Computers and Mathematics with Applications 200 (2025) 349–363

Contents lists available at ScienceDirect

Computers and Mathematics with Applications

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

Efficient shallow Ritz method for 1D diffusion problems ☆

Zhiqiang Cai [a], Anastassia Doktorova [a], Robert D. Falgout [b], César Herrera [a,*]

[a] Department of Mathematics, Purdue University, West Lafayette, IN, United States of America
[b] Lawrence Livermore National Laboratory, Livermore, CA, United States of America

ARTICLE INFO

Keywords:
Fast iterative solvers
Neural network
Ritz formulation
ReLU activation
Diffusion problems
Elliptic problems
Newton's method

ABSTRACT

This paper studies the shallow Ritz method for solving the one-dimensional diffusion problem. It is shown that the shallow Ritz method improves the order of approximation dramatically for non-smooth problems. To realize this optimal or nearly optimal order of the shallow Ritz approximation, we develop a damped block Newton (dBN) method that alternates between updates of the linear and non-linear parameters. Per each iteration, the linear and the non-linear parameters are updated by exact inversion and one step of a modified, damped Newton method applied to a reduced non-linear system, respectively. The computational cost of each dBN iteration is $\mathcal{O}(n)$.

Starting with the non-linear parameters as a uniform partition of the interval, numerical experiments show that the dBN is capable of efficiently moving mesh points to nearly optimal locations. To improve the efficiency of the dBN further, we propose an adaptive damped block Newton (AdBN) method by combining the dBN with the adaptive neuron enhancement (ANE) method [28].

1. Introduction

In the past decade, the use of neural networks (NNs) has surged in popularity, finding applications in artificial intelligence, natural language processing, image recognition, and various other domains within machine learning. Consequently, the use of NNs has extended to diverse fields, including numerically solving partial differential equations (PDEs) [4,10,19,20,33,35]. The idea of using NNs to solve PDEs may be traced back to the 90s [18,25,24] based on a simplified Bramble-Schatz least-squares (BSLS) formulation [5] in 1970.

This simplified version of the BSLS applies the L^2 norm least-squares (LS) principle to the strong form of the underlying PDE and boundary conditions. For non-smooth problems, the BSLS formulation is *not* equivalent to the original problem because the critical interface condition is ignored (see, e.g., (3.2)). For various NN methods based on equivalent LS formulations, see [10] for the convection-diffusion-reaction problem, [7] for the advection-reaction problem, and [8] for the scalar non-linear hyperbolic conservation laws.

For a class of self-adjoint problems with a natural minimization principle, one may employ the Ritz formulation instead of manufactured LS formulations. Recently, the deep Ritz method was introduced in [20] for the Poisson equation and uses deep neural

☆ This work was supported in part by the National Science Foundation under grant DMS-2110571 and by the Department of Energy under NO. B665416. This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-JRNL-862433 and No. B665416).

* Corresponding author.
E-mail addresses: caiz@purdue.edu (Z. Cai), adoktoro@purdue.edu (A. Doktorova), rfalgout@llnl.gov (R.D. Falgout), herre125@purdue.edu (C. Herrera).

https://doi.org/10.1016/j.camwa.2025.10.020

Available online 27 October 2025

networks (DNNs) as approximating functions. Since the Ritz formulation automatically enforces the interface condition weakly, the deep Ritz method is applicable to non-smooth problems such as elliptic interface problems.

In one dimension, the shallow ReLU NN is sufficient to accurately approximate non-smooth functions because it produces the same class of approximating functions as the free knots splines (FKS). The FKS was introduced in the 1960s and studied by many researchers [3,17,23,34]. Spline approximations to non-smooth functions can be improved dramatically with free knots [6] (see Section 2). Nevertheless, determining optimal knot locations (the non-linear parameters of a shallow NN) becomes a complicated, computationally intensive non-convex optimization problem. Although FKS was a subject of many research articles on various optimization methods such as the DFP method [16,21], the Gauss-Newton method [22], a method moving each knot locally [17,29], etc. (see, e.g., [23,32] and references therein), to the best of our knowledge, there is no optimization method, including those commonly used in scientific machine learning, that can move mesh points (the non-linear parameters) to nearly optimal locations efficiently and effectively.

The main purpose of this paper is to address this challenging issue on the non-convex optimization problem arising from the shallow Ritz approximation of one-dimensional diffusion problems. The shallow ReLU NN has two types of parameters: linear and non-linear. They are corresponding to the weights and biases of the output and hidden layers, respectively. Moreover, the non-linear ones are mesh points of the continuous piecewise linear NN approximation, and the linear ones are the coefficients of the linear combination of one and neuron functions. Due to their geometrical meanings, it is then natural and efficient to employ the commonly used outer-inner iterative method by alternating between updates of the linear and non-linear parameters.

The optimality condition for the linear parameters leads to a system of linear equations whose coefficient matrix and right-hand side vector depend on the non-linear parameters. Due to the non-local support of neuron functions, this stiffness matrix is dense and ill-conditioned. Specifically, we show that its condition number is bounded by $\mathcal{O}\left(n\, h_{\min}^{-1}\right)$ (see Lemma 4.3), where n is the number of neurons and $h_{\min}$ is the smallest distance between two neighboring mesh points. To overcome this difficulty in the linear solver, we find that the inverse of the stiffness matrix is a tridiagonal matrix (see Lemma 5.1).

The optimality condition for the non-linear parameters is a nearly decoupled system of non-linear algebraic equations with a linear coupling through the penalized boundary term. This algebraic structure implies that the Newton method could be the best choice for solving this non-linear system. However, the gradient of the energy functional with respect to the non-linear parameters is not differentiable at the physical interface for the elliptic interface problem, and the Hessian matrix may not be invertible due to either some vanishing linear parameters or vanishing main parts of some diagonals of the Hessian matrix. In both cases, we identify that the corresponding non-linear parameters are either unneeded or can be fixed. Hence, they may be removed from the non-linear system. The reduced non-linear system is then differentiable and its Hessian is invertible.

Combining the exact inversion for the linear parameters and one step damped Newton method for the reduced non-linear system, we introduce the damped block Newton (dBN) method. Computational cost of each dBN iteration is $\mathcal{O}(n)$. Starting with uniformly partitioned non-linear parameters of the interval, numerical experiments show that the dBN is capable of efficiently moving mesh points to nearly optimal locations. To improve the efficiency of the dBN further, we propose an adaptive damped block Newton (AdBN) method by combining the dBN with the adaptive neuron enhancement (ANE) method [28]. Numerical examples demonstrate the ability of AdBN not only to move mesh points quickly and efficiently but also to achieve a nearly optimal order of approximation. Both the dBN and AdBN outperform BFGS in terms of cost and accuracy.

Related work. In the context of the shallow NN for multi-dimension, the active neuron least squares (ANLS) method was recently introduced in [1,2] to efficiently update the non-linear parameters. By utilizing both the quadratic structure of the functional and the NN structure, the structure-guided Gauss-Newton (SgGN) method was newly proposed in [13] for solving the non-linear least-squares problem. For several one- and two-dimensional least-squares test problems which are difficult for the commonly used training algorithms in machine learning such as BFGS and Adam, the SgGN shows superior convergence. However, the mass matrix for the linear parameters and the layer Gauss-Newton matrix are ill-conditioned even though they are symmetric and positive definite.

The paper is structured as follows. Section 2 describes FKS and shallow ReLU NNs as well as their equivalence. The shallow Ritz method for the diffusion equation is presented and analyzed in Section 3. Optimality conditions of the shallow Ritz discretization are derived and the condition number of the stiffness matrix is estimated in Section 4. The damped block Newton (dBN) method and its adaptive version are introduced in Sections 5 and 6, respectively. Finally, numerical results are presented in Section 7.

2. Free-knot splines and shallow neural networks

This section describes continuous linear free-knot splines (FKS) [34] and shallow neural networks in one dimension as well as their equivalence.

Set $b_{-1} = b_0 = 0$ and $b_{n+2} = b_{n+1} = 1$. Let $\phi_i(x)$ be the standard hat basis function with support on (b_{i-1}, b_{i+1}) given by

$$\phi_i(x) = \begin{cases} (x - b_{i-1})/(b_i - b_{i-1}), & x \in (b_{i-1}, b_i), \\ (b_{i+1} - x)/(b_{i+1} - b_i), & x \in (b_i, b_{i+1}), \\ 0, & \text{otherwise.} \end{cases} \tag{2.1}$$

Then the linear FKS on the interval $I = [0, 1]$ is defined as

$$S_n(I) = \left\{ \sum_{i=0}^{n+1} c_i \phi_i(x) : \; c_i, b_i \in \mathbb{R} \text{ and } 0 < b_1 < \ldots < b_n < 1 \right\}. \tag{2.2}$$

The $\{b_i\}_{i=1}^n$ are referred to as the knots, or breaking points, and are the parameters to be determined.

