem.

We convert the exact solution (33) in the Cartesian coordinate to get the reference solution and use the DNN method to compute the numerical solution. The implementation of the DNN method has been intensively discussed in Section 3 and Section 4.1. The network that we used is shown in Fig. 1 and Fig. 2. We have increased the number of hidden layers and the number of neurons within each layer and found that the error does not decrease, which means that the error comes from other sources. In addition, we fixed $w = 15$ in Fig. 2 and choose $w = 8$, $w = 10$ and $w = 12$ in Fig. 1, we obtain almost same results, which verifies that choosing a shallow network can maintain accuracy and reduce computational costs.

In the learning process, i.e., the running of the SGD method, we choose the batch number (number of samples per gradient update) to be 4096 (that contains 3840 points in the interior domain of D to evaluate the first term in Eq. (8) and 256 points on the boundary ∂D to evaluate the second term in Eq. (8); see Eq. (20) for more details) and generate a new batch every 10 steps of updating. The learning rate η is 5×10^{-4}. Once we have a uniform sampler, the network automatically deals with the interface without knowing locations of the interface a priori.

In Fig. 4, we show the corresponding numerical results. In Fig. 4a and Fig. 4b, we plot the profiles of a shallow network approximation of the boundary condition $g(x)$ and the deep network approximation of solution $u'(x)$ to the auxiliary PDE (13), respectively. In Fig. 4d and Fig. 4e, we show the comparison between the DNN solution and the reference solution. One can see that the DNN method provides an accurate result for this interface problem.

In Fig. 4c and Fig. 4f, we plot the decay of the Lagrangian and the L_2 relative error between the DNN solution and reference solution during the training process. Interestingly we observe that optimization process gets stuck at a local

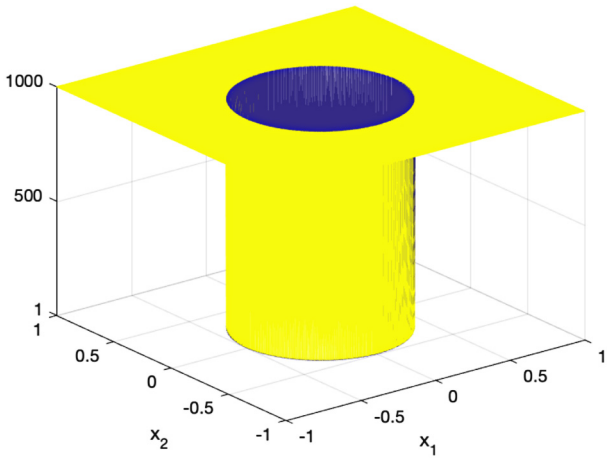


Fig. 3. Profile of a high-contrast coefficient on D.

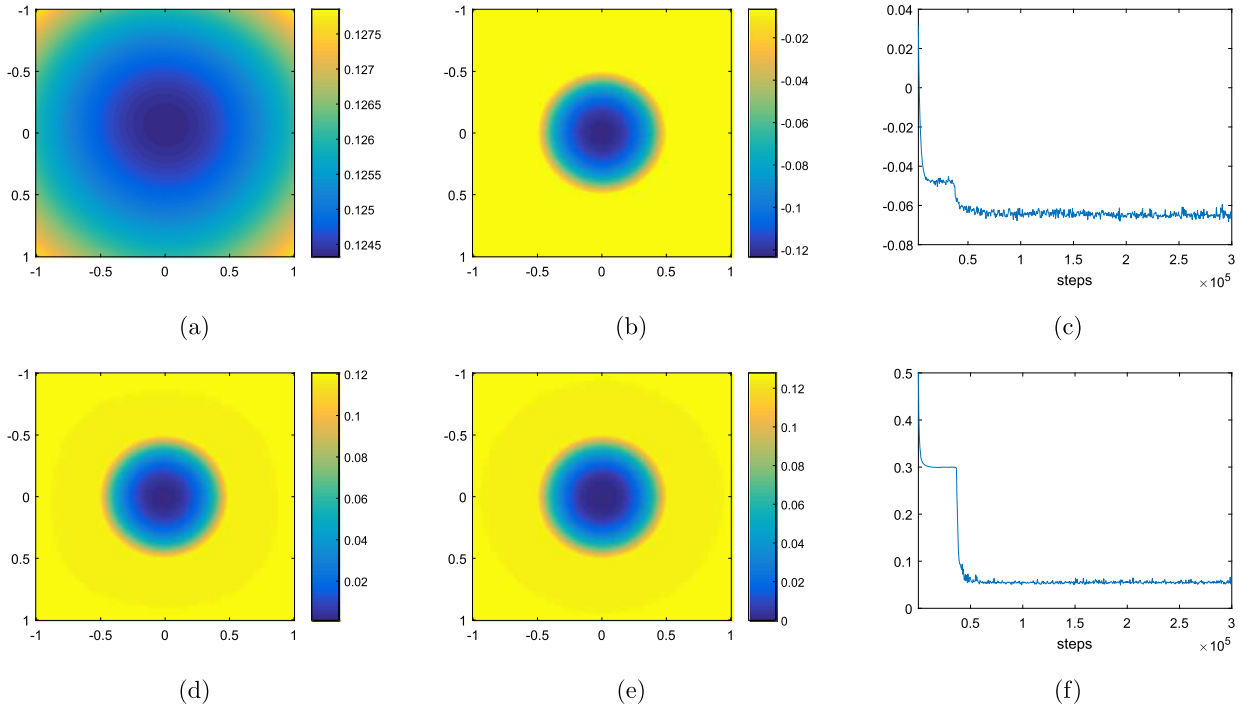


Fig. 4. High contrast problem, $\alpha_0 = 1000$, $\alpha_1 = 1$ case: (a) profile of $\tilde{g}$; (b) profile of u'; (c) decay of the Lagrangian during the training process; (d) profile of the DNN solution u at the final step; (e) profile of the reference solution; (f) decay of the L_2 relative error during the training process.

