STRUCTURE PROBING NEURAL NETWORK DEFLATION

YIQI GU

DEPARTMENT OF MATHEMATICS, NATIONAL UNIVERSITY OF SINGAPORE, 10 LOWER KENT RIDGE ROAD, SINGAPORE, 119076 (MATGUY@NUS.EDU.SG)

CHUNMEI WANG

DEPARTMENT OF MATHEMATICS & STATISTICS, TEXAS TECH UNIVERSITY, 1108 MEMORIAL CIRCLE, LUBBOCK, TX 79409, USA (CHUNMEI.WANG@TTU.EDU)

HAIZHAO YANG

DEPARTMENT OF MATHEMATICS, PURDUE UNIVERSITY, 150 N UNIVERSITY ST, WEST LAFAYETTE, IN 47907, USA (HAIZHAO@PURDUE.EDU)

Abstract. Deep learning is a powerful tool for solving nonlinear differential equations, but usually, only the solution corresponding to the flattest local minimizer can be found due to the implicit regularization of stochastic gradient descent. This paper proposes Structure Probing Neural Network Deflation (SP-NND) to make deep learning capable of identifying multiple solutions that are ubiquitous and important in nonlinear physical models. First, we introduce deflation operators built with known solutions to make known solutions no longer local minimizers of the optimization energy landscape. Second, to facilitate the convergence to the desired local minimizer, a structure probing technique is proposed to obtain an initial guess close to the desired local minimizer. Together with neural network structures carefully designed in this paper, the new regularized optimization can converge to new solutions efficiently. Due to the mesh-free nature of deep learning, SP-NND is capable of solving high-dimensional problems on complicated domains with multiple solutions, while existing methods focus on merely one or two-dimensional regular domains and are more expensive than SP-NND in operation counts. Numerical experiments also demonstrate that SP-NND could find more solutions than exiting methods.

Key words. Neural Networks Deflation; Structure Probing; Nonlinear Differential Equations; High Dimension; Deep Residual Method; Convergence.

AMS subject classifications. 65M75; 65N75; 62M45;

1. Introduction.

1.1. Problem statement.

Nonlinear differential equations are ubiquitous in various important physical models such as fluid dynamics, plasma physics, solid mechanics, and quantum field theory [27, 15, 20, 42], as well as chemical and biological models [60, 16]. Solving nonlinear differential equations has been a very challenging problem especially when it is important to find multiple distinct solutions. The nonlinearity of the differential equation may cause traditional iterative solvers to stop at a spurious solution if the initial guess is not close to a physically meaningful solution. When multiple distinct solutions are of interest, a naive strategy is to try different initial guesses as many as possible so that iterative solvers can return distinct solutions as many as possible. However, most of the initial guesses would lead to either spurious solutions or repeated solutions, making this approach usually time-consuming and inefficient unless a priori estimate of the solutions is available.

Recently, neural network-based optimization has become a powerful tool for solving nonlinear differential equations especially in high-dimensional spaces [7, 41, 34, 10, 66, 39, 33]. As a form of nonlinear parametrization through compositions of simple functions [30], deep neural networks (DNNs) can efficiently approximate various useful classes of functions without the curse of dimensionality [6, 51, 53, 57, 64, 48, 52] and achieve exponential approximation rates [63, 53, 48, 53, 45, 24, 56]. Therefore, applying DNNs to parametrize the solution space of differential equations (including boundary value problems, initial value problems, and eigenvalue problems)

and seeking a solution via energy minimization from variational formulation have become a popular choice, e.g., the residual method [7, 41] as a special case of variational formulation, the Ritz method [25], the Nitsche method [47].

However, neural network-based optimization usually can only find the smoothest solution with the fastest decay in the frequency domain due to the implicit regularization of network structures and the stochastic gradient descent (SGD) for solving the minimization problem, no matter how the initial guess is randomly selected. It was shown through the frequency principle of neural networks [67, 68, 49] and the neural tangent kernel [11] that neural networks have an implicit bias towards functions that decay fast in the Fourier domain and the gradient descent method tends to fit a low-frequency function better than a high-frequency function. Through the analysis of the optimization energy landscape of SGD, it was shown that SGD with small batches tends to converge to the flattest minimum [54, 44, 18]. Therefore, designing an efficient algorithm for neural network-based optimization to find distinct solutions as many as possible is a challenging problem.

To tackle the challenging problem just above and find distinct solutions as many as possible, we propose the structure probing neural network deflation (SP-NND) in this paper. The key idea of NND is to introduce deflation operators built with known solutions to regularize deep learning optimization, making known solutions no longer local minimizers of the optimization energy landscape while preserving unknown solutions as local minimizers. In particular, we introduce a deflation functional mapping known solutions to infinity. We multiply this deflation functional to the original optimization loss function, then the known solutions will be removed from consideration and unknown solutions can be found by optimizing the regularized loss function via SGD. Furthermore, to facilitate the convergence of SGD, we propose special network structures incorporating boundary conditions of differential equations to simplify the optimization loss function. Finally, a novel structure-probing (SP) algorithm is proposed to initialize the NND optimization making it more powerful to identify distinct solutions with desired structures.

As a general framework, NND can be applied to all neural network-based optimization methods for differential equations. In this paper, we will take the example of boundary value problem (BVP) and the residual method (RM) without loss of generality. The generalization to other problems and methods is similar. Consider the boundary value problem (BVP)

$$\begin{aligned} &\mathcal{D}u(\boldsymbol{x}) = f(u(\boldsymbol{x}), \boldsymbol{x}), \text{ in } \Omega, \\ &\mathcal{B}u(\boldsymbol{x}) = g(\boldsymbol{x}), \text{ on } \partial\Omega, \end{aligned} \tag{1.1}$$

where $\mathcal{D} : \Omega \to \Omega$ is a differential operator that can be nonlinear, $f(u(\boldsymbol{x}), \boldsymbol{x})$ can be a nonlinear function in u, Ω is a bounded domain in $\mathbb{R}^d$, and $\mathcal{B}u = g$ characterizes the boundary condition. Other types of problems like initial value problems can also be formulated as a BVP as discusssed in [33]. Then RM seeks a solution $u(\boldsymbol{x}; \boldsymbol{\theta})$ as a neural network with a parameter set $\boldsymbol{\theta}$ via the following optimization problem

$$\min_{\boldsymbol{\theta}} L_{\mathrm{RM}} := \|\mathcal{D}u(\boldsymbol{x}; \boldsymbol{\theta}) - f(u, \boldsymbol{x})\|^2_{L^2(\Omega)} + \lambda \|\mathcal{B}u(\boldsymbol{x}; \boldsymbol{\theta}) - g(\boldsymbol{x})\|^2_{L^2(\partial\Omega)}, \tag{1.2}$$

where L_{RM} is the loss function measuring the L^2 norms of the differential equation residual $\mathcal{D}u(\boldsymbol{x}; \boldsymbol{\theta}) - f(u, \boldsymbol{x})$ and the boundary residual $\mathcal{B}u(\boldsymbol{x}; \boldsymbol{\theta}) - g(\boldsymbol{x})$, and $\lambda > 0$ is a regularization parameter.

As we shall see, SP-NND enjoys four main advantages compared to traditional methods not based on deep learning:

- Numerical examples show that SP-NND can identify more solutions than other existing methods, e.g., see Test Case 5 in Section 5.

- As a neural network-based method, SP-NND can be applied to solve high-dimensional nonlinear differential equations with multiple solutions while existing methods are only applicable to low-dimensional problems. For example, there is a 6-dimensional Yamabe's equation in Test Case 6 in Section 5.
- SP-NND can be applied to problems with complex domains due to the flexibility of neural network parametrization, e.g., see Test Cases 5 & 6 in Section 5.
- As we shall discuss in Section 3.4, SP-NND admits lower computational complexity of $O(N^2)$ compared to the computational complexity of $O(N^3)$ in existing methods like the original deflation method in [26], where N is the degree of freedom for each method.

1.2. Related work. The deflation technique can be traced back to the last century for identifying distinct roots of scalar polynomials [59]. This technique was extended to find roots of systems of nonlinear algebraic equations by Brown and Gearhart in [9], where deflation matrices were constructed with old roots to transform the residual of a system of nonlinear algebraic equations so that iterative methods applied to the new residual will only converge to a new root. In [26], Ferrell et al. extended the theoretical framework of Brown and Gearhart [9] to the case of infinite-dimensional Banach spaces with new classes of deflation operators, enabling the Newton-Kantorovitch iteration to converge to several distinct solutions of nonlinear differential equations even with the same initial guess.

Another well-established method for distinct solutions of differential equations is based on the numerical continuation [4, 3, 12, 14], where the basic idea of which is to transform the known solutions of a simple start system gradually to the desired solutions of a difficult target system. For example, [5] proposed coefficient-parameter polynomial continuation for computing all geometrically isolated solutions to polynomial systems. [35] put forward a bootstrapping approach for computing multiple solutions of differential equations using a homotopy continuation method with domain decomposition to speed up computation. For more examples of homotopy-based methods and theory in the literature, the reader is referred to [46].

The third kind of methods to identify distinct solutions of nonlinear systems is the numerical integration of the Davidenko differential equation associated with the original nonlinear problem [8, 19]. The basic idea is to introduce an artificial time parameter s such that solving the original nonlinear equation $F(u(\boldsymbol{x})) = 0$ to identify a solution $u_0(\boldsymbol{x})$ is equivalent to finding a steady state solution of a time-dependent nonlinear equation $\frac{dF(u(s,\boldsymbol{x}))}{ds} + F(u(s,\boldsymbol{x})) = 0$, which provides a gradient flow of $u(s,\boldsymbol{x})$. The gradient flow forms an ordinary differential equation with a solution converging to a solution to the original problem, i.e., $\lim_{s\to\infty} u(s,\boldsymbol{x}) = u_0(\boldsymbol{x})$. This method is indeed a broad framework containing the Newton's method as a special example.

1.3. Organization. This paper is organized as follows. In Section 2, we will review the fully connected feed-forward neural network, introduce the formulation of RM for BVP, and design special network structures for four types of boundary conditions. In Section 3, the detailed formulation and implementation of NND will be presented. In Section 4, the SP initialization is introduced. Various numerical experiments are provided in Section 5 to verify the efficiency of SP-NND. Finally, we conclude this paper in Section 6.

2. Network-based Methods for Differential Equations. In this section, we introduce the network-based residual method based on fully connected feed-forward neural networks and (1.2) for solving the BVP (1.1). Moreover, special network structures for common boundary conditions are introduced to simplify the loss function in (1.2) to facilitate the convergence to the desired PDE solution. Vectors are written in the bold font to distinguish from scalars in our presentation.