The FKS was introduced in the late 1960s and has been extensively studied since then. By adjusting the breaking points, the FKS gains substantially in approximation order for non-smooth functions. For example, let $0 < \alpha < 1$, then the order of the best approximation to x^α on I is $\mathcal{O}(n^{-\alpha})$ when using linear finite elements on the uniform mesh. Whereas for the FKS, the order is $\mathcal{O}(n^{-1})$ (see, e.g., [15,34]). Despite this remarkable approximation capability of FKS, numerical analysts have largely moved away from this method due to two main challenges: (1) no successful extension of FKS to two or higher dimensions has been achieved, and (2) determining the optimal placement of $\{b_i\}_{i=1}^n$ has remained a difficult and unresolved problem.

Denote by $\sigma(x) = \max\{0, x\}$ the ReLU activation function. Set $b_0 = 0$ and $b_{n+1} = 1$. Then the collection of the shallow neural network (NN) functions introduced by Rosenblatt in 1958 is given by

$$\mathcal{M}_n(I) = \left\{ c_{-1} + \sum_{i=0}^{n} c_i \sigma(x - b_i) \,:\, c_i, b_i \in \mathbb{R} \text{ and } 0 < b_1 < \ldots < b_n < 1 \right\}, \tag{2.3}$$

where $\{b_i\}_{i=1}^n$ are the interior "mesh" points to be determined and the typically present weights $\{\omega_i\}_{i=0}^n$ are normalized (i.e., $\omega_i = 1$). Clearly, any function in $\mathcal{M}_n(I)$ is continuous piecewise linear with respect to the partition of the interval I by the breaking points $\{b_i\}_{i=1}^n$ and the boundary $\partial I = \{0, 1\} = \{b_0, b_{n+1}\}$. Therefore, the shallow ReLU NN is equivalent to the FKS, i.e. $\mathcal{S}_n(I) = \mathcal{M}_n(I)$, and can be extended to two or higher dimensions directly. This resolves the first challenge of the FKS due to the change of the basis functions from local support to global support. However, the price for this gain is complexity and ill-condition of the resulting algebraic structure. We will address this second fundamental challenge of FKS in section 5.

3. Shallow Ritz method

Consider the following one-dimensional diffusion equation

$$\begin{cases} -(a(x)u'(x))' = f(x), \quad x \in I = (0, 1), \\ u(0) = \alpha, \quad u(1) = \beta, \end{cases} \tag{3.1}$$

where $f(x)$ is a given real-valued function defined on I. Assume that the diffusion coefficient $a(x)$ is bounded below by a positive constant $\mu > 0$ almost everywhere on I. When $a(x)$ is a piecewise constant, (3.1) is referred to as an elliptic interface problem. Clearly, the strong form in (3.1) is invalid on the physical interface Γ where a is discontinuous. For completion, an additional *interface condition* (3.2) must be supplemented on the interface Γ:

$$(au')^+\Big|_\Gamma - (au')^-\Big|_\Gamma = 0. \tag{3.2}$$

Due to the non-local support of neurons, one of the Dirichlet boundary conditions is enforced strongly and the other can be enforced either algebraically (see Appendix A for the formulation) or weakly by penalizing the energy functional. We opt for the latter; the modified Ritz formulation of problem (3.1) is to find $u \in H^1(I) \cap \{u(0) = \alpha\}$ such that

$$J(u) = \min_{v \in H^1(I) \cap \{v(0) = \alpha\}} J(v), \tag{3.3}$$

where the modified energy functional is given by

$$J(v) = \frac{1}{2} \int_0^1 a(x)(v'(x))^2 dx - \int_0^1 f(x)v(x)dx + \frac{\gamma}{2}(v(1) - \beta)^2.$$

Here, $\gamma > 0$ is a penalization constant. Then the Ritz neural network approximation is to find $u_n \in \mathcal{M}_n(I) \cap \{u_n(0) = \alpha\}$ such that

$$J(u_n) = \min_{v \in \mathcal{M}_n(I) \cap \{v(0) = \alpha\}} J(v). \tag{3.4}$$

Below we estimate the error bound for the solution u_n to (3.4). To this end, define the bilinear and linear forms by

$$a(u, v) := \int_0^1 a(x)u'(x)v'(x)dx + \gamma u(1)v(1) \quad \text{and} \quad f(v) := \int_0^1 f(x)v(x)dx + \gamma\beta\, v(1), \tag{3.5}$$

respectively, and denote the induced norm of the bilinear form by $\|v\|_a^2 = a(v, v)$. Then the modified energy function is given by

$$J(v) = \frac{1}{2}a(v, v) - f(v) + \frac{1}{2}\gamma\beta^2. \tag{3.6}$$

Lemma 3.1. *Let u and u_n be the solutions of problems (3.3) and (3.4), respectively. Then*

$$\|u - u_n\|_a \le \sqrt{3} \inf_{v \in \mathcal{M}_n(I) \cap \{v(0) = \alpha\}} \|u - v\|_a + 2\sqrt{2}\left|a(1)u'(1)\right|\gamma^{-1/2}. \tag{3.7}$$

Proof. It is easy to see that the solution u of (3.1) satisfies

$$a(u,v)+\alpha a(0)u'(0)-a(1)u'(1)v(1)=f(v) \tag{3.8}$$

for any $v\in H^1(I)\cap\{v(0)=\alpha\}$. Together with (3.6), we have

$$a(v-u,v-u)=2(J(v)-J(u))+2a(1)u'(1)(u(1)-v(1)),$$

which, combining with the inequality that $2c\,d\le 2\gamma^{-1}c^2+\frac{\gamma}{2}d^2$, implies

$$\|v-u\|_a^2-4\left|a(1)u'(1)\right|^2\gamma^{-1}\le 4(J(v)-J(u))\le 3\|v-u\|_a^2+4\left|a(1)u'(1)\right|^2\gamma^{-1}.$$

Now, together with the fact that $J(u_n)\le J(v)$ for any $v\in\mathcal{M}_n(I)\cap\{v(0)=\alpha\}$, we have

$$\|u_n-u\|_a^2\le 4(J(v)-J(u))+4\left|a(1)u'(1)\right|^2\gamma^{-1}\le 3\|v-u\|_a^2+8\left|a(1)u'(1)\right|^2\gamma^{-1}.$$

This implies (3.7) and proves the lemma. □

Since $\mathcal{M}_n(I)=S_n(I)$, it is reasonable to assume that there exists a constant $C(u)$ depending on u such that

$$\inf_{v\in\mathcal{M}_n(I)}\|a^{1/2}(u-v)'\|_{L^2(I)}\le C(u)\,n^{-1}, \tag{3.9}$$

provided that u has certain smoothness. Obviously, (3.9) is valid for $u\in H^2(I)=W^{2,2}(I)$ even when the breaking points are fixed and form a quasi-uniform partition of the interval I. It is expected that (3.9) is also valid for $u\in W^{2,1}(I)$ when the breaking points are free [34].

Proposition 3.2. *Let u and u_n be the solutions of the problem (3.3) and (3.4), respectively. Assume that $a\in L^\infty(I)$, then there exists a constant C depending on u such that*

$$\|u-u_n\|_a\le C\left(n^{-1}+\gamma^{-1/2}\right). \tag{3.10}$$

Proof. The proposition is a direct consequence of Lemma 3.1 and (3.9). □

We close this section by introducing a concept on optimal breaking points.

Definition 3.3. The breaking points $\{\hat{b}_i\}_{i=1}^n$ of a NN function $\hat{u}_n\in\mathcal{M}_n(I)$ are said to be optimal if $\hat{u}_n\in\mathcal{M}_n(I)$ satisfies the error bound in (3.10).

4. Optimality conditions

This section derives optimality conditions of (3.4) and estimates the condition number of the stiffness matrix.

To this end, let $u_n(x)=\alpha+\sum_{i=0}^n c_i\sigma(x-b_i)$ be a solution of (3.4). Denote by $\mathbf{c}=(c_0,\dots,c_n)^T$ and $\mathbf{b}=\left(b_1,\dots,b_n\right)^T$ the respective linear and non-linear parameters. Set

$$\mathbf{\Sigma}=\mathbf{\Sigma}(x)=(\sigma(x-b_0),\dots,\sigma(x-b_n))^T\quad\text{and}\quad\mathbf{H}=\mathbf{H}(x)=(H(x-b_0),\dots,H(x-b_n))^T,$$

where $H(t)=\sigma'(t)$ is the Heaviside step function given by

$$H(t)=\sigma'(t)=\begin{cases}1, & t>0,\\ 0, & t<0.\end{cases}$$

Clearly, we have the following identities

$$u_n(x)=\alpha+\mathbf{\Sigma}^T\mathbf{c},\quad u_n'(x)=\mathbf{H}^T\mathbf{c},\quad\nabla_{\mathbf{c}}u_n(x)=\mathbf{\Sigma},\quad\text{and}\quad\nabla_{\mathbf{c}}u_n'(x)=\mathbf{H}, \tag{4.1}$$

where $\nabla_{\mathbf{c}}$ and $\nabla_{\mathbf{b}}$ are the gradient operators with respect to the $\mathbf{c}$ and $\mathbf{b}$, respectively.

