A STRUCTURE-GUIDED GAUSS-NEWTON METHOD FOR SHALLOW RELU NEURAL NETWORK *

ZHIQIANG CAI†, TONG DING†, MIN LIU‡, XINYU LIU†, AND JIANLIN XIA†

Abstract. In this paper, we propose a structure-guided Gauss-Newton (SgGN) method for solving least squares problems using a shallow ReLU neural network. The method effectively takes advantage of both the least squares structure and the neural network structure of the objective function. By categorizing the weights and biases of the hidden and output layers of the network as nonlinear and linear parameters, respectively, the method iterates back and forth between the nonlinear and linear parameters. The nonlinear parameters are updated by a damped Gauss-Newton method and the linear ones are updated by a linear solver. Moreover, at the Gauss-Newton step, a special form of the Gauss-Newton matrix is derived for the shallow ReLU neural network and is used for efficient iterations. It is shown that the corresponding mass and Gauss-Newton matrices in the respective linear and nonlinear steps are symmetric and positive definite under reasonable assumptions. Thus, the SgGN method naturally produces an effective search direction without the need of additional techniques like shifting in the Levenberg-Marquardt method to achieve invertibility of the Gauss-Newton matrix. The convergence and accuracy of the method are demonstrated numerically for several challenging function approximation problems, especially those with discontinuities or sharp transition layers that pose significant challenges for commonly used training algorithms in machine learning.

Key words. structure-guided Gauss-Newton method, neural network, least squares, mass matrix, Hessian matrix, positive definiteness

AMS subject classifications. 65D15, 65K10

1. Introduction.

When a neural network (NN) is used as a model for least-squares data fitting, the procedure for determining the values of the NN parameters is a high-dimensional non-convex optimization problem. This optimization problem tends to be computationally intensive and complicated. Popular and widely used optimization algorithms (iterative solvers) in machine learning are generally first-order gradient-based methods (see, e.g., survey papers [3, 11, 24]) because of their moderate computational cost per iteration and their ease for implementation. However, their efficiency depends heavily on hyper-parameters, the learning rate is cumbersome to tune, and the methods usually converge slowly. Moreover, they exhibit the so-called plateau phenomenon in many training tasks (see, e.g., [1]).

Recently, there has been a lot of interest in using second-order methods such as BFGS [4, 10, 12, 22] to solve optimization problems that arise from NN-based machine learning applications. For attractive features and recent progress in overcoming the challenges of second-order methods, see survey papers [3, 11, 24]. The Gauss-Newton (GN) method is a popular and widely used optimization technique for solving general least-squares problems, as described in [9, 21]. Originating from the classical Newton's method, this approach makes use of the least squares structure and approximates the Hessian matrix by its principal part. In recent years, the GN method has found practical applications in the realm of machine learning. The Kronecker-factored approximate curvature (KFAC) method [17] enhances the optimization process of neural networks by leveraging the Kronecker-factored approximation of the curvature matrix to mitigate computational challenges associated with second-order optimization techniques. Building upon this foundation, the GN method was advanced in KFRA [2] for deep learning with various practical aspects addressed, such as the computational efficiency, the recursion relationship between layers, and step-size choices. The authors

*Submitted to the editors DATE.

Funding: This work of Zhiqiang Cai and Min Liu was supported in part by the National Science Foundation under grant DMS-2110571. The work of Jianlin Xia was supported in part by the National Science Foundation under grant DMS-2111007.

†Department of Mathematics, Purdue University, West Lafayette, IN (caiz@purdue.edu, ding158@purdue.edu,, liu1957@purdue.edu, xiaj@purdue.edu).

‡School of Mechanical Engineering, Purdue University, West Lafayette, IN (liu66@purdue.edu)

demonstrated the potential of the GN method to provide a more reliable and efficient alternative to conventional first-order optimization techniques for training deep neural networks.

The purpose of this paper is to design and investigate a novel structure-guided Gauss-Newton (SgGN) iterative method for solving least squares optimization problems using shallow ReLU NN. The method utilizes both the least squares structure and the ReLU NN structure. Following the study of the set of approximating functions generated by shallow ReLU NN and the establishment of the linearly independence of neurons, we set up a least-squares optimization problem and the corresponding algebraic systems for the stationary points. To quickly and accurately solve the optimization problem, we categorize the NN parameters into linear parameters (the weights $\mathbf{c}$ and bias $\boldsymbol{\alpha}$ of the output layer) and nonlinear parameters (the weights $\mathbf{w}$ and bias b of the hidden layer), denoted by $(\mathbf{c}, \boldsymbol{\alpha})$ and $\mathbf{r} = (b, \mathbf{w}^T)^T$, respectively. Such a classification leads to a natural block iterative process for solving the optimization problem by iterating back and forth between $(\mathbf{c}, \boldsymbol{\alpha})$ and $\mathbf{r}$. $(\mathbf{c}, \boldsymbol{\alpha})$ and $\mathbf{r}$ are updated by a linear solver and a damped Gauss-Newton nonlinear iterative solver, respectively.

At each SgGN iteration, the linear solver involves a mass matrix $\mathcal{A}(\mathbf{r})$ defined in (3.4) below. The nonlinear Gauss-Newton iterative solver is based on the following newly derived form of a *Gauss-Newton matrix* for shallow ReLU NN:

$$\big(D(\mathbf{c}) \otimes I_{d+1}\big)\mathcal{H}(\mathbf{r})\big(D(\mathbf{c}) \otimes I_{d+1}\big),$$

where d is the input data dimension, I_{d+1} is the order-$(d+1)$ identity matrix, $D(\mathbf{c})$ is a diagonal matrix consisting of the linear parameter $\mathbf{c}$, and $\mathcal{H}(\mathbf{r})$ depends on $\mathbf{r}$ and is referred as the *layer Gauss-Newton matrix*. A specific form of $\mathcal{H}(\mathbf{r})$ is given in (4.2). Both $\mathcal{A}(\mathbf{r})$ and $\mathcal{H}(\mathbf{r})$ are functions of the nonlinear parameter $\mathbf{r}$ and are independent of the linear parameters $(\mathbf{c}, \boldsymbol{\alpha})$.

Theoretically, we show that both $\mathcal{A}(\mathbf{r})$ and $\mathcal{H}(\mathbf{r})$ are *symmetric and positive definite* under the condition that all neurons are linearly independent (see Lemmas 3.1 and 4.2). Hence, a significant distinction between the SgGN method and the usual Gauss-Newton method is that there is no need to use additional techniques like shifting in the Levenberg-Marquardt method [14, 16] to achieve *invertibility* of the Gauss-Newton matrix.

The SgGN method provides an innovative way to effectively take advantage of both the quadratic structure and the NN structure in least-squares optimization problems arising from shallow ReLU NN approximations. The natural positive definiteness of $\mathcal{A}(\mathbf{r})$ and $\mathcal{H}(\mathbf{r})$ means many efficient direct/iterative linear solvers may be used for updating linear parameters (see (4.4)) and for computing search directions of nonlinear parameters (see (4.5)).

The SgGN method works for both continuous and discrete least-squares approximations. Its convergence and accuracy are demonstrated numerically for several one and two dimensional problems, especially those with discontinuities or sharp transition layers that pose significant challenges for commonly used training algorithms in machine learning such as BFGS [4, 10, 12, 22] and Adam [13]. The loss curves for all test problems clearly show that the SgGN method significantly outperforms those methods in terms of the convergence and accuracy. This conclusion is further enhanced by examining the ability of the methods in effectively moving the breaking hyper-planes (points for one dimension and lines for two dimensions). The breaking hyper-planes are determined by the nonlinear parameters (weights and biases of the hidden layer). A data fitting application is also tested to show that the SgGN method can be naturally extended to discrete least-squares optimization, which makes it useful for many data science applications.

