A SUBSPACE CORRECTION METHOD FOR RELU NEURAL NETWORKS FOR SOLVING PDES

Anonymous authors
Paper under double-blind review

ABSTRACT

In this paper, we propose a novel algorithm called Neuron-wise Parallel Subspace Correction Method (NPSC) for training ReLU neural networks for numerical solution of partial differential equations (PDEs). Despite of extremely extensive research activities in applying neural networks for numerical PDEs, there is still a serious lack of training algorithms that can be used to obtain approximation with adequate accuracy. Based on recent results on the spectral properties of linear layers and landscape analysis for single neuron problems, we develop a special type of subspace correction method that deals with the linear layer and each neuron in the nonlinear layer separately. An optimal preconditioner that resolves the ill-conditioning of the linear layer is presented, so that the linear layer is trained in a uniform number of iterations with respect to the number of neurons. In each single neuron problem, a good local minimum is found by a superlinearly convergent algorithm, avoiding regions where the loss function is flat. Performance of the proposed method is demonstrated through numerical experiments for function approximation problems and PDEs.

1 INTRODUCTION

Neural networks, thanks to the universal approximation property (Cybenko, 1989; Pinkus, 1999), are promising tools for numerical solutions of partial differential equations (PDEs). Moreover, it was shown in Siegel & Xu (2022b) that the approximation properties of neural networks have higher asymptotic approximation rates than that of traditional numerical algorithms such as finite element methods. Such powerful approximation properties, however, can hardly be observed in numerical experiments even in simple tasks of function approximation. It was shown in Hong et al. (2022) that conventional training algorithms such as gradient descent converge too slowly to an accurate solution even for a very simple function. Therefore, applying neural networks to solutions of PDEs must require novel training algorithms, different from the conventional ones for regression, image classification, and pattern recognition tasks. In this viewpoint, there are many works on designing and analyzing training algorithms for neural networks to try to narrow the gap between the theoretical optimum and training results. A hybrid least squares/gradient descent method was proposed in Cyr et al. (2020). Plateau phenomena (Park et al., 2000) occurring in the gradient dynamics of ReLU shallow neural networks were analyzed in a rigorous manner in Ainsworth & Shin (2021), and a training algorithm called active neuron least squares was designed to avoid plateau phenomena in Ainsworth & Shin (2022). As a completely different approach, the orthogonal greedy algorithm was shown to achieve an optimal convergence rate (Siegel & Xu, 2022c) with respect to the number of neurons when it is applied to certain shallow neural networks in Siegel & Xu (2022a).

A surprising recent result on training of neural networks is that optimizing the linear layer parameters in a neural network is one bottleneck that leads to a large number of iterations of a gradient-based method. More precisely, it was proven in Hong et al. (2022) that optimizing the linear layer parameters in a ReLU shallow neural network requires solving a very ill-conditioned linear problem in general. This work motivates us to separately design efficient solvers for the outer linear layer and the inner nonlinear layer, respectively, and train them alternately.

Meanwhile, there have been some encouraging works on learning a single neuron, a more simplified model than a nonlinear layer. Convergence analyses of gradient methods for the single neuron problem with ReLU activation were presented in Soltanolkotabi (2017); Tian (2017); Yehudai & Ohad

(2020) under various assumptions on input distributions. The case of a single ReLU neuron with bias was analyzed in Vardi et al. (2021). All of these results show that the global convergence of gradient methods for the single ReLU neuron problem can be attained under certain conditions. Hence, learning a single neuron can be regarded as a much more hopeful task than learning a nonlinear layer with multiple neurons.

In this paper, we use the idea of subspace correction (Xu, 1992), which is well-known in the fields of numerical analysis and scientific computing. Subspace correction methods provide a unified framework to design and analyze many modern iterative numerical methods such as block coordinate descent, multigrid, and domain decomposition methods. In subspace correction methods, the solution space of a target problem is decomposed into a sum of subspaces, and local problems defined on subspaces are solved separately in either sequential or parallel manner. Then a solution of the whole problem is constructed by assembling the solutions of the local problems appropriately. Mathematical theory of subspace correction methods for convex optimization problems can be found in Park (2020); Tai & Xu (2002). Subspace correction methods have been successfully applied to various nonlinear optimization problems appearing in engineering fields (see, e.g., Badea & Krause (2012); Lee & Park (2019)). In particular, there have been successful applications of block coordinate descent methods (Wright, 2015) for training of neural networks (Zeng et al., 2019; Zhang & Brand, 2017). Therefore, we expect that the idea of the subspace correction method is also suitable for training of ReLU shallow neural networks.

Inspired by the above works, we propose a new training algorithm called Neuron-wise Parallel Subspace Correction Method (NPSC), which is a special type of subspace correction method for a ReLU shallow neural network. The proposed method utilizes a space decomposition for the linear layer and each individual neuron. In the first step of each epoch of the NPSC, the linear layer is fully trained by solving a relevant linear system. This resolves ill-conditioning of the linear layer. We design an optimal preconditioner for the linear layer problem in one-dimension, based on the relation between ReLU neural networks and linear finite elements investigated in He et al. (2020); Hong et al. (2022). In the second step, we train each single neuron in parallel, taking advantages of better convergence properties of learning a single neuron explained above. Each neuron is trained by a superlinearly convergent algorithm (Marquardt, 1963). Finally, an update for the parameters in the nonlinear layer is computed by assembling the corrections obtained in the local problems for each neuron. Due to the intrinsic parallel structure, NPSC is suitable for parallel computation on distributed memory computers. We present applications of NPSC to various function approximation problem and PDEs, and numerically verify that it outperforms conventional training algorithms.

The rest of this paper is organized as follows. In Section 2, we summarize key features of ReLU shallow networks and describe our model problem. The proposed NPSC is presented in Section 3. Applications of NPSC to various function approximation problems and PDEs are presented in Section 4 to demonstrate the effectiveness of NPSC. We conclude the paper with remarks in Section 5.

2 Model Problem

We consider a shallow neural network with n neurons and d-dimensional inputs $x \in \mathbb{R}^d$ given by

$$u(x;\theta) = \sum_{i=1}^{n} a_i \sigma(\omega_i \cdot x + b_i), \quad \theta = \{a, \omega, b\} = \left\{(a_i)_{i=1}^n, (\omega_i)_{i=1}^n, (b_i)_{i=1}^n\right\}, \tag{1}$$

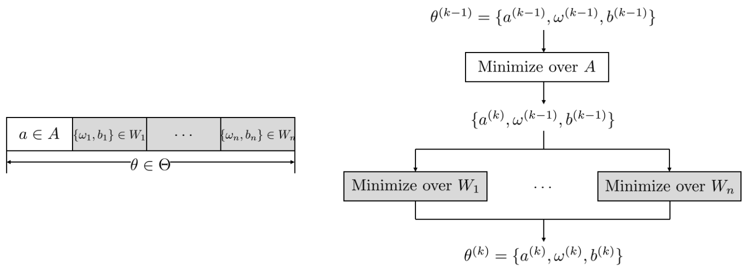
where θ is the collection of parameters consisting of $a_i \in \mathbb{R}$, $\omega_i \in \mathbb{R}^d$, and $b_i \in \mathbb{R}$ for $1 \leq i \leq n$, and $\sigma\colon \mathbb{R} \to \mathbb{R}$ is a ReLU activation function defined by $\sigma(x) = \max\{0, x\}$. The neural network (1) possesses total $(d+2)n$ parameters.

Let $\Omega \subset \mathbb{R}^d$ be a bounded domain and let $f \in L^2(\Omega)$. As a model problem, we consider the following minimization problem:

$$\min_{\theta} \left\{ E(\theta) := \frac{1}{2} a\left(u(x;\theta), u(x;\theta)\right) - \int_\Omega f(x) u(x;\theta)\, dx \right\}, \tag{2}$$

where $a(\cdot,\cdot)$ is a continuous, coercive, and symmetric bilinear form defined on a Hilbert space $V \subset L^2(\Omega)$. The problem (2) can be seen as the Galerkin approximation of the problem

$$\min_{u \in V} \left\{ \frac{1}{2} a(u,u) - \int_\Omega f u \, dx \right\},$$


$a \in A$
$\{\omega_1, b_1\} \in W_1$
$\cdots$
$\{\omega_n, b_n\} \in W_n$
$\theta \in \Theta$
$\theta^{(k-1)} = \{a^{(k-1)}, \omega^{(k-1)}, b^{(k-1)}\}$
Minimize over A
$\{a^{(k)}, \omega^{(k-1)}, b^{(k-1)}\}$
Minimize over W_1
$\cdots$
Minimize over W_n
$\theta^{(k)} = \{a^{(k)}, \omega^{(k)}, b^{(k)}\}$

(a) (b)

Figure 1: **(a)** Space decomposition of the solution space Θ of (2) into subspaces A and $\{W_i\}_{i=1}^n$. **(b)** Subspace correction procedure of NPSC.

which can represent various elliptic PDEs (E & Yu, 2018; Xu, 2020), on the space

$$\Sigma_n^1 = \left\{ v(x) = \sum_{i=1}^n a_i \sigma(\omega_i \cdot x + b_i) : a_i \in \mathbb{R},\ \omega_i \in \mathbb{R}^d,\ b_i \in \mathbb{R} \right\}.$$

The space Σ_n^1 enjoys the universal approximation property (Cybenko, 1989; Pinkus, 1999), namely, any function with sufficient regularity can be uniformly approximated by functions in Σ_n^1. Recent results on the approximation property of Σ_n^1 can be found in Siegel & Xu (2022c).