By (4.1), we have

$$\nabla_{\mathbf{c}}J(u_n)=\int_0^1 a(x)u_n'(x)\nabla_{\mathbf{c}}u_n'(x)dx-\int_0^1 f(x)\nabla_{\mathbf{c}}u_n(x)dx+\gamma(u_n(1)-\beta)\nabla_{\mathbf{c}}u_n(1)$$

$$= \left[\int_0^1 a(x)\mathbf{H}(x)\mathbf{H}(x)^T dx \right] \mathbf{c} - \int_0^1 f(x)\boldsymbol{\Sigma}(x) dx + \gamma \left(\mathbf{d}^T \mathbf{c} + \alpha - \beta \right) \mathbf{d},$$

where $\mathbf{d} = \boldsymbol{\Sigma}(1)$. Denote by

$$A(\mathbf{b}) = \int_0^1 a(x)\mathbf{H}(x)\mathbf{H}(x)^T dx \quad \text{and} \quad \mathbf{f}(\mathbf{b}) = \int_0^1 f(x)\boldsymbol{\Sigma}(x) dx,$$

the stiffness matrix and right hand side vector, respectively, with components

$$a_{ij}(\mathbf{b}) = \int_0^1 a(x) H(x - b_{i-1}) H(x - b_{j-1}) dx \quad \text{and} \quad f_i = \int_0^1 f(x)\sigma(x - b_{i-1}) dx.$$

Now, the optimality condition of (3.4) with respect to the linear parameters $\mathbf{c}$ is given by

$$\mathbf{0} = \nabla_{\mathbf{c}} J(u_n) = \mathcal{A}(\mathbf{b})\,\mathbf{c} - \mathcal{F}(\mathbf{b}), \tag{4.2}$$

where $\mathcal{A}(\mathbf{b}) = A(\mathbf{b}) + \gamma \mathbf{d}\mathbf{d}^T$ and $\mathcal{F}(\mathbf{b}) = \mathbf{f}(\mathbf{b}) + \gamma(\beta - \alpha)\mathbf{d}$.

Next, we derive the optimality condition of (3.4) with respect to the non-linear parameters $\mathbf{b}$. For $j = 1, \dots, n$, let

$$q_j = \int_{b_j}^1 f(x)\, dx - a(b_j) u_n'(b_j), \quad \text{where} \quad u_n'(b_j) := \frac{u_n'(b_j^+) + u_n'(b_j^-)}{2} = \sum_{i=0}^{j-1} c_i + c_j/2. \tag{4.3}$$

Additionally, let

$$\mathbf{q} = (q_1, \dots, q_n)^T, \quad \hat{\mathbf{c}} = (c_1, \dots, c_n), \quad \text{and} \quad \mathbf{1} = (1, \dots, 1)^T.$$

Lemma 4.1. *The optimality condition of* (3.4) *with respect to* $\mathbf{b}$ *is given by*

$$\nabla_{\mathbf{b}} J(u_n) = \mathbf{D}(\hat{\mathbf{c}}) \left\{ \mathbf{q} - \gamma(u_n(1) - \beta)\mathbf{1} \right\} = \mathbf{0}, \tag{4.4}$$

where $\mathbf{D}(\hat{\mathbf{c}})$ *is a diagonal matrix consisting of the linear parameters* $(c_1, \dots, c_n)$.

Proof. For each $j = 0, 1, \dots, n$, we have

$$\int_{b_j}^{b_{j+1}} a(x)(u_n'(x))^2\, dx = \int_{b_j}^{b_{j+1}} a(x) \left(\sum_{i=0}^{j} c_i H(x - b_i) \right)^2 dx = \left(\sum_{i=0}^{j} c_i \right)^2 \int_{b_j}^{b_{j+1}} a(x) dx,$$

which yields

$$\frac{\partial}{\partial b_j} \int_0^1 a(x)(u_n'(x))^2 dx = a(b_j) \left(\sum_{i=0}^{j-1} c_i \right)^2 - a(b_j) \left(\sum_{i=0}^{j} c_i \right)^2 = -2c_j a(b_j) \left(\sum_{i=0}^{j-1} c_i + \frac{c_j}{2} \right).$$

It is easy to see that

$$\frac{\partial u_n(x)}{\partial b_j} = -c_j H(x - b_j) \quad \text{and} \quad \frac{\partial u_n(1)}{\partial b_j} = -c_j,$$

which implies

$$\frac{\partial}{\partial b_j} \int_0^1 f(x) u_n(x)\, dx = -c_j \int_{b_j}^1 f(x)\, dx \quad \text{and} \quad \frac{\partial}{\partial b_j} (u_n(1) - \beta)^2 = -2c_j (u_n(1) - \beta).$$

Hence, the optimality condition of (3.4) with respect to b_j is

$$0 = \frac{\partial}{\partial b_j} J(u_n) = -c_j \left[a(b_j) \left(\sum_{i=0}^{j-1} c_i + \frac{c_j}{2} \right) - \int_{b_j}^1 f(x)\, dx + \gamma(u_n(1) - \beta) \right].$$

This completes the proof of the lemma. □

Remark 4.2. If $c_l = 0$ for some $l \in \{1, \dots, n\}$, then there is no l^{th} equation of the optimality condition in (4.4). In this case, the l^{th} neuron has no contribution to the approximation $u_n(x)$ and hence can be removed or redistributed.

Let $h_i = b_{i+1} - b_i$ for $i = 0, \dots, n$ and $h_{\min} = \min_{0 \le i \le n} h_i$. The next lemma provides an upper bound for the condition number of the stiffness matrix $A(\mathbf{b})$.

Lemma 4.3. *Let $a(x) = 1$ in (3.1), then the condition number of the stiffness matrix $A(\mathbf{b})$ is bounded by $\mathcal{O}\left(n\, h_{\min}^{-1}\right)$.*

Proof. For any vector $\boldsymbol{\xi} = (\xi_0, \dots, \xi_n)^T \in \mathbb{R}^n$, denote its magnitude by $\left|\boldsymbol{\xi}\right| = \left(\sum_{i=0}^{n} \xi_i^2\right)^{1/2}$. By the Cauchy-Schwarz inequality, we have

$$\begin{aligned}\boldsymbol{\xi}^t A(\mathbf{b})\boldsymbol{\xi} &= \int_0^1 \left(\sum_{i=0}^{n} \xi_i H(x - b_i)\right)^2 dx \le |\boldsymbol{\xi}|^2 \int_0^1 \left(\sum_{i=0}^{n} H(x - b_i)^2\right) dx \\ &= |\boldsymbol{\xi}|^2 \sum_{i=0}^{n} (1 - b_i) < (n+1)|\boldsymbol{\xi}|^2. \end{aligned} \tag{4.5}$$

To estimate the lower bound of the quadratic form $\boldsymbol{\xi}^t A(\mathbf{b})\boldsymbol{\xi}$, note that

$$\boldsymbol{\xi}^t A(\mathbf{b})\boldsymbol{\xi} = \sum_{j=0}^{n} \int_{b_j}^{b_{j+1}} \left(\sum_{i=0}^{j} \xi_i H(x - b_i)\right)^2 dx = \sum_{j=0}^{n} h_j \left(\sum_{i=0}^{j} \xi_i\right)^2 \ge h_{\min} \sum_{j=0}^{n} \left(\sum_{i=0}^{j} \xi_i\right)^2. \tag{4.6}$$

Now, the lemma is a direct consequence of (4.5), (4.6), and the fact that

$$|\boldsymbol{\xi}|^2 = \sum_{j=0}^{n} \left(\sum_{i=0}^{j} \xi_i - \sum_{i=0}^{j-1} \xi_i\right)^2 \le 2 \sum_{j=0}^{n} \left(\sum_{i=0}^{j} \xi_i\right)^2 + 2 \sum_{j=0}^{n} \left(\sum_{i=0}^{j-1} \xi_i\right)^2 \le 4 \sum_{j=0}^{n} \left(\sum_{i=0}^{j} \xi_i\right)^2.$$

This completes the proof of the lemma. □

5. A damped block Newton method

The optimality conditions in (4.2) and (4.4) are systems of non-linear algebraic equations. One may use the Variable Projection (VarPro) method of Golub-Pereyra [22] that substitutes the solution $\mathbf{c}$ of (4.2) into (4.4) to produce a reduced non-linear system for the $\mathbf{b}$. However, this approach can significantly complicate the non-linear structure of the system.

In this section, we introduce a damped block Newton (dBN) method for solving the resulting non-convex minimization problem in (3.4). The method employs a commonly used outer-inner iterative method by alternating between updates for $\mathbf{c}$ and $\mathbf{b}$. Per each outer iteration, the $\mathbf{c}$ and the $\mathbf{b}$ are updated by exact inversion and one step of a damped Newton method, respectively. To do so, we need to study the inversions of the stiffness and Hessian matrices.