minimum at the beginning, i.e., the first four thousand steps, where the Lagrangian functional does not have decay and the error between the DNN solution and reference solution keeps as a constant. Beyond that, the optimization process jumps out the local minimum, which makes the Lagrangian functional and the error continue to decay. Finally, the error oscillates around 5%. In this experiment, the data size is about 1 GB and iteration for 3×10^5 steps costs about 3700 seconds.

In Fig. 5a, Fig. 5b, and Fig. 5c, we respectively show the solution profile when the iteration is at the initial stage, near the local minima and get out of a local minimum. One can see that the solution profile in Fig. 5b captures the profile of the exact solution to a certain extent. For such a 2D elliptic PDE, the corresponding deep learning based optimization problem has complicated structures. More in-depth research will be carried out in our future work.

In our second experiment, we choose $\alpha_0 = 1$ and $\alpha_1 = 10^3$ in (32). The profile of the new coefficient looks like an upside-down of the profile shown in Fig. 3. We do not show it here. Again, we use the exact solution (33) to serve as the reference solution and the DNN method to compute the numerical solution. The setting of the DNN method is the same as the first experiment.

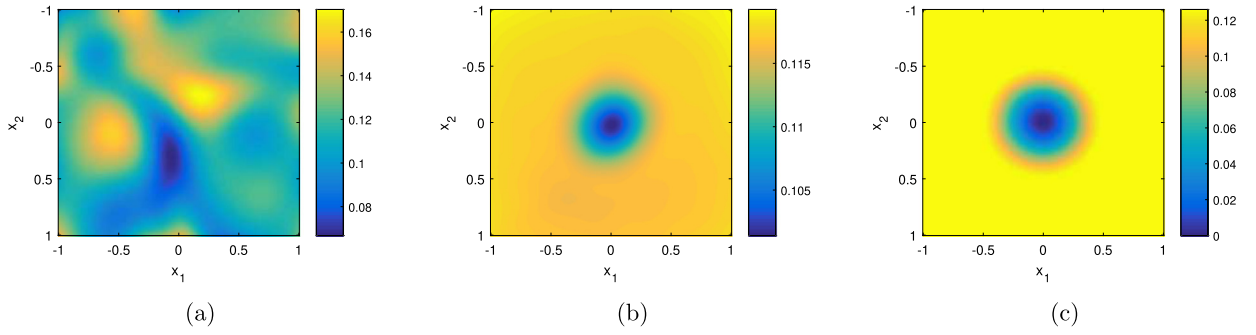


Fig. 5. High contrast problem, $\alpha_0 = 1000$, $\alpha_1 = 1$ case: (a) Solution profile when the iteration is at the initial stage; (b) Solution profile when the iteration is near the local minima; (c) Solution profile when the iteration gets out of the local minima.