3 Neuron-wise Parallel Subspace Correction Method (NPSC)

In this section, we introduce NPSC, a subspace correction method (Xu, 1992) that deals with each neuron in parallel for the problem (2). We first present a space decomposition for NPSC.

Let $\Theta = \mathbb{R}^{(d+2)n}$ denote the solution space of (2), i.e., the space for the parameter θ. It admits a natural decomposition $\Theta = A \oplus W$, where $A = \mathbb{R}^n$ and $W = \mathbb{R}^{(d+1)n}$ are the spaces for a and $\{\omega, b\}$, respectively, and $\oplus$ denotes direct sum. Since any $\{\omega, b\} \in W$ consists of the parameters $(\{\omega_i, b_i\})_{i=1}^n$ from n neurons, W can be further decomposed as $W = \bigoplus_{i=1}^n W_i$, where $W_i = \mathbb{R}^{d+1}$ is the space for $\{\omega_i, b_i\}$. Finally, we have the following space decomposition of Θ:

$$\Theta = A \oplus \bigoplus_{i=1}^n W_i. \tag{3}$$

A graphical description for the space decomposition (3) is presented in Figure 1(a).

NPSC, our proposed method, is presented in Algorithm 1. It is a subspace correction method (Xu, 1992) for (2) based on the space decomposition (3). At the kth epoch, NPSC updates the parameter a first by minimizing the loss function with respect to a, then it updates the parameters in each neuron by minimizing E with respect to $\{\omega_i, b_i\}$ in parallel. The update of $\{\omega_i, b_i\}$ is relaxed by an appropriate learning rate $\tau_k > 0$ as in the existing parallel subspace correction methods for optimization problems (Lee & Park, 2019; Tai & Xu, 2002). The overall structure of NPSC is depicted in Figure 1(b). In the remainder of this section, we discuss parts of Algorithm 1 in detail.

3.1 Adjustment of Parameters

We call that a neuron with parameter $\{\omega_i, b_i\}$ is dead if $\sigma(\omega_i \cdot x + b_i)$ vanishes almost everywhere (a.e.) on the domain Ω. In Vardi et al. (2021, Theorem 3.1), it was shown that a neuron is initialized as dead with probability close to half if we employ a usual random initialization scheme. A similar observation was made in Lu et al. (2020). Such a dead neuron makes the functions $\{\sigma(\omega_i \cdot x + b_i)\}_{i=1}^n$ linearly dependent, i.e., there exist $a_i \in \mathbb{R}$, $1 \leq i \leq n$, not all zero, such that $\sum_{i=1}^n a_i \sigma(\omega_i \cdot x + b_i) = 0$. Meanwhile, the linear dependence may occur by a combination of several non-dead neurons. Theorem 1 characterizes some situations when $\{\sigma(\omega_i \cdot x + b_i)\}_{i=1}^n$ becomes linearly dependent; see Appendix A for a proof.

Theorem 1. *Consider the shallow neural network* (1). *The followings hold.*

Algorithm 1 Neuron-wise Parallel Subspace Correction Method (NPSC) for (2)

```
Choose an initial guess $\theta^{(0)} = \{a^{(0)}, \omega^{(0)}, b^{(0)}\}$ and an initial learning rate $\tau_0 = 1$.
for $k = 1, \dots, T$ do
    Adjust $\{\omega^{(k-1)}, b^{(k-1)}\}$ to avoid linear dependence of the neurons (see Algorithm 2).
    $a^{(k)} \in \arg\min_{a \in \mathbb{R}^n} E(\{a, \omega^{(k-1)}, b^{(k-1)}\})$ (see (6))
    for $i = 1, \dots, n$ in parallel do
        $\{\omega_i^{(k-\frac{1}{2})}, b_i^{(k-\frac{1}{2})}\} \in \underset{\{\omega_i, b_i\} \in \mathbb{R}^{d+1}}{\arg\min} E\left(\left\{a^{(k)}, \omega_i \oplus \bigoplus_{j=1, j\neq i}^{n} \omega_j^{(k-1)}, b_i \oplus \bigoplus_{j=1, j\neq i}^{n} b_j^{(k-1)}\right\}\right)$    (4)
    end for
    Determine the learning rate $\tau_k$ by backtracking (see Algorithm 3).
    for $i = 1, \dots, n$ in parallel do
        $\omega_i^{(k)} = (1-\tau_k)\omega_i^{(k-1)} + \tau_k \omega_i^{(k-\frac{1}{2})}$
        $b_i^{(k)} = (1-\tau_k) b_i^{(k-1)} + \tau_k b_i^{(k-\frac{1}{2})}$
    end for
end for
```

1. *If a neuron satisfies* $b_i \leq -\operatorname{ess\,sup}_{x \in \Omega} \omega_i \cdot x$*, then it is dead.*
2. *If there are more than* $d+1$ *neurons such that* $b_i \geq -\operatorname{ess\,inf}_{x\in\Omega} \omega_i \cdot x$*, then the functions* $\sigma(\omega_i \cdot x + b_i)$ *corresponding to these neurons are linearly dependent in* Ω*.*

Linear dependence of the functions $\{\sigma(\omega_i \cdot x + b_i)\}_{i=1}^n$ in the neural network (1) should be avoided because it means that several neurons are redundant and they are not utilized to improve the approximability of (1). Theorem 1 motivates us to consider an adjustment procedure for $\{\omega^{(k-1)}, b^{(k-1)}\}$ when we enter the kth iteration of NPSC so that linear dependence does not occur. Algorithm 2 summarizes the adjustment procedure for a given $\{\omega, b\} \in W$.

Algorithm 2 Adjustment for $\{\omega, b\}$ in Algorithm 1

```
for $i = 1, \dots, n$ in parallel do
    Set $\omega_i \leftarrow \frac{\omega_i}{|\omega_i|}$ and $b_i \leftarrow \frac{b_i}{|\omega_i|}$.
    if $b_i \notin \left(-\max_{x\in\Omega} \omega_i \cdot x, -\min_{x\in\Omega} \omega_i \cdot x\right)$ then
        Reset $b_i \in \mathbb{R}$ such that $b_i \sim \text{Uniform}\left(-\max_{x\in\Omega} \omega_i \cdot x, -\min_{x\in\Omega} \omega_i \cdot x\right)$.
    end if
end for
```

Since ω_i determines the direction of a function $\sigma(\omega_i \cdot x + b_i)$, we may normalize $\{\omega_i, b_i\}$ so that $|\omega_i| = 1$ in Algorithm 2. Then, for each b_i that is not on the interval between the minimum and maximum of $\omega_i \cdot x$, we relocate it on the interval. Since the extrema in Ω agree with the essential extrema under mild conditions on Ω, this relocation step lets the neurons avoid the scenario of linear dependence described in Theorem 1.

In Algorithm 2, evaluations of the extrema of $\omega_i \cdot x$ are simple linear programs, so that they can be done efficiently by conventional algorithms for linear programming (Nocedal & Wright, 2006). In particular, if Ω is a polyhedral domain, then we can evaluate the extrema of $\omega_i \cdot x$ by hands; we simply compute $\omega_i \cdot x$ at all the vertices of Ω and take the extrema among them.

3.2 a-MINIMIZATION PROBLEMS

In Algorithm 1, a-minimization problems have the general form

$$\min_{a\in\mathbb{R}^n} E(\{a,\omega,b\}), \tag{5}$$

where $\{\omega,b\}\in W$. The first-order optimality condition for (5) reads as

$$Ka=\beta, \tag{6}$$

where $K\in\mathbb{R}^{n\times n}$ and $\beta\in\mathbb{R}^n$ are given by

$$K_{ij}=a\left(\sigma(\omega_j\cdot x+b_j),\sigma(\omega_i\cdot x+b_i)\right),\ \beta_i=\int_\Omega f(x)\sigma(\omega_i\cdot x+b_i)\,dx,\quad 1\le i,j\le n. \tag{7}$$

Assuming the basis functions $\{\sigma(\omega_i\cdot x+b_i)\}_{i=1}^n$ are linearly independent in Ω, the matrix K is symmetric and positive definite. Hence, the system (6) can be solved efficiently by the conjugate gradient method, and the efficiency can be improved with an appropriate preconditioner.