The stiffness matrix A has the form of

$$A(\mathbf{b}) = \begin{pmatrix} \int_{b_0}^1 a(x)dx & \int_{b_1}^1 a(x)dx & \int_{b_2}^1 a(x)dx & \cdots & \int_{b_n}^1 a(x)dx \\ \int_{b_1}^1 a(x)dx & \int_{b_1}^1 a(x)dx & \int_{b_2}^1 a(x)dx & \cdots & \int_{b_n}^1 a(x)dx \\ \int_{b_2}^1 a(x)dx & \int_{b_2}^1 a(x)dx & \int_{b_2}^1 a(x)dx & \cdots & \int_{b_n}^1 a(x)dx \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \int_{b_n}^1 a(x)dx & \int_{b_n}^1 a(x)dx & \int_{b_n}^1 a(x)dx & \cdots & \int_{b_n}^1 a(x)dx \end{pmatrix}.$$

Lemma 5.1. *The inversion of the stiffness matrix $A(\mathbf{b})$ is tri-diagonal given by*

$$A(\mathbf{b})^{-1} = \begin{pmatrix} \frac{1}{s_1} & -\frac{1}{s_1} & 0 & 0 & \cdots & 0 & 0 \\ -\frac{1}{s_1} & \frac{1}{s_1} + \frac{1}{s_2} & -\frac{1}{s_2} & 0 & \cdots & 0 & 0 \\ 0 & -\frac{1}{s_2} & \frac{1}{s_2} + \frac{1}{s_3} & -\frac{1}{s_3} & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \frac{1}{s_{n-1}} + \frac{1}{s_n} & -\frac{1}{s_n} \\ 0 & 0 & 0 & 0 & \cdots & -\frac{1}{s_n} & \frac{1}{s_n} + \frac{1}{s_{n+1}} \end{pmatrix}, \tag{5.1}$$

where $s_i = \int_{b_{i-1}}^{b_i} a(x)dx$ for $i = 1, \dots, n+1$. Moreover, the inversion of the coefficient matrix $\mathcal{A}(\mathbf{b})$ is given by

$$\mathcal{A}(\mathbf{b})^{-1} = A(\mathbf{b})^{-1} - \frac{\gamma A(\mathbf{b})^{-1}\mathbf{d}\mathbf{d}^T A(\mathbf{b})^{-1}}{1+\gamma \mathbf{d}^T A(\mathbf{b})^{-1}\mathbf{d}}. \tag{5.2}$$

Proof. It is easy to verify that $A(\mathbf{b})^{-1}A(\mathbf{b}) = I$. (5.2) is a direct consequence of (5.1) and the Sherman-Morrison formula. □

Now, we study how to solve the system of non-linear algebraic equations in (4.4). As stated in Remark 4.2, if $c_l = 0$, then b_l is removed from the system in (4.4). Therefore, we may assume that $c_j \neq 0$ for all $j \in \{1, \dots, n\}$.

Lemma 5.2. *Assume that $c_j \neq 0$ for all $j \in \{1, \dots, n\}$ and that $a(x)$ is differentiable at $\{b_j\}_{j=1}^n$. Then we have*

$$\nabla_{\mathbf{b}}^2 J(u_n) \equiv \mathcal{H}(\mathbf{c}, \mathbf{b}) = \mathbf{D}(\hat{\mathbf{c}})\left(\mathbf{D}(\mathbf{g}) + \gamma \mathbf{1}\hat{\mathbf{c}}^T\right), \tag{5.3}$$

where $\mathbf{D}(\mathbf{g}) = diag\ \left(g_1, \dots, g_n\right)$ is a diagonal matrix with the j^{th} entry

$$g_j = \frac{\partial}{\partial b_j} q_j = -f(b_j) - a'(b_j)u_n'(b_j).$$

Proof. Since $u_n'(b_j) = \sum_{i=0}^{j-1} c_i + c_j/2$ is independent of b_j, then we have

$$\frac{\partial}{\partial b_k} q_j = \begin{cases} g_j, & k = j, \\ 0, & k \neq j \end{cases} \quad \text{for} \quad k = 1, \dots, n.$$

Now (5.3) is a direct consequence of Lemma 4.1 and the fact that $\nabla_{\mathbf{b}}(u_n(1) - \beta) = -\hat{\mathbf{c}}$. □

If $a(x)$ is not differential at b_l for some $l \in \{1, \dots, n\}$, then b_l lies at the physical interface. Hence, it should be fixed without further update. That is, it is removed from the system in (4.4).

If $g_l = 0$ for some $l \in \{1, \dots, n\}$, then the matrix $\mathbf{D}(\mathbf{g})$ is singular. By (3.1), we have

$$0 = g_l = -f(b_l) - a'(b_l)u_n'(b_l) \approx -f(b_l) - a'(b_l)u'(b_l) = a(b_l)u''(b_l). \tag{5.4}$$

This indicates that b_l is an approximate inflection or undulation point of the solution $u(x)$. In the case of the former, b_l may be removed or redistributed. Otherwise, b_l should be fixed without further update. In both cases, b_l is again removed from the system in (4.4). To distinguish these two cases, notice that the linear parameter

$$c_l = u_n'(b_l^+) - u_n'(b_l^-)$$

represents the change in slope of $u_n(x)$ at $x = b_l$. Hence, given a prescribed tolerance $\tau_1 > 0$, if

$$|c_l| < \tau_1, \tag{5.5}$$

then b_l corresponds to an approximate inflection point; otherwise, it corresponds to an approximate undulation point.

Lemma 5.3. *Under the assumptions of Lemma 5.2, if $g_i \neq 0, c_i \neq 0$ for all $i \in \{1, \dots, n\}$, and $\zeta \equiv 1 - \gamma \sum_{i=1}^n c_i/g_i \neq 0$, then the Hessian matrix $\mathcal{H}(\mathbf{c}, \mathbf{b})$ is invertible. Moreover, its inverse is given by*

$$\mathcal{H}(\mathbf{c}, \mathbf{b})^{-1} = -\mathbf{D}(\mathbf{g})^{-1}\left[I + \frac{\gamma}{\zeta}\mathbf{1}\hat{\mathbf{c}}^T\mathbf{D}(\mathbf{g})^{-1}\right]\mathbf{D}(\hat{\mathbf{c}})^{-1}. \tag{5.6}$$

Proof. Clearly, by Lemma 5.2, the assumptions imply that $\mathcal{H}(\mathbf{c}, \mathbf{b})$ is invertible. (5.6) is a direct consequence of the Sherman-Morrison formula. □

Now, we are ready to describe the dBN method (see Algorithm 5.1 for a pseudocode). Given prescribed tolerances $\tau_1 > 0$ and $\tau_2 > 0$, let $\mathbf{b}^{(k)}$ be the previous iterate, then the current iterate $\left(\mathbf{c}^{(k+1)}, \mathbf{b}^{(k+1)}\right)$ is computed as follows:

(i) *Compute the linear parameters*

$$\mathbf{c}^{(k+1)} = \mathcal{A}\left(\mathbf{b}^{(k)}\right)^{-1}\mathcal{F}\left(\mathbf{b}^{(k)}\right)$$

(ii) *Compute the search direction* $\mathbf{p}^{(k)} = \left(p_1^{(k)}, \dots, p_n^{(k)}\right)$. Let $S = S_1 \cup S_2$, where S_1 is the set of indices for the non-contributing neurons

$$S_1 = \left\{ i \in \{1, \dots, n\} : \left|c_i^{(k)}\right| < \tau_1 \text{ or } b_i^{(k)} \notin I \right\}$$

and S_2 is the set of indices for which the corresponding neurons are either redistributed or fixed

$$S_2 = \left\{ i \in \{1, \dots, n\} : \left| g_i^{(k)} \right| < \tau_2 \text{ or DNE} \right\}.$$

Compute

$$p_i^{(k)} = \begin{cases} 0, & i \in S, \\ \dfrac{1}{g_i^{(k)}} \left(\gamma \bar{\beta} - q_i^{(k)} + \gamma \bar{\gamma} / \bar{\xi} \right), & i \notin S, \end{cases} \tag{5.7}$$

where $\bar{\beta} = \left(u_n(1) - \beta \right)$, $\bar{\xi} = 1 - \gamma \sum_{j \notin S} c_j^{(k)} / g_j^{(k)}$, and $\bar{\gamma} = \sum_{j \notin S} c_j^{(k)} \left(\gamma \bar{\beta} - q_j^{(k)} \right) / g_j^{(k)}$.

(iii) *Calculate the stepsize* η_k by performing a one-dimensional optimization

$$\eta_k = \underset{\eta \in \mathbb{R}^+}{\arg\min}\, J\left(u_n\left(x; \mathbf{c}^{(k+1)}, \mathbf{b}^{(k)} + \eta \mathbf{p}^{(k)} \right) \right).$$

(iv) *Compute the non-linear parameters*

$$\mathbf{b}^{(k+1)} = \mathbf{b}^{(k)} + \eta_k \mathbf{p}^{(k)}.$$

(v) *Redistribute non-contributing breakpoints* $b_i^{(k+1)}$ for all $i \in S_1$ (e.g., per Remark 5.4) and sort $\mathbf{b}^{(k+1)}$.

Remark 5.4. In the implementation of Algorithm 5.1 in Section 7, the particular redistribution for a neuron $b_l^{(k+1)}$ satisfying (5.5) is:

$$b_l^{(k+1)} \leftarrow \frac{b_{m-1}^{(k+1)} + b_m^{(k+1)}}{2},$$

where $m \in \{1, \dots, n+1\}$ is an integer chosen uniformly at random.

Algorithm 5.1 A damped block Newton (dBN) method for (3.4).