2.1. Fully connected feed-forward neural network (FNN). FNNs are one of the most popular DNNs and are widely applied to network-based methods for differential equations. Mathematically speaking, for a fixed nonlinear activation function σ, FNN is the composition of L simple nonlinear functions, called hidden layer functions, in the following formulation:

$$\phi(\boldsymbol{x};\boldsymbol{\theta}) := \boldsymbol{a}^T \boldsymbol{h}_L \circ \boldsymbol{h}_{L-1} \circ \cdots \circ \boldsymbol{h}_1(\boldsymbol{x}) \quad \text{for } \boldsymbol{x} \in \mathbb{R}^d,$$

where $\boldsymbol{a} \in \mathbb{R}^{N_L}$ and $\boldsymbol{h}_\ell(\boldsymbol{x}_\ell) = \sigma\left(\boldsymbol{W}_\ell \boldsymbol{x}_\ell + \boldsymbol{b}_\ell\right)$ with $\boldsymbol{W}_\ell \in \mathbb{R}^{N_\ell \times N_{\ell-1}}$ and $\boldsymbol{b}_\ell \in \mathbb{R}^{N_\ell}$ for $\ell = 1, \ldots, L$. The activation function σ can be chosen as a rectified linear unit (ReLU) function $\sigma(x) = \max\{x, 0\}$, its polynomial $\sigma(x) = \max\{x^3, 0\}$, a hyperbolic tangent function $\sigma(x) = \tanh(x)$, etc. With the abuse of notations, $\sigma(\boldsymbol{x})$ means that σ is applied entry-wise to a vector $\boldsymbol{x}$ to obtain another vector of the same size. N_ℓ is the width of the ℓ-th layer and L is the depth of the FNN. $\boldsymbol{\theta} := \{\boldsymbol{a}, \boldsymbol{W}_\ell, \boldsymbol{b}_\ell : 1 \le \ell \le L\}$ is the set of all parameters in ϕ to determine the underlying neural network. Other kinds of neural networks are also suitable in our proposed methods, but we will adopt FNNs for simplicity.

2.2. Residual method. The RM is a least-squares optimization approach to solve general differential equations. Specifically, let $u(\boldsymbol{x};\boldsymbol{\theta})$ be a neural network to approximate the solution $u(\boldsymbol{x})$ of BVP (1.1), then the RM is formulated as

$$\min_{\boldsymbol{\theta}} L_{\mathrm{RM}}(\boldsymbol{\theta}) := \|\mathcal{D}u(\boldsymbol{x};\boldsymbol{\theta}) - f(\boldsymbol{x})\|^2_{L^2(\Omega)} + \lambda \|\mathcal{B}u(\boldsymbol{x};\boldsymbol{\theta}) - g(\boldsymbol{x})\|^2_{L^2(\partial\Omega)}, \tag{2.1}$$

where L_{RM} is the loss function measuring the weighted magnitude of the differential equation residual $\mathcal{D}u(\boldsymbol{x};\boldsymbol{\theta}) - f(\boldsymbol{x})$ and the boundary residual $\mathcal{B}u(\boldsymbol{x};\boldsymbol{\theta}) - g(\boldsymbol{x})$ in the sense of L^2-norm with a weight parameter $\lambda > 0$.

The goal of (2.1) is to find an appropriate set of parameters θ such that the network $u(\boldsymbol{x};\boldsymbol{\theta})$ minimizes the loss L_{RM}. If the loss L_{RM} is minimized to zero with some $\boldsymbol{\theta}$, then $u(\boldsymbol{x};\boldsymbol{\theta})$ satisfies $\mathcal{D}u(\boldsymbol{x};\boldsymbol{\theta}) - f(\boldsymbol{x}) = 0$ in Ω and $\mathcal{B}u(\boldsymbol{x};\boldsymbol{\theta}) - g(\boldsymbol{x}) = 0$ on $\partial\Omega$, implying that $u(\boldsymbol{x};\boldsymbol{\theta})$ is exactly a solution of (1.1). If L_{RM} is minimized to a nonzero but small positive number, $u(\boldsymbol{x};\boldsymbol{\theta})$ is close to the true solution as long as (1.1) is well-posed (e.g. the elliptic PDE with Neumann boundary condition, see Theorem 4.1 in [33]).

In general, the optimization problem (2.1) is solved by stochastic gradient descent (SGD) method or its variants (e.g. Adagrad [22], Adam [43] and AMSGrad [58]) in the deep-learning framework. The optimization and mesh-free setting of RM with neural networks admit several advantageous features that led to its great success and popularity including but not limited to 1) the capacity to solve high-dimensional problems; 2) the flexibility to solve equations of various forms on complicated problem domains; 3) the simple and high-performance implementation with automatic differential programming in existing open-source software.

2.3. Special network structures for boundary conditions. In numerical implementation, the RM loss function in (2.1) heavily relies on the selection of a suitable weight parameter λ and a suitable initial guess. If λ is not appropriate, it may be difficult to identify a reasonably good minimizer of (2.1). For instance, in the BVP (1.1) with $g \equiv 0$, if we solve (2.1) by SGD with an initial guess $\boldsymbol{\theta}^0$ such that $u(\boldsymbol{x};\boldsymbol{\theta}^0) \approx 0$, SGD might converge to a local minimizer corresponding to a solution neural network close to a constant zero, which is far away from the desired solution, especially when the differential operator $\mathcal{D}$ is highly nonlinear or λ is too large. The undesired local minimizer is due to the fact that the boundary residual $\|\mathcal{B}u(\boldsymbol{x};\boldsymbol{\theta}) - g(\boldsymbol{x})\|$ overwhelms the equation residual $\|\mathcal{D}u(\boldsymbol{x};\boldsymbol{\theta}) - f(\boldsymbol{x})\|$ in the loss function.

The issue just above motivates us to design special networks $u(\boldsymbol{x};\boldsymbol{\theta})$ that satisfy the boundary condition $\mathcal{B}u(\boldsymbol{x};\boldsymbol{\theta}) = g(\boldsymbol{x})$ automatically and hence we can simplify the RM loss function from

(2.1) to

$$\min_{\boldsymbol{\theta}} L_{\mathrm{RM}}(\boldsymbol{\theta}) := \|\mathcal{D}u(\boldsymbol{x};\boldsymbol{\theta}) - f(\boldsymbol{x})\|^2_{L^2(\Omega)}. \tag{2.2}$$

As we shall see in the numerical section, our numerical tests show that such simplification can help SGD to converge to desired solutions rather than spurious solutions. The design of these special neural networks depends on the type of boundary conditions. We will discuss four common types of boundary conditions by taking one-dimensional problems defined in the domain $\Omega = [a, b]$ as an example. Network structures for more complicated boundary conditions in high-dimensional domains can be constructed similarly. In what follows, denote by $\hat{u}(x;\boldsymbol{\theta})$ a generic deep neural network with trainable parameters $\boldsymbol{\theta}$. We will augment $\hat{u}(x;\boldsymbol{\theta})$ with several specially designed functions to obtain a final network $u(x;\boldsymbol{\theta})$ that satisfies $\mathcal{B}u(x;\boldsymbol{\theta}) = g(x)$ automatically.

Case 1. Dirichlet boundary condition $u(a) = a_0$, $u(b) = b_0$.

In this case, we can introduce two special functions $h_1(x)$ and $l_1(x)$ to augment $\hat{u}(x;\boldsymbol{\theta})$ to obtain the final network $u(x;\boldsymbol{\theta})$:

$$u(x;\boldsymbol{\theta}) = h_1(x)\hat{u}(x;\boldsymbol{\theta}) + l_1(x). \tag{2.3}$$

Note $h_1(x)$ and $l_1(x)$ are chosen such that $u(x;\boldsymbol{\theta})$ automatically satisfies the Dirichlet $u(a;\boldsymbol{\theta}) = a_0,\ u(b;\boldsymbol{\theta}) = b_0$ no matter what $\boldsymbol{\theta}$ is. Then $u(x;\boldsymbol{\theta})$ is used to approximate the true solution of the differential equation and is trained through (2.2).

For the purpose, $l_1(x)$ is taken by a lifting function which satisfies the given Dirichlet boundary condition, i.e. $l_1(a) = a_0$, $l_1(b) = b_0$; $h_1(x)$ is taken by a special function which satisfies the homogeneous Dirichlet boundary condition, i.e. $h_1(a) = 0$, $h_1(b) = 0$. A straightforward choice for $l_1(x)$ is the linear function given by

$$l_1(x) = (b_0 - a_0)(x - a)/(b - a) + a_0.$$

For $h_1(x)$, we can set it as a (possibly fractional) polynomial with roots a and b, namely,

$$h_1(x) = (x - a)^{p_a}(x - b)^{p_b},$$

with $0 < p_a,\ p_b \leq 1$. To obtain an accurate approximation, p_a and p_b should be chosen to be consistent with the orders of a and b of the true solution, hence no singularity will be brought into the network structure. For regular solutions, we take $p_a = p_b = 1$; for singular solutions, p_a and p_b would take fractional values. For instance, in the case of a fractional Laplace equation $(-\Delta)^s u = f$ for $0 < s < 1$ on the domain $\Omega = [-1, 1]$ with boundary conditions $u(\pm 1) = 0$, the true solution $u(x)$ has the property that $u(x) = (x-1)^s(x+1)^s v(x)$ with $v(x)$ as a smooth function [1, 23]. Then in the construction of $u(x;\boldsymbol{\theta})$, it is reasonable to choose $h_1(x) = (x - 1)^s(x - 1)^s$ and $l_1(x) = 0$.

Case 2. one-sided condition $u(a) = a_0$, $u'(a) = a_1$.

Similarly to Case 1, the special network can be constructed by $u(x;\boldsymbol{\theta}) = h_2(x)\hat{u}(x;\boldsymbol{\theta}) + l_2(x)$, where the lifting function $l_2(x)$ is given by

$$l_2(x) = a_1(x - a) + a_0,$$

and $h_2(x)$ is set as

$$h_2(x) = (x - a)^{p_a}, \tag{2.4}$$

with $1 < p_a \leq 2$. Such p_a guarantees $h_2(x)\hat{u}(x;\boldsymbol{\theta})$ and its first derivative both vanish at $x = a$.

Case 3. mixed boundary condition $u'(a) = a_0$, $u(b) = b_0$.

In this case, the special network is constructed by $u(x;\boldsymbol{\theta}) = \tilde{u}(x;\boldsymbol{\theta}) + l_3(x)$ with a lifting function $l_3(x)$ chosen as a linear function satisfying the mixed boundary conditions, e.g.,

$$l_3(x) = a_0 x + b_0 - a_0 b,$$

and $\tilde{u}(x;\boldsymbol{\theta})$ satisfying the homogeneous mixed boundary conditions. In the construction of $\tilde{u}(x;\boldsymbol{\theta})$, it is inappropriate to naively take $\tilde{u}(x;\boldsymbol{\theta}) = (x-a)^{p_a}(x-b)^{p_b}$ with $1 < p_a \leq 2$ and $0 < p_b \leq 1$, following the approaches in the preceding two cases, because such $\tilde{u}(x;\boldsymbol{\theta})$ satisfies a redundant condition $\tilde{u}(a;\boldsymbol{\theta}) = 0$. Instead, we assume

$$\tilde{u}(x;\boldsymbol{\theta}) = (x-a)^{p_a}\hat{u}(x;\boldsymbol{\theta}) + c, \tag{2.5}$$

where $1 < p_a \leq 2$ and c is a network-related constant to be determined. Clearly, (2.5) implies $\tilde{u}'(a;\boldsymbol{\theta}) = 0$, whereas $\tilde{u}(a;\boldsymbol{\theta})$ has not been specified. Next, the constraint $\tilde{u}(b;\boldsymbol{\theta}) = 0$ gives $c = -(b-a)^{p_a}\hat{u}(b;\boldsymbol{\theta})$. Therefore, the special network for mixed boundary conditions can be constructed via

$$u(x;\boldsymbol{\theta}) = (x-a)^{p_a}\hat{u}(x;\boldsymbol{\theta}) - (b-a)^{p_a}\hat{u}(b;\boldsymbol{\theta}) + l_3(x). \tag{2.6}$$

Case 4. Neumann boundary condition $u'(a) = a_0$, $u'(b) = b_0$.