Theorem 2 presents an optimal preconditioner in one-dimension. While it was proven in Hong et al. (2022) that the matrix K is usually ill-conditioned, the preconditioner in Theorem 2 can resolve this issue. We refer readers to Appendix A for a proof of Theorem 2. Computational aspects and numerical results regarding the preconditioner are presented in Appendix B.

Theorem 2. *Let $\Omega=[0,1]\subset\mathbb{R}$. Assume that $0\le x_1<x_2<\cdots<x_n\le 1$ under an appropriate reordering, where $x_i=-b_i/\omega_i$ for $1\le i\le n$. Then we can find a matrix $P\in\mathbb{R}^{n\times n}$ explicitly such that the condition number $\kappa(PK)$ of a preconditioned operator PK has an upper bound independent of n, ω, and b, i.e., $\kappa(PK)=O(1)$.*

3.3 $\{\omega_i,b_i\}$-MINIMIZATION PROBLEMS

Training the nonlinear layer is challenging because of its nonconvexity. Moreover, it was shown in Lu et al. (2020) that training of ReLU networks usually suffers from dead neurons. In the following, we claim that training each neuron in the nonlinear layer separately has an advantage that some "good" local minima that avoid dead neurons can be found (cf. Vardi et al. (2021)).

We consider the $\{\omega_i,b_i\}$-minimization problem (4) in Algorithm 1 for a fixed i. We may assume that $a_i\neq 0$; otherwise, the minimization problem becomes trivial. Under some elementary manipulations, (4) is rewritten as

$$\min_{\{\omega_i,b_i\}\in\mathbb{R}^{d+1}}\left\{E_i(\omega_i,b_i):=\frac{1}{2}a\left(\sigma(\omega_i\cdot x+b_i),\sigma(\omega_i\cdot x+b_i)\right)-\int_\Omega F(x)\sigma(\omega_i\cdot x+b_i)\,dx\right\}, \tag{8}$$

where $F\in L^2(\Omega)$ is a function determined in terms of f, $a^{(k)}$, $\omega_j^{(k-1)}$, and $b_j^{(k-1)}$ for $j\neq i$.

A notable advantage of single neuron training (8) is that we can easily avoid regions where the loss function is flat. The following theorem says that all the critical points of (8) on the flat regions make the neuron dead, and that all the critical points outside the flat regions have less loss values than those on the flat regions. A proof of Theorem 3 can be found in Appendix A.

Theorem 3. *Consider the single neuron problem (8). The followings hold:*

1. *If $\{\omega_i^*,b_i^*\}\in\mathbb{R}^{d+1}$ is a critical point of (8) such that $\sigma(\omega_i^*\cdot x+b_i^*)$ vanishes on Ω, then we have $E_i(\omega_i^*,b_i^*)=E_i(\mathbf{0})$.*
2. *If $\{\omega_i^*,b_i^*\}\in\mathbb{R}^{d+1}$ is a critical point of (8) such that $\sigma(\omega_i^*\cdot x+b_i^*)$ does not vanish on a subset of Ω with nonzero measure, then we have $E_i(\omega_i^*,b_i^*)<E_i(\mathbf{0})$.*

Theorem 3 implies that, if we choose an initial guess $\{\omega_i^{(0)},b_i^{(0)}\}$ such that $E_i(\omega_i^{(0)},b_i^{(0)})<E_i(\mathbf{0})$ for a monotone training algorithm, then the algorithm converges to a good local minimum that prevents the neuron to be dead. That is, each $\{\omega_i,b_i\}$-minimization problem of NPSC finds a good local solution. Note that the condition $E_i(\omega_i^{(0)},b_i^{(0)})<E_i(\mathbf{0})$ is satisfied by a random initialization of $\{\omega_i^{(0)},b_i^{(0)}\}$ around $\mathbf{0}$ under some mild conditions; see Vardi et al. (2021, Theorem 5.4).

Various optimization algorithms including first- and second-order methods can be used to solve (8). Among them, a good option is the Levenberg–Marquardt algorithm (Marquardt, 1963); PDEs usually arise from physics, so that the dimension $d+1$ of $\{\omega_i, b_i\}$ is not very big in general. The major computational cost of each iteration of the Levenberg–Marquardt algorithm is to solve a linear system with $d+1$ unknowns; it is not time-consuming when $d+1$ is small. The Levenberg–Marquardt algorithm does not require explicit assembly of the Hessian, and it converges to a local minimum with the superlinear convergence rate (Yamashita & Fukushima, 2001), which is much faster than that of the first-order methods.

3.4 Backtracking for Learning Rates

Recently, it was shown in Park (2022) that parallel subspace correction methods for convex optimization problems can be accelerated by adopting a backtracking scheme. Hence, it is natural to utilize a backtracking scheme to find a suitable value for the learning rate τ_k in Algorithm 1. Due to the nonconvexity of the loss function E of (2), we are not able to adopt the backtracking scheme proposed in Park (2022) directly to Algorithm 1. Moreover, since ReLU activation in (2) makes E nonsmooth, conventional backtracking schemes such as Calatroni & Chambolle (2019); Scheinberg et al. (2014) are not applicable. A simple but effective backtracking scheme for finding τ_k is presented in Algorithm 3. By allowing adaptive increase and decrease of τ_k along the epochs, the convergence rate of NPSC is improved.

Algorithm 3 Backtracking scheme to find τ_k in Algorithm 1

```
Choose a minimum learning rate $\tau_{\min} = 10^{-12}$.
$\tau_k \leftarrow 2\tau_{k-1}$
repeat
    for $i = 1, \dots, n$ in parallel do
        $\hat{\omega}_i = (1-\tau_k)\omega_i^{(k-1)} + \tau_k \omega_i^{(k-\frac{1}{2})}$
        $\hat{b}_i = (1-\tau_k) b_i^{(k-1)} + \tau_k b_i^{(k-\frac{1}{2})}$
    end for
    if $E(\{a^{(k)}, \hat{\omega}, \hat{b}\}) > E(\{a^{(k)}, \omega^{(k-1)}, b^{(k-1)}\})$ then
        $\tau_k \leftarrow \tau_k / 2$
    end if
until $E(\{a^{(k)}, \hat{\omega}, \hat{b}\}) \leq E(\{a^{(k)}, \omega^{(k-1)}, b^{(k-1)}\})$ or $\tau_k < \tau_{\min}$
```

4 Numerical Results

In this section, we present numerical results of NPSC applied to various function approximation problems and PDEs of the form (2). The following algorithms are compared with NPSC in our numerical experiments: gradient descent (GD), Adam (Kingma & Ba, 2015), hybrid least-squares/gradient descent (LSGD) (Cyr et al., 2020), and the proposed NPSC. All the algorithms were implemented in ANSI C with OpenMPI compiled by Intel C++ Compiler. They were executed on a computer cluster equipped with multiple Intel Xeon SP-6148 CPUs (2.4GHz, 20C) and the operating system CentOS 7.4 64bit.

In all experiments, we use the He initialization (He et al., 2015) to initialize the parameters of (2). That is, we set $a_i \sim N(0, 2/n)$, $\omega_i \sim (N(0, 2/d))^d$, and $b_i \sim (N(0, 2/d))$ in (2). All the numerical results presented in this section are averaged over 10 random initializations. As shown in Appendix C, the performances of conventional training algorithms can be improved by utilizing the backtracking scheme presented in Algorithm 3. Hence, for GD, Adam, and LSGD, we employ Algorithm 3 to find learning rates. In LSGD and NPSC, a-minimization problems (6) are solved by the preconditioned conjugate gradient method with the preconditioner P in Theorem 2 and the stop criterion

$$\frac{\|Ka^{(k)} - \beta\|_{\ell^2}}{\|Ka^{(0)} - \beta\|_{\ell^2}} < 10^{-10}. \tag{9}$$

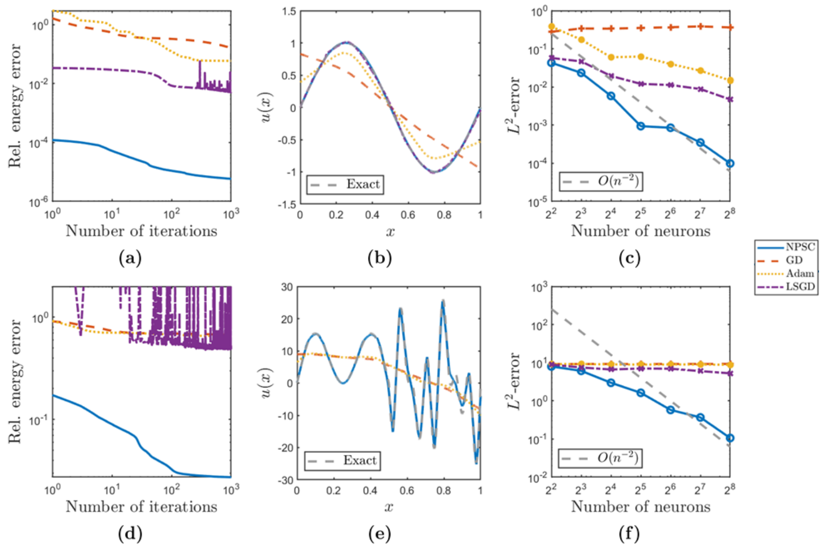


Figure 2: Numerical results for the function approximation problems **(a–c)** (11) and **(d–f)** (12). **(a, d)** Decay of the relative energy error $\frac{E(\theta^{(k)})-E^*}{|E^*|}$ in various training algorithms ($n = 2^5$). **(b, e)** Exact solution and its approximations ($n = 2^5$, 10^3 epochs). **(c, f)** L^2-errors with respect to the number of neurons (10^3 epochs). The dotted line represents the theoretical optimal approximation rate.