Input: Initial network parameters $\mathbf{b}^{(0)}$, coefficient function $a(x)$, the right-hand side function $f(x)$, boundary data α and β.
Output: Network parameters $\mathbf{c}$, $\mathbf{b}$
for $k = 0, 1 \dots$ **do**
▷ *Linear parameters*
$\mathbf{c}^{(k+1)} \leftarrow \mathcal{A}\left(\mathbf{b}^{(k)}\right)^{-1} \mathcal{F}\left(\mathbf{b}^{(k)}\right)$
▷ *Non-linear parameters*
Compute the search direction $\mathbf{p}^{(k)}$ *as in* (5.7)
$\eta_k \leftarrow \underset{\eta \in \mathbb{R}^+}{\arg\min}\, J(u_n(x; \mathbf{c}^{(k+1)}, \mathbf{b}^{(k)} + \eta \mathbf{p}^{(k)}))$
$\mathbf{b}^{(k+1)} \leftarrow \mathbf{b}^{(k)} + \eta_k \mathbf{p}^{(k)}$
▷ *Redistribute non-contributing neurons and sort* $\mathbf{b}^{(k+1)}$
end for

By (5.2) and (5.6), the linear parameters $\mathbf{c}^{(k+1)}$ and the direction vector $\mathbf{p}^{(k)}$ may be calculated in $8(n+1)$ and $4(n+1)$ operations, respectively. Therefore, the computational cost of Algorithm 5.1 per step is $\mathcal{O}(n)$. This is significantly less than Quasi-Newton approaches like BFGS, where the computational cost per iteration is $\mathcal{O}(n^2)$ (see [31], Chapter 6).

Remark 5.5. The minimization problem in (3.4) is non-convex and has many local and global minima. The desired one is obtained only if we start from a close enough first approximation. Because the non-linear parameters $\mathbf{b}$ correspond to the breaking points that partition the interval $I = [0, 1]$, as in *[7,27]*, in this paper we initialize the non-linear parameters $\mathbf{b}$ to form a uniform partition of the interval $[0, 1]$ and the linear parameters $\mathbf{c}$ to be the corresponding solution on this uniform mesh.

6. Adaptive dBN (AdBN) method

Starting with the uniform distribution of the breakpoints $\{b_i\}_{i=1}^n$ as an initial (see Remark 5.5), the dBN efficiently moves them to a nearly optimal location (see Section 7). However, the location of the resulting breakpoints may not be optimal for achieving the optimal convergence order of the shallow Ritz approximation in Proposition 3.2. This is because the uniform distribution is often not at the dBN basin of convergence when the number of breakpoints is relatively small.

To circumvent this difficulty, we employ the adaptive neuron enhancement (ANE) method [27,28,9] using the dBN as the non-linear solver per adaptive step, and the resulting method is referred to as the adaptive dBN (AdBN) method. The ANE integrates the "moving mesh" feature of the shallow Ritz method with the local mesh refinement of adaptive finite element method (aFEM). It starts with a relatively small NN and adaptively adds new neurons based on the previous approximation. Moreover, the newly added

neurons are initialized at points where the previous approximation is not accurate. At each adaptive step, we use the dBN method to numerically solve the minimization problem in (3.4).

A key component of the ANE is the marking strategy, defined by error indicators, which determines the number of new neurons to be added. Below, we describe the local indicators and the marking strategy used in this paper.

Letting $\mathcal{K} = [c, d] \subseteq [0, 1]$ be a subinterval, a modified local indicator of the ZZ type on $\mathcal{K}$ (see, e.g., [14]) is defined by

$$\xi_{\mathcal{K}} = \|a^{-1/2}\left(G(au_n') - au_n'\right)\|_{L^2(\mathcal{K})},$$

where $G(au_n')$ is the projection of au_n' onto the space of the continuous piecewise linear functions. The corresponding relative error estimator is defined by

$$\xi = \frac{\|a^{-1/2}\left(G(au_n') - au_n'\right)\|_{L^2(I)}}{\|u_n'\|_{L^2(I)}}. \tag{6.1}$$

For a given $u_n \in \mathcal{M}_n(I)$ with the breakpoints

$$0 = b_0 < b_1 < \ldots < b_n < b_{n+1} = 1,$$

let $\mathcal{K}^i = [b_i, b_{i+1}]$, then $\mathcal{K}_n = \left\{\mathcal{K}^i\right\}_{i=0}^{n}$ is a partition of the interval $[0, 1]$. Define a subset $\widehat{\mathcal{K}}_n \subset \mathcal{K}_n$ by using the following average marking strategy:

$$\widehat{\mathcal{K}}_n = \left\{ K \in \mathcal{K}_n : \xi_{\mathcal{K}} \geq \frac{1}{\#\mathcal{K}_n} \sum_{\mathcal{K} \in \mathcal{K}_n} \xi_{\mathcal{K}} \right\}, \tag{6.2}$$

where $\#\mathcal{K}_n$ is the number of elements in $\mathcal{K}_n$. The AdBN method is described in Algorithm 6.1.

Algorithm 6.1 Adaptive damped block Newton (AdBN) method.

Input: Initial number of neurons n_0, parameters $a(x)$, $f(x)$, α, and β, tolerance ϵ,
(1) Compute an approximation to the solution u_n of the optimization problem in (3.4) by the dBN method;
(2) Compute the estimator $\xi_n = \left(\sum_{K \in \mathcal{K}} \xi_K^2\right)^{1/2} / |u_n|_{H^1(I)}$;
(3) If $\xi_n \leq \epsilon$, then stop; otherwise go to step (4);
(4) Mark elements in $\widehat{\mathcal{K}}_n$ and denote by $\#\widehat{\mathcal{K}}_n$ the number of elements in $\widehat{\mathcal{K}}_n$;
(5) Add $\#\widehat{\mathcal{K}}_n$ new neurons to the network and initialize them at the midpoints of elements in $\widehat{\mathcal{K}}_n$, then go to step (1)

For numerical experiments presented in the subsequent section, the dBN method in step (1) of Algorithm 6.1 is stopped if the difference of the relative residuals for two consecutive iterates is less than a prescribed tolerance.

7. Numerical experiments

This section presents numerical results of the dBN and AdBN methods for solving (3.1). In all the experiments, the parameters τ_1 and τ_2 were set to 10^{-10} and 10^{-6}, respectively. The penalization parameter γ was set to 10^4 for the first two test problems (subsection 7.1 and subsection 7.2) and 10^{13} for the third one (subsection 7.3). For the AdBN method, a refinement occurred when the absolute difference of the relative residuals for two consecutive iterates was less than 10^{-3}.

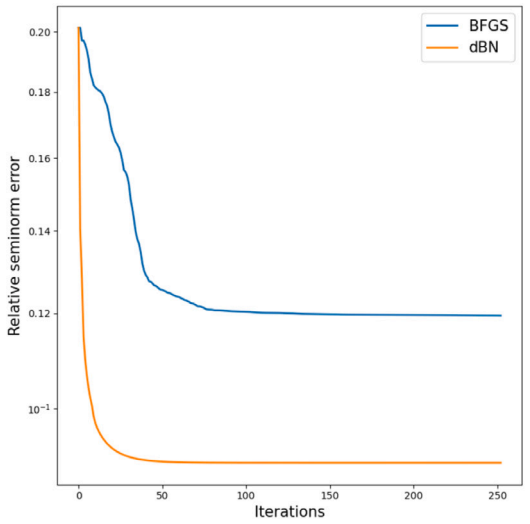
In order to evaluate the performance of the iterative solvers described in Algorithm 5.1 and Algorithm 6.1, we compare the resulting approximations to the true solution. For each test problem, let u and u_n be the exact solution and its approximation in $\mathcal{M}_n(I)$, respectively. Denote the relative H_1 seminorm error by

$$e_n = \frac{|(u - u_n)|_{H^1(I)}}{|u|_{H^1(I)}}.$$

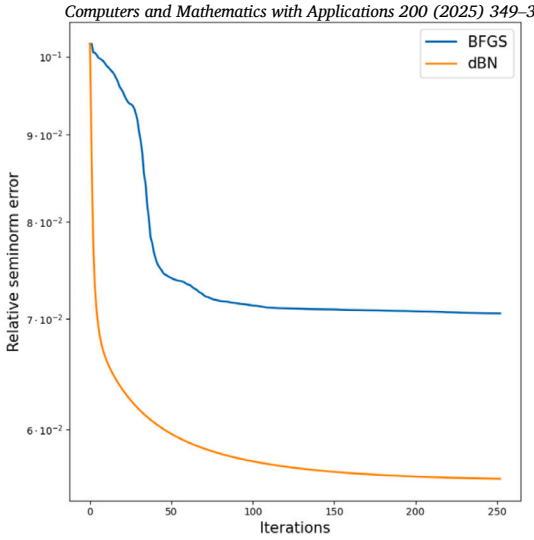
Aside from observing the true error, we would like to evaluate the solver by estimating the order of convergence for select examples. Recall that according to Proposition 3.2, it is theoretically possible to achieve an order of convergence of $\mathcal{O}(n^{-1})$. However, since (3.4) is a non-convex minimization problem, the existence of local minima makes it challenging to achieve this order. Therefore, given the neural network approximation u_n to u provided by the dBN method, assume that

$$e_n = \left(\frac{1}{n}\right)^r,$$

for some $r > 0$. The larger this r is, the better the approximation. In subsection 7.1, we can see that r improves when introducing adaptivity to dBN.