Similarly to Case 3, we construct the network by $u(x;\boldsymbol{\theta}) = \tilde{u}(x;\boldsymbol{\theta}) + l_4(x)$ with a lifting function $l_4(x)$ satisfying the Neumann boundary condition given by

$$l_4(x) = \frac{(b_0 - a_0)}{2(b-a)}(x-a)^2 + a_0 x.$$

And $\tilde{u}(x;\boldsymbol{\theta})$ satisfying the homogeneous Neumann boundary condition is assumed to be

$$\tilde{u}(x;\boldsymbol{\theta}) = (x-a)^{p_a}\check{u}(x;\check{\boldsymbol{\theta}}) + c_1, \tag{2.7}$$

where $1 < p_a \leq 2$, $\check{u}(x;\check{\boldsymbol{\theta}})$ is an intermediate network to be determined later, and c_1 is a network parameter to be trained together with $\check{\boldsymbol{\theta}}$. It is easy to check that $\tilde{u}'(a;\boldsymbol{\theta}) = 0$. Next, by the constraint $\tilde{u}'(b;\boldsymbol{\theta}) = p_a(b-a)^{p_a-1}\check{u}(b;\check{\boldsymbol{\theta}}) + (b-a)^{p_a}\check{u}'(b;\check{\boldsymbol{\theta}}) = 0$, we have

$$p_a\check{u}(b;\check{\boldsymbol{\theta}}) + (b-a)\check{u}'(b;\check{\boldsymbol{\theta}}) = 0,$$

which can be reformulated as

$$\Big(\exp(\frac{p_a x}{b-a})\check{u}(x;\check{\boldsymbol{\theta}})\Big)'_{x=b} = 0.$$

Therefore, we have

$$\exp(\frac{p_a x}{b-a})\check{u}(x;\check{\boldsymbol{\theta}}) = (x-b)^{p_b}\hat{u}(x;\hat{\boldsymbol{\theta}}) + c_2, \tag{2.8}$$

where $1 < p_b \leq 2$ and c_2 is another network parameter to be trained together with $\hat{\boldsymbol{\theta}}$. Finally, by combining (2.7) and (2.8), we obtain the following special network satisfying the given Neumann condition, i.e.

$$u(x;\boldsymbol{\theta}) = \exp(\frac{p_a x}{a-b})(x-a)^{p_a}\big((x-b)^{p_b}\hat{u}(x;\hat{\boldsymbol{\theta}}) + c_2\big) + c_1 + l_4(x), \tag{2.9}$$

where $\boldsymbol{\theta} = \{\hat{\boldsymbol{\theta}}, c_1, c_2\}$.

3. Neural Network Deflation. In this section, we propose the general formulation, the detailed implementation, and the computational complexity of NND. As we shall see, our method is easy to implement on high-dimensional and complex domains with a lower computational cost per iteration than other traditional deflation methods.

3.1. Formulation. A nonlinear BVP (1.1) might have multiple distinct solutions and each solution is a local minimizer of the corresponding network-based optimization, say

$$\min_{\boldsymbol{\theta}} L(u(\boldsymbol{x};\boldsymbol{\theta})), \tag{3.1}$$

where L is a generic loss function for solving differential equations. One example is the residual loss in (2.2) and L can also be other loss functions. However, due to the implicit regularization of SGD, only local minimizers in flat energy basins are likely to be found. Hence, no matter how to initialize the SGD and how to choose hyper-parameters, usually, only a few solutions can be found by minimizing (3.1) directly.

Neural network deflation (NND) is therefore introduced in this paper. We multiply the original loss function L by a prefactor from deflation operators [26] to modify the energy landscape using know solutions. The prefacor is able to exclude known solutions as local minimizers while preserving unknown solutions as local minimizers. Specifically, let $u_k(\boldsymbol{x})$ $(k = 1, \cdots, K)$ be the known solutions of the BVP (1.1), which are named deflated solutions. Then NND is formulated as the following optimization problem,

$$\min_{\boldsymbol{\theta}} L_{\mathrm{NND}} := \Big(\sum_{k=1}^{K} \frac{1}{\|u(\boldsymbol{x};\boldsymbol{\theta}) - u_k(\boldsymbol{x})\|_{L^2(\Omega)}^{p_k}} + \alpha\Big) L(u(\boldsymbol{x};\boldsymbol{\theta})), \tag{3.2}$$

where p_k is a positive power of the deflated solutions u_k for $k = 1, \cdots, K$ and $\alpha > 0$ is a shift constant. Note that a nonzero $\alpha > 0$ is used to prevent u from approaching to infinity during the training process as discussed in [26]. The modified loss function (3.2) has a new energy landscape where the known solutions $u_k(\boldsymbol{x})$ take values equal to infinity and are no longer local minimizers, while unknown solutions are still local minimizers [26]. Consequently, new solutions can be found through the regularized loss function L_{NND} via SGD.

3.2. Deflation with a varying shift. The original deflation operator introduced in [26] fixes the shift α in (3.2) as a constant. In this paper, we propose a new variant of deflation operators with a varying shift α along with the SGD iteration. Note that when α is equal or close to 0, the deflation term $\sum_{k=1}^{K} \frac{1}{\|u(\boldsymbol{x};\boldsymbol{\theta})-u_k(\boldsymbol{x})\|_{L^2(\Omega)}^{p_k}}$ dominates the loss and hence gradient descent tends to converge to what is far away from the known solutions. When α is moderately large, the original loss function $L(u(\boldsymbol{x};\boldsymbol{\theta}))$ dominates the loss and the gradient descent process tends to converge to a solution with a smaller residual. Therefore, α in this paper can be set to be a monotonically increasing function of the SGD iteration. In the early stage, α is chosen to be close to 0 such that the current solution will be pushed away from known solutions. During this stage, a large learning rate is preferable. In the latter stage when the current solution is roughly stable, α is set to be large and a small learning rate is used to obtain a small residual loss.

Practically, we increase α exponentially with a linearly increasing power over time. The shift α in the n-th SGD iteration, denoted by α_n, is given by

$$\alpha_n = 10^{p_0 + n(p_1 - p_0)/N_{\mathrm{I}}}, \tag{3.3}$$

where p_0 and p_1 are two prescribed powers with $p_0 \le p_1$, and N_{I} is the total number of iterations.

3.3. Discretization. The continuous loss functions in (2.2) and (3.2) can be approximately evaluated by stochastic sampling. The L^2-norm can be interpreted as an expectation of a random function with a random variable $\boldsymbol{x}$ in a certain domain. Hence, the expectation can be approximated by sampling $\boldsymbol{x}$ several times and computing the average function value as an approximant. Let us take $\|u(\boldsymbol{x})\|_{L^2(\Omega)}$ as an example. We generate $N_{\rm p}$ random samples $\boldsymbol{x}_i$, $i = 1, \cdots, N_{\rm p}$, which are uniformly distributed in Ω. Denote $\boldsymbol{X} := \{\boldsymbol{x}_i\}_{i=1}^{N_{\rm p}}$, then $\|u(\boldsymbol{x})\|_{L^2(\Omega)}$ is evaluated as the discrete L^2-norm denoted as $\|u(\boldsymbol{x})\|_{L^2(\boldsymbol{X})}$ via

$$\|u(\boldsymbol{x})\|_{L^2(\boldsymbol{X})} := \Big(\frac{1}{N_{\rm p}} \sum_{\boldsymbol{x}_i \in \boldsymbol{X}} |u(\boldsymbol{x}_i)|^2\Big)^{\frac{1}{2}}. \tag{3.4}$$

The discretization technique above is applied to discretize the L^2-norms in all loss functions in this paper. In the n-th iteration of gradient descent for minimizing the NND optimization problem in (3.2), assuming that the shift α is set to be α_n and the set of random samples is denoted as $\boldsymbol{X}_n$, then the discrete NND loss function is calculated by

$$\widehat{L}^{(n)}_{\rm NND}(\boldsymbol{\theta}) := \Big(\sum_{k=1}^{K} \frac{1}{\|u(\boldsymbol{x};\boldsymbol{\theta}) - u_k(\boldsymbol{x})\|^{p_k}_{L^2(\boldsymbol{X}_n)}} + \alpha_n\Big)\widehat{L}(u(\boldsymbol{x};\boldsymbol{\theta})),$$

where $\widehat{L}(u(\boldsymbol{x};\boldsymbol{\theta}))$ is a discrete approximation to $L(u(\boldsymbol{x};\boldsymbol{\theta}))$ using the same set of samples, e.g.,

$$\widehat{L}(u(\boldsymbol{x};\boldsymbol{\theta})) = \|\mathcal{D}u(\boldsymbol{x};\boldsymbol{\theta}) - f(\boldsymbol{x})\|^2_{L^2(\boldsymbol{X}_n)}$$

when the RM loss in (2.2) is applied. Then the network parameter $\boldsymbol{\theta}$ is updated by

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \tau_n \nabla_{\boldsymbol{\theta}} \widehat{L}^{(n)}_{\rm NND}(\boldsymbol{\theta}),$$

where $\tau_n > 0$ is the learning rate in the n-th iteration. In our implementation, $\boldsymbol{X}_n$ is renewed in every iteration.

3.4. Computational Complexity. Let us estimate the computational complexity of the SGD algorithm for the NND optimization (3.2) with RM loss function (2.2). Recall that $N_{\rm p}$ denotes the number of random samples in each iteration. Assume that the FNN has L layers and N neurons in each hidden layer. Note that evaluating the FNN or computing its derivative with respect to its parameters or input $\boldsymbol{x}$ via the forward or backward propagation takes $O(dN + LN^2)$ FLOPS (floating point operations per second) for each sample $\boldsymbol{x}$. Moreover, as in most existing approaches, we assume $f(\boldsymbol{x})$ in the BVP can be evaluated with $O(d)$ FLOPS for a single $\boldsymbol{x}$. Therefore, $L(u(\boldsymbol{x};\boldsymbol{\theta}))$ in (2.2) and its derivative $\nabla_{\boldsymbol{\theta}} L(u(\boldsymbol{x};\boldsymbol{\theta}))$ can be calculated with $O(N_{\rm p}(dN + LN^2))$ FLOPS using the discrete L^2-norm in (3.4), if the differential operator $\mathcal{D}$ is evaluated through finite difference approximation. Similarly, assuming the number of known solutions K is $O(1)$ and the known solutions $\{u_k(\boldsymbol{x})\}_{k=1}^K$ are stored as neural networks of width N and depth L, then the deflation factor and its gradient with respect to $\boldsymbol{\theta}$ can also be calculated with $O(N_{\rm p}(dN+LN^2))$ FLOPS. Finally, the total complexity in each gradient descent iteration of the NND optimization is $O(N_{\rm p}(dN + LN^2))$.

In existing methods [26, 17, 2], a given nonlinear differential equation is discretized via traditional discretization techniques, e.g. the finite difference method (FDM) and finite element method (FEM), resulting in a nonlinear system of algebraic equations. The solutions of the system of algebraic equations provide numerical solutions to the original nonlinear differential equation. By multiplying different deflation terms to the nonlinear system of algebraic equations, existing methods are able to identify distinct solutions via solving the deflated system by Newton's iteration.

The number of algebraic equations N_e derived by FDM is exactly the number of grid points; and the number of equations derived by FEM is exactly the number of trial functions in the Galerkin formulation.