Finally, $\{\omega_i, b_i\}$-minimization problems (8) in NPSC are solved by the Levenberg–Marquardt algorithm (Marquardt, 1963) with the stop criterion

$$\frac{|E_i^{(n-1)} - E_i^{(n)}|}{|E_i^{(n)}|} < 10^{-10}.$$

MPI parallelization is applied to NPSC in a way that each $\{\omega_i, b_i\}$-minimization problem is assigned to a single processor and solved in parallel.

4.1 L^2-FUNCTION APPROXIMATION PROBLEMS

If we set

$$V = L^2(\Omega), \quad a(u,v) = \int_\Omega uv\,dx, \quad u,v \in V \tag{10}$$

in (2), then we obtain the L^2-function approximation problem, which is the most elementary instance of (2). That is, the solution of (2) in this case is the best L^2-approximation of f found in Σ_n^1. As our first example, we consider L^2-approximation (10) for a sine function; we set

$$\Omega = [0,1] \subset \mathbb{R}, \quad u(x) = \sin 2\pi x \tag{11}$$

in (2). For numerical integration, we use the trapezoidal rule on 10,000 uniformly sampled points; see Appendix D. Figure 2(a) plots the relative energy errors $\frac{E(\theta^{(k)})-E^*}{|E^*|}$ obtained by NPSC, GD, Adam, and LSGD per epoch, where E^* is the loss corresponding to the exact solution. It is clear from the loss decay that NPSC outperforms all the other methods. The exact solution of (11) and its approximations obtained by 10^3 epochs of the training algorithms with 2^5 neurons are depicted in Figure 2(b). One can readily observe that the NPSC result is most accurate among the approximations. The L^2-errors between the exact solution and its approximations for various numbers of neurons are presented in Figure 2(c). Only the NPSC result seems to be comparable to $O(n^{-2})$, the theoretical optimal rate derived in Siegel & Xu (2022c).

The second example is the following highly oscillatory instance of (10):

$$\Omega = [0,1] \subset \mathbb{R}, \quad u(x) = \begin{cases} 10\,(\sin 2\pi x + \sin 6\pi x)\,, & \text{if } 0 \le x < \frac{1}{2}, \\ 10\,(\sin 8\pi x + \sin 18\pi x + \sin 26\pi x)\,, & \text{if } \frac{1}{2} \le x \le 1, \end{cases} \tag{12}$$

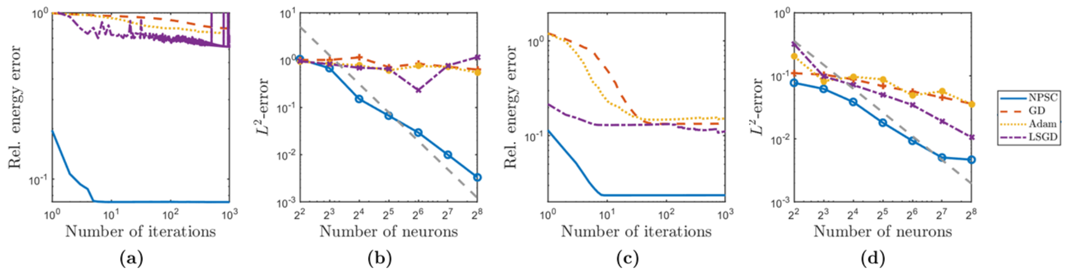


Figure 3: Numerical results for the elliptic PDEs **(a, b)** (14) and **(c, d)** (15). **(a, c)** Decay of the relative energy error $\frac{E(\theta^{(k)})-E^*}{|E^*|}$ in various training algorithms ($n = 2^5$). **(b, d)** L^2-errors with respect to the number of neurons (10^3 epochs). The dotted line represents the theoretical optimal approximation rate.

in which a similar problem appeared in Cai et al. (2020). We note that it was demonstrated in Krishnapriyan et al. (2021) that training for complex functions like (12) is a much harder task than training for simple functions. Same as in (11), we use the trapezoidal rule on 10,000 uniformly sampled points for numerical integration. Figures 2(d–f) present numerical results for the problem (12). In Figure 2(d), NPSC shows stable decay of the loss while the losses of GD and Adam are stagnant along the epochs and that of LSGD blows up at in the beginning epochs and varies very rapidly. In Figure 2(e), one can observe that the NPSC result captures both low- and high-frequency parts of the target function well, while the other ones captures the low-frequency part only (Xu et al., 2020). Moreover, as shown in Figure 2(f), the L^2-error of the NPSC decreases when the number of neurons increases, while those of the other algorithms seem to stagnate. This highlights the robustness of NPSC for complex problems.

4.2 Elliptic Partial Differential Equations

Next, we consider the case

$$V = H^1(\Omega), \quad a(u,v) = \int_\Omega (\nabla u \cdot \nabla v + uv)\, dx, \quad u, v \in V \tag{13}$$

in the problem (2), where $H^1(\Omega)$ is the usual Sobolev space consisting of $L^2(\Omega)$-functions with square-integrable gradients. It is well-known that (2) becomes the Galerkin approximation of the weak formulation of the following elliptic PDE on Σ_n^1 (Xu, 2020):

$$-\Delta u + u = f \quad \text{in } \Omega, \quad \frac{\partial u}{\partial n} = 0 \quad \text{on } \partial\Omega,$$

where $\partial u/\partial n$ denotes the normal derivative of u to $\partial\Omega$. Hence, one can find a numerical approximation for the solution of the above PDE by solving (13). We set

$$\Omega = [-1, 1] \subset \mathbb{R}, \quad f(x) = (1 + 9\pi^2)\cos 3\pi x + (1 + 121\pi^2)\cos 11\pi x \tag{14}$$

in (13); the exact solution is $u(x) = \cos 3\pi x + \cos 11\pi x$. We can observe in Figure 3(a) that NPSC achieves a superior level of accuracy such that the other algorithms do not reach even with a large number of epochs. As shown in Figures 3(b), L^2–errors of the NPSC results decrease as the number of neuron increases, and their decreasing rates are comparable to the theoretical optimal rates in (Siegel & Xu, 2022c).

Finally, we consider the following two-dimensional example for (13):

$$\Omega = [0, 1]^2 \subset \mathbb{R}^2, \quad f(x_1, x_2) = (1 + 2\pi^2)\cos \pi x_1 \cos \pi x_2, \tag{15}$$

whose exact solution is $u(x_1, x_2) = \cos \pi x \cos \pi y$. Since the preconditioner in Theorem 2 is applicable for one-dimension only, in this example, a-minimization problems are solved by n iterations of the unpreconditioned conjugate gradient method. We use the quasi-Monte Carlo method (Caflisch, 1998) with 10,000 sampling points for numerical integration; see Appendix D for details. It is verified by the numerical results for (15) presented in Figure 3(c, d) that NPSC outperforms the other methods and provides reasonable L^2-errors in high-dimensional problems as well.

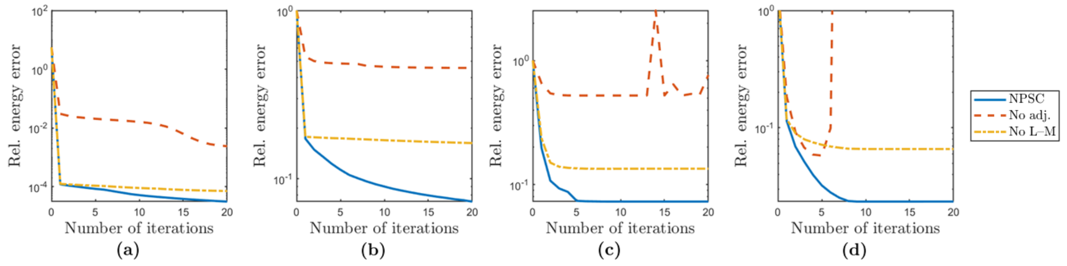


Figure 4: Ablation studies for NPSC on the relative energy error $\frac{E(\theta^{(k)})-E^*}{|E^*|}$. "No adj." and "No L–M" denote NPSC without the adjustment step and the Levenberg–Marquardt algorithm, respectively.