(a) Relative error e_n vs number of iterations using 25 neurons, the ratio between the final errors is 0.753



(b) Relative error e_n vs number of iterations using 50 neurons, the ratio between the final errors is 0.794

Fig. 1. Comparison between BFGS and dBN for approximating function (7.1).

7.1. Exponential solution

The first test problem involves the function

$$u(x) = x\left(\exp\left(-\frac{(x-1/3)^2}{0.01}\right) - \exp\left(-\frac{4}{9\times 0.01}\right)\right), \tag{7.1}$$

serving as the exact solution of (3.1). This example highlights the performance of dBN in comparison to current solvers and is used to motivate the addition of adaptivity.

We start by comparing the dBN method with a commonly used method: BFGS. In this comparison, we utilized a Python BFGS implementation from 'scipy.optimize'. The initial network parameters for both algorithms were set to be the uniform mesh for $\mathbf{b}^{(0)}$ with $\mathbf{c}^{(0)}$ given by solving (4.2). We see in Fig. 1 that dBN requires about 20 iterations to get an accuracy that BFGS cannot achieve after 250 iterations. Recall that the computational cost per iteration of dBN is $\mathcal{O}(n)$ while each iteration of BFGS has cost $\mathcal{O}(n^2)$. Not only does dBN decrease the relative error much more quickly than BFGS, but dBN also achieves a lower final error.

Next we observe that our methods influence the movement of breakpoints, enhancing the overall approximation. In Fig. 2 (a), we present the initial neural network approximation of the exact solution in (7.1), obtained by using uniform breakpoints and determining the linear parameters through the solution of (4.2). The approximations generated by the dBN and AdBN methods are shown in Fig. 2 (b) and (c), respectively.

Worth noting is that the placement of breakpoints by AdBN appears to be more optimal than that achieved by dBN in terms of relative error. To verify this observation, we estimate the order of convergence for the approximation to (7.1) in Algorithm 5.1. In Table 1, dBN is applied for 1000 iterations for different values n, and the resulting r is calculated. Since

$$0.78 < r < 0.83,$$

it can be inferred that the algorithm gets stuck in a local minimum. Henceforth, to improve this order of convergence, we can use the AdBN method.

In fact, adding adaptivity improves the r value. Table 2 illustrates AdBN starting with 20 neurons, refining 8 times, and reaching a final count of 269 neurons. The stopping tolerance was set to $\epsilon = 0.01$. The recorded data in Table 2 includes the relative seminorm error and the error estimator for each iteration of the adaptive process. Additionally, Table 2 provides the results for dBN with a fixed 190 and 269 neurons after 300 iterations. Comparing these results to the adaptive run with the same number of neurons, we observe a significant improvement in rate, error estimator, and seminorm error within the adaptive run. The experiments confirm that AdBN improves the overall error. Furthermore, the order of convergence notably improves, especially with a larger number of neurons.

7.2. Non-smooth solution

The exact solution of the second test problem is

$$u(x) = x^{2/3}, \tag{7.2}$$

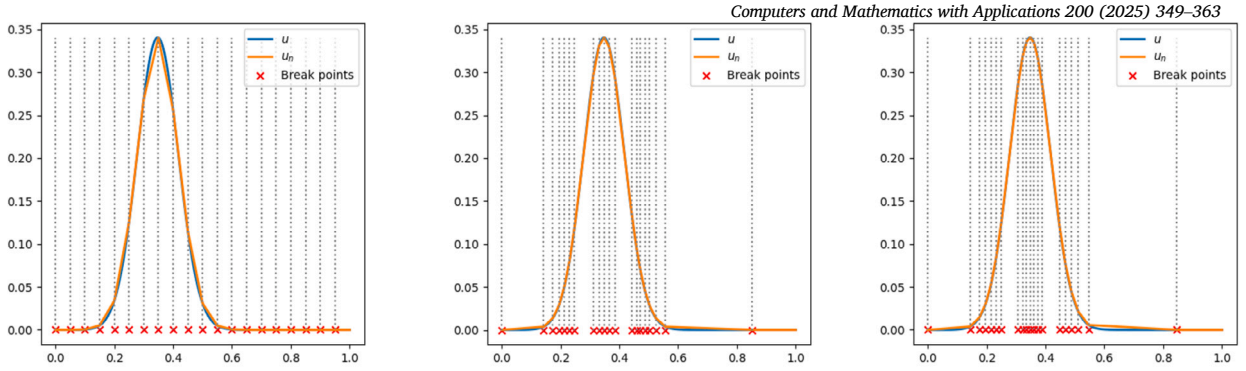


Fig. 2. (a) initial approximation with 20 uniform breakpoints, $e_n = 0.250$, (b) approximation after 500 iterations, $e_n = 0.104$, (c) adaptive approximation ($n = 13, 16, 20$), $e_n = 0.092$.

Table 1
Relative errors e_n and powers r for different numbers of breakpoints after 1000 iterations.

Breakpoints	e_n	r
60	4.07×10^{-2}	0.782
90	2.88×10^{-2}	0.789
120	1.92×10^{-2}	0.826
150	1.89×10^{-2}	0.792
180	1.61×10^{-2}	0.796
210	1.26×10^{-2}	0.818
240	1.16×10^{-2}	0.814
270	1.07×10^{-2}	0.811
300	9.54×10^{-3}	0.816
330	8.94×10^{-3}	0.813

Table 2
Comparison of an adaptive network with fixed networks for relative errors e_n, relative error estimators ξ_n, and powers r.

NN (breakpoints)	e_n	ξ_n	r
Adaptive (26)	7.06×10^{-2}	0.097	0.814
Adaptive (33)	5.68×10^{-2}	0.073	0.820
Adaptive (38)	3.98×10^{-2}	0.049	0.838
Adaptive (67)	2.87×10^{-2}	0.033	0.844
Adaptive (98)	1.92×10^{-2}	0.021	0.862
Adaptive (134)	1.42×10^{-2}	0.015	0.868
Adaptive (190)	9.92×10^{-3}	0.011	0.879
Adaptive (269)	7.14×10^{-3}	0.007	0.883
Fixed (190)	1.46×10^{-2}	0.016	0.806
Fixed (269)	1.15×10^{-2}	0.012	0.798

noting that $u(x) \in H^{1+1/6-\epsilon}(I)$ for any $\epsilon > 0$. As discussed in Section 2, the order of approximation with n uniform breakpoints is at most $1/6$, i.e., the error bound is $\mathcal{O}\left(n^{-1/6}\right)$. For $n = 22$ breakpoints, Fig. 3 depicts approximations on (a) the uniform mesh, (b) the moving mesh after 500 iterations of the dBN, and (c) the adaptively refined moving mesh ($n = 11, 16, 22$). The breakpoints move to the left where the solution exhibits the most curvature. As expected, the AdBN is more effective than the dBN.

Table 3 compares dBN, AdBN, and aFEM. In this experiment, the dBN method was run for 300 iterations for 14, 18, and 23 breakpoints. Both AdBN and aFEM start with 10 breakpoints and are refined 4 and 16 times, respectively. The same marking strategy (6.2) is used to determine intervals for refinement for aFEM. The true error of the dBN using 23 breakpoints is slightly more accurate than that of aFEM after 11 refinements (45 breakpoints), and the error of AdBN after 4 refinements (31 breakpoints) is more accurate to that of aFEM after its 16^{th} refinement (122 breakpoints).

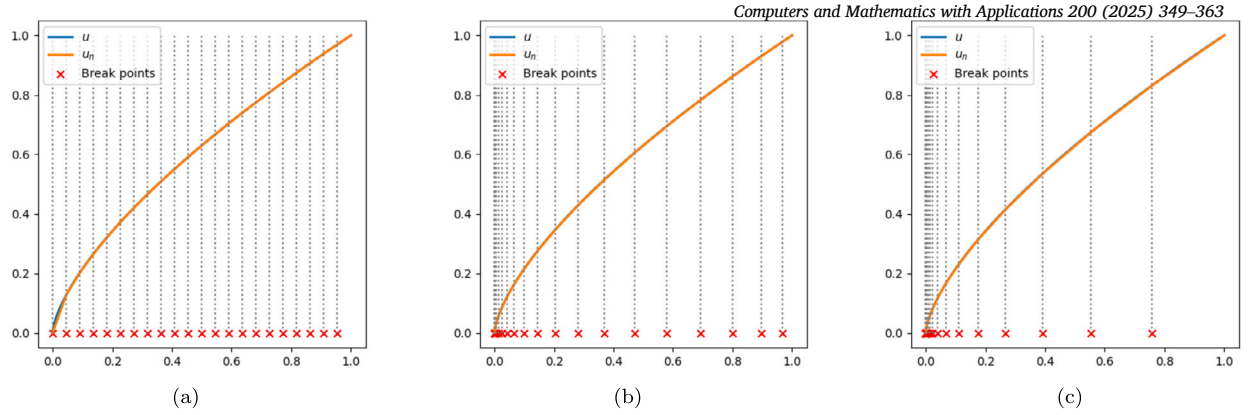


Fig. 3. (a) initial approximation with 22 uniform breakpoints, $e_n = 0.300$, (b) approximation after 500 iterations, $e_n = 0.086$, (c) adaptive approximation ($n = 11, 16, 22$), $e_n = 0.063$.