Now we compare NND with existing deflation methods in [26, 17, 2] in terms of the computational complexity under the assumption that the degrees of freedom of these methods are equal, i.e., the number of grid points or trial functions in existing methods is equal to the number of parameters in NND, which guarantees that these methods have almost the same accuracy to find a solution. Denote the degree of freedom of these methods by W. Then by the above discussion, we have $W = N_e = O(dN + LN^2)$. Therefore, the total computational complexity in each iteration of NND is $O(N_p W)$, where N_p is usually chosen as a hyper-parameter much smaller than W. In existing methods, the Jacobian matrix in each Newton's iteration is a low-rank matrix plus a sparse matrix of size W by W. Typically, each iteration of Newton's method requires solving a linear system of the Jacobian matrix, which usually requires $O(W^2)$ FLOPS. If a good preconditioner exists or a sparse direct solver for inverting the Jacobian matrix exists, the operation count may be reduced. Consequently, the total complexity in each iteration of existing methods would be more expensive than NND depending on the performance of preconditioners.

4. Structure Probing Initialization. The initialization of parameters plays a critical role in training neural networks and has a significant impact on the ultimate performance. In the training of a general FNN, network parameters are usually randomly initialized using normal distributions with zero-mean. One popular technique is the Xavier initialization [29]: for each layer ℓ, the weight matrix $\boldsymbol{W}_\ell \in \mathbb{R}^{N_\ell \times N_{\ell-1}}$ is chosen randomly from a normal distribution with mean 0 and variance $1/N_{l-1}$; the bias vector $\boldsymbol{b}_\ell$ is initialized to be zero. As a variant of Xavier initialization, the He initialization [37] takes a slightly different variance $2/(N_{l-1} + N_l)$ for W_ℓ and $2/N_{l-1}$ for b_ℓ. In general, FNNs initialized randomly have a smooth function configuration, and hence their Fourier transform coefficients decay quickly.

The least-squares optimization problem, either for regression problems or solving linear partial differential equations, with over-parameterized FNNs and random initialization admits global convergence by gradient descent with a linear convergence rate [40, 21, 65, 13, 50]. However, the speed of convergence depends on the spectrum of the target function. The training of a randomly initialized DNN tends to first capture the low-frequency components of a target solution quickly. The high-frequency fitting error cannot be improved significantly until the low-frequency error has been eliminated, which is referred to as F-principle [62]. Related works on the learning behavior of DNNs in the frequency domain is further investigated in [49, 68, 67, 11]. In the case of nonlinear differential equations where multiple solutions exist, these theoretical works imply that deep learning-based solvers converge to solutions in the low-frequency domain unless the DNN is initialized near a solution with high-frequency components.

The discussion just above motivates us to propose the structure probing initialization that helps the training converge to multiple structured solutions. The structure probing initialization incorporates desired structures in the initialization and training of DNNs. For example, to obtain oscillatory solutions of a differential equation, we initialize the DNN with high-frequency components for the purpose of making the initialization closer to the desired oscillatory solution. During the optimization process, the magnitudes of these high-frequency components will be optimized to fit the desired solution. One choice to probe an oscillatory solution is to take a linear combination of structure probing functions with various frequencies, e.g., $\{\xi_j(\boldsymbol{x}) = e^{\mathrm{i}\boldsymbol{k}_j\cdot\boldsymbol{x}}, |\boldsymbol{k}_j| = j, j = 1, \cdots, J\}$ with $\boldsymbol{k}_j$ randomly selected. Then the following network u_J with a set of random parameters $\boldsymbol{\theta}$ can

serve as an oscillatory initial guess:

$$u_J(\boldsymbol{x};\boldsymbol{\theta}_J) = u(\boldsymbol{x};\boldsymbol{\theta}) + \sum_{j=1}^{J} c_j \xi_j(\boldsymbol{x}), \tag{4.1}$$

where $\boldsymbol{\theta}_J := \{\boldsymbol{\theta}, \{c_j\}_{j=1}^J\}$ is trainable after initialization. In the initialization, $\{c_j\}$ can be set as random numbers or manually determined hyper-parameters with large magnitudes. Instead of planewaves, radial basis functions are also a popular structure in the solution of differential equations. In this case, we can choose $\{\xi_j(\boldsymbol{x}) = \sin(j\pi|\boldsymbol{x}|), j = 1, \cdots, J\}$ for example. The idea of structure probing initialization is not limited to the above two types of structures and is application dependent.

The above paragraph has sketched out the main idea of the structure probing initialization. Now we are ready to discuss its special cases when we need to make the structure probing network u_J in (4.1) satisfy the boundary condition $\mathcal{B}u_J = g$ in the BVP (1.1), which is important for the convergence of deep learning-based solvers as discussed in Section 2.3. For this purpose, we first construct a special network $u(\boldsymbol{x};\boldsymbol{\theta})$ such that $\mathcal{B}u(\boldsymbol{x};\boldsymbol{\theta}) = g$ by the approaches described in Section 2.3. Next, the structured probing functions $\{\xi_j(\boldsymbol{x})\}$ are specifically chosen to satisfy $\mathcal{B}\xi_j(\boldsymbol{x}) = 0$ for each j. As an example, let us take the one-dimensional mixed boundary condition on $[a, b]$:

$$u'(a) = a_0, \quad u(b) = b_0 \tag{4.2}$$

for any constants a_0 and b_0. Then a feasible choice for $\xi_j(\boldsymbol{x})$ can be $\xi_j(x) = \cos(\frac{(2j-1)\pi(x-a)}{2(b-a)})$. Finally, it is easy to check that $\mathcal{B}u_J(\boldsymbol{x};\boldsymbol{\theta}) = g$.

5. Numerical Examples. In this section, several numerical examples are provided to show the performance of SP-NND in solving BVP (1.1). We choose the loss function of RM as the general loss function $L(u(\boldsymbol{x};\boldsymbol{\theta}))$ in (3.2), then the optimization problem of SP-NND is formulated as

$$\min_{\boldsymbol{\theta}} L_{\mathrm{NND}}(\boldsymbol{\theta}) := \Big(\sum_{k=1}^{K} \frac{1}{\|u(\boldsymbol{x};\boldsymbol{\theta}) - u_k(\boldsymbol{x})\|_{L^2(\Omega)}^{p_k}} + \alpha\Big) \|\mathcal{D}u(\boldsymbol{x};\boldsymbol{\theta}) - f(\boldsymbol{x})\|_{L^2(\Omega)}^2, \tag{5.1}$$

where $u(\boldsymbol{x};\boldsymbol{\theta})$ is the neural network of the approximate solution to be determined. Remark that the optimization problem can also be formulated by other optimization-based methods instead of least squares.

To verify the effectiveness of special networks that satisfy boundary conditions automatically, we use NND without the special network for boundary conditions as a comparison, where the loss function of NND becomes

$$\begin{aligned}\min_{\boldsymbol{\theta}} L_{\mathrm{NND}}(\boldsymbol{\theta}) := \Big(\sum_{k=1}^{K} \frac{1}{\|u(\boldsymbol{x};\boldsymbol{\theta}) - u_k(\boldsymbol{x})\|_{L^2(\Omega)}^{p_k}} + \alpha\Big) \cdot \\ \Big(\|\mathcal{D}u(\boldsymbol{x};\boldsymbol{\theta}) - f(u,\boldsymbol{x})\|_{L^2(\Omega)}^2 + \lambda \|\mathcal{B}u(\boldsymbol{x};\boldsymbol{\theta}) - g(\boldsymbol{x})\|_{L^2(\partial\Omega)}^2\Big).\end{aligned} \tag{5.2}$$

The overall setting for all examples is summarized as follows.

- **Environment.** The experiments are performed in Python 3.7 environment. We utilize PyTorch library for neural network implementation and CUDA 10.0 toolkit for GPU-based parallel computing. One-dimensional examples (Test Case 1-4) can be implemented on a laptop and high-dimensional examples (Test Case 5-6) are implemented on a scientific workstation.

- **Optimizer.** In all examples, the optimization problems are solved by *adam* subroutine from PyTorch library with default hyper parameters. This subroutine implements the Adam algorithm in [43].
- **Learning rate.** In each example, the learning rate is set to decay exponentially with linearly decreasing powers. Specifically, the learning rate in the n-th iteration is set as

$$\tau_n = 10^{q_0 + n(q_1 - q_0)/N_{\mathrm{I}}}, \tag{5.3}$$

  where $q_0 > q_1$ are the initial and final powers, respectively, and N_{I} denotes the total number of iterations.
- **Network setting.** In each example, we construct a special network that satisfies the given boundary condition as discussed in Section 2.3. The special network involves a generic FNN, denoted by $\hat{u}$. In all examples, we set the depth of $\hat{u}$ as a fixed number $L = 3$, but the width depends on individual examples. Unless specified particularly, all weights and biases of $\hat{u}$ are initialized by $\boldsymbol{W}_l, \boldsymbol{b}_l \sim U(-\sqrt{N_{l-1}}, \sqrt{N_{l-1}})$. The activation function of $\hat{u}$ is chosen as $\sigma(x) := \max(0, x^3)$.
- **Varying shifts in deflation operators.** In one-dimensional examples (Test Case 1-4), using constant shifts is sufficient to find all solutions. In high-dimensional examples (Test Case 5-6), varying shifts will help to find more distinct solutions. In these examples, we set varying shifts according to (3.3).

We also summarize the numerical examples in this section in Table 5.1 below, which could help the reader to better understand how the extensive numerical examples demonstrate the advantages of our new ideas in this paper: 1) neural network deflation (NND); 2) structure probing initialization (SP); 3) special network for boundary conditions (BC); 4) varying shifts in deflation operators (VS).

Test Case	NND	SP	BC	VS	Justified Ideas
Case 1	1/0	0	1/0	0	NND and BC
Case 2	1/0	0	1/0	0	NND and BC
Case 3	1/0	0	1/0	0	NND and BC
Case 4	1/0	1/0	1/0	0	NND, SP and BC
Case 5	1/0	1/0	1	1/0	NND, SP, and VS
Case 6	1/0	1/0	1	1/0	NND, SP, and VS

Table 5.1: Summary of numerical examples and goals. In this table, "1" represents an idea is used and "0" means the idea is not used. "1/0" indicates that a comparison with/without the idea is tested.

In each example, necessary parameters to obtain each solution are listed in a table right next to the example. In these tables, we use N, N_{p}, N_{I}, and I_{lr} to denote the width for $\hat{u}$, the batch size, the number of iterations, and the range of learning rates (i.e. $[10^{q_1}, 10^{q_0}]$), respectively. In each iteration of the optimization, N_{p} random samples will be renewed. The value of the shift α for each solution found by SP-NND is listed in the table as a constant for a a fixed α or an interval $[10^{p_0}, 10^{p_1}]$ for a varying α.

5.1. Numerical tests in one-dimension. First of all, we will provide four numerical tests for problems in one-dimension. These numerical tests show that the proposed NND works as well as existing methods [26, 35].

Test Case 1. We consider second-order the Painlevé equation [38, 28, 55] that seeks $u(x)$

	u_1	u_2	$\bar{u}_1$	$\bar{u}_2$
N	100	100	100	100
N_{I}	10000	10000	10000	10000
N_{p}	1000	1000	1000	1000
I_{lr}	$[10^{-3}, 10^{-2}]$	$[10^{-3}, 10^{-2}]$	$[10^{-3}, 10^{-2}]$	$[10^{-3}, 10^{-2}]$
deflation source	/	u_1 $(p_1 = 2)$	u_1 $(p_1 = 2)$	u_1 $(p_1 = 2)$
shift α	/	1	1	1

Table 5.2: *Parameters for 1-D Painlevé equations* (5.4)-(5.5). *"/" means the corresponding item is not used (the same as below)*

satisfying

$$\frac{\mathrm{d}^2 u}{\mathrm{d}x^2} = 100u^2 - 1000x, \qquad \text{in } \Omega = (0,1), \tag{5.4}$$

$$u(0) = 0, \quad u(1) = \sqrt{10}. \tag{5.5}$$

It has been shown in [36] that the Painlevé equation (5.4)-(5.5) has exactly two solutions, denoted by u_1 and u_2, which satisfy $u_1'(0) > 0$ and $u_2'(0) < 0$, respectively.