Remark 1. When we train a neural network (1) for solving the PDE (13) with a gradient-based method, we have to evaluate the second-order derivative of $\sigma(x)$, which is the Dirac delta function. In our experiments, we simply ignore Dirac delta terms in numerical integration. This motivates us to consider high-order activation functions like ReLU^k (Siegel & Xu, 2022b) as a future work.

4.3 Ablation Studies

Key components of the proposed NPSC are the adjustment step for parameters (Algorithm 2), the optimal preconditioner for a-minimization problems (Theorem 2), the Levenberg–Marquardt algorithm for $\{\omega_i, b_i\}$-minimization problems, and the backtracking scheme for learning rates (Algorithm 3). In order to validate the effects of these components, we conduct ablation studies for the adjustment step and the Levenberg–Marquardt algorithm; relevant results for the preconditioner and the backtracking scheme can be found in Appendices B and C, respectively.

Figure 4 depicts numerical comparisons among three algorithms: NPSC, NPSC without Algorithm 2, and NPSC without the Levenberg–Marquardt algorithm. In the last algorithm, each $\{\omega_i, b_i\}$-minimization problem is solved approximately by 20 iterations of GD. Since the variants of NPSC achieve slower convergence rates than NPSC in all the examples, one can conclude that both Algorithm 2 and the Levenberg–Marquardt algorithm contribute to the fast convergence of NPSC. We also note that Algorithm 2 helps NPSC to avoid unstable convergence behaviors like Figure 4(c, d).

5 Conclusion

In this paper, we proposed NPSC for training ReLU neural networks for numerical solution of PDEs. Separately designing efficient solvers for the linear layer and each neuron and training them alternately, NPSC yields accurate results for function approximation problems and PDEs.

This paper leaves us several interesting and important topics for future research. Although NPSC adopts the well-established framework of subspace correction methods, its rigorous convergence analysis is still open due to the nonconvexity of the model. To solve high-order PDEs, we should generalize NPSC so that it can be applied to different kind of activation functions. In addition, we should consider optimal preconditioners for optimizing linear layer in high dimensions. Generalization of NPSC to many layers is not clear yet, and we consider this topic as a future work.

Reducing the gap between the theoretical approximation properties and training results is a challenging issue even for simple neural networks. To the best of our knowledge, NPSC is the first result that deals with the ill-conditioning of layers and successfully reduces the gap. We believe that this paper can play a role of a good preliminary work for efficient and accurate training of general network architectures.

References

Mark Ainsworth and Yeonjong Shin. Plateau phenomenon in gradient descent training of ReLU networks: Explanation, quantification, and avoidance. *SIAM J. Sci. Comput.*, 43(5):A3438–A3468, 2021.

Mark Ainsworth and Yeonjong Shin. Active Neuron Least Squares: A training method for multivariate rectified neural networks. *SIAM J. Sci. Comput.*, 44(4):A2253–A2275, 2022.

Lori Badea and Rolf Krause. One-and two-level Schwarz methods for variational inequalities of the second kind and their application to frictional contact. *Numer. Math.*, 120(4):573–599, 2012.

Haim Brezis. *Functional Analysis, Sobolev Spaces and Partial Differential Equations*. Springer, New York, 2011.

Richard L. Burden, J. Douglas Faires, and Annette M. Burden. *Numerical Analysis*. Cengage Learning, Boston, MA, 2015.

Russel E. Caflisch. Monte Carlo and quasi-Monte Carlo methods. *Acta Numer.*, 7:1–49, 1998.

Wei Cai, Xiaoguang Li, and Lizuo Liu. A phase shift deep neural network for high frequency approximation and wave problems. *SIAM J. Sci. Comput.*, 42(5):A3285–A3312, 2020.

Luca Calatroni and Antonin Chambolle. Backtracking strategies for accelerated descent methods with smooth composite objectives. *SIAM J. Optim.*, 29(3):1772–1798, 2019.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Math. Control Signals Syst.*, 2(4):303–314, 1989.

Eric C. Cyr, Mamikon A. Gulian, Ravi G. Patel, Mauro Perego, and Nathaniel A. Trask. Robust training and initialization of deep neural networks: An adaptive basis viewpoint. In *Proceedings of The First Mathematical and Scientific Machine Learning Conference*, volume 107 of *Proceedings of Machine Learning Research*, pp. 512–536. PMLR, 2020.

Weinan E and Bing Yu. The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems. *Commun. Math. Stat.*, 6(1):1–12, 2018.

Juncai He, Lin Li, Jinchao Xu, and Chunyue Zheng. ReLU deep neural networks and linear finite elements. *J. Comput. Math.*, 38(3):502–527, 2020.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1026–1034, 2015.

Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, Philadelphia, PA, second edition, 2002.

Qingguo Hong, Qinyang Tan, Jonathan W. Siegel, and Jinchao Xu. On the activation function dependence of the spectral bias of neural networks. *arXiv preprint arXiv:2208.04924*, 2022.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

Ladislav Kocis and William J. Whiten. Computational investigations of low-discrepancy sequences. *ACM Trans. Math. Software*, 23(2):266–294, 1997.

Aditi Krishnapriyan, Amir Gholami, Shandian Zhe, Robert Kirby, and Michael W. Mahoney. Characterizing possible failure modes in physics-informed neural networks. In *Advances in Neural Information Processing Systems*, volume 34, pp. 26548–26560. Curran Associates, Inc., 2021.

Chang-Ock Lee and Jongho Park. Fast nonoverlapping block Jacobi method for the dual Rudin–Osher–Fatemi model. *SIAM J. Imaging Sci.*, 12(4):2009–2034, 2019.

Lu Lu, Yeonjong Shin, Yanhui Su, and George E. Karniadakis. Dying ReLU and initialization: Theory and numerical examples. *Commun. Comput. Phys.*, 28(5):1671–1706, 2020.

Donald W. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *J. Soc. Ind. Appl. Math.*, 11(2):431–441, 1963.

Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, New York, second edition, 2006.

Hyeyoung Park, S.-I. Amari, and Kenji Fukumizu. Adaptive natural gradient learning algorithms for various stochastic models. *Neural Netw.*, 13(7):755–764, 2000.

Jongho Park. Additive Schwarz methods for convex optimization as gradient methods. *SIAM J. Numer. Anal.*, 58(3):1495–1530, 2020.

Jongho Park. Additive Schwarz methods for convex optimization with backtracking. *Comput. Math. Appl.*, 113:332–344, 2022.

Allan Pinkus. Approximation theory of the MLP model in neural networks. *Acta Numer.*, 8:143–195, 1999.

Katya Scheinberg, Donald Goldfarb, and Xi Bai. Fast first-order methods for composite convex optimization with backtracking. *Found. Comput. Math.*, 14(3):389–417, 2014.

Jonathan W. Siegel and Jinchao Xu. Optimal convergence rates for the orthogonal greedy algorithm. *IEEE Trans. Inform. Theory*, 68(5):3354–3361, 2022a.

Jonathan W. Siegel and Jinchao Xu. High-order approximation rates for shallow neural networks with cosine and ReLU^k activation functions. *Appl. Comput. Harmon. Anal.*, 58:1–26, 2022b.

Jonathan W. Siegel and Jinchao Xu. Sharp bounds on the approximation rates, metric entropy, and n-widths of shallow neural networks. *Found. Comput. Math.*, 2022c. doi: 10.1007/s10208-022-09595-3.

Mahdi Soltanolkotabi. Learning ReLUs via gradient descent. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Xue-Cheng Tai and Jinchao Xu. Global and uniform convergence of subspace correction methods for some convex optimization problems. *Math. Comp.*, 71(237):105–124, 2002.

Yuandong Tian. An analytical formula of population gradient for two-layered ReLU network and its applications in convergence and critical point analysis. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pp. 3404–3413. PMLR, 2017.

Gal Vardi, Gilad Yehudai, and Ohad Shamir. Learning a single neuron with bias using gradient descent. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

Stephen J. Wright. Coordinate descent algorithms. *Math. Program.*, 151(1):3–34, 2015.

Jinchao Xu. Iterative methods by space decomposition and subspace correction. *SIAM Rev.*, 34(4): 581–613, 1992.

Jinchao Xu. Finite neuron method and convergence analysis. *Commun. Comput. Phys.*, 28(5): 1707–1745, 2020.

Zhi-Qin John Xu, Yaoyu Zhang, Tao Luo, Yanyang Xiao, and Zheng Ma. Frequency principle: Fourier analysis sheds light on deep neural networks. *Commun. Comput. Phys.*, 28(5):1746–1767, 2020.

N. Yamashita and M. Fukushima. On the rate of convergence of the Levenberg-Marquardt method. In *Topics in Numerical Analysis*, volume 15 of *Comput. Suppl.*, pp. 239–249. Springer, Vienna, 2001.