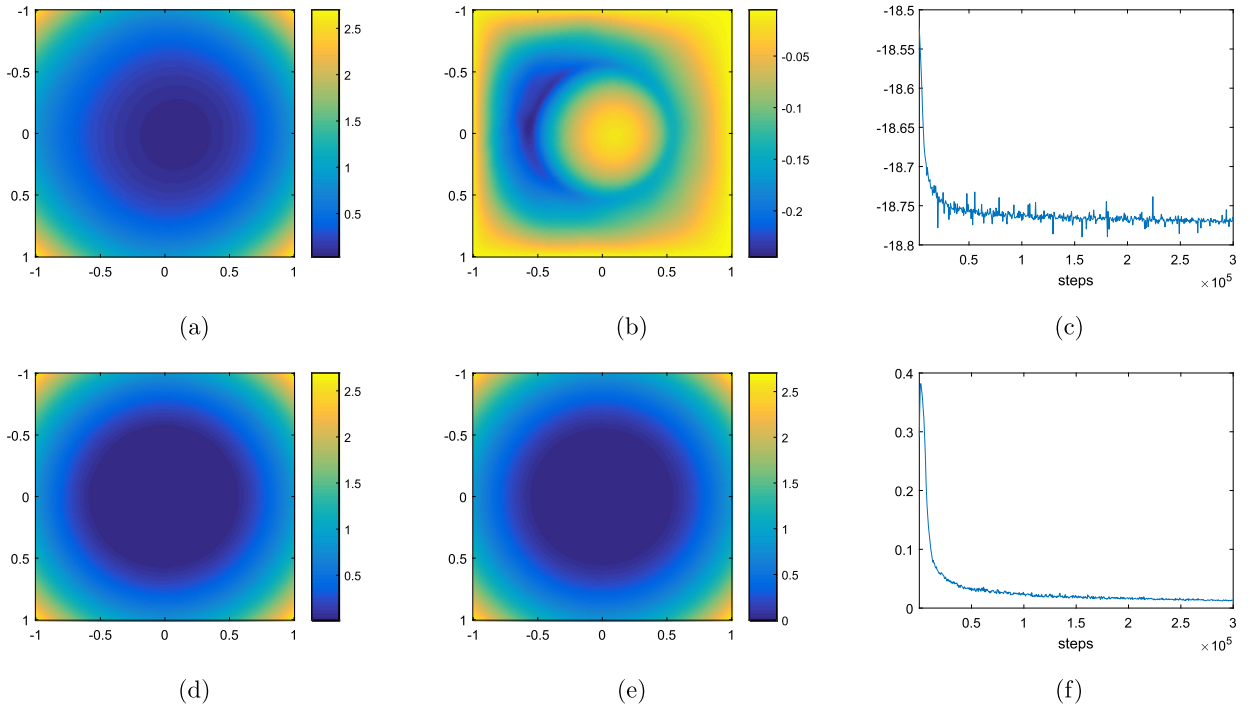
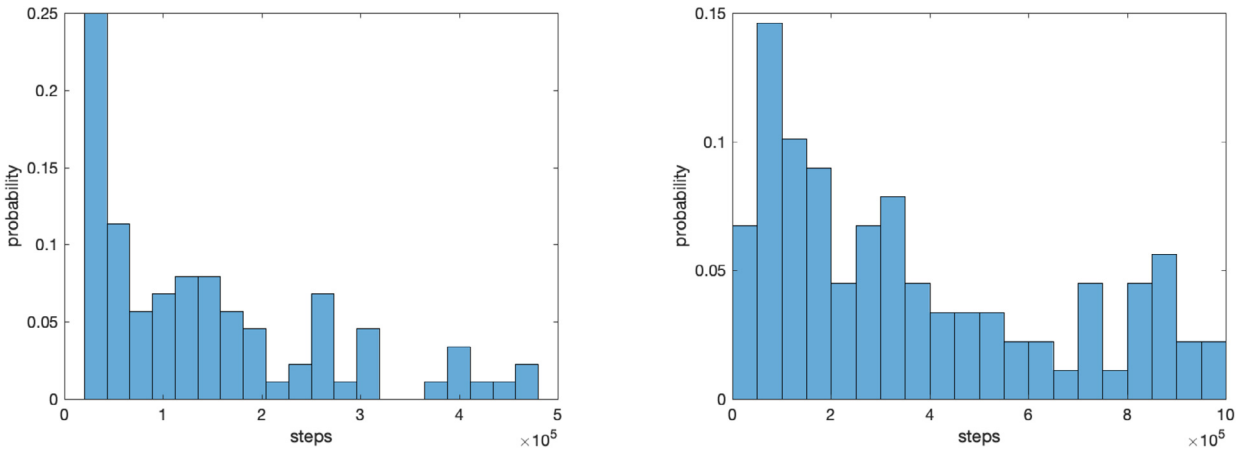


Fig. 6. High contrast problem, $\alpha_0 = 1$, $\alpha_1 = 1000$ case: (a) profile of $\tilde{g}$; (b) profile of u'; (c) decay of the Lagrangian during the training process; (d) profile of the DNN solution u at the final step; (e) profile of the reference solution; (f) decay of the L_2 relative error during the training process.

In Fig. 6, we show the corresponding numerical results. In Fig. 6a and Fig. 6b, we plot the profiles of a shallow network approximation of the boundary condition $g(x)$ and the deep network approximation of solution $u'(x)$ to the auxiliary PDE (13), respectively. In Fig. 6d and Fig. 6e, we show the comparison between the DNN solution and the reference solution. The DNN method again provides an accurate result for this interface problem.

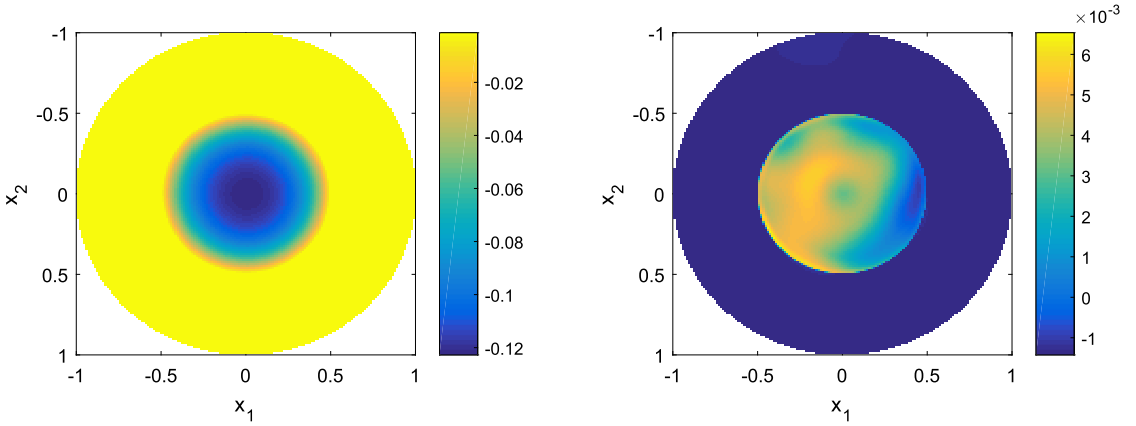
In Fig. 6c and Fig. 6f, we plot the decay of the Lagrangian and the L_2 relative error between the DNN solution and reference solution during the training process. We find that the decay pattern of the second experiment is different from the first one. The Lagrangian functional has instant fluctuations during the optimization process. However, it does not get stuck at a local minimum. The error function is a monotonic decreasing function. Finally, the error is reduced to about 2%.