Table 3
Comparison of dBN, AdBN and aFEM for relative errors e_n, estimator errors ξ_n and powers r. The initial number of breakpoints for AdBN and aFEM is 10.

Method (breakpoints)	e_n	ξ_n	r
dBN (10)	1.38×10^{-1}	0.163	0.859
dBN (14)	1.17×10^{-1}	0.136	0.813
dBN (18)	1.01×10^{-1}	0.116	0.794
dBN (23)	8.99×10^{-2}	0.077	0.768
AdBN (14)	9.22×10^{-2}	0.106	0.903
AdBN (18)	7.59×10^{-2}	0.081	0.892
AdBN (23)	6.09×10^{-2}	0.063	0.892
AdBN (31)	4.74×10^{-2}	0.048	0.888
aFEM (10)	3.39×10^{-1}	0.103	0.451
aFEM (45)	9.77×10^{-2}	0.033	0.611
aFEM (99)	6.12×10^{-2}	0.020	0.608
aFEM (122)	5.45×10^{-2}	0.017	0.606

7.3. Interface problem

The third test problem is an elliptic interface problem whose solution has a discontinuous derivative. Specifically, the diffusion coefficient is $a(x) = 1 + (k-1)H(x - 1/2)$, and the right-hand side and true solution are given respectively by

$$f(x) = \begin{cases} 8k(3x-1), & x \in (0, 1/2), \\ 4k(k+1), & x \in (1/2, 1), \end{cases} \quad \text{and} \quad u(x) = \begin{cases} 4kx^2(1-x), & x \in (0, 1/2), \\ [2(k+1)x - 1](1-x), & x \in (1/2, 1). \end{cases}$$

This test problem was introduced in [10] to show that the Bramble-Schatz least-squares NN method (i.e., the physics-informed neural network (PINN)) produces an unacceptable numerical approximation, that completely misses the interface physics (see Figure 4.4 (c) in [10]), because the interface condition was not enforced.

Notice that the diffusion coefficient $a(x)$ is piecewise constant with interface at $x = 1/2$ and that its derivative is given by

$$a'(x) = \delta(x - 1/2) = \begin{cases} +\infty, & x = 1/2, \\ 0, & x \neq 1/2. \end{cases}$$

As explained in Section 5, if $b_l = 1/2$ for a $l \in \{1, \ldots, n\}$, then the breakpoint b_l remains unchanged.

The test problem employs 15 neurons so that the uniformly distributed breakpoints do not contain the interface point $x = 1/2$ upon initialization. For $k = 10^6$, this initial approximation is depicted in Fig. 4 (a). The approximation after 100 iterations of dBN is depicted in Fig. 4 (b) and is significantly more accurate because one of the breakpoints moved close to the interface point $x = 1/2$. Table 4 shows the relative errors of the shallow Ritz approximations on the uniform distribution and after 100 iterations of dBN for various values of k.

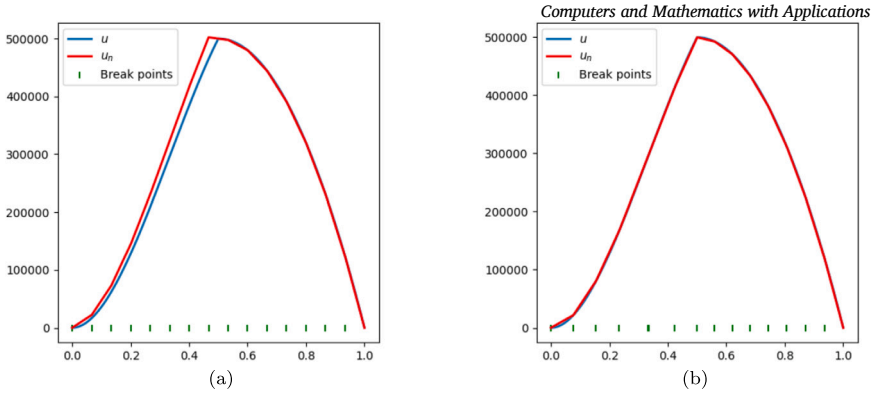


Fig. 4. For $k = 10^6$: (a) initial approximation with 15 uniform breakpoints, $e_n = 0.204$ and (b) approximation after 100 iterations, $e_n = 0.073$.

Table 4
Relative energy error e_n for 15 neurons: initial approximation on uniform breakpoints and approximation after 100 iterations of dBN.

k	e_n (initial)	e_n (dBN)
10	1.71×10^{-1}	6.86×10^{-2}
10^2	2.00×10^{-1}	7.06×10^{-2}
10^3	2.03×10^{-1}	6.48×10^{-2}
10^4	2.04×10^{-1}	7.27×10^{-2}
10^5	2.04×10^{-1}	7.28×10^{-2}
10^6	2.04×10^{-1}	7.30×10^{-2}
10^7	2.04×10^{-1}	6.70×10^{-2}
10^8	2.04×10^{-1}	7.46×10^{-2}

8. Conclusion and discussion

The shallow Ritz method improves the order of approximation dramatically for one-dimensional non-smooth diffusion problems. However, determining optimal mesh locations (the non-linear parameters of a shallow NN) is a complicated, computationally intensive non-convex optimization problem. We have addressed this challenging problem by developing the dBN method, a specially designed non-linear algebraic solver that is capable of efficiently moving the uniformly distributed mesh points to nearly optimal locations.

In the process of developing the dBN, we discovered that the exact inversion of the dense stiffness matrix is tri-diagonal; more importantly, we identified and overcame two difficulties of the Newton method for computing the non-linear parameters: (1) non-differentiable optimality condition and (2) vanishing linear parameters and diagonal elements of the Hessian matrix. The reduced non-linear system is differentiable and its Hessian is invertible. The computational cost of each dBN iteration is $\mathcal{O}(n)$.

For any given size n of shallow NN, the uniformly distributed mesh points may not always be a good initial for the non-linear parameters and possibly prevent the dBN from moving mesh points to optimal locations. This was addressed by proposing the AdBN which combines the dBN with the adaptive neuron enhancement (ANE) method for adaptively adding neurons and is initialized at where the previous approximation is inaccurate.

For non-smooth problems, the commonly used adaptive finite element method (aFEM) improves the inefficiency of the standard finite element method by adaptively refining the mesh, while the shallow Ritz method achieves the same goal by locating mesh points at optimal places. Both methods are non-linear, and their efficiencies depend on the local error indicator and the dBN, respectively. The AdBN is the combination of these two methods that efficiently moves and adaptively refines the mesh.

When using the shallow ReLU neural network, the condition number of the coefficient matrix for the diffusion problem is bounded by $\mathcal{O}\left(n\, h_{\min}^{-1}\right)$ (see Lemma 4.3), the same as that obtained when using local hat basis functions. For applications to general elliptic partial differential equations and least-squares data fitting problems, the corresponding dBN method requires inversion of the mass matrix. However, the condition number of the mass matrix is extremely large. This difficulty will be addressed in a forthcoming paper.

Extension of all components of the dBN to multi-dimension and to deep (two or more hidden layers) NNs are not straightforward, and some of them may not be possible. For example, the exact inversion of the stiffness matrix in multi-dimension would not be sparse, and the Hessian would no longer be diagonal or diagonal plus a low-rank update. Nevertheless, they do have some special structures that could be used for developing fast solvers (see, e.g., [11]). The most significant contribution of the dBN is how to handle singularities of the Hessian directly by using the physical meaning of the nonlinear parameters. This idea is valid for multi-dimension. For example, the structure-guided Gauss-Newton (SgGN) method was developed in [12] for solving shallow NN least-squares approximation in multi-dimension, and outperforms the commonly used Levenberg–Marquardt (LM) algorithm [26,30] by a large margin. The LM deals with singularities of the GN matrix by adding a regularization and hence changes the GN search

direction. In summary, the methodology presented in this paper has a great potential to be extended to multi-dimension and deep NNs for achieving better efficiency, accuracy, and reliability than the commonly used *generic* training algorithms in machine learning.

Code availability statement

The code used to generate the findings of this study is openly available in GitHub at https://doi.org/10.5281/zenodo.17336056.