Gilad Yehudai and Shamir Ohad. Learning a single neuron with gradient methods. In *Proceedings of Thirty Third Conference on Learning Theory*, volume 125 of *Proceedings of Machine Learning Research*, pp. 3756–3786. PMLR, 2020.

Jinshan Zeng, Tim T.-K. Lau, Shaobo Lin, and Yuan Yao. Global convergence of block coordinate descent in deep learning. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pp. 7313–7323. PMLR, 2019.

Ziming Zhang and Matthew Brand. Convergent block coordinate descent for training tikhonov regularized deep neural networks. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

A Deferred Proofs

In this appendix, we provide proofs of Theorems 1–3. First, we provide the proof of Theorem 1.

Proof of Theorem 1. If $b_i \leq -\operatorname{ess\,sup}_{x \in \Omega} \omega_i \cdot x$ for some i, then we have

$$\omega_i \cdot x + b_i \leq \operatorname*{ess\,sup}_{x \in \Omega} \omega_i \cdot x + b_i \leq 0 \quad \text{a.e. in } \Omega.$$

Hence, $\sigma(\omega_i \cdot x + b_i) = 0$ a.e. in Ω, which means that the corresponding neuron is dead.

Now, we suppose that $b_i \geq -\operatorname{ess\,inf}_{x \in \Omega} \omega_i \cdot x$ for $1 \leq i \leq d+2$. Then we have

$$\omega_i \cdot x + b_i \geq \operatorname*{ess\,inf}_{x \in \Omega} \omega_i \cdot x + b_i \geq 0 \quad \text{a.e. in } \Omega.$$

This implies that $\sigma(\omega_i \cdot x + b_i) = \omega_i \cdot x + b_i$ a.e. in Ω, i.e., each $\sigma(\omega_i \cdot x + b_i)$ is affine in Ω. Since the dimension of the space of all affine functions in Ω is $d+1$, $\{\sigma(\omega_i \cdot x + b_i)\}_{i=1}^{d+2}$ are linearly dependent in Ω. □

Next, we introduce some preliminaries that will be used in the proof of Theorem 2. For brevity, we only consider the case $0 < x_1 < x_2 < \cdots < x_n < 1$; the cases $x_1 = 0$ and $x_n = 1$ can be proven similarly. By the Lax–Milgram theorem (Brezis, 2011, Corollary 5.8), there exists a bijective linear operator $\mathcal{A} \colon V \to V$ such that

$$a(u, v) = \int_\Omega (\mathcal{A}u) v \, dx, \quad u, v \in V. \tag{16}$$

We denote

$$\psi_i(x) = \sigma(\omega_i x + b_i), \quad 1 \leq i \leq n,$$

and write them as a column vector $\Psi(x) = [\psi_1(x), \dots, \psi_n(x)]^{\mathrm{T}}$. Then it follows that

$$K = \int_\Omega (\mathcal{A}\Psi(x)) \Psi(x)^{\mathrm{T}} \, dx,$$

where K was defined in (7) and $\mathcal{A}$ is applied entrywise. In addition, we define

$$\psi_0(x) = \sigma(\omega_0 x + b_0) = \sigma(x), \quad \psi_{n+1}(x) = \sigma(\omega_{n+1} x + b_{n+1}) = \sigma(-x + 1),$$

and write $\overline{\Psi}(x) = [\psi_0(x), \psi_1(x), \dots, \psi_{n+1}(x)]^{\mathrm{T}}$. We readily have $\Psi(x) = R\overline{\Psi}(x)$, where

$$R = [\mathbf{0}, I_n, \mathbf{0}] \in \mathbb{R}^{n \times (n+2)}. \tag{17}$$

In (17), $\mathbf{0}$ and I_n denote the zero column vector of length n and the $\mathbb{R}^{n \times n}$ identity matrix, respectively. If we write

$$\overline{K} = \int_\Omega (\mathcal{A}\overline{\Psi}(x)) \overline{\Psi}(x)^{\mathrm{T}} \, dx, \tag{18}$$

then we get $M = R\overline{K}R^{\mathrm{T}}$. Invoking He et al. (2020, Theorem 2.1) implies that $\{\psi_i(x)\}_{i=0}^{n+1}$ are linearly independent, so that they form a basis for the space V_n of continuous and piecewise linear functions on the grid $0 = x_0 < x_1 < \cdots < x_{n+1} = 1$, where $x_i = -b_i/\omega_i$.

Meanwhile, V_n admits another set of basis functions $\{\phi_i(x)\}_{i=0}^{n+1}$, which is given by

$$\phi_i(x) = \begin{cases} \dfrac{x - x_{i-1}}{x_i - x_{i-1}}, & \text{if } x \in (x_{i-1}, x_i], \\ \dfrac{x - x_{i+1}}{x_i - x_{i+1}}, & \text{if } x \in [x_i, x_{i+1}), \\ 0, & \text{otherwise}, \end{cases} \quad 0 \leq i \leq n+1,$$

with conventions $x_{-1} = 0$ and $x_{n+2} = 1$. That is, $\{\phi_i(x)\}_{i=0}^{n+1}$ is the collection of the standard hat basis functions defined on the grid $\{x_i\}_{i=0}^{n+1}$ Similar to (18), we write $\overline{\Phi}(x) = [\phi_0(x), \phi_1(x), \dots, \phi_n(x)]^{\mathrm{T}}$ and set

$$\overline{K}_\phi = \int_\Omega (\mathcal{A}\overline{\Phi}(x)) \overline{\Phi}(x)^{\mathrm{T}} \, dx.$$

The following lemma, which can be deduced by using elementary linear algebra, is an ingredient of the proof of Theorem 2.

Lemma 1. *For two positive integers $m \geq n$, let A, $B \in \mathbb{R}^{m\times m}$ be nonsingular matrices and let $R \in \mathbb{R}^{n\times m}$ be a surjective matrix. Then $RB^{-1}R^{\mathrm{T}}$ is nonsingular and*

$$\kappa\left(\left(RB^{-1}R^{\mathrm{T}}\right)^{-1}RAR^{\mathrm{T}}\right) \leq \kappa(BA).$$

Proof. Since B^{-1} is nonsingular and R^{T} is injective, we get

$$\alpha^{\mathrm{T}}RB^{-1}R^{\mathrm{T}}\alpha = 0 \quad \Leftrightarrow \quad R^{\mathrm{T}}\alpha = \mathbf{0} \quad \Leftrightarrow \quad \alpha = \mathbf{0}$$

for $\alpha \in \mathbb{R}^n$, which implies that $RB^{-1}R^{\mathrm{T}}$ is nonsingular.

Let $\lambda_{\min}$ and $\lambda_{\max}$ stand for the minimum and maximum eigenvalues, respectively. Then it follows that

$$\begin{aligned}\lambda_{\min}\left(\left(RB^{-1}R^{\mathrm{T}}\right)^{-1}RAR^{\mathrm{T}}\right) &= \min_{\alpha\neq 0}\frac{\alpha^{\mathrm{T}}RAR^{\mathrm{T}}\alpha}{\alpha^{\mathrm{T}}RB^{-1}R^{\mathrm{T}}\alpha} \\ &= \min_{\substack{\bar{\alpha}\in\operatorname{ran}R^{\mathrm{T}},\\ \bar{\alpha}\neq 0}}\frac{\bar{\alpha}^{\mathrm{T}}A\bar{\alpha}}{\bar{\alpha}^{\mathrm{T}}B^{-1}\bar{\alpha}} \geq \min_{\bar{\alpha}\neq 0}\frac{\bar{\alpha}^{\mathrm{T}}A\bar{\alpha}}{\bar{\alpha}^{\mathrm{T}}B^{-1}\bar{\alpha}} = \lambda_{\min}(BA). \quad (19)\end{aligned}$$

In the same manner, we have

$$\lambda_{\max}\left(\left(RB^{-1}R^{\mathrm{T}}\right)^{-1}RAR^{\mathrm{T}}\right) \leq \lambda_{\max}(BA). \quad (20)$$

Combining (19) and (20) yields the desired result. □

Now, we are ready to present our proof of Theorem 2.