The DNN method is a stochastic method since the initial value of parameters in the network, i.e. $\theta \in \Theta$ and the Adam SGD optimizer are random. We are interested in investigating the convergence speed when $\alpha_1 = 1$ and $\alpha_0 \gg 1$, which is a 'harder' case of the high-contrast problem since the optimization process of the DNN method gets stuck at a local minimum. Let n record the steps that the iteration gets out of the local minimum; see the staircase shown in Fig. 4c and Fig. 4f. In Fig. 7, we plot the histogram of the n when $\alpha_0 = 1000$ and $\alpha_0 = 10000$, respectively. The total number of iteration is set to be 5×10^5 when $\alpha_0 = 1000$ and 10^6 when $\alpha_0 = 10000$. We find that a higher contrast in the coefficient will lead to a slower convergence in the DNN method. We observe that when the contrast of the coefficient is higher, the optimization process of the DNN method has a bigger chance to get stuck at a local minimum. We also observe that about 7% of trials failed to converge within the designed steps.



(a) $\alpha_0 = 1000$, $\alpha_1 = 1$. (b) $\alpha_0 = 10000$, $\alpha_1 = 1$.

Fig. 7. Histogram of the number of steps to get out of local minima.



(a) Profile of u (b) Profile of error

Fig. 8. Problem defined on a disk domain.

To further show the benefit of the mesh-free nature of the DNN method, we consider a 2D elliptic PDE (30) defined on a closed disk D. Specifically, the domain D is a disk with radius one and centered at the origin. The coefficient $a(x)$ is defined in (32), where $\alpha_0 = 1000$ and $\alpha_1 = 1$. The source term is $f(x) = -9(x_1^2 + x_2^2)^{1/2}$.

In this experiment, we impose zero boundary condition so the reference solution differs from the exact solution in Eq. (33) only by a constant, i.e. $\frac{1}{\alpha_0} + (\frac{1}{\alpha_1} - \frac{1}{\alpha_0})r_0^3$. The implementation of the DNN method is exactly the same as the previous two experiments. In Fig. 8, we show the numerical solution obtained by our method and the numerical error. The final L_2 relative error is about 4.5%. This numerical result demonstrates that once we have a sampling method to generate collocation points in the interior domain and on the boundary, we can use the DNN method to solve PDEs, where we do not need to specially treat the locations of the interface and/or the shape of the boundary. Thus, the DNN method can be used to solve PDEs defined in irregular domains.

5.2. More discussions on our method based on a 2D high-contrast problem

In our proposed method, there are some parameters that determine the accuracy of the DNN method, such as the batch number (the number of collocation points used in computing the integration in the variational problem), the learning rate in the SGD, etc. In this subsection, we shall carry out several numerical experiments to study the performance of our method on those parameters.

We consider a 2D multiscale elliptic PDE (30) on the domain $D = [0, 1]^2$, where the coefficient $a(x)$ contains high-contrast inclusions and channels; see Fig. 9. This type of coefficient is used to mimic complicated permeability fields in the reservoir simulation [10]. The performance of the proposed method for inhomogeneous boundary conditions has been

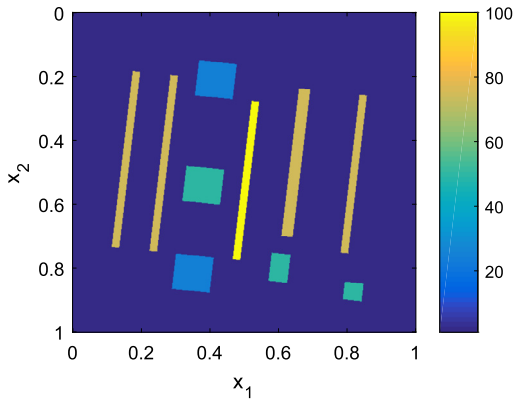


Fig. 9. Profile of a high-contrast coefficient on D.

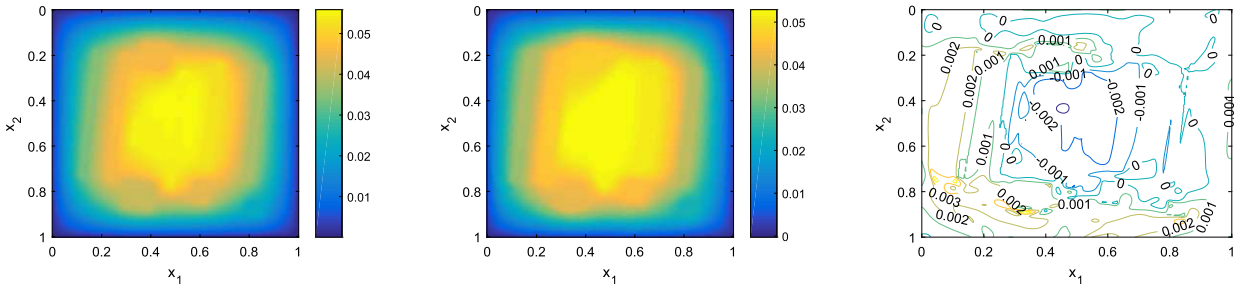


Fig. 10. (a) reference solution; (b) numerical solution; (c) contour plot of the error.

verified in the previous subsection. Here, we choose homogeneous boundary condition $g(x) = 0$ on ∂D. The networks that we used for approximating the boundary condition and the solution are shown in Fig. 1 and Fig. 2, respectively.