In our experiments, we take the following special network

$$u(x; \boldsymbol{\theta}) = x(x-1)\hat{u}(x; \boldsymbol{\theta}) + \sqrt{10}x \tag{5.6}$$

that automatically satisfies the boundary conditions and use parameters in Table 5.2. The initial guess of $\boldsymbol{\theta}$ is randomly initialized as mentioned previously. The first solution u_1 can be easily found by the RM using (2.2), and the second solution u_2 is found by NND with u_1 as the deflation source and $p_1 = 2$. Other parameters associated with these solutions are listed in Table 5.2. Figure 5.1 visualizes the identified solutions u_1 and u_2 with the same function configurations as in [26].

To verify the effectiveness of special networks that satisfy the boundary conditions (5.5), we use NND without the special network for boundary conditions as a comparison. Hence, the loss function of NND is given by (5.2) with a solution network $u(\boldsymbol{x}; \boldsymbol{\theta})$ as a generic FNN of the same structure as $\hat{u}$ in (5.6). To show that the results of (5.2) are quite independent of the weight λ, $\lambda = 1$ and $\lambda = 100$ are used and the corresponding solutions are denoted as $\bar{u}_1$ and $\bar{u}_2$, respectively. As listed in Table 5.2, other parameters to identify these two solutions are the same as those for identifying u_2 for a fair comparison. It is clear that these two solutions do not satisfy the boundary condition at the endpoint $x = 0$ (see Figure 5.1). This verifies the importance of using special networks that satisfy the boundary conditions automatically.

Test Case 2. We consider a fourth-order nonlinear BVP that seeks u such that

$$\frac{\mathrm{d}^4 u}{\mathrm{d}x^4} = \beta x(1 + u^2) \qquad \text{in } \Omega = (0,1), \tag{5.7}$$

$$u(0) = u'(1) = u''(1) = 0, \quad u''(0) - u''(\gamma) = 0, \tag{5.8}$$

where β and γ are two given constants. Graef et al. [32, 31] have proven that the problem (5.7)-(5.8) has at least two positive solutions when $\beta = 10$ and $\gamma = 1/5$.

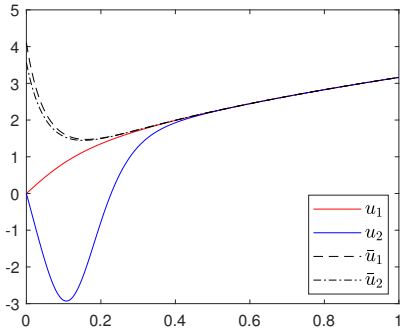

Fig. 5.1: *Identified solutions of the 1-D Painlevé equations* (5.4)-(5.5) *by RM/NND. All correct solutions,* u_1 *and* u_2*, are identified with special networks for boundary conditions. Spurious solutions,* $\bar{u}_1$ *and* $\bar{u}_2$*, are found if the special networks are not used.*

	u_1	u_2	$\bar{u}_1$	$\bar{u}_2$
N	100	100	100	100
N_{I}	5000	5000	5000	5000
N_{p}	1000	1000	1000	1000
I_{lr}	$[10^{-3}, 10^{-2}]$	$[10^{-2}, 10^{-1}]$	$[10^{-2}, 10^{-1}]$	$[10^{-2}, 10^{-1}]$
deflation source	/	u_1 $(p_1 = 1)$	u_1 $(p_1 = 1)$	u_1 $(p_1 = 1)$
shift α	/	1	1	1

Table 5.3: *Parameters for the equation* (5.7)-(5.8).

The three-point boundary condition (5.8) is more complicated than usual. Accordingly, we construct the following special network for it,

$$u(x;\boldsymbol{\theta}) = (x-1)^3\hat{u}(x;\boldsymbol{\theta}) + \hat{u}(0;\boldsymbol{\theta}) + c_\gamma x(x-1)^3, \tag{5.9}$$

where

$$c_\gamma = \frac{1}{-12\gamma^2 + 18\gamma}\Big(\frac{\mathrm{d}^2}{\mathrm{d}x^2}\big((x-1)^3\hat{u}(x;\boldsymbol{\theta})\big)|_{x=\gamma} - \frac{\mathrm{d}^2}{\mathrm{d}x^2}\big((x-1)^3\hat{u}(x;\boldsymbol{\theta})\big)|_{x=0}\Big). \tag{5.10}$$

It can be verified that (5.9) indeed satisfies the boundary condition (5.8) independent of $\boldsymbol{\theta}$.

In our experiment, we find the first solution, denoted by u_1, by applying RM (2.2). With deflation source u_1 $(p_1 = 1)$, we find the second solution, denoted by u_2, by using NND (5.1). The parameters and solutions are demonstrated in Table 5.3 and Figure 5.2.

Similarly, we test NND without special networks for boundary conditions as a comparison under the same setting as Test Case 1. We find two solutions, denoted by $\bar{u}_1$ and $\bar{u}_2$, from $\lambda = 1$ and $\lambda = 100$, respectively (see Figure 5.2). It is clear that both solutions are spurious since their configurations do not take the prescribed boundary value 0 at $x = 0$ (see Figure 5.2), which implies the effectiveness of using special networks for boundary conditions.

Test Case 3. We consider the fourth-order nonlinear equation describing the steady laminar flow of a viscous incompressible fluid in a porous channel [61]. For simplicity, we consider

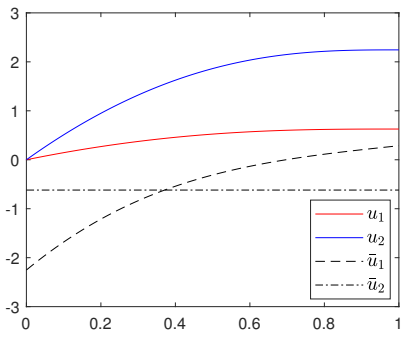

Fig. 5.2: *Identified solutions of the equation* (5.7)-(5.8) *by RM/NND. All correct solutions,* u_1 *and* u_2*, are identified with special networks for boundary conditions. Spurious solutions,* $\bar{u}_1$ *and* $\bar{u}_2$*, are found if the special networks are not used.*

the one-dimensional problem that seeks u such that

$$\frac{\mathrm{d}^4 u}{\mathrm{d}x^4} + \gamma\left(x\frac{\mathrm{d}^3 u}{\mathrm{d}x^3} + 3\frac{\mathrm{d}^2 u}{\mathrm{d}x^2}\right) + R\left(u\frac{\mathrm{d}^3 u}{\mathrm{d}x^3} - \frac{\mathrm{d}u}{\mathrm{d}x}\frac{\mathrm{d}^2 u}{\mathrm{d}x^2}\right) = 0, \quad 0 < x < 1, \tag{5.11}$$

$$u(0) = 0, \quad u''(0) = 0, \quad u(1) = 1, \quad u'(1) = 0, \tag{5.12}$$

where R is the cross-flow Reynolds number and γ is a physical constant related to the wall expansion ratio. Xu et al. [61] have proven that the problem (5.11)-(5.12) admits multiple solutions by analytic approaches. Three solutions were found by homotopy analysis method (HAM) in [46] for the setting $R = -11$ and $\gamma = 1.5$.

In our experiments, we take the same R and γ as in [46]. The special network for the boundary condition (5.12) is chosen as

$$u(x; \boldsymbol{\theta}, \hat{c}) = x(x-1)^2(x^2\hat{u}(x; \boldsymbol{\theta}) + c)e^{2x} + \sin(\pi x/2), \tag{5.13}$$

where c is a network parameter to be trained together with $\boldsymbol{\theta}$. In this case, we initialize $\boldsymbol{b}_3 = \mathbf{0}$ and $c \sim U[0, -5]$. Other network parameters are initialized as mentioned above. Firstly, one solution u_1 is found by RM (2.2). Next, the second solution u_2 is obtained by NND (5.1) with deflation source u_1 ($p_1 = 2$). Moreover, the third solution u_3 is obtained by NND (5.1) with deflation sources u_1 and u_2 ($p_1 = p_2 = 2$). Corresponding parameters are shown in Table 5.4. The three found solutions and their first derivatives are plotted in Figure 5.3, which are the same solutions found in [46].

Also, a comparison test is performed to seek u_2 by NND (5.2) with the same setting as above, except for using a generic solution network without the special structure for boundary conditions. We find two solutions, denoted by $\hat{u}_1$ and $\hat{u}_2$, using $\lambda = 1$ and $\lambda = 100$. Neither of them takes the prescribed boundary value 0 at $x = 0$ or 1 at $x = 1$ and, hence, they are spurious solutions (see Figure 5.3).

Test Case 4. We consider the following second-order problem that seeks u such that

$$\frac{\mathrm{d}^2 u}{\mathrm{d}x^2} = f(u), \quad 0 < x < 1, \tag{5.14}$$

$$u'(0) = 0, \quad u(1) = 0, \tag{5.15}$$

	u_1	u_2	u_3	$\hat{u}_1$	$\hat{u}_2$
N	50	50	50	50	50
N_{I}	20000	10000	20000	10000	10000
N_{p}	1000	1000	1000	1000	1000
I_{lr}	$[10^{-3}, 10^{-2}]$	$[10^{-3}, 10^{-2}]$	$[10^{-3}, 10^{-2}]$	$[10^{-3}, 10^{-2}]$	$[10^{-3}, 10^{-2}]$
deflation source	/	u_1 $(p_1 = 2)$	u_1, u_2 $(p_1 = p_2 = 2)$	u_1 $(p_1 = 2)$	u_1 $(p_1 = 2)$
shift α	/	1	1	1	1

Table 5.4: *Parameters for the channel flows equation* (5.11)-(5.12).

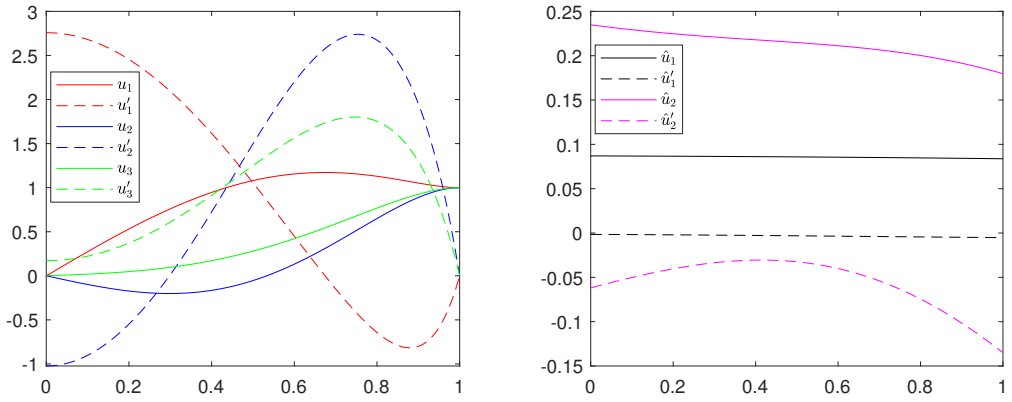

Fig. 5.3: *Identified solutions and their derivatives of the channel flows equation* (5.11)-(5.12) *by RM/NND. All correct solutions,* u_1*,* u_2 *and* u_3*, are identified with special networks for boundary conditions. Spurious solutions,* $\bar{u}_1$ *and* $\bar{u}_2$*, are found if the special networks are not used.*

where $f(u)$ is a polynomial function of u. The existence of multiple solutions for the problem (5.14) has been studied by the bootstrapping method [35].