Proof of Theorem 2. First, we define $\overline{\Psi}^{+}(x) = [\psi_0^+(x), \psi_1^+(x), \dots, \psi_{n+1}^+(x)]^{\mathrm{T}}$, where each $\psi_i^+(x)$ is given by

$$\psi_i^+(x) = \begin{cases}\sigma(x - x_i), & \text{if } 0 \leq i \leq n,\\ \sigma(-x + x_i), & \text{if } i = n+1.\end{cases}$$

It follows by direct calculation that

$$\psi_i(x) = \begin{cases}|\omega_i|\psi_i^+(x), & \text{if } \omega_i > 0 \text{ or } i = n+1,\\ \omega_i(1 - x_i)\psi_0^+(x) - \omega_i\psi_i^+(x) - \omega_i x_i\psi_{n+1}^+(x), & \text{otherwise.}\end{cases} \quad (21)$$

Using (21), it is straightforward to construct a matrix $C_1 \in \mathbb{R}^{(n+2)\times(n+2)}$ satisfying

$$\overline{\Psi}(x) = C_1\overline{\Psi}^{+}(x). \quad (22)$$

The matrix C_1 is nonsingular since both $\{\psi_i(x)\}_{i=0}^{n+1}$ and $\{\psi_i^+(x)\}_{i=0}^{n+1}$ forms bases for V_n (He et al., 2020). Meanwhile, one can verify the following relation between $\{\{\phi_i(x)\}_{i=0}^{n+1}$ and $\{\psi_i^+(x)\}_{i=0}^{n+1}$ by direct calculation:

$$\begin{aligned}\phi_0(x) &= -\frac{1 - x_1}{x_1}\psi_0^+(x) + \frac{1}{x_1}\psi_1^+(x) + \psi_{n+1}^+(x),\\ \phi_i(x) &= \frac{1}{x_i - x_{i-1}}\psi_{i-1}^+(x) - \frac{x_{i+1} - x_{i-1}}{(x_{i+1} - x_i)(x_i - x_{i-1})}\psi_i^+(x) + \frac{1}{x_{i+1} - x_i}\psi_{i+1}^+(x),\ 1 \leq i \leq n-1,\\ \phi_n(x) &= \frac{1}{x_n - x_{n-1}}\psi_{n-1}^+(x) - \frac{1 - x_{n-1}}{(1 - x_n)(x_n - x_{n-1})}\psi_n^+(x),\\ \phi_{n+1}(x) &= \frac{1}{1 - x_n}\psi_n^+(x).\end{aligned} \quad (23)$$

Hence, we can construct a matrix $C_2 \in \mathbb{R}^{(n+2)\times(n+2)}$ such that

$$\overline{\Phi}(x) = C_2\overline{\Psi}^{+}(x) \quad (24)$$

explicitly using (23). Combining (22) and (24) yields

$$\overline{\Phi}(x) = C\overline{\Psi}(x), \quad (25)$$

where $C = C_2 C_1^{-1}$. See Appendix B for how to compute multiplication by C_1^{-1} efficiently. Equation (25) implies that two matrices $\overline{K}$ and $\overline{K}_\phi$ are related as follows:

$$\overline{K} = \int_\Omega (\mathcal{A}\overline{\Psi}(x))\overline{\Psi}(x)^{\mathrm{T}}\,dx = \int_\Omega C^{-1}(\mathcal{A}\overline{\Phi}(x))\overline{\Phi}(x)^{\mathrm{T}}C^{-\mathrm{T}}\,dx = C^{-1}\overline{K}_\phi C^{-\mathrm{T}}. \tag{26}$$

Since $M = R\overline{K}R^{\mathrm{T}}$, invoking Lemma 1, setting $\overline{P} = C^{\mathrm{T}}\overline{K}_\phi^{-1}C$ and

$$P = \left(R\overline{P}^{-1}R^{\mathrm{T}}\right)^{-1} \tag{27}$$

completes the proof, where R was defined in (17). A computationally cheap way of multiplication by P is presented in Appendix B. □

Remark 2. In the case of the L^2-function approximation problem (10), we can construct an alternative preconditioner P_{alt} whose computational cost is a bit cheaper than P. Let $\overline{P}_{\mathrm{alt}} = C^{\mathrm{T}}\operatorname{diag}(\overline{K}_\phi)^{-1}C$. It follows from (26) that

$$\overline{P}_{\mathrm{alt}}\overline{K} = C^{\mathrm{T}}\operatorname{diag}(\overline{K}_\phi)^{-1}C\overline{K} = C^{\mathrm{T}}\operatorname{diag}(\overline{K}_\phi)^{-1}\overline{K}_\phi C^{-\mathrm{T}}.$$

In (10), $\overline{K}_\phi$ is a mass matrix for the linear finite element method defined on the grid $\{x_i\}_{i=0}^{n+1}$, so that it satisfies $\kappa(\operatorname{diag}(\overline{K}_\phi)^{-1}\overline{K}_\phi) = O(1)$. Then we have

$$\kappa(\overline{P}_{\mathrm{alt}}\,\overline{K}) = \kappa(C^{\mathrm{T}}\operatorname{diag}(\overline{K}_\phi)^{-1}\overline{K}_\phi C^{-\mathrm{T}}) = \kappa(\operatorname{diag}(\overline{K}_\phi)^{-1}\overline{K}_\phi) = O(1).$$

Therefore, $P_{\mathrm{alt}} = (R\overline{P}_{\mathrm{alt}}^{-1}R^{\mathrm{T}})^{-1}$ satisfies $\kappa(P_{\mathrm{alt}}K) = O(1)$ by Lemma 1. It is evident that the computational cost of $\operatorname{diag}(\overline{K}_\phi)^{-1}$ is cheaper than that of $\overline{K}_\phi^{-1}$.

Finally, we present our proof of Theorem 3.

Proof of Theorem 3. Let $\{\omega_i^*, b_i^*\}$ be a critical point of (8). If $\sigma(\omega_i^*\cdot x + b_i^*)$ vanishes on Ω, then we readily get

$$E_i(\omega_i^*, b_i^*) = 0 = E_i(\mathbf{0}).$$

Now, we suppose that $\sigma(\omega_i^*\cdot x + b_i^*)$ is nontrivial, i.e., the set

$$D = \{x \in \Omega : \omega_i^*\cdot x + b_i^* > 0\}$$

has nonzero measure. Note that $\sigma(\omega_i^*\cdot x + b_i^*) = \omega_i^*\cdot x + b_i^*$ and $\sigma'(\omega_i^*\cdot x + b_i^*) = 1$ on D. Since $\nabla E_i(\omega_i^*, b_i^*) = \mathbf{0}$, we obtain

$$\begin{aligned}
\mathbf{0} &= \nabla E_i(\omega_i^*)\\
&= \int_\Omega (\mathcal{A}\sigma(\omega_i^*\cdot x + b_i^*) - F(x))\,\sigma'(\omega_i^*\cdot x + b_i^*)\begin{bmatrix} x\\ 1\end{bmatrix}dx = \int_D (\mathcal{A}(\omega_i^*\cdot x + b_i^*) - F(x))\begin{bmatrix} x\\ 1\end{bmatrix}dx,
\end{aligned} \tag{28}$$

where the operator $\mathcal{A}$ was given in (16) and the integrals are done entrywise. It follows that

$$\begin{aligned}
E_i(\omega_i^*, b_i^*) &= \frac{1}{2}\int_D [\mathcal{A}(\omega_i^*\cdot x + b_i^*)]\,(\omega_i^*\cdot x + b_i^*)\,dx - \int_D F(x)(\omega_i^*\cdot x + b_i^*)\,dx\\
&= \begin{bmatrix}\omega_i^*\\ b_i^*\end{bmatrix}\cdot\int_D [\mathcal{A}(\omega_i^*\cdot x + b_i^*) - F(x)]\begin{bmatrix} x\\ 1\end{bmatrix}dx - \frac{1}{2}\int_D [\mathcal{A}(\omega_i^*\cdot x + b_i^*)]\,(\omega_i^*\cdot x + b_i^*)\,dx\\
&\overset{(28)}{=} -\frac{1}{2}\int_D [\mathcal{A}(\omega_i^*\cdot x + b_i^*)]\,(\omega_i^*\cdot x + b_i^*)\,dx\\
&< 0 = E_i(\mathbf{0}),
\end{aligned}$$

which completes the proof. □

Remark 3. In Vardi et al. (2021, Theorem 4.2), a stronger result than Theorem 3 for the following special case was presented:

$$a(u,v) = \int_\Omega uv\,dx, \quad F(x) = \sigma(\hat{w}_i\cdot x + \hat{b}_i)$$

for some $\hat{w}_i$ and $\hat{b}_i$. It was proven that any critical point ω_i^* of (8) such that $\sigma(\omega_i^*\cdot x + b_i^*)$ is nontrivial is a global minimum.

B COMPUTATIONAL ASPECTS OF THE PRECONDITIONER IN THEOREM 2

In this appendix, we present computational aspects of the preconditioner P presented in Theorem 2. Although the preconditioner P defined in (27) seems a bit complicated at the first glance, its computation requires only a cheap cost. We provide a detailed explanation on how to deal with the preconditioner P in implementation. In addition, we present numerical comparisons between the preconditioned and unpreconditioned conjugate gradient methods to support our theoretical result and highlight the computational efficiency of the preconditioner P.