Appendix A. Enforcing the Dirichlet boundary condition algebraically

Another way to make a function $u_n \in \mathcal{M}_n(I) \cap \{u_n(0) = \alpha\}$ satisfy the Dirichlet boundary condition $u_n(1) = \beta$ is by enforcing this algebraically. Consider the energy functional given by

$$\mathcal{J}(v) = \frac{1}{2}\int_0^1 a(x)(v'(x))^2 dx - \int_0^1 f(x)v(x)dx.$$

Let $\mathbf{d} = \boldsymbol{\Sigma}(1)$ and consider the Lagrangian function

$$\begin{aligned}\mathcal{L}(\mathbf{c}, \mathbf{b}, \lambda) &= \mathcal{J}(u_n(x; \mathbf{c}, \mathbf{b})) + \lambda(u_n(1) - \beta) \\ &= \mathcal{J}(u_n(x; \mathbf{c}, \mathbf{b})) + \lambda(\mathbf{d}^T\mathbf{c} + \alpha - \beta),\end{aligned}$$

hence, if u_n minimizes $\mathcal{J}(v)$ for $v \in \mathcal{M}_n(I) \cap \{u_n(0) = \alpha\}$, subject to the constraint $u_n(1) = \beta$, then by the KKT conditions:

$$\nabla_{\mathbf{c}}\mathcal{L}\left(u_n\right) = \mathbf{0}, \quad \frac{\partial}{\partial\lambda}\mathcal{L}\left(u_n\right) = 0 \quad \text{and} \quad \nabla_{\mathbf{b}}\mathcal{L}\left(u_n\right) = \mathbf{0}. \tag{A.1}$$

The first two equations in (A.1) can be written as

$$\begin{pmatrix} A(\mathbf{b}) & \mathbf{d} \\ \mathbf{d}^T & 0 \end{pmatrix}\begin{pmatrix} \mathbf{c} \\ \lambda \end{pmatrix} = \begin{pmatrix} \mathbf{f}(\mathbf{b}) \\ \beta - \alpha \end{pmatrix}, \tag{A.2}$$

which can be solved efficiently, by writing the matrix on the left-hand side as

$$\begin{pmatrix} A(\mathbf{b}) & \mathbf{d} \\ \mathbf{d}^T & 0 \end{pmatrix} = \begin{pmatrix} I & 0 \\ \mathbf{d}^T A(\mathbf{b})^{-1} & 1 \end{pmatrix}\begin{pmatrix} A(\mathbf{b}) & \mathbf{d} \\ 0 & -\mathbf{d}^T A(\mathbf{b})^{-1}\mathbf{d} \end{pmatrix},$$

and since

$$\begin{pmatrix} I & 0 \\ \mathbf{d}^T A(\mathbf{b})^{-1} & 1 \end{pmatrix}^{-1} = \begin{pmatrix} I & 0 \\ -\mathbf{d}^T A(\mathbf{b})^{-1} & 1 \end{pmatrix},$$

it follows that (A.2) is equivalent to

$$\begin{pmatrix} A(\mathbf{b}) & \mathbf{d} \\ 0 & -\mathbf{d}^T A(\mathbf{b})^{-1}\mathbf{d} \end{pmatrix}\begin{pmatrix} \mathbf{c} \\ \lambda \end{pmatrix} = \begin{pmatrix} I & 0 \\ -\mathbf{d}^T A(\mathbf{b})^{-1} & 1 \end{pmatrix}\begin{pmatrix} \mathbf{f}(\mathbf{b}) \\ \beta - \alpha \end{pmatrix}$$

which can be solved by finding λ first and then solving for $\mathbf{c}$. According to (5.1), the computational cost is $\mathcal{O}(n)$.

On the other hand, for the non-linear parameters $\mathbf{b}$, the Hessian matrix $\nabla^2_{\mathbf{b}}\mathcal{L}(u_n) = \mathbf{D}(\hat{\mathbf{c}})\mathbf{D}(\mathbf{g})$, where $\mathbf{D}(\mathbf{g})$ is defined in (5.3). Therefore, as described in Section 5, we solve (3.1) iteratively, alternating between solving exactly for $(\mathbf{c}^T, \lambda)^T$ and performing a Newton step for $\mathbf{b}$.

Data availability

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

References

[1] M. Ainsworth, Y. Shin, Plateau phenomenon in gradient descent training of ReLU networks: explanation, quantification and avoidance, SIAM J. Sci. Comput. 43 (2020) A3438–A3468.
[2] M. Ainsworth, Y. Shin, Active neuron least squares: a training method for multivariate rectified neural networks, SIAM J. Sci. Comput. 44 (4) (2022) A2253–A2275.
[3] D.L. Barrow, C.K. Chui, P.W. Smith, J.D. Ward, Unicity of best mean approximation by second order splines with variable knots, Math. Comput. 32 (144) (1978) 1131–1143.
[4] J. Berg, K. Nyström, A unified deep artificial neural network approach to partial differential equations in complex geometries, Neurocomputing 317 (2018) 28–41.
[5] J.H. Bramble, A.H. Schatz, Rayleigh-Ritz-Galerkin-methods for Dirichlet's problem using subspaces without boundary conditions, Commun. Pure Appl. Math. 23 (1970) 653–675.
[6] H.G. Burchard, Splines (with optimal knots) are better, Appl. Anal. 3 (1974) 309–319.
[7] Z. Cai, J. Chen, M. Liu, Least-squares ReLU neural network (LSNN) method for linear advection-reaction equation, J. Comput. Phys. 443 (2021) 110514.

[8] Z. Cai, J. Chen, M. Liu, Least-squares neural network (LSNN) method for scalar nonlinear hyperbolic conservation laws: discrete divergence operator, J. Comput. Appl. Math. 433 (2023) 115298.
[9] Z. Cai, J. Chen, M. Liu, Self-adaptive deep neural network: numerical approximation to functions and PDEs, J. Comput. Phys. 455 (2022) 111021.
[10] Z. Cai, J. Chen, M. Liu, Xinyu Liu, Deep least-squares methods: an unsupervised learning-based numerical method for solving elliptic PDEs, J. Comput. Phys. 420 (2020) 109707.
[11] Z. Cai, T. Ding, M. Liu, X. Liu, J. Xia, Matrix analysis for shallow relu neural network least-squares approximations, Manuscript.
[12] Z. Cai, T. Ding, M. Liu, X. Liu, J. Xia, A structure-guided Gauss-Newton method for shallow neural network, arXiv:2404.05064 [cs.LG], 2024.
[13] Z. Cai, T. Ding, M. Liu, X. Liu, J. Xia, A structure-guided Gauss-Newton method for shallow ReLU neural network, arXiv:2404.05064v1 [cs.LG], 2024.
[14] Z. Cai, S. Zhang, Recovery-based error estimators for interface problems: conforming linear elements, SIAM J. Numer. Anal. 47 (3) (2009) 2132–2156.
[15] I. Daubechies, R. DeVore, S. Foucart, B. Hanin, G. Petrova, Nonlinear approximation and (deep) ReLU networks, Constr. Approx. 55 (1) (February 2022) 127–172.
[16] W.C. Davidon, Variable metric method for minimization, SIAM J. Optim. 1 (1) (1991) 1–17.
[17] C. deBoor, J.R. Rice, Least squares cubic spline approximation II: variable knots, Department of Computer Sciences Purdue University, 1968, CSD TR 21.
[18] M.W.M.G. Dissanayake, N. Phan-Thien, Neural network based approximations for solving partial differential equations, Commun. Numer. Methods Eng. 10 (3) (1994) 195–201.
[19] T. Dockhorn, A discussion on solving partial differential equations using neural networks, arXiv:1904.07200, 2019.
[20] W. E, B. Yu, The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems, Commun. Math. Stat. 6 (1) (March 2018) 1–12.
[21] R. Fletcher, M.J.D. Powell, A rapidly convergent descent method for minimization, Comput. J. 6 (2) (1963) 163–168.
[22] G.H. Golub, V. Pereyra, The differentiation of pseudo-inverses and nonlinear least squares problems whose variables separate, SIAM J. Numer. Anal. 10 (2) (1973) 413–432.
[23] D.L.B. Jupp, Approximation to data by splines with free knots, SIAM J. Numer. Anal. 15 (2) (1978) 328–343.
[24] I.E. Lagaris, A. Likas, D.I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, IEEE Trans. Neural Netw. 9 (5) (1998) 987–1000.
[25] I.E. Lagaris, A.C. Likas, D.G. Papageorgiou, Neural-network methods for boundary value problems with irregular boundaries, IEEE Trans. Neural Netw. 11 (5) (2000) 1041–1049.
[26] K. Levenberg, A method for the solution of certain non-linear problems in least squares, Q. Appl. Math. 2 (1944) 164–168.
[27] M. Liu, Z. Cai, Adaptive two-layer ReLU neural network: II. Ritz approximation to elliptic pdes, Comput. Math. Appl. 113 (May 2022) 103–116.
[28] M. Liu, Z. Cai, J. Chen, Adaptive two-layer ReLU neural network: I. Best least-squares approximation, Comput. Math. Appl. 113 (May 2022) 34–44.
[29] P.D. Loach, A.J. Wathen, On the best least squares approximation of continuous functions using linear splines with free knots, IMA J. Numer. Anal. 11 (3) (July 1991) 393–409.
[30] D.W. Marquardt, An algorithm for least-squares estimation of nonlinear parameters, J. Soc. Ind. Appl. Math. 11 (2) (1963) 431–441.
[31] J. Nocedal, S.J. Wright, Numerical Optimization, 2nd edition, Springer, New York, NY, USA, 2006.
[32] M.R. Osborne, Some special nonlinear least squares problems, SIAM J. Numer. Anal. 12 (1975) 571–592.
[33] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707.
[34] L. Schumaker, Spline Functions: Basic Theory, Wiley, New York, 1981.
[35] J. Sirignano, K. Spiliopoulos, DGM: a deep learning algorithm for solving partial differential equations, J. Comput. Phys. 375 (2018) 1339–1364.