First, we set the right-hand side of the problem (5.14) as $f(u) = \lambda(1 + u^4)$. It is shown in [35] that there are two solutions for $0 < \lambda < \lambda^* = 1.30107$. In our experiments, we take $\lambda = 1.2$. The special network for the boundary condition (5.15) is given by

$$u(x; \boldsymbol{\theta}) = x^2 \hat{u}(x; \boldsymbol{\theta}) - \hat{u}(1; \boldsymbol{\theta}). \tag{5.16}$$

The first solution u_1 is found by RM (2.2) and the second solution u_2 is found by NND (5.1) with deflation source u_1 $(p_1 = 2)$. Similarly to preceding cases, we perform a comparison test without the special network structure for boundary conditions and two spurious solutions $\hat{u}_1$ (for $\lambda = 1$) and $\hat{u}_2$ (for $\lambda = 100$) are found by NND (5.2). The parameters for all these solutions are shown in Table 5.5 and all solutions are plotted in Figure 5.4.

Second, we repeat the test by choosing $f(u) = -\frac{\pi^2}{4} u^2(u^2 - 10)$. [35] has proved that there exist eight solutions in total. Note that $u_0 = 0$ is a trivial solution. In this case, we start from NND (5.1) with the special network (5.16) and the deflation source u_0 $(p_0 = 2)$ to find the first solution u_1, which is quite close to u_0. We would like to emphasize that it is sufficient to use NND without the structure probing initialization to identify u_0 and u_1. However, we were not able to identify any other solutions without the structure probing initialization even

	u_1	u_2	$\hat{u}_1$	$\hat{u}_2$
N	100	100	100	100
$N_{\rm I}$	10000	10000	10000	10000
$N_{\rm p}$	1000	1000	1000	1000
$I_{\rm lr}$	$[10^{-3}, 10^{-2}]$	$[10^{-3}, 10^{-2}]$	$[10^{-3}, 10^{-2}]$	$[10^{-3}, 10^{-2}]$
deflation source	/	u_1 $(p_1 = 2)$	u_1 $(p_1 = 2)$	u_1 $(p_1 = 2)$
shift α	/	1	1	1

Table 5.5: *Parameters for the nonlinear problem* (5.14)-(5.15) *with* $f(u) = \lambda(1 + u^4)$.

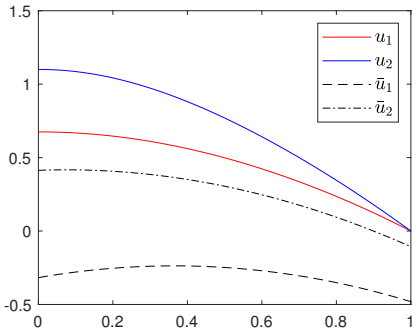

Fig. 5.4: *Identified solutions of the nonlinear problem* (5.14)-(5.15) *with* $f(u) = \lambda(1 + u^4)$ *by RM/NND. All correct solutions,* u_1 *and* u_2*, are identified with special networks for boundary conditions. Spurious solutions,* $\bar{u}_1$ *and* $\bar{u}_2$*, are found if the special networks are not used.*

if we tried our best to tune parameters and use different random initialization. To perform a wider search for other solutions, we employ the following structure probing initialization

$$u_J(x; \boldsymbol{\theta}, \hat{c}_j) = x^2 \hat{u}(x; \boldsymbol{\theta}) - \hat{u}(1; \boldsymbol{\theta}) + \sum_{j=1}^{J} \hat{c}_j \cos((2j-1)\pi x/2), \tag{5.17}$$

with initial setting $c_j = 0$ for $j = 1, \cdots, J-1$ and $c_J \sim U(-5, 5)$. Two solutions, denoted by u_2 and u_3, are found by NND (5.1) with deflation source u_0 $(p_0 = 2)$ and the structure probing network (5.17) with $J = 1$. Another two solutions, denoted by u_4 and u_6, are found by NND (5.1) with deflation source u_0 $(p_0 = 2)$ and the network (5.17) with $J = 2$. Two more solutions, denoted by u_5 and u_7, are found by NND (5.1) with deflation sources u_4 $(p_4 = 2)$ and u_6 $(p_6 = 2)$, respectively, and the network (5.17) with $J = 2$. Corresponding parameters, including the initial value of c_J actually randomized for each solution, are listed in Table 5.6. All the 7 nontrivial solutions are plotted in Figure 5.5.

5.2. Numerical tests in high-dimension. In this subsection, we will provide numerical tests in high-dimensional domains $(d \geq 2)$.

Test Case 5. We consider 2-D Yamabe's equation that seeks u such that

$$\begin{aligned} -8\Delta u - 0.1u + \frac{u^5}{|\boldsymbol{x}|^3} &= 0, \quad \text{in } \Omega = \{\boldsymbol{x} \in \mathbb{R}^2 : r < |\boldsymbol{x}| < R\}, \\ u &= 1, \quad \text{on } \partial\Omega, \end{aligned} \tag{5.18}$$

	u_1	u_2	u_3	u_4
N	100	100	100	100
N_{I}	5000	5000	10000	20000
N_{p}	1000	1000	1000	1000
I_{lr}	$[10^{-3}, 10^{-2}]$	$[10^{-3}, 10^{-2}]$	$[10^{-4}, 10^{-3}]$	$[10^{-4}, 10^{-3}]$
J	0	1	1	2
initial c_J	/	-3.48	4.61	-3.67
deflation source	u_0 $(p_0 = 2)$	u_0 $(p_0 = 2)$	u_0 $(p_0 = 2)$	u_0 $(p_0 = 2)$

	u_5	u_6	u_7	
N	100	100	100	
N_{I}	20000	20000	20000	
N_{p}	1000	1000	1000	
I_{lr}	$[10^{-4}, 10^{-3}]$	$[10^{-4}, 10^{-3}]$	$[10^{-4}, 10^{-3}]$	
J	2	2	2	
initial c_J	-4.12	3.64	3.44	
deflation source	u_4 $(p_4 = 2)$	u_0 $(p_0 = 2)$	u_6 $(p_6 = 2)$	

Table 5.6: *Parameters for the nonlinear problem* (5.14)-(5.15) *with* $f(u) = -\frac{\pi^2}{4}u^2(u^2 - 10)$.

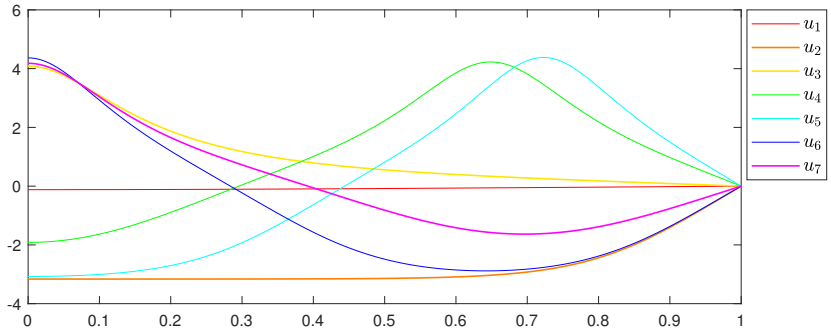

Fig. 5.5: *Identified solutions of the nonlinear DE* (5.14)-(5.15) *with* $f(u) = -\frac{\pi^2}{4}u^2(u^2 - 10)$ *by NND.*

where r and R are set as 1 and 100. Nine solutions were found by using non-network deflation techniques and various initial guesses in [26].

In our experiments, the solutions are approximated by the following special network

$$u_J(\boldsymbol{x}; \boldsymbol{\theta}) = \hat{u}(\boldsymbol{x}; \boldsymbol{\theta}) \sin\left(\pi \frac{|\boldsymbol{x}| - r}{R - r}\right) + 1 \tag{5.19}$$

if the random initialization without the structure probing technique is used, or the following network

$$u_J(\boldsymbol{x}; \boldsymbol{\theta}, c_j) = \hat{u}(\boldsymbol{x}; \boldsymbol{\theta}) \sin\left(\pi \frac{|\boldsymbol{x}| - r}{R - r}\right) + \sum_{j=1}^{J} c_j \sin(j\pi \frac{|\boldsymbol{x}| - r}{R - r}) + 1 \tag{5.20}$$

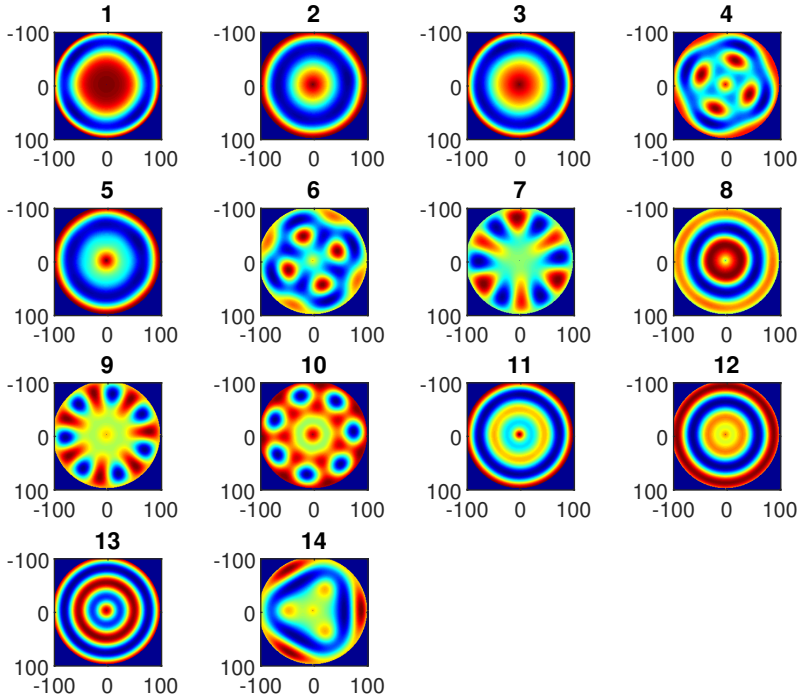

Fig. 5.6: *Identified solutions of the 2-D Yamabe's equation* (5.18) *by SP-NND with varying shifts.*

with the structure probing initialization, where the initial values are $c_j = 0$ for $j = 1, \cdots, J-1$ and $c_J \sim U(-1, 1)$. Note that both (5.19) and (5.20) satisfy the given boundary condition automatically.

In our proposed framework of SP-NND with varying shift, we always follow the four steps: 1) use the RM (2.2) to find the first few solutions; 2) use NND without SP and varying shifts to find other solutions; 3) use SP-NND without varying shifts to find more distinct solutions; 4) finally, use SP-NND with varying shifts to find extra distinct solutions. Following these procedures, we find 14 solutions in total for the 2D Yamabe's equation as plotted in Figure 5.6 with parameters specified in Table 5.7.