We use the FEM with fine mesh $h = \frac{1}{128}$ to compute the reference solution and the DNN method to compute the numerical solution. In the learning process, i.e., the running of the SGD method, we choose the batch number to be 4096 (that contains 3840 points in the interior domain of D and 256 points on the boundary ∂D) and generate a new batch every 10 steps of updating. The learning rate is $\eta = 2 \times 10^{-3}$. Once we have a uniform sampler, the network automatically deals with the interface without knowing locations of the interface a priori. Finally, the final L_2 relative error is about 3%.

In Fig. 10, we respectively show the reference solution, the numerical solution and the contour plot of the error. In this example, the solution contains sharp edges due to the heterogeneity in the coefficient. Numerical results show that the DNN method can capture those sharp edges well.

In what follows, we test the influence of hyper-parameters on the accuracy of the DNN method. We first choose different batch numbers, i.e. $N = 1024$, $N = 4096$ and $N = 16384$, and keep other parameters in the DNN method unchanged. Again, we put 6.25%N points on the boundary and the remaining points in the interior domain. We show the numerical solutions obtained by different batch numbers in Fig. 11a - Fig. 11c. Moreover, we plot the decay of the L_2 relative error during the training process together in Fig. 11d. One can see that when the batch number is not big enough, say $N = 1024$, the DNN solution cannot capture the edges well thus leads to a big numerical error. The DNN method with $N = 4096$ and $N = 16384$ batch numbers gives almost identical solutions. These numerical results indicate that the setting of the batch number is essential in the accuracy of the DNN method.

We also test the influence of the learning rate on the accuracy of the DNN method. Notice that the learning rate and training step are highly related. In this experiment, we choose four different learning rates, i.e., $\eta = 1 \times 10^{-3}$, $\eta = 2 \times 10^{-3}$, $\eta = 4 \times 10^{-3}$ and $\eta = 8 \times 10^{-3}$, and keep other parameters in the DNN method unchanged, where the batch number is $N = 4096$. We plot the decay of the L_2 relative error vs *learning rate* $\times$ *training steps* during the training process together in Fig. 12. One can see that the final L_2 errors of each experiment are close thus it seems to suggest that the DNN method is robust in terms of the learning rate. However, one can also see that a larger learning rate leads to greater fluctuations in the decay of error during the training process.

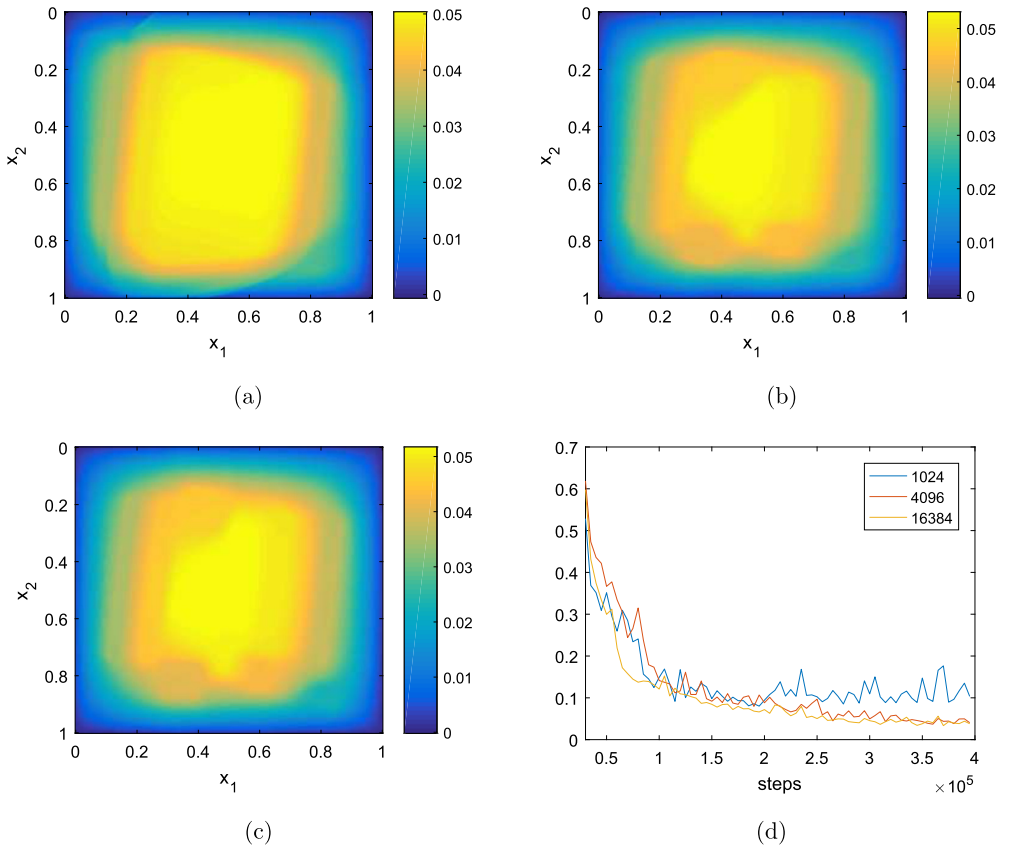


Fig. 11. Influence of batch number: (a) $N = 1024$; (b) $N = 4096$; (c) $N = 16384$; (d) L_2 relative error during the training process.