B.1 IMPLEMENTATION ISSUES

Three nontrivial parts in the preconditioner P are the inverses of the matrices C_1, $R\overline{P}^{-1}R^{\mathrm{T}}$, and $\overline{K}_\phi$. First, we consider how to compute $C_1^{-1}\alpha$ when a vector $\alpha \in \mathbb{R}^{n+2}$ is given. We solve a linear system $C_1\beta = \alpha$ in order to obtain $C_1^{-1}\alpha$. Thanks to the sparsity pattern of C_1, this linear system can be solved directly by the following $O(n)$ elementary arithmetic operations:

$$\begin{aligned}
\beta_1 &= \frac{\alpha_1}{(C_1)_{1,1}}, \\
\beta_i &= \frac{1}{(C_1)_{ii}}\left(\alpha_i - \frac{(C_1)_{i1}}{(C_1)_{1,1}}\alpha_1 - \frac{(C_1)_{i,n+2}}{(C_1)_{n+2,n+2}}\alpha_{n+2}\right), \quad 2 \le i \le n+1, \\
\beta_{n+2} &= \frac{\alpha_{n+2}}{(C_1)_{n+2,n+2}}.
\end{aligned}$$

Similarly, for $\alpha \in \mathbb{R}^n$, one can compute $(R\overline{P}^{-1}R^{\mathrm{T}})^{-1}\alpha$ by solving a linear system

$$R\overline{P}^{-1}R^{\mathrm{T}}\beta = \alpha. \tag{29}$$

In (29), by the definition (17) of R, we get

$$\overline{P}^{-1}R^{\mathrm{T}}\beta = \begin{bmatrix} p \\ \alpha \\ q \end{bmatrix}$$

for some $p, q \in \mathbb{R}$. Using the fact that the first and last entries of $R^{\mathrm{T}}\beta$ are zero, two constants p and q can be determined by the following system of linear equations:

$$\begin{aligned}
\overline{P}_{1,1}p + \overline{P}_{1,n+2}q &= -\overline{P}_{1,2:n+1}\alpha, \\
\overline{P}_{n+2,1}p + \overline{P}_{n+2,n+2}q &= -\overline{P}_{n+2,2:n+1}\alpha,
\end{aligned}$$

where $2 : n+1$ means "from the second to $(n+1)$th columns." Now, β is obtained by

$$\beta = R\overline{P}\begin{bmatrix} p \\ \alpha \\ q \end{bmatrix}.$$

Finally, we consider how to compute $\overline{K}_\phi^{-1}\alpha$ when a vector $\alpha \in \mathbb{R}^{n+2}$ is given. In most applications, the bilinear form $a(\cdot,\cdot)$ is defined in terms of differential operators as in (13). This implies that the matrix $\overline{K}_\phi$ is tridiagonal, so that $\overline{K}_\phi^{-1}\alpha$ can be obtained by the Thomas algorithm (see, e.g., Higham (2002, Section 9.6)), which requires only $O(n)$ elementary arithmetic operations. In conclusion, the computation of the preconditioner P can be done efficiently.

B.2 NUMERICAL RESULTS

We turn our attention to numerical experiments for the preconditioner P. We consider the a-minimization problem (6) appearing in training of neural networks that solve the L^2-function approximation problem (12) and the elliptic PDE (14). In (6), we set the parameters ω and b as follows:

$$\omega_i = 1, \quad b_i = -\frac{i}{n+1}, \quad 1 \le i \le n.$$

n	Problem (12)		Problem (14)	
	Prec.	Unprec.	Prec.	Unprec.
2^4	2	34	3	19
2^5	2	107	2	37
2^6	2	310	4	59
2^7	3	1018	4	103
2^8	3	3470	5	173

Table 1: Number of preconditioned and unpreconditioned conjugate gradient iterations to solve the a-minimization problems (6) corresponding to the L^2-function approximation problem (12) and the elliptic PDE (14), where n denotes the number of neurons.

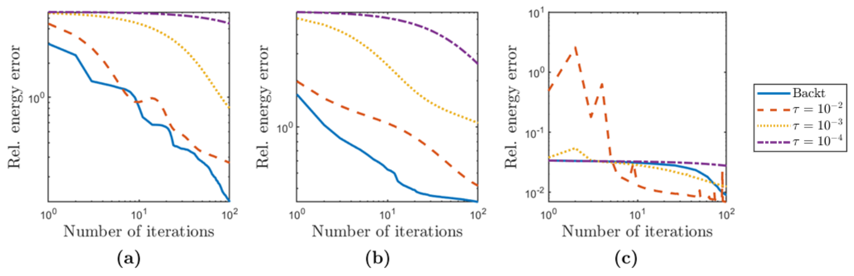


Figure 5: Decay of the relative energy error $\frac{E(\theta^{(k)})-E^*}{|E^*|}$ in various training algorithms for solving (11). "Backt" denotes the backtracking scheme presented in Algorithm 3, and τ denotes the fixed learning rate.

We compare the numerical performances of the preconditioned and unpreconditioned conjugate gradient methods solving (6). The initial guess $a^{(0)}$ is given by the zero vector. Table 1 presents the number of iterations of the preconditioned and unpreconditioned conjugate gradient methods to meet the stop condition (9) with respect to various numbers of neurons n. While the number of unpreconditioned iterations increases rapidly as n increases, the number of preconditioned iterations is uniformly bounded with respect to n. This supports our theoretical result presented in Theorem 2, which claims that the condition number of the preconditioned operator PK is $O(1)$. Since the computational cost of P is cheap, we can conclude that the preconditioner P is numerically efficient.

C Effect of Backtracking for Conventional Training Algorithms

In this appendix, we present numerical results that show that the backtracking scheme presented in Algorithm 3 is useful not only for the proposed NPSC but also for conventional training algorithms such as GD, Adam (Kingma & Ba, 2015), and LSGD (Cyr et al., 2020).

Figure 5 plots the relative energy error $\frac{E(\theta^{(k)})-E^*}{|E^*|}$ of GD, Adam, and LSGD for solving the problem (11), averaged over 10 random initializations, where k denotes the number of epochs and E^* is the energy corresponding to the exact solution of the problem. The number of neurons used is 2^5; while we can observe similar results for the other numbers of neurons, we only provide the result of 2^5 neurons for brevity. We observe that the algorithms equipped with the backtracking scheme outperform those with constant learning rates $\tau = 10^{-2}$, 10^{-3}, and 10^{-4} in the sense of the convergence rate. That is, Algorithm 3 seems to successfully find a good learning rate at each iteration of conventional training algorithms as well. Hence, in Section 4, we employ Algorithm 3 to find learning rates of GD, Adam, and LSGD.

D NUMERICAL INTEGRATION

This appendix is devoted to numerical integration schemes for computing the integral in our model problem (2). If $d = 1$, i.e., if the domain $\Omega = [x_{\mathrm{L}}, x_{\mathrm{R}}] \subset \mathbb{R}$ for some $x_{\mathrm{L}}, x_{\mathrm{R}} \in \mathbb{R}$, then we approximate the integral of a function $g(x)$ defined on Ω by the following simple trapezoidal rule:

$$\int_{\Omega} g(x)\,dx \approx \sum_{j=1}^{N-1} \frac{g(x_j) + g(x_{j+1})}{2}(x_{j+1} - x_j), \tag{30}$$

where $x_{\mathrm{L}} = x_1 < x_2 < \cdots < x_N = x_{\mathrm{R}}$ are N uniform sampling points between x_{L} and x_{R}, i.e., $x_j = x_{\mathrm{L}} + \frac{j-1}{N-1}(x_{\mathrm{R}} - x_{\mathrm{L}})$, $1 \leq j \leq N$. Approximation properties of the trapezoidal rule (30) can be found in standard textbooks on numerical analysis; see, e.g., Burden et al. (2015).

When $d \geq 2$, we adopt the quasi-Monte Carlo method (Caflisch, 1998) based on Halton sequences (Kocis & Whiten, 1997), which is known to overcome the curse of dimensionality in the sense that approximation error bounds independent of the dimension d are available. In the quasi-Monte Carlo method, the integral of a function $g(x)$ defined on Ω is approximated by the average of the function evaluated at N sampling points:

$$\int_{\Omega} g(x)\,dx \approx \frac{1}{N}\sum_{j=1}^{N} g(x_j), \tag{31}$$

where $\{x_j\}_{j=1}^N$ is a low-discrepancy sequence in Ω defined in terms of Halton sequences. For the sake of description, we assume that $\Omega = [0,1]^d \in \mathbb{R}^d$. Then the kth coordinate of x_j ($1 \leq j \leq N$, $1 \leq k \leq d$) is the number j written in p_k-ary representation, inverted, and written after the decimal point, where p_k is the kth smallest prime number. For example, if $d = 3$, then the first four points of $\{x_j\}_{j=1}^N$ is given by

$$x_1 = \left(\frac{1}{2}, \frac{1}{3}, \frac{1}{5}\right), \; x_2 = \left(\frac{1}{2^2}, \frac{2}{3}, \frac{2}{5}\right), \; x_3 = \left(\frac{3}{2^2}, \frac{1}{3^2}, \frac{3}{5}\right), \; x_4 = \left(\frac{1}{2^3}, \frac{4}{3^2}, \frac{4}{5}\right).$$

Approximation properties of (31) can be found in Caflisch (1998).