More precisely, u_1 and u_{11} are found by RM (2.2) and the others are found by NND (5.1) with previously found solutions as deflation sources ($p_k = 2$ for all k). In NND, we employ the technique of varying shifts in deflation operators, which helps to find more distinct solutions. All solutions are found by using networks (5.19) or (5.20) (specified in Table 5.7) with their corresponding initialization as mentioned previously, except that we take the network (5.19) with $2 - u_9$ as the initial guess to find u_{10}. We would like to remark that both the structure probing initialization and the varying shifts are key techniques to find more distinct solutions for high-dimensional problems. Without any of them, we cannot find 14 distinct solutions even if we have tried our best to tune parameters with commonly used random initialization in the literature.

	u_1	u_2	u_3	u_4	u_5
N	100	100	100	100	100
N_{I}	2000	2000	2000	5000	2000
N_{p}	10000	10000	10000	10000	10000
I_{lr}	$[10^{-3}, 10^{-1}]$	$[10^{-2}, 10^{-1}]$	$[10^{-2}, 10^{-1}]$	$[10^{-2}, 10^{-1}]$	$[10^{-2}, 10^{-1}]$
network	(5.19)	(5.19)	(5.19)	(5.19)	(5.19)
α	/	1	1	$[0.01, 100]$	$[0.01, 100]$
deflation source	/	u_1	u_2	u_3	u_1
	u_6	u_7	u_8	u_9	u_{10}
N	100	100	100	100	100
N_{I}	5000	10000	20000	20000	10000
N_{p}	10000	10000	10000	20000	10000
I_{lr}	$[10^{-2}, 10^{-1}]$	$[10^{-2}, 10^{-1}]$	$[10^{-2}, 10^{-1}]$	$[10^{-2}, 10^{-1}]$	$[10^{-2}, 10^{-1}]$
network	(5.19)	(5.19)	(5.19)	(5.19)	(5.19)
α	$[0.01, 10]$	$[0.01, 10]$	$[0.01, 10]$	$[0.01, 10]$	$[0.01, 10]$
deflation source	u_1,u_4	u_1,u_2	u_1,u_2	u_1,u_2	u_9
	u_{11}	u_{12}	u_{13}	u_{14}	
N	100	100	100	100	
N_{I}	2000	10000	10000	10000	
N_{p}	10000	10000	10000	10000	
I_{lr}	$[10^{-3}, 10^{-1}]$	$[10^{-2}, 10^{-1}]$	10^{-2}	$[10^{-2}, 10^{-1}]$	
network	(5.20) ($J = 4$)	(5.20) ($J = 4$)	(5.20) ($J = 4$)	(5.19)	
α	/	0.01	$[0.01, 10]$	1	
deflation source	/	u_{11}	u_8,u_{11}	u_{11}	

Table 5.7: *Parameters for the 2-D Yamabe's equation* (5.18) *($p_k = 2$ for all deflation sources for the solutions obtained by NND).*

Test Case 6. The high-dimensional Yamabe's equation seeks u such that

$$
\begin{aligned}
-\frac{4(d-1)}{(d-2)}\Delta u - 0.125u + \frac{u^{\frac{d+2}{(d-2)}}}{|\boldsymbol{x}|^3} &= 0, \quad \text{in } \Omega = \{1 < |\boldsymbol{x}| < 100\}, \\
u &= 1, \quad \text{on } \partial\Omega,
\end{aligned} \tag{5.21}
$$

where $d \geq 3$ is the dimension of the problem.

We continue applying the network (5.19) without structure probing initialization and the network (5.20) with the structure probing initialization as solution networks to solve the Yamabe's equation when $d = 3$ and $d = 6$. The initialization parameters are the same as in the $2D$ case.

Again, in our proposed framework of SP-NND with varying shift, we follow the four steps: 1) use the RM (2.2) to find the first few solutions; 2) use NND without SP and varying shifts to find other solutions; 3) use SP-NND without varying shifts to find more distinct solutions; 4) finally, use SP-NND with varying shifts to find extra distinct solutions. Following these procedures, we obtain 11 solutions when $d = 3$ and 9 solutions when $d = 6$. The corresponding parameters are shown in Tables 5.8 and 5.9 for $d = 3$ and $d = 6$, respectively. The solutions are visualized in Figures 5.7 and 5.8 for $d = 3$ and $d = 6$,

	u_1	u_2	u_3	u_4	u_5
N	100	100	100	100	100
N_{I}	20000	20000	20000	20000	20000
N_{p}	10000	10000	10000	10000	10000
I_{lr}	$[10^{-2}, 10^{-1}]$	$[10^{-2}, 10^{-1}]$	$[10^{-2}, 10^{-1}]$	$[10^{-2}, 10^{-1}]$	$[10^{-2}, 10^{-1}]$
network	(5.19)	(5.19)	(5.19)	(5.19)	(5.19)
α	/	$[0.01, 10]$	1	0.1	0.01
deflation source	/	u_1	u_1,u_2	u_1,u_2	u_1,u_2

	u_6	u_7	u_8	u_9	u_{10}
N	100	100	100	100	100
N_{I}	20000	20000	20000	20000	20000
N_{p}	10000	10000	10000	10000	10000
I_{lr}	$[10^{-2}, 10^{-1}]$	$[10^{-2}, 10^{-1}]$	$[10^{-2}, 10^{-1}]$	$[10^{-2}, 10^{-1}]$	$[10^{-2}, 10^{-1}]$
network	(5.20) ($J = 4$)	(5.20) ($J = 6$)	(5.20) ($J = 4$)	(5.19)	(5.19)
α	0.01	0.1	$[0.01, 10]$	$[0.01, 10]$	$[0.01, 10]$
deflation source	u_1,u_2	u_1,u_2	u_1,u_4	u_1,u_2,u_3	u_1,u_2,u_5

	u_{11}
N	10
N_{I}	20000
N_{p}	10000
I_{lr}	$[10^{-2}, 10^{-1}]$
network	(5.20) ($J = 4$)
α	$[0.01, 10]$
deflation source	u_1,u_2,u_6

Table 5.8: *Parameters for the 3-D Yamabe's equation* (5.21) *($p_k = 2$ for all deflation sources for the solutions obtained by SP-NND).*

respectively. We would like to remark that both the structure probing initialization and the varying shifts are key techniques to find more distinct solutions for high-dimensional problems. Without any of them, we cannot find several distinct solutions even if we have tried our best to tune parameters with commonly used random initialization in the literature.

In these tests, the deflation powers p_k are set as 2 for all k whenever deflation is used. In the case of $d = 3$, most networks are initialized using (5.19) or (5.20), except for u_8, u_9 and u_{10}, which are found by using initial guesses $2 - u_4$, $2 - u_3$ and $2 - u_5$, respectively. In the case of $d = 6$, we also try the initialization with a constant minus a known solution. However, this initialization method does not lead to new solutions.

6. Conclusion. In this paper, we proposed the structure probing neural network deflation to find distinct solutions to nonlinear differential equations. The original optimization energy landscape of network-based methods is regularized by neural network deflation so that known solutions are no longer local minimizers while preserving unknown solutions as local minimizers. To obtain a new solution with the desired features, a structure probing algorithm is applied to obtain an initial guess that is close to the desired solution. Finally, special network structures that satisfy various boundary conditions automatically are introduced to simplify the objective function of network-based methods. These techniques

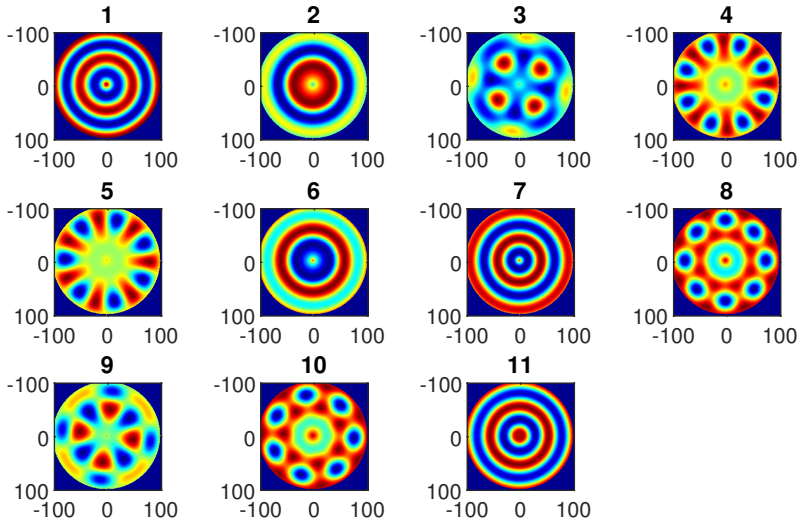

Fig. 5.7: *Identified solutions of the 3-D Yamabe's equation* (5.21) *by SP-NND. We visualize these solutions by projecting them in the first two coordinates.*

	u_1	u_2	u_3	u_4	u_5
N	100	100	100	100	100
N_{I}	20000	20000	20000	20000	20000
N_{p}	10000	10000	10000	10000	10000
I_{lr}	$[10^{-2}, 10^{-1}]$	$[10^{-2}, 10^{-1}]$	$[10^{-2}, 10^{-1}]$	$[10^{-3}, 10^{-1}]$	$[10^{-3}, 10^{-1}]$
network	(5.19)	(5.19)	(5.19)	(5.19)	(5.19)
α	/	$[0.01, 10]$	0.1	10	$[0.01, 10]$
deflation source	/	u_1	u_1	u_1	u_1,u_2

	u_6	u_7	u_8	u_9	
N	100	100	100	100	
N_{I}	20000	20000	20000	20000	
N_{p}	10000	10000	10000	10000	
I_{lr}	$[10^{-3}, 10^{-2}]$	$[10^{-3}, 10^{-2}]$	$[10^{-2}, 10^{-1}]$	$[10^{-2}, 10^{-1}]$	
network	(5.19)	(5.20) $(J = 6)$	(5.20) $(J = 6)$	(5.20) $(J = 6)$	
α	$[0.1, 1]$	/	$[0.01, 10]$	1	
deflation source	u_1,u_2	/	u_7	u_7	

Table 5.9: *Parameters for the 6-D Yamabe's equation* (5.21) *($p_k = 2$ for all deflation sources for the solutions obtained by SP-NND).*

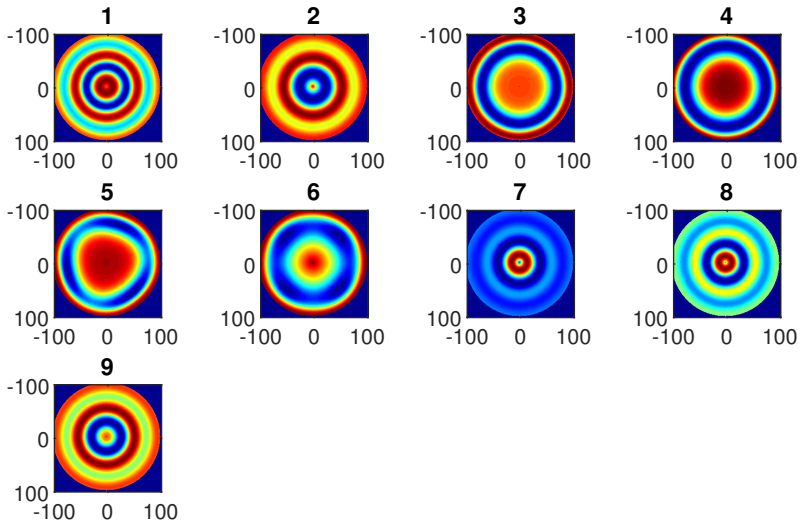

Fig. 5.8: *Identified solutions of the 6-D Yamabe's equation* (5.21) *by SP-NND. We visualize these solutions by projecting them in the first two coordinates.*

form a new framework for identifying distinct solutions of nonlinear differential equations. Compared to existing methods, SP-NND is capable of solving high-dimensional problems on complex domains with a lower computational cost and can identify more distinct solutions.

Acknowledgments. Y. G. was partially supported by the Ministry of Education in Singapore under the grant MOE2018-T2-2-147. H. Y. was partially supported by the US National Science Foundation under award DMS-1945029.