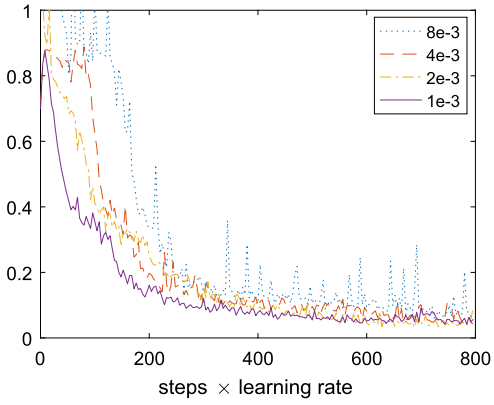


Fig. 12. L_2 relative error with different learning rate η.

5.3. 2D Linear elasticity interface problem

We consider the linear elasticity PDE with a discontinuous stress tensor defined in (26), where $x = (x_1, x_2)$, the domain $D = [-8, 8] \times [-8, 8]$, $\mathbf{u} = (u_1, u_2)^T$, the elasticity tensor $\mathbb{C}$ is defined by (23) or (24) with $\lambda = 1$ and $\mu = 1$. We choose $\alpha = 1$ in the jump of the stress tensor across the cell.

In the cell model [41], keratocytes typically have a roughly circular shape with an annular lamellipodium surrounding the nucleus, when they are in a stationary state. Contact and force transmission with the substrate occurs only at the lamellipodium and not the nucleus and organelles. Accordingly, we choose the initial lamellipodium region Ω to be an annulus in the center of the square domain D, with the nucleus excluded; see Fig. 13.

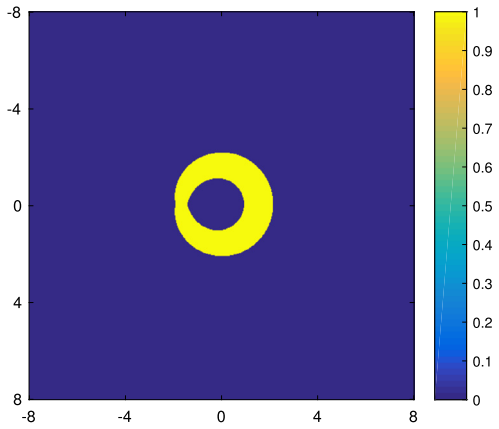


Fig. 13. Value of χ on D, where the yellow region is Ω.

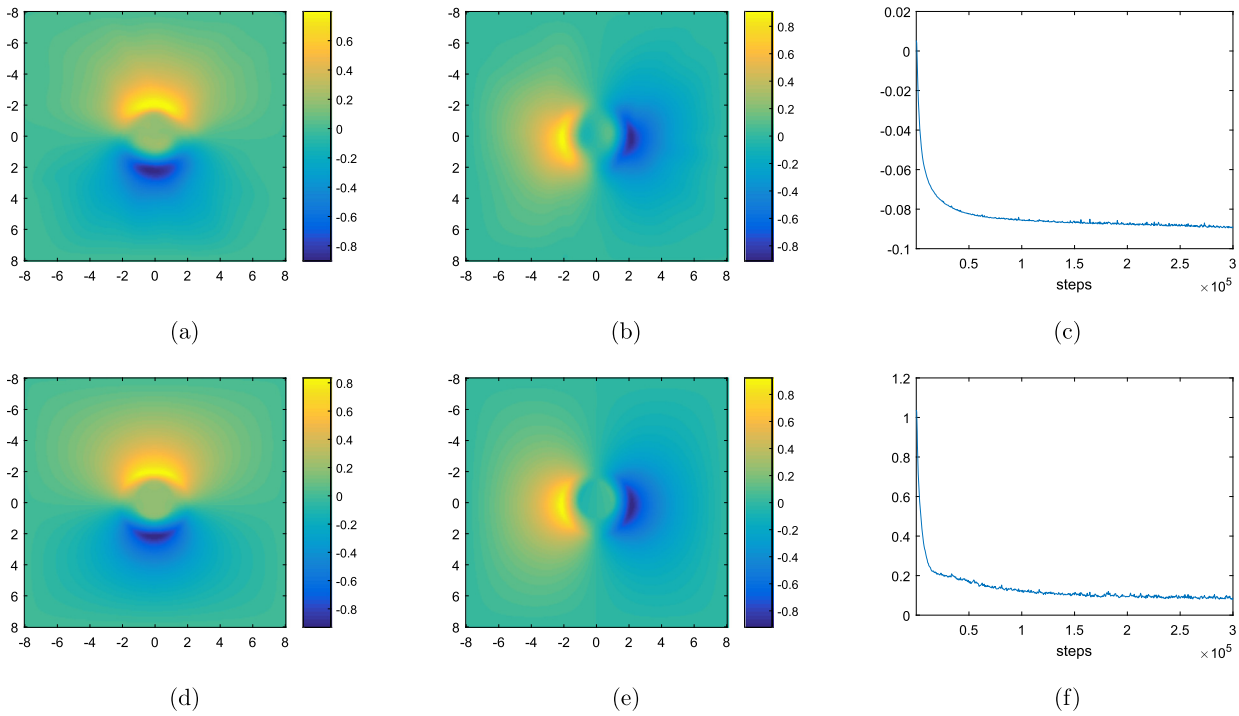


Fig. 14. 2D Linear elasticity interface problem: (a) profile of DNN solution u_1; (b) profile of DNN solution u_2; (c) decay of the Lagrangian during the training process; (d) profile of reference solution u_1; (e) profile of reference solution u_2; (f) decay of the L_2 relative error during the training process.