The paper is organized as follows. Section 2 introduces the set of approximating functions generated by shallow ReLU neural networks and establishes linear independence of neurons. The least-squares optimization problem and the corresponding nonlinear algebraic system for stationary points are described in Section 3. In Section 4, the structure of the Gauss-Newton matrix for the nonlinear parameters is derived and the resulting SgGN method is proposed. Section 5 presents the SgGN method for discrete least-squares optimization. Numerical results are given in Section 6 for

various function approximations and data science applications, following by some conclusions and discussions in Section 7.

2. Shallow ReLU Neural Network. This section describes shallow ReLU NN as a set of continuous piece-wise linear functions from $\mathbb{R}^d$ to $\mathbb{R}$ and discusses some analytical and geometrical properties of the network. Here the output dimension is restricted to one for simplicity of presentation since the extension of materials covered by the paper to higher output dimensions is straightforward.

ReLU refers to the rectified linear activation function defined by

$$\sigma(t) = \max\{0, t\} = \begin{cases} t, & t > 0, \\ 0, & t \le 0. \end{cases} \tag{2.1}$$

Its first- and second-order derivatives are the Heaviside (unit) step and the Dirac delta functions given by

$$H(t) = \sigma'(t) = \begin{cases} 1, & t > 0, \\ 0, & t < 0 \end{cases} \quad \text{and} \quad \delta(t) = \sigma''(t) = H'(t) = \begin{cases} \infty, & t = 0, \\ 0, & t \neq 0, \end{cases} \tag{2.2}$$

respectively.

Let Ω be a connected, bounded open domain in $\mathbb{R}^d$. For any $\mathbf{x} = (x_1, \ldots, x_d)^T \in \Omega \subset \mathbb{R}^d$, by appending 1 to the inhomogeneous $(x_1, \ldots, x_d)$-coordinates, we have the following homogeneous coordinates:

$$\mathbf{y}^T = \left(1, \mathbf{x}^T\right) = (1, x_1, \ldots, x_d).$$

A standard shallow ReLU neural network with n neurons may be viewed as the set of continuous piece-wise linear functions from $\Omega \subset \mathbb{R}^d$ to $\mathbb{R}$ as follows:

$$\hat{\mathcal{M}}_n(\Omega) = \left\{ \sum_{i=1}^{n} c_i \sigma(\mathbf{r}_i \cdot \mathbf{y}) + \alpha_0 : \mathbf{x} \in \Omega,\ c_i \in \mathbb{R},\ \mathbf{r}_i^T = (b_i, \mathbf{w}_i^T),\ b_i \in \mathbb{R},\ \mathbf{w}_i \in \mathcal{S}^{d-1},\ \alpha_0 \in \mathbb{R} \right\}, \tag{2.3}$$

where $\mathbf{c} = (c_1, \ldots, c_n)^T$ and α_0 are the respective output weight and bias, $\mathbf{r}_i = (b_i, \mathbf{w}_i^T) \in \mathbb{R}^{d+1}$ is for the parameters of the i^{th} neuron with b_i and $\mathbf{w}_i$ the respective bias and weight of the neuron, and $\mathcal{S}^{d-1}$ is the unit sphere in $\mathbb{R}^d$. The constraint that the weight of each neuron belongs to the unit sphere is a consequence of normalization for the ReLU activation function (see [15]) and may narrow down the solution set of a given approximating problem. For convenience, denote

$$\mathbf{r}^T = \left(\mathbf{r}_1^T, \ldots, \mathbf{r}_n^T\right) = \left(b_1, \mathbf{w}_1^T, \ldots, b_n, \mathbf{w}_n^T\right). \tag{2.4}$$

Notice that the ReLU activation function $\sigma(t)$ is a continuous piece-wise linear function with one *breaking point* at $t = 0$. Hence each ridge function $\sigma(\mathbf{r}_i\, \mathbf{y}) = \sigma(\mathbf{w}_i \cdot \mathbf{x} + b_i)$ is a continuous piece-wise linear function with a *breaking hyper-plane* (see [5, 15]):

$$\mathcal{P}_i(\mathbf{r}_i) = \left\{ \mathbf{x} \in \Omega \subset \mathbb{R}^d : \mathbf{w}_i \cdot \mathbf{x} + b_i = 0 \right\}. \tag{2.5}$$

For the set $\hat{\mathcal{M}}_n(\Omega)$ of neural network functions defined in (2.3) with fixed weights and biases $\mathbf{r}$, there are n breaking hyper-planes $\{\mathcal{P}_i(\mathbf{r}_i)\}_{i=1}^n$. Together with the boundary of the domain Ω, these hyper-planes form a *physical partition*, denoted by $\mathcal{K}(\mathbf{r})$, of Ω [8, 15]. This partition $\mathcal{K}(\mathbf{r})$ consists of irregular, polygonal sub-domains of Ω (see Figures 6.6(e) and 6.6(i) below for some examples). Each function in $\hat{\mathcal{M}}_n(\Omega)$ is then a continuous piece-wise linear function with respect to $\mathcal{K}(\mathbf{r})$.

Below, we discuss linear independence of some ridge functions for fixed parameter $\mathbf{r}$ in (2.4). To this end, let $\sigma_0(\mathbf{x}) = 1$ and for $i = 1, \ldots, n$, let

$$\sigma_i(\mathbf{x}) = \sigma(\mathbf{r}_i \cdot \mathbf{y}) \quad \text{and} \quad H_i(\mathbf{x}) = H(\mathbf{r}_i \cdot \mathbf{y}),$$

where σ and H are the ReLU activation and Heaviside step functions given in (2.1) and (2.2), respectively. It is already known from Lemma 2.1 in [15] that the set of functions $\{\sigma_i(\mathbf{x})\}_{i=0}^n$ is linearly independent if the hyper-planes $\{\mathcal{P}_i(\mathbf{r}_i)\}_{i=1}^n$ are distinct. We further have the following result.

LEMMA 2.1. *For fixed* $\mathbf{r}$ *in* (2.4)*, assume that the hyper-planes* $\{\mathcal{P}_i(\mathbf{r}_i)\}_{i=1}^n$ *are distinct. Then the set of functions* $\{H_i(\mathbf{x}), x_1 H_i(\mathbf{x}), \ldots, x_d H_i(\mathbf{x})\}_{i=1}^n$ *is linearly independent.*

Proof. For each $i = 1, \ldots, n$, the linear independence of $\{1, x_1, \ldots, x_d\}$ implies that the set of functions

$$\big\{H_i(\mathbf{x}), x_1 H_i(\mathbf{x}), \ldots, x_d H_i(\mathbf{x})\big\} = H_i(\mathbf{x})\{1, x_1, \ldots, x_d\}$$

is linearly independent. Now, the linear independence of $\big\{H_i(\mathbf{x}), x_1 H_i(\mathbf{x}), \ldots, x_d H_i(\mathbf{x})\big\}_{i=1}^n$ follows from the assumption on the distinct hyper-planes. ∎

We conclude this section in discussing a possible restriction on the biases of all neurons in $\hat{\mathcal{M}}_n(\Omega)$. For each $\mathbf{r}_i$, there are two ridge functions:

$$\sigma(\mathbf{w}_i \cdot \mathbf{x} + b_i) \quad \text{and} \quad \sigma(-\mathbf{w}_i \cdot \mathbf{x} - b_i).$$

They share the same breaking hyper-plane $\mathcal{P}_i(\mathbf{r}_i)$ and are linearly independent. Nevertheless, the following identity is well known:

$$\sigma(\mathbf{w}_i \cdot \mathbf{x} + b_i) - \sigma(-\mathbf{w}_i \cdot \mathbf{x} - b_i) = \mathbf{w}_i \cdot \mathbf{x} + b_i$$

for all $\mathbf{x} \in \mathbb{R}^d$. This identity implies that clearly only one of the two ridge functions is needed in $\hat{\mathcal{M}}_n(\Omega)$, if $\hat{\mathcal{M}}_n(\Omega)$ contains all linear functions

$$\mathbb{P}_1 = span\,\{\phi_i(\mathbf{x})\}_{i=0}^d \quad \text{with } \phi_0(\mathbf{x}) = 1,\ \phi_i(\mathbf{x}) = x_i.$$

Hence, we may assume that $b_i \in \mathbb{R}_0^+ = [0, +\infty)$ for all i to further narrow down the solution set.