REFERENCES

[1] G. Acosta, J.P. Borthagaray, O. Bruno, and M. Maas. Regularity theory and high order numerical methods for the (1d)-Fractional Laplacian. *arXiv e-prints*, arXiv:1608.08443, 2016.
[2] J.H. Adler, D.B. Emerson, P.E. Farrell, and S.P. MacLachlan. A deflation technique for detecting multiple liquid crystal equilibrium states. *arXiv e-prints*, arXiv:1601.07383, 2016.
[3] E.L. Allgower and K. Georg. Continuation and path following. *Acta Numer.*, 2:1–64, 1993.
[4] E.L. Allgower and K. Georg. *Introduction to Numerical Continuation Methods.* Classics Appl. Math., SIAM, Philadelphia, 2003.
[5] A.J. Sommese A.P. Morgan. Coefficient-parameter polynomial continuation. *Appl. Math. Comput.*, 29(2):123–160, 1989.
[6] A. R. Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Trans. Inf. Theory*, 39(3):930–945, 1993.
[7] J. Berg and K. Nyström. A Unified Deep Artificial Neural Network Approach to Partial Differential Equations in Complex Geometries. *Neurocomputing*, 317:28 – 41, 2018.
[8] F.H. Branin. Widely convergent method for finding multiple solutions of simultaneous nonlinear equations.

IBM J. Res. Develop, 16:504–522, 1972.

[9] K.M. Brown and W.B. Gearhart. Deflation techniques for the calculation of further solutions of a nonlinear system. *Numer. Math.*, 16:334–342, 1971.

[10] W. Cai and Z.J. Xu. Multi-scale deep neural networks for solving high dimensional PDEs. *arXiv e-prints*, arXiv:1910.11710, 2019.

[11] Yuan Cao, Zhiying Fang, Yue Wu, Ding-Xuan Zhou, and Quanquan Gu. Towards understanding the spectral bias of deep learning, 2020.

[12] K.-S. Chao, D.-K. Liu, and C.-T. Pan. A systematic search method for obtaining multiple solutions of simultaneous nonlinear equations. *IEEE Trans. Circuits Systems*, 22:748–753, 1975.

[13] Zixiang Chen, Yuan Cao, Difan Zou, and Quanquan Gu. How much over-parameterization is sufficient to learn deep relu networks? *CoRR*, arXiv:1911.12360, 2019.

[14] M.-J. Chien. Searching for multiple solutions of nonlinear systems. *IEEE Trans. Circuits Systems*, 26:817–827, 1979.

[15] C.J. Chyan and J. Henderson. Multiple solutions for 2mth order sturmliouville boundary value problems. *Comput. Math. Appl.*, 40:231–237, 2000.

[16] L.H. Clark, P.M. Schlosser, and J.F. Selgrade. Multiple stable periodic solutions in a model for hormonal control of the menstrual cycle. *Bull. Math. Biol.*, 65:157173, 2003.

[17] M. Croci and P.E. Farrell. Distinct solutions of finite-dimensional complementarity problems. *arXiv e-prints*, arXiv:1510.02433, 2015.

[18] X. Dai and Y. Zhu. Towards theoretical understanding of large batch training in stochastic gradient descent. *CoRR*, abs/1812.00542, 2018.

[19] D.F. Davidenko. On a new method of numerical solution of systems of nonlinear equations. *Dokl. Akad. Nauk SSSR*, 88:601–602, 1953.

[20] J.M. Davis, L.H. Erbe, and J. Henderson. Multiplicity of positive solutions for higher order sturm-liouville problems. *Rocky Mountain J. Math.*, 31:169–184, 2001.

[21] S. S. Du, X. Zhai, B. Poczos, and A. Singh. Gradient descent provably optimizes over-parameterized neural networks. *arXiv e-prints*, arXiv:1810.02054, 2018.

[22] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res*, 12:2121–2159, 2011.

[23] B. Dyda, A. Kuznetsov, and M. Kwaśnicki. Eigenvalues of the fractional laplace operator in the unit ball. *J. Lond. Math. Soc.*, 95(2):500–518, 2017.

[24] W. E and Q. Wang. Exponential convergence of the deep neural network approximation for analytic functions. *CoRR*, abs/1807.00297, 2018.

[25] W. E and B. Yu. The deep ritz method: a deep learning-based numerical algorithm for solving variational problems. *Commun. Math. Stat.*, 6:1–12, 2018.

[26] P.E. Farrell, Á. Birkisson, and S.W. Funke. Deflation techniques for finding distinct solutions of nonlinear partial differential equations. *SIAM J. Sci. Comput.*, 37(4):A2026–A2045, 2015.

[27] M.C. Ferris and J.S. Pang. Engineering and economic applications of complementarity problems. *SIAM Rev.*, 39:669–713, 1997.

[28] B. Fornberg and J.A.C. Weideman. A numerical methodology for the Painlevé equations. *J. Comput. Phys.*, 230(15):5957 – 5973, 2011.

[29] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 2010. PMLR.

[30] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, Cambridge, 2016.

[31] J.R. Graef, C. Qian, and B. Yang. Multiple positive solutions of a boundary value problem for ordinary differential equations. *Electron. J. Qual. Theo. [electronic only]*, 2003:Paper No. 11, 13 p., electronic only–Paper No. 11, 13 p., electronic only, 2003.

[32] J.R. Graef, C. Qian, and B. Yang. A three point boundary value problem for nonlinear fourth order differential equations. *J. Math. Anal. Appl.*, 287(1):217 – 233, 2003.

[33] Y. Gu, H. Yang, and C. Zhou. SelectNet: Self-paced Learning for High-dimensional Partial Differential Equations. *arXiv e-prints*, arXiv:2001.04860, 2020.

[34] J. Han, A. Jentzen, and W. E. Solving high-dimensional partial differential equations using deep learning. *Proc. Natl. Acad. Sci.*, 115(34):8505–8510, 2018.

[35] W. Hao, J.D. Hauenstein, B. Hu, and A.J. Sommese. A bootstrapping approach for computing multiple solutions of differential equations. *J. Comput. Appl. Math.*, 258:181–190, 2014.

[36] S.P. Hastings and W.C. Troy. On some conjectures of turcotte, spence, bau, and holmes. *SIAM J. Math. Anal.*, 20(3):634–642, 1989.

[37] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[38] P. Holmes and D. Spence. On a Painlevé-type boundary-value problem. *Q. J. Mech. Appl. Math.*, 37(4):525–538, 1984.

[39] J. Huang, H. Wang, and H. Yang. Int-Deep: A Deep Learning Initialized Iterative Method for Nonlinear Problems. *arXiv e-prints*, arXiv:1910.01594, 2019.

[40] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *CoRR*, abs/1806.07572, 2018.

[41] S. Justin and S. Konstantinos. Dgm: A deep learning algorithm for solving partial differential equations. *J. Comput. Phys.*, 375:1339–1364, 2018.

[42] A.A. Kilbas, H.M. Srivastava, and J.J. Trujillo. *Theory and Applications of Fractional Differential Equations*. Elsevier Science, Amsterdam, The Netherlands, 2006.

[43] D. P. Kingma and J. Ba. Adam: a method for stochastic optimization. *arXiv e-prints*, arXiv:1412.6980, 2014.

[44] D. Lei, Z. Sun, Y. Xiao, and W. Y. Wang. Implicit regularization of stochastic gradient descent in natural language processing: observations and implications. *arXiv e-prints*, arXiv:1811.00659, 2018.

[45] S. Liang and R. Srikant. Why deep neural networks? *CoRR*, abs/1610.04161, 2016.

[46] S. Liao. *Homotopy Analysis Method in Nonlinear Differential Equations*. Springer, Berlin, Heidelberg, 2012.

[47] Y. Liao and P. Ming. Deep Nitsche method: deep Ritz method with essential boundary conditions. *arXiv e-prints*, arXiv:1912.01309, 2019.

[48] J. Lu, Z. Shen, H. Yang, and S. Zhang. Deep Network Approximation for Smooth Functions. *arXiv e-prints*, arXiv:2001.03040, 2020.

[49] T. Luo, Z. Ma, Z.J. Xu, and Y. Zhang. Theory of the frequency principle for general deep neural networks. *CoRR*, abs/1906.09235, 2019.

[50] Tao Luo and Haizhao Yang. Two-Layer Neural Networks for Partial Differential Equations: Optimization and Generalization Theory. *arXiv e-prints*, arXiv:2006.15733, 2020.

[51] H. Montanelli and Q. Du. New error bounds for deep networks using sparse grids. *arXiv e-prints*, arXiv:1712.08688, 2017.

[52] H. Montanelli and H. Yang. Error bounds for deep relu networks using the kolmogorovarnold superposition theorem. *Neural Networks*, 129:1–6, 2020.

[53] H. Montanelli, H. Yang, and Q. Du. Deep ReLU networks overcome the curse of dimensionality for bandlimited functions. *arXiv e-prints*, arXiv:1903.00735, 2019.

[54] B. Neyshabur, R. Tomioka, R. Salakhutdinov, and N. Srebro. Geometry of optimization and implicit regularization in deep learning. *arXiv e-prints*, arXiv:1705.03071, 2017.

[55] V.A. Noonburg. A separating surface for the Painlevé differential equation $x'' = x^2 - t$. *J. Math. Anal. Appl.*, 193(3):817 – 831, 1995.

[56] J.A.A. Opschoor, C. Schwab, and J. Zech. Exponential relu dnn expression of holomorphic maps in high dimension. Technical report, Zurich, 2019.

[57] T. Poggio, H.N. Mhaskar, L. Rosasco, B. Miranda, and Q. Liao. Why and when can deep—but not shallow—networks avoid the curse of dimensionality: A review. *International Journal of Automation and Computing*, 14:503–519, 2017.

[58] S. J. Reddi, S. Kale, and S. Kumar. On the convergence of Adam and beyond. *arXiv e-prints*, arXiv:1904.09237, 2019.

[59] J.H. Wilkinson. Rounding errors in algebraic processes. *Natl. Phys. Lab. Notes Appl. Sci.*, 32:334–342, 1963.

[60] L.R. Williams and R.W. Leggett. Unique and multiple solutions of a family of differential equations modeling chemical reactions. *SIAM J. Math. Anal.*, 13:122–133, 1982.

[61] H. Xu, Z. Lin, S. Liao, J. Wu, and J. Majdalani. Homotopy based solutions of the navier-stokes equations for a porous channel with orthogonally moving walls. *Phys. Fluids*, 22:053601, 2010.

[62] Z. J. Xu, Y. Zhang, and Y. Xiao. Training behavior of deep neural network in frequency domain. In *Neural Information Processing*, pages 264–274. Springer International Publishing, 2019.

[63] D. Yarotsky. Error bounds for approximations with deep relu networks. *Neural Networks*, 94:103–114, 2017.

[64] D. Yarotsky and A. Zhevnerchuk. The phase diagram of approximation rates for deep neural networks. *arXiv e-prints*, arXiv:1906.09477, 2019.

[65] Z. Song Z. A.-Zhu, Y. Li. A convergence theory for deep learning via over-parameterization. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 242–252, Long Beach, California, USA, 2019. PMLR.

[66] Y. Zang, G. Bao, X. Ye, and H. Zhou. Weak adversarial networks for high-dimensional partial differential equations. *J. Comput. Phys.*, 411:109409, 2020.

[67] Y. Zhang, Z.J. Xu, T. Luo, and Z. Ma. Explicitizing an implicit bias of the frequency principle in two-layer neural networks. *CoRR*, abs/1905.10264, 2019.

[68] Y. Zhang, Z.J. Xu, T. Luo, and Z. Ma. A type of generalization error induced by initialization in deep neural networks. *CoRR*, abs/1905.07777, 2019.