We set $u_1 = u_2 = 0$ on the boundary of D, which gives a null displacement or traction-free boundary condition. On the boundary of the cell Ω, we impose the jump conditions (27).

We use the immersed-interface FEM with a fine mesh $h = 1/32$ to compute the reference solution and the DNN method to compute the numerical solution. The network that we used has four intermediate layers and the width of each layer is twenty, where the network is densely connected. In the running of the SGD method, we choose the batch number to be 2048 (that contains 1920 points in the interior domain of D and 128 points on the boundary ∂D) and generate a new batch every 10 steps of updating. The learning rate η is 5×10^{-4}.

In Fig. 14, we show the corresponding numerical results. In Fig. 14a and Fig. 14b, we plot the profiles of DNN solutions u_1 and u_2, which are the displacements in x_1 and x_2 coordinates, respectively. The corresponding reference solutions are shown in Fig. 14d and Fig. 14e. We find that the DNN solutions agree well with the reference solutions. In Fig. 14c and Fig. 14f, we plot the decay of the Lagrangian and the L_2 relative error between the DNN solution and reference solution during the training process. We find that both the Lagrangian functional and error function are monotonic decreasing functions. Finally the error is reduced to about 4%. Our numerical results imply that the DNN method is efficient in solving the 2D Linear elasticity interface problem (26). Most importantly, its implementation is very simple.

6. Conclusions

In this paper, we proposed a DNN method to solve interface problems. By formulating the PDEs into variational problems, we convert the interface problems into optimization problems. Since the ReLU-DNN can be used to approximate the linear space spanned by FEM nodal basis functions. Thus, we parameterize the PDE solutions by using the ReLU-DNNs and solve the interface problems by searching the minimizer of the associated optimization problems. In this framework, once we have samplers of collocation points in the domain and on the boundary, we do not need any special treatment to deal with the interface inside the domain and/or the shape of the boundary. Therefore, the proposed method is easy to implement and mesh-free.

Finally, we use the DNN method to solve elliptic PDEs with discontinuous and high-contrast coefficients and linear elasticity with discontinuous stress tensors. We find the ReLU-DNN with enough hidden layers and enough neurons with each layer can approximate the solutions of the target PDEs well. Although the parameter space of the ReLU-DNN is huge, the SGD method can efficiently solve the optimization problems. Numerical results show that the accuracy of the DNN method depends on the expressive power of the DNNs and the batch number in the SGD method. It seems that the DNN method is not very sensitive to the learning rate. Therefore, the DNN method provides an effective alternative to solve interface problems.

There are several issues remain open. For instance, we do not get the convergence rate for the DNN method and we have little understanding about the parameter space of the DNN. In addition, the issue of local minima and saddle points in the optimization problem is highly nontrivial. We are interested in studying these issues in our future research.

Declaration of competing interest

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

Acknowledgements

The research of Z. Wang is partially supported by the Hong Kong PhD Fellowship Scheme. The research of Z. Zhang is supported by Hong Kong RGC grants (Projects 27300616, 17300817, and 17300318), National Natural Science Foundation of China (Project 11601457), Basic Research Programme (JCYJ20180307151603959) of the Science, Technology and Innovation Commission of Shenzhen Municipality, Seed Funding Programme for Basic Research (HKU), and an RAE Improvement Fund from the Faculty of Science (HKU). The computations were performed using the HKU ITS research computing facilities that are supported in part by the Hong Kong UGC Special Equipment Grant (SEG HKU09). We would like to thank Professor Thomas Hou for stimulating discussions.