In general, $\hat{\mathcal{M}}_n(\Omega)$ does not contain $\mathbb{P}_1$. Nevertheless, this may be resolved by either expanding $\hat{\mathcal{M}}_n(\Omega)$ to contain $\mathbb{P}_1$ or fixing the weights and biases of d neurons in $\hat{\mathcal{M}}_n(\Omega)$ such that the corresponding d hyper-planes are linearly independent and do not intersect with the domain Ω. For convenience, in this paper we choose the former. That is, we use the following set of approximating functions:

$$\mathcal{M}_n(\Omega) = \left\{ \sum_{i=1}^n c_i \sigma_i(\mathbf{x}) + \sum_{i=0}^d \alpha_i \phi_i(\mathbf{x}) : \mathbf{x} \in \Omega,\, c_i \in \mathbb{R},\, \alpha_i \in \mathbb{R},\, b_i \in \mathbb{R}_0^+,\, \mathbf{w}_i \in \mathcal{S}^{d-1} \right\}.$$

The SgGN method developed in this paper can be applied to the standard shallow ReLU neural network $\hat{\mathcal{M}}_n(\Omega)$ directly without enforcing $b_i \in \mathbb{R}_0^+$.

3. Continuous Least-Squares Optimization Problems. Given a function $u(\mathbf{x})$ defined on Ω, the best least-squares approximation to u in $\mathcal{M}_n(\Omega)$ is to find $u_n(\mathbf{x}) \in \mathcal{M}_n(\Omega)$ such that

$$\mathcal{J}_\mu(u_n) = \min_{v \in \mathcal{M}_n(\Omega)} \mathcal{J}_\mu(v) \quad \text{with } \mathcal{J}_\mu(v) = \frac{1}{2}\|v - u\|_\mu^2 \tag{3.1}$$

where $\|\cdot\|_\mu$ is the weighted $L^2(\Omega)$ norm given by

$$\|f\|_\mu = \left(\int_\Omega \mu(\mathbf{x}) f^2(\mathbf{x})\, d\mathbf{x}\right)^{1/2}.$$

The corresponding weighted $L^2(\Omega)$ inner product is denoted by $\langle\cdot,\cdot\rangle_\mu$.

Below, we use the optimality condition to derive the corresponding systems of algebraic equations. To this end, let

$$u_n(\mathbf{x}) = \sum_{i=1}^{n} c_i \sigma_i(\mathbf{x}) + \sum_{i=0}^{d} \alpha_i \phi_i(\mathbf{x})$$

be a solution of (3.1). Then the *linear parameter*

$$\hat{\mathbf{c}} = (\mathbf{c}, \boldsymbol{\alpha})^T = (c_1, \ldots, c_n, \alpha_0, \ldots, \alpha_d)^T$$

and the *nonlinear parameter* $\mathbf{r}$ defined in (2.4) are critical points of the loss function $\mathcal{J}_\mu(u_n)$. That is, $\hat{\mathbf{c}}$ and $\mathbf{r}$ satisfy the following systems of algebraic equations

$$\nabla_{\hat{\mathbf{c}}} \mathcal{J}_\mu(u_n) = \mathbf{0} \quad \text{and} \quad \nabla_{\mathbf{r}} \mathcal{J}_\mu(u_n) = \mathbf{0}, \tag{3.2}$$

where $\nabla_{\hat{\mathbf{c}}}$ and $\nabla_{\mathbf{r}}$ denote the gradients with respect to the respective parameters $\hat{\mathbf{c}}$ and $\mathbf{r}$.

In the following, we derive specific forms of the algebraic equations in (3.2). Notice that

$$\nabla_{\hat{\mathbf{c}}} u_n(\mathbf{x}) = (\sigma_1, \ldots, \sigma_n, \phi_0, \ldots, \phi_d)^T.$$

Let

$$\begin{cases} a_{ij}(\mathbf{r}) = \langle \sigma_j, \sigma_i \rangle\,, \ f_i(\mathbf{r}) = \langle u, \sigma_i \rangle\,, & \text{for } i,j = 1,\ldots,n, \\ b_{ij} = \langle \phi_j, \phi_i \rangle\,, \ g_i = \langle u, \phi_i \rangle\,, & \text{for } i,j = 0,1,\ldots,d, \\ c_{ij}(\mathbf{r}) = \langle \phi_j, \sigma_i \rangle\,, & \text{for } i = 1,\ldots,n,\ j = 0,1,\ldots,d, \end{cases}$$

and let

$$\mathcal{A}_{11}(\mathbf{r}) = \big[a_{ij}(\mathbf{r})\big]_{n\times n},\ \mathcal{A}_{12}(\mathbf{r}) = \big[c_{ij}(\mathbf{r})\big]_{n\times d},\ \mathcal{A}_{22} = \big[b_{ij}\big]_{d\times d},\ \mathbf{f}_1(\mathbf{r}) = \big[f_i(\mathbf{r})\big]_{n\times 1},\ \mathbf{f}_2 = \big[g_i\big]_{d\times 1}.$$

It is easy to see that the first equation in (3.2) implies

$$\mathcal{A}(\mathbf{r})\,\hat{\mathbf{c}} = \mathbf{f}(\mathbf{r}), \tag{3.3}$$

where $\mathcal{A}(\mathbf{r})$ and $\mathbf{f}(\mathbf{r})$ are the *mass matrix* and the right-hand side vector given by

$$\mathcal{A}(\mathbf{r}) = \begin{bmatrix} \mathcal{A}_{11}(\mathbf{r}) & \mathcal{A}_{12}(\mathbf{r}) \\ \mathcal{A}_{12}^T(\mathbf{r}) & \mathcal{A}_{22} \end{bmatrix} \quad \text{and} \quad \mathbf{f}(\mathbf{r}) = \begin{bmatrix} \mathbf{f}_1(\mathbf{r}) \\ \mathbf{f}_2 \end{bmatrix}, \tag{3.4}$$

respectively.

LEMMA 3.1. *Under the assumption of* Lemma 2.1*, additionally assume that for all* $i = 1,\ldots,n$

$$b_i \geq 0 \quad \textit{and} \quad \mathcal{P}_i(\mathbf{r}_i) \cap \Omega \neq \emptyset, \tag{3.5}$$

where $\mathcal{P}_i(\mathbf{r}_i)$ *is the breaking hyper-plane defined in* (2.5)*. Then the mass matrix* $\mathcal{A}(\mathbf{r})$ *is symmetric and positive definite.*

Proof. Obviously, $\mathcal{A}(\mathbf{r})$ is symmetric. For any $\hat{\mathbf{c}} = (\mathbf{c}^T, \boldsymbol{\alpha}^T)^T \in \mathbb{R}^{n+d+1}$, it is easy to see that

$$\hat{\mathbf{c}}^T \mathcal{A}(\mathbf{r})\, \hat{\mathbf{c}} = \left\| \sum_{i=1}^{n} c_i \sigma_i(\mathbf{x}) + \sum_{i=0}^{d} \alpha_i \phi_i(\mathbf{x}) \right\|_{\mu}^2 .$$

Hence, in order to show the positive definiteness of $\mathcal{A}(\mathbf{r})$, it suffices to prove that

$$\{\sigma_i(\mathbf{x})\}_{i=1}^{n} \bigcup \{\phi_i(\mathbf{x})\}_{i=0}^{d}$$

is linearly independent. This can be done through proof by induction. To do so, first notice that the second assumption in (3.5) implies that $\sigma_i(\mathbf{x})$ vanishes in a non-empty subdomain Ω_i of Ω. The linear independence of $\{\sigma_1(\mathbf{x})\} \bigcup \{\phi_i(\mathbf{x})\}_{i=0}^{d}$ is a direct consequence of the fact that $\sigma_1(\mathbf{x})|_{\Omega_1} \equiv 0$ and the linear independence of $\{\phi_i(\mathbf{x})\}_{i=0}^{d}$ in $\Omega \backslash \Omega_1$. Similarly, the linear independence of $\{\sigma_i(\mathbf{x})\}_{i=1}^{k} \bigcup \{\phi_i(\mathbf{x})\}_{i=0}^{d}$ follows from the fact that $\sigma_k(\mathbf{x})|_{\Omega_k} \equiv 0$ and the assumption on the linear independence of $\{\sigma_i(\mathbf{x})\}_{i=1}^{k-1} \bigcup \{\phi_i(\mathbf{x})\}_{i=0}^{d}$. This completes the proof of the lemma. ∎

Next, we calculate $\nabla_{\mathbf{r}} \mathcal{J}_{\mu}(u_n)$. To simplify expression of formulas, we use the Kronecker product, denoted by $\otimes$, of two matrices. Let

$$\mathbf{H}(\mathbf{x}) = \big(H_1(\mathbf{x}), \ldots, H_n(\mathbf{x})\big)^T .$$

For $i, j = 1, \ldots, n$, the fact that

$$\nabla_{\mathbf{r}_i} \sigma_j(\mathbf{x}) = \begin{cases} \mathbf{0}, & i \neq j, \\ H_j(\mathbf{x})\mathbf{y}, & i = j \end{cases}$$

implies

$$\nabla_{\mathbf{r}} u_n(\mathbf{x}) = \left(D(\mathbf{c}) \otimes I_{d+1}\right) \left(\mathbf{H}(\mathbf{x}) \otimes \mathbf{y}\right), \tag{3.6}$$

where $D(\mathbf{c}) = \mathrm{diag}(c_1, \ldots, c_n)$ is the diagonal matrix with the i^{th}-diagonal element c_i.

Let

$$\mathbf{G}(\mathbf{c}, \boldsymbol{\alpha}, \mathbf{r}) = \int_{\Omega} \mu(\mathbf{x}) \big(u_n(\mathbf{x}) - u(\mathbf{x})\big) \mathbf{H}(\mathbf{x}) \otimes \mathbf{y}\, d\mathbf{x}. \tag{3.7}$$

It is easy to check that the second equation in (3.2) becomes

$$\nabla_{\mathbf{r}} \mathcal{J}_{\mu}(u_n) = \left(D(\mathbf{c}) \otimes I_{d+1}\right) \mathbf{G}(\mathbf{c}, \boldsymbol{\alpha}, \mathbf{r}) = 0. \tag{3.8}$$

(3.3) and (3.8) define two algebraic systems that may be used to solve for the linear parameter $\mathbf{c}$ and the nonlinear parameter $\mathbf{r}$, respectively. For convenience, (3.3) and (3.8) are referred to as the *linear and nonlinear problems*, respectively.

4. A structure-guided Gauss-Newton (SgGN) method. In this section, we introduce our SgGN method for solving the minimization problem in (3.1), guided by both the least squares structure and the ReLU NN structure. The method utilizes a block structure of the systems of algebraic equations given in (3.3) and (3.8) and iterates back and forth between the linear parameter $\mathbf{c}$ by solving (3.3) and the nonlinear parameter $\mathbf{r}$ by using the Gauss-Newton method.

To efficiently apply the Gauss-Newton method, below we derive a special "Gauss-Newton" matrix by making use of the neural network structure. To this end, let $\delta_i(\mathbf{x}) = \delta(\mathbf{r}_i \cdot \mathbf{y})$ for

$i = 1, \ldots, n$, where $\delta(t)$ is the Dirac delta function defined in (2.2). Denote the $n \times n$ diagonal matrix with the i^{th}-diagonal element $\delta_i(\mathbf{x})$ by

$$\Lambda(\mathbf{x}) = \operatorname{diag}(\delta_1(\mathbf{x}), \ldots, \delta_n(\mathbf{x})).$$

For $i, j = 1, \ldots, n$, it is easy to check that

$$\nabla_{\mathbf{r}_i} H_j(\mathbf{x}) = \begin{cases} \mathbf{0}, & i \neq j, \\ \delta_j(\mathbf{x})\mathbf{y}, & i = j. \end{cases}$$

This implies

$$\nabla_{\mathbf{r}} \mathbf{H}^T(\mathbf{x}) = \Lambda(\mathbf{x}) \otimes \mathbf{y}. \tag{4.1}$$

Now we introduce the following $n(d+1) \times n(d+1)$ matrix:

$$\mathcal{H}(\mathbf{r}) = \int_\Omega \mu \left[\mathbf{H}\mathbf{H}^T\right] \otimes \left[\mathbf{y}\mathbf{y}^T\right] d\mathbf{x}, \tag{4.2}$$

where we write $\mu(\mathbf{x})$ as μ and $\mathbf{H}(\mathbf{x})$ as $\mathbf{H}$ for the ease of notation. Lemma 4.1 below shows that $\mathcal{H}(\mathbf{r})$ is the principal part of the Gauss-Newton matrix and is referred as the *layer Gauss-Newton matrix* in this paper.

LEMMA 4.1. *The Hessian matrix has the form of*

$$\nabla_{\mathbf{r}}\big(\nabla_{\mathbf{r}} \mathcal{J}_p(u_n)\big)^T = \big(D(\mathbf{c}) \otimes I_{d+1}\big)\mathcal{H}(\mathbf{r})\big(D(\mathbf{c}) \otimes I_{d+1}\big) + \hat{\mathcal{H}}(\mathbf{c}, \mathbf{r})\big(D(\mathbf{c}) \otimes I_{d+1}\big), \tag{4.3}$$

where $\hat{\mathcal{H}}(\mathbf{c}, \mathbf{r})$ *is given by*

$$\hat{\mathcal{H}}(\mathbf{c}, \mathbf{r}) = \int_\Omega \mu\left(u_n(\mathbf{x}) - u(\mathbf{x})\right) \Lambda(\mathbf{x}) \otimes \left(\mathbf{y}\mathbf{y}^T\right) d\mathbf{x}.$$

Proof. It follows from (3.8), the product rule, (3.6), and (4.1) that

$$\begin{aligned} \nabla_{\mathbf{r}} \mathbf{G}(\mathbf{c}, \boldsymbol{\alpha}, \mathbf{r}) &= \int_\Omega \mu\big(\nabla_{\mathbf{r}} u_n\big)\left(\mathbf{H} \otimes \mathbf{y}\right)^T d\mathbf{x} + \int_\Omega \mu\left(u_n - u\right)\left(\nabla_{\mathbf{r}} \mathbf{H}^T\right) \otimes \mathbf{y}^T d\mathbf{x} \\ &= \left(D(\mathbf{c}) \otimes I_{d+1}\right) \int_\Omega \mu\big(\mathbf{H} \otimes \mathbf{y}\big)\big(\mathbf{H} \otimes \mathbf{y}\big)^T d\mathbf{x} + \int_\Omega \mu\left(u_n - u\right) \Lambda(\mathbf{x}) \otimes \left(\mathbf{y}\mathbf{y}^T\right) d\mathbf{x}, \end{aligned}$$

which, together with (3.8) and the transpose rule of the Kronecker product, implies (4.3). This completes the proof of the lemma. ∎

LEMMA 4.2. *Under the assumption of* Lemma 2.1, *the Gauss-Newton matrix* $\mathcal{H}(\mathbf{r})$ *is symmetric and positive definite.*

Proof. Clearly, $\mathcal{H}(\mathbf{r})$ is symmetric. For any $\mathbf{v}^T = (\boldsymbol{\beta}_1^T, \ldots, \boldsymbol{\beta}_n^T) \in \mathbb{R}^{n(d+1)}$ with $\boldsymbol{\beta}_i \in \mathbb{R}^{d+1}$, let

$$v(\mathbf{x}) = \sum_{i=1}^n \left(\boldsymbol{\beta}_i^T \mathbf{y}\right) H_i(\mathbf{x}).$$

It is easy to check that

$$\mathbf{v}^T \mathcal{H}(\mathbf{r}) \mathbf{v} = \mathbf{v}^T \left(\int_\Omega \mu(\mathbf{x})\big(\mathbf{H} \otimes \mathbf{y}\big)\big(\mathbf{H} \otimes \mathbf{y}\big)^T d\mathbf{x} \right) \mathbf{v} = \langle v, v \rangle_\mu \geq 0,$$

which, together with Lemma 2.1, implies that $\mathcal{H}(\mathbf{r})$ is positive definite. This completes the proof of the lemma. ∎

With the Gauss-Newton matrix $\mathcal{H}(\mathbf{r})$ defined in (4.2), Lemma 4.1, and Lemma 4.2, we are ready to describe one step of the SgGN method. Given the previous iterate $\big(\hat{\mathbf{c}}^{(k)}, \mathbf{r}^{(k)}\big) = \big(\mathbf{c}^{(k)}, \boldsymbol{\alpha}^{(k)}, \mathbf{r}^{(k)}\big)$, compute the current iterate $\big(\hat{\mathbf{c}}^{(k+1)}, \mathbf{r}^{(k+1)}\big) = \big(\mathbf{c}^{(k+1)}, \boldsymbol{\alpha}^{(k+1)}, \mathbf{r}^{(k+1)}\big)$ as follows.

(i) Compute the linear parameter $\hat{\mathbf{c}}^{(k+1)} = \big(\mathbf{c}^{(k+1)}, \boldsymbol{\alpha}^{(k+1)}\big)$ by solving the system of linear equations

$$\mathcal{A}\big(\mathbf{r}^{(k)}\big)\,\hat{\mathbf{c}}^{(k+1)} = \mathbf{f}\big(\mathbf{r}^{(k)}\big), \tag{4.4}$$

using either a direct or iterative solver.

(ii) In the case that every entry of $\mathbf{c}^{(k+1)}$ is nonzero, compute the search direction

$$\mathbf{p}^{(k+1)} = \Big(D^{-1}\big(\mathbf{c}^{(k+1)}\big) \otimes I_{d+1}\Big)\,\mathbf{s}^{(k+1)},$$

where $\mathbf{s}^{(k)}$ is the solution of the Gauss-Newton system

$$\mathcal{H}\big(\mathbf{r}^{(k)}\big)\,\mathbf{s}^{(k+1)} = -\mathbf{G}\big(\mathbf{c}^{(k+1)}, \boldsymbol{\alpha}^{(k+1)}, \mathbf{r}^{(k)}\big). \tag{4.5}$$

(iii) In the case that $\mathbf{c}^{(k+1)}$ has some entries with magnitudes less than a certain small threshold, the search direction $\mathbf{p}^{(k+1)}$ is obtained in a similar fashion as in (ii) by updating the entries of $\mathbf{p}^{(k)}$ that corresponds to the rest of entries of $\mathbf{c}^{(k+1)}$. The biases corresponding to those small entries will be assigned randomly.

(iv) Compute the nonlinear parameter

$$\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} + \gamma_{k+1}\mathbf{p}^{(k+1)},$$

where the damping parameter γ_{k+1} is the solution of a one-dimensional optimization problem

$$\gamma_{k+1} = \arg\min_{\gamma \in \mathbb{R}_0^+} \mathcal{J}_\mu\left(u_n\big(\cdot; \mathbf{c}^{(k+1)}, \mathbf{r}^{(k)} + \gamma\mathbf{p}^{(k+1)}\big)\right).$$

See Algorithm 4.1 for a pseudocode of the SgGN method.

The SgGN method needs the algebraic solutions of multiple linear systems in (4.4) and (4.5) during the iterations. Since the focus of this work is on investigating the SgGN method for shallow ReLU networks, here we just briefly mention some numerical issues considered in the method and leave the details to a forthcoming paper [7]. The linear systems may be solved with direct or iterative solvers. If $\mathbf{r}^{(k)}$ satisfies the assumption in Lemma 2.1, both the matrices $\mathcal{A}\left(\mathbf{r}^{(k)}\right)$ and $\mathcal{H}\left(\mathbf{r}^{(k)}\right)$ are symmetric and positive definite (see Lemmas 3.1 and 4.2). Nevertheless, both of them could be *nearly* singular in some applications, such as when the underlying target function has sharp layers. This is especially the case when the SgGN iterations start to converge so that some basis functions get close to each other. (A rigorous way to characterize how close they are will be given in [7].) In this work, we use direct solvers for the purpose of verifying the convergence of the SgGN algorithm.

5. SgGN for Discrete Least-Squares Optimization Problems. We then show how the SgGN method can be extended to discrete least-squares optimization problems. For a given data set $\big\{\big(\mathbf{x}^i, u^i\big)\big\}_{i=1}^m$ with $\mathbf{x}^i \in \Omega$ and $u^i \in \mathbb{R}$ and a given distribution function $0 \le \mu(\mathbf{x}) \le 1$, consider the following discrete least-squares minimization problem: finding $u_n(\mathbf{x}) \in \hat{\mathcal{M}}_n(\Omega)$ such that

$$\mathcal{J}_{m,\mu}(u_n) = \min_{v \in \hat{\mathcal{M}}_n(\Omega)} \mathcal{J}_{m,\mu}(v), \tag{5.1}$$

where $\mathcal{J}_{m,\mu}(v)$ is the weighted discrete least-squares loss function given by

$$\mathcal{J}_{m,\mu}(v) = \frac{1}{2}\sum_{i=1}^m \mu(\mathbf{x}^i)\left(v(\mathbf{x}^i) - u^i\right)^2 = \frac{1}{2}\|v - u\|_{m,\mu}^2.$$

Algorithm 4.1 A structure-guided Gauss-Newton (SgGN) method for (3.1)

Input: network parameters $\mathbf{r} = (\mathbf{r}_1, \ldots, \mathbf{r}_n)$, data set $\{(\mathbf{x}^i, u^i)\}_{i=1}^N$, density function $\mu(\mathbf{x})$
Output: network parameters $\mathbf{c}$, $\mathbf{r}$
Initialize $\mathbf{r}^{(0)}$ in (2.4)
for $k = 0, 1, \ldots$ **do**
▷ *Linear parameter* $\mathbf{c}$
Form $\mathcal{A}(\mathbf{r}^{(k)})$, $\mathbf{f}(\mathbf{r}^{(k)})$ in (3.4).
$\mathbf{c}^{(k+1)} \leftarrow \mathcal{A}^{-1}(\mathbf{r}^{(k)})\mathbf{f}(\mathbf{r}^{(k)})$
▷ *Nonlinear parameter* $\mathbf{r}$
Form $\mathbf{G}(\mathbf{c}^{(k+1)}, \mathbf{r}^{(k)})$, $\mathcal{H}(\mathbf{r}^{(k)})$ in Equations (3.7) and (4.2)
$\mathbf{s}^{(k)} \leftarrow -\mathcal{H}^{-1}\left(\mathbf{r}^{(k)}\right)\mathbf{G}\left(\mathbf{c}^{(k+1)}, \mathbf{r}^{(k)}\right)$
$\mathbf{p}^{(k)} \leftarrow \left(D^{-1}\left(\mathbf{c}^{(k+1)}\right) \otimes I_{d+1}\right)\mathbf{s}^{(k)}$
$\gamma_{k+1} \leftarrow \underset{\gamma \in \mathbb{R}_0^+}{\arg\min} \mathcal{J}_\mu\left(u_n\left(\cdot; \mathbf{c}^{(k+1)}, \mathbf{r}^{(k)} + \alpha \mathbf{p}^{(k)}\right)\right)$
$\mathbf{r}^{(k+1)} \leftarrow \mathbf{r}^{(k)} + \gamma_{k+1}\mathbf{p}^{(k)}$
if a desired loss or a specified number of iterations is reached **then**
Return $\mathbf{c}^{(k+1)}, \mathbf{r}^{(k+1)}$
end if
end for

Here, $\|\cdot\|_{m,\mu}$ is the weighted discrete $L^2(\Omega)$ norm given by

$$\|f\|_{m,\mu} = \left(\sum_{i=1}^m \mu(\mathbf{x}^i) f^2(\mathbf{x}^i)\right)^{\frac{1}{2}},$$

and the corresponding weighted discrete $L^2(\Omega)$ inner product is denoted by $\langle \cdot, \cdot \rangle_{m,\mu}$.

Problem (5.1) may be regarded as a discretization of (3.1) using a numerical integration of Monte-Carlo type. Hence, the SgGN method proposed in the previous section can be applied directly for iteratively solving the discrete least-squares minimization problem in (5.1) by simply replacing integrals by summations. For the reader's convenience, we describe the corresponding components of the SgGN method.

To this end, let

$$u_n(\mathbf{x}) = \sum_{i=1}^n c_i \sigma_i(\mathbf{x}) + \alpha_0 = \sum_{i=1}^n c_i \sigma(\mathbf{w}_i \mathbf{x} + b_i) + \alpha_0$$

be a solution of (5.1), then

$$\hat{\mathbf{c}} = (c_1, \ldots, c_n, \alpha_0)^T \quad \text{and} \quad \mathbf{r}^T = \left(\mathbf{r}_1^T, \ldots, \mathbf{r}_n^T\right) = \left(b_1, \mathbf{w}_1^T, \ldots, b_n, \mathbf{w}_n^T\right)$$

are the linear and nonlinear parameters of $u_n(\mathbf{x})$, respectively. The blocks of $\mathcal{A}(\mathbf{r})$ and $\mathbf{f}(\mathbf{r})$ in (3.4) are given by

$$\mathcal{A}_{11}(\mathbf{r}) = \left[a_{ij}(\mathbf{r})\right]_{n\times n},\ \mathcal{A}_{12}(\mathbf{r}) = \left[b_i(\mathbf{r})\right]_{n\times 1},\ \mathcal{A}_{22} = \langle 1, 1\rangle_{m,\mu},\ \mathbf{f}_1(\mathbf{r}) = \left[f_i(\mathbf{r})\right]_{n\times 1},\ f_2 = \langle u, 1\rangle_{m,\mu},$$

where entries of the block matrices are defined by

$$\begin{cases} a_{ij}(\mathbf{r}) = \langle \sigma_j, \sigma_i\rangle_{m,\mu}, \ f_i(\mathbf{r}) = \langle u, \sigma_i\rangle_{m,\mu}, & \text{for } i, j = 1, \ldots, n, \\ b_i(\mathbf{r}) = \langle \sigma_i, 1\rangle_{m,\mu}, & \text{for } i = 1, \ldots, n, \end{cases}$$

respectively. The gradient vector of $\mathcal{J}_{m,\mu}(u_n)$ with respect to $\mathbf{r}$ is

$$\mathbf{G}(\mathbf{c},\boldsymbol{\alpha},\mathbf{r}) = \sum_{i=1}^{m} \mu(\mathbf{x}^i)\big(u_n(\mathbf{x}^i) - u(\mathbf{x}^i)\big)\mathbf{H}(\mathbf{x}^i) \otimes \mathbf{y}^i,$$

and the layer Gauss-Newton matrix is

$$\mathcal{H}(\mathbf{r}) = \sum_{i=1}^{m} \mu(\mathbf{x}^i)\big(\mathbf{H}(\mathbf{x}^i)\mathbf{H}^T(\mathbf{x}^i)\big) \otimes \big(\mathbf{y}^i(\mathbf{y}^i)^T\big).$$

The damping parameter γ_{k+1} is the solution of a one-dimensional optimization problem

$$\gamma_{k+1} = \underset{\gamma\in\mathbb{R}_0^+}{\arg\min} \mathcal{J}_{m,\mu}\left(u_n\big(\cdot;\mathbf{c}^{(k+1)},\mathbf{r}^{(k)} + \gamma\mathbf{p}^{(k+1)}\big)\right).$$

6. Numerical Experiments. In this section, a series of numerical experiments is conducted to test the effectiveness and accuracy of the proposed SgGN algorithm. We also compare it with several existing optimization algorithms commonly used for neural networks: Adam [13], BFGS [4, 10, 12, 22], and the Gauss-Newton-based KFRA method introduced in [2] which is considered more applicable than the original Gauss-Newton-based KFAC method [17].

Our test problems encompass various types of target functions, including step functions in one and two dimensions, a delta-like function in 1D, and a continuous piece-wise linear function in 2D. These functions are well-suited for accurate approximation using shallow neural networks. However, they pose significant challenges for optimization algorithms due to the presence of discontinuities or sharp transition layers. As indicated in [5, 15], the nonlinear parameters $\mathbf{r}_i = (b_i, \mathbf{w}_i^T)^T \in \mathbb{R}^{d+1}$ correspond to the breaking points/lines of the neurons which form a physical partition of the domain. Therefore, the efficiency of an optimization algorithm can be measured as the efficiency of moving those breaking points/lines from a uniform distribution to the nearly "optimal" locations according to the underlying target function.

In the comparison study, we used BFGS as a baseline. For each test, BFGS was first repeated 30 times, and we report the median loss and the corresponding approximation result. We then run the other two methods, KFAC and our SgGN, using the same number of iterations. For the first-order Adam optimizer, we run all test problems with a relatively larger number of iterations until the corresponding loss functions plateau.

It is important to note that different methods may have varying computation complexities per iteration. For instance, computational cost of BFGS [20] and the KFRA [2] is $O(n^2)$ per iteration, while cost of Adam is only $O(n)$. Our SgGN involves solutions of two dense linear system with coefficient matrices $\mathcal{A}(\mathbf{r}^{(k)})$ and $\mathcal{H}(\mathbf{r}^{(k)})$, respectively, as in Algorithm 4.1. These matrices are typically very ill conditioned for the test problems considered here. In our current implementation, truncated SVDs are used for the solution and its cost is $O(rn^2)$ with r depending on the accuracy. This suffices our purpose of comparing the convergence of the SgGN with the other methods. Although we use the number of iterations as one of the reference metrics for the efficiency performance of the optimization, our major focus for comparison in this study is the quality of the solution. As shown for the test problems, the SgGN can often converge to much more accurate approximation than the other methods.

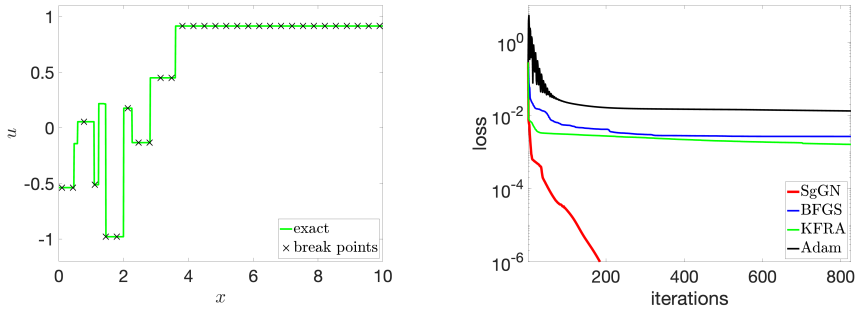
The detailed parameter setting for each method is listed in Table 6.1. All the methods start with the same initial as listed in the Initialization section in Table 6.1. The integration of the loss function $\mathcal{J}_\mu(u_n)$ is computed by the composite mid-point rule over a uniform partition with the mesh size $h = 0.01$.

6.1. One-dimensional piece-wise constant function. The first test problem is a one-dimensional piece-wise constant function defined in the interval $[0, 10]$ with ten pieces from a skewed

Table 6.1: List of parameters used in the methods, where the parameters for BFGS and Adam are referred to the MATLAB deep learning toolbox [18].

BFGS	
`net.trainParam.min_grad`	minimum performance gradient with value 0
`net.trainParam.max_fail`	maximum validation failures with value 10^4
`net.trainParam.epochs`	maximum number of epochs to train with value 10^4
KFRA	
γ	A damping parameter for the approximated Gauss-Newton matrix induced by the full Gauss-Newton
Adam	
`InitialLearnRate`	initial learning rate α_1
`DropRateFactor`	multiplicative factor α_f by which the learning rate drops
`DropPeriod`	number of epochs that passes between adjustments to the learning rate, denoted by T
Initialization	
Linear coefficient $\mathbf{c}$	initialized by a narrow normal distribution $\mathcal{N}(0, 0.01)$
nonlinear parameter $\mathbf{r}_i$	the corresponding breaking hyper-planes uniformly partition the domain

distribution (see Figure 6.1(a)). We use these skewed pieces to test whether an optimizer can move the uniformly initialized breaking points to catch those discontinuities in a target function. Theoretically, 20 neurons are enough to approximate a ten-piece step function with a given accuracy $\epsilon > 0$ [5, 6]. Due to the uncertainty of solving a non-convex optimization problem, 30 neurons are used in the test. For Adam method, we adjusted the learning rate for the best performance and the reported results are obtained using $\alpha_1 = 0.1$, $\alpha_f = 0.5$, and $T = 1000$. For the KFRA, $\gamma = 0.01$.



(a) Target function u and initial breaking points

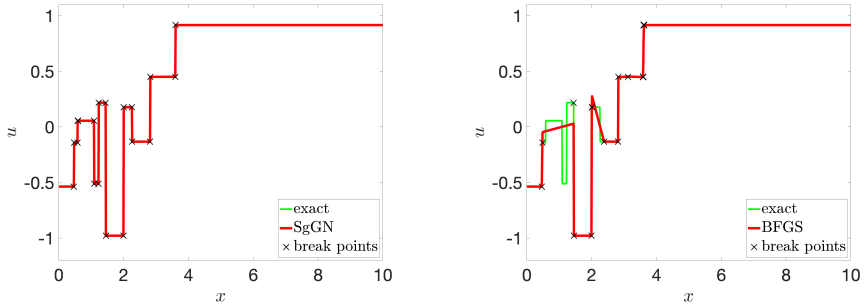
(b) Loss curves of the optimization methods

Fig. 6.1: One-dimensional piece-wise constant function approximation: target function, initial breaking points, and optimization loss curves.

The loss decay curves are depicted in Figure 6.1(b). While the loss curve for SgGN continues to decline even after 200 iterations, the curves for the other three methods decay slowly, reaching values close to their final training loss. Table 6.2 compares the least squares loss values of the

Table 6.2: Comparison for one-dimensional piece-wise constant function.

Method	SgGN		BFGS		KFRA	Adam
Iteration	**9**	825	207	825	825	10,000
$\mathcal{J}_{m,\mu}$	8.76E-4	**6.56E-9**	4.03E-3	2.65E-3	1.61E-3	8.14E-3



(a) SgGN u_n (b) BFGS u_n

(c) KFRA u_n (d) Adam u_n

Fig. 6.2: One-dimensional piece-wise constant function approximation results.

four methods. Notably, SgGN achieves a loss value of magnitude 10^{-9}, significantly lower than the 10^{-3} magnitude obtained by the other methods. Further insight into these loss comparisons can be derived from the approximation results shown in Figure 6.2. Specifically, only SgGN accurately captures all steps (Figure 6.2(a)), by precisely aligning the breaking points to the discontinuities. In contrast, the other methods (Figures 6.2(b) to 6.2(d)) either overlook certain discontinuous steps or induce overshooting.

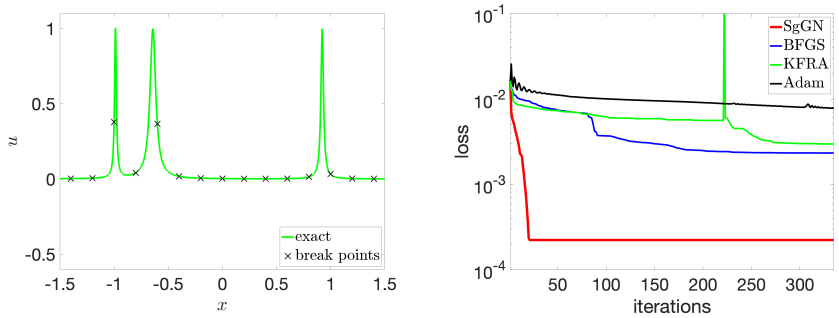
6.2. One-dimensional delta-like function. In the second experiment, we consider a smooth but sharp delta-like function [23]

$$u(x) = \sum_{i=1}^{k} \frac{1}{d_i(x - x_i)^2 + 1}, \quad x \in [-1.5, 1.5],$$

where k is the number of centers, x_i is the center position, and d_i is to control center width so that the larger d_i is, the narrower width is.

In the experiment, we set $k = 3$ with centers $\{x_1, x_2, x_3\} = \left\{-\frac{\pi^2}{10}, -(\pi - \frac{5}{2}), \frac{\sqrt{85}}{10}\right\}$, and width parameters $\{d_1, d_2, d_3\} = \left\{10^4, 10^3, 5 \times 10^3\right\}$. We employed 15 neurons in the first hidden layer for our tests. To evaluate the optimization methods, we initialized the neural network with uniformly distributed breaking points, as illustrated in Figure 6.3(a). Our primary objectives are twofold: first, to assess each optimization method's ability to accurately capture all three irrational peak centers from an rationally initialized breaking points and second, to determine whether the method can adaptively allocate the 15 breaking points to account for the differing widths of the centered delta-like peaks. For Adam, $\alpha_1 = 0.02$, $\alpha_f = 0.6$ and $T = 2000$, while in KFRA, $\gamma = 0.0001$.

As illustrated in Figure 6.3(b), the loss curve for SgGN not only decays more rapidly than those of the other three methods but also converges to a better solution within about 50 iterations. The SgGN method converges at a loss magnitude of 10^{-4}, which is substantially lower than the $10^{-3} \sim 10^{-2}$ magnitudes exhibited by the other methods (see Table 6.3). As further shown in Figure 6.4, SgGN successfully moves the breaking points to align with all center positions and adaptively distributes the remaining neuron breaking points to accurately approximate sharp peaks of varying widths. However, the other methods (see Figures 6.4(b) to 6.4(d)) fail to capture these nuances; they either fail to capture all peaks or do not distribute the breaking points proportionally.



(a) Target function u and initial breaking points

(b) Loss curves of four optimization methods

Fig. 6.3: One-dimensional delta-like function approximation: target function, initial breaking points, and optimization loss curves.

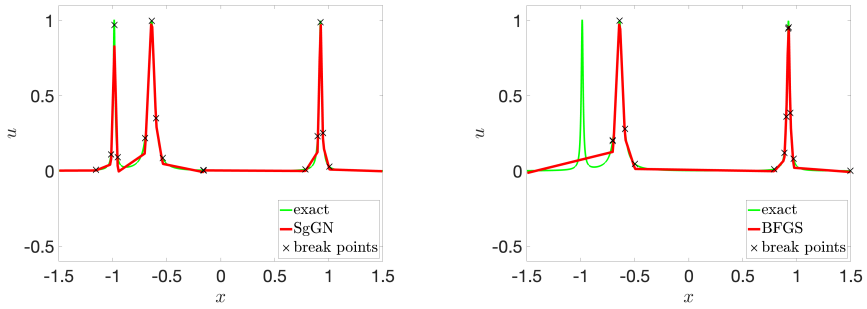
Table 6.3: Comparison for one-dimensional delta-like function.

Method	SgGN		BFGS		KFRA	Adam
Iteration	**12**	334	91	334	334	10,000
$\mathcal{J}_{m,\mu}$	2.24E-3	**2.23E-4**	3.74E-3	2.33E-3	2.97E-3	3.94E-3

6.3. Two-dimensional piece-wise constant function. Next, we consider a 2D piece-wise constant function defined in the domain $[-1, 1]^2$:

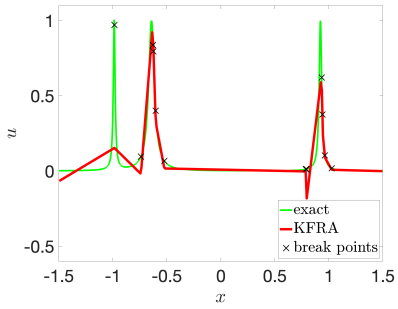
$$u(x) = \begin{cases} 1, & -0.5 \le x + y \le 0.5, \\ -1, & \text{otherwise.} \end{cases}$$

In the previous two examples, we used more neurons than the minimum required to mitigate the

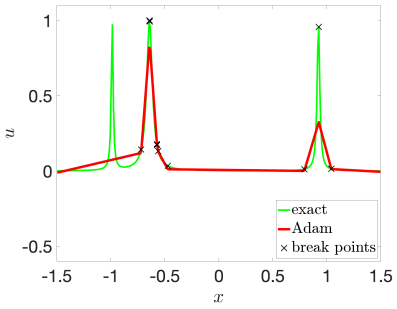


(a) SgGN u_n

(b) BFGS u_n



(c) KFRA u_n



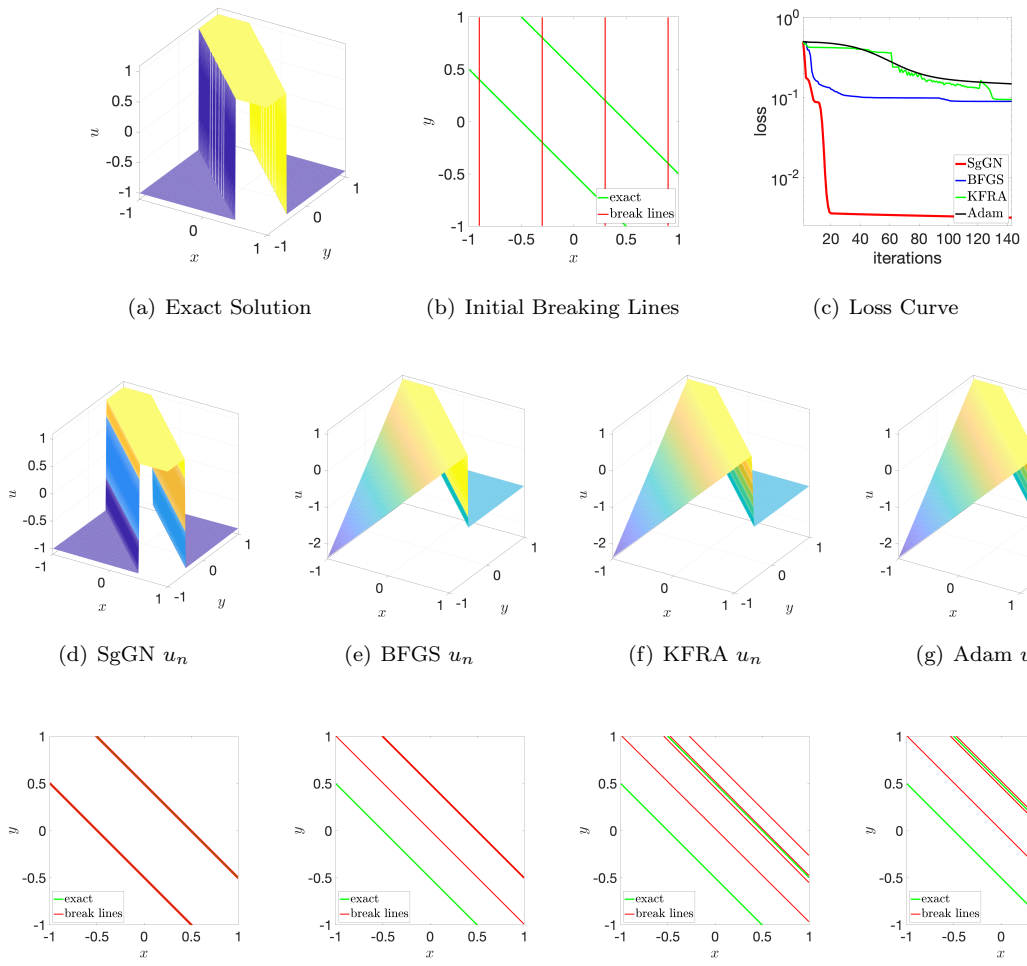
(d) Adam u_n

Fig. 6.4: One-dimensional delta-like function approximation: optimization loss curves and approximation results using four optimization methods.

uncertainties arising from the non-convex optimization problems