References

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, Tensorflow: a system for large-scale machine learning, in: OSDI, vol. 16, 2016, pp. 265–283.
[2] I. Babuška, The finite element method for elliptic equations with discontinuous coefficients, Computing 5 (3) (1970) 207–213.
[3] C. Bernardi, R. Verfürth, Adaptive finite element methods for elliptic equations with non-smooth coefficients, Numer. Math. 85 (4) (2000) 579–608.
[4] L. Bottou, Large-scale machine learning with stochastic gradient descent, in: Proceedings of COMPSTAT'2010, Springer, 2010, pp. 177–186.
[5] Z. Chen, J. Zou, Finite element methods and their convergence for elliptic and parabolic interface problems, Numer. Math. 79 (2) (1998) 175–202.
[6] C. Chu, I. Graham, T.Y. Hou, A new multiscale finite element method for high-contrast elliptic interface problems, Math. Comput. 79 (2010) 1915–1955.
[7] N. Cohen, O. Sharir, A. Shashua, On the expressive power of deep learning: a tensor analysis, in: Conference on Learning Theory, 2016, pp. 698–728.
[8] G. Cybenko, Approximation by superpositions of a sigmoidal function, Math. Control Signals Syst. 2 (4) (1989) 303–314.
[9] Y. Efendiev, J. Galvis, X. Wu, Multiscale finite element methods for high-contrast problems using local spectral basis functions, J. Comput. Phys. 230 (4) (2011) 937–955.
[10] Y. Efendiev, T.Y. Hou, Multiscale Finite Element Methods. Theory and Applications, Springer-Verlag, New York, 2009.
[11] S. Ellacott, Aspects of the numerical analysis of neural networks, Acta Numer. 3 (1994) 145–202.
[12] R. Fedkiw, T. Aslam, B. Merriman, S. Osher, A non-oscillatory eulerian approach to interfaces in multimaterial flows (the ghost fluid method), J. Comput. Phys. 152 (2) (1999) 457–492.
[13] Y. Gong, B. Li, Z. Li, Immersed-interface finite-element methods for elliptic interface problems with nonhomogeneous jump conditions, SIAM J. Numer. Anal. 46 (1) (2008) 472–495.
[14] I. Goodfellow, Y. Bengio, A. Courville, Y. Bengio, Deep Learning, vol. 1, MIT Press, Cambridge, 2016.
[15] J. Han, A. Jentzen, W. E, Solving high-dimensional partial differential equations using deep learning, Proc. Natl. Acad. Sci. 115 (34) (2018) 8505–8510.
[16] J. He, L. Li, J. Xu, C. Zheng, Relu deep neural networks and linear finite elements, arXiv:1807.03973, 2018.
[17] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural Netw. 2 (5) (1989) 359–366.
[18] S. Karumuri, R. Tripathy, I. Bilionis, J. Panchal, Simulator-free solution of high-dimensional stochastic elliptic partial differential equations using deep neural networks, arXiv:1902.05200, 2019.
[19] Y. Khoo, J. Lu, L. Ying, Solving parametric pde problems with artificial neural networks, arXiv:1707.03351, 2017.
[20] D. Kingma, J. Ba, Adam: a method for stochastic optimization, arXiv:1412.6980, 2014.
[21] I. Lagaris, A. Likas, D. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, IEEE Trans. Neural Netw. 9 (5) (1998) 987–1000.
[22] Yann LeCun, Yoshua Bengio, Geoffrey Hinton, Deep learning, Nature 521 (7553) (2015) 436.
[23] H. Lee, Neural algorithm for solving differential equations, J. Comput. Phys. 91 (1990) 110–131.

[24] R. Leveque, Z. Li, The immersed interface method for elliptic equations with discontinuous coefficients and singular sources, SIAM J. Numer. Anal. 31 (4) (1994) 1019–1044.
[25] Z. Li, T. Lin, X. Wu, New cartesian grid methods for interface problems using the finite element formulation, Numer. Math. 96 (1) (2003) 61–98.
[26] A. Meade, A. Fernandez, The numerical solution of linear ordinary differential equations by feedforward neural networks, Math. Comput. Model. 19 (12) (1994) 1–25.
[27] H. Montanelli, Q. Du, New error bounds for deep ReLU networks using sparse grids, SIAM J. Math. Data Sci. 1 (1) (2019) 78–92.
[28] C. Peskin, Numerical analysis of blood flow in the heart, J. Comput. Phys. 25 (3) (1977) 220–252.
[29] C. Peskin, The immersed boundary method, Acta Numer. 11 (2002) 479–517.
[30] A. Pinkus, Approximation theory of the MLP model in neural networks, Acta Numer. 8 (1999) 143–195.
[31] M. Raissi, P. Perdikaris, G. Karniadakis, Multistep neural networks for data-driven discovery of nonlinear dynamical systems, arXiv:1801.01236, 2018.
[32] M. Raissi, P. Perdikaris, G. 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.
[33] C. Schwab, J. Zech, Deep Learning in High Dimension, Research Report, 2017, 2017.
[34] J. Sirignano, K. Spiliopoulos, DGM: a deep learning algorithm for solving partial differential equations, J. Comput. Phys. 375 (2018) 1339–1364.
[35] B. Wang, K. Xia, G. Wei, Matched interface and boundary method for elasticity interface problems, J. Comput. Appl. Math. 285 (2015) 203–225.
[36] Y. Wang, S. Cheung, E. Chung, Y. Efendiev, M. Wang, Deep multiscale model learning, arXiv:1806.04830, 2018.
[37] Weinan E, Jiequn Han, Arnulf Jentzen, Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations, Commun. Math. Stat. 5 (4) (2017) 349–380.
[38] Weinan E, Bing Yu, The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems, Commun. Math. Stat. 6 (1) (2018) 1–12.
[39] X. Yang, B. Li, Z. Li, The immersed interface method for elasticity problems with interface, Dyn. Contin. Discrete Impuls. Syst., Ser. A Math. Anal. 10 (2003) 783–808.
[40] D. Yarotsky, Error bounds for approximations with deep ReLU networks, Neural Netw. 94 (2017) 103–114.
[41] Z. Zhang, P. Rosakis, T. Hou, G. Ravichandran, A minimal mechanosensing model predicts keratocyte evolution on flexible substrates, arXiv:1803.09220, 2018.
[42] Y. Zhu, N. Zabaras, Bayesian deep convolutional encoder-decoder networks for surrogate modeling and uncertainty quantification, J. Comput. Phys. 366 (2018) 415–447.
[43] Y. Zhu, N. Zabaras, P. Koutsourelakis, P. Perdikaris, Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data, J. Comput. Phys. 394 (2019) 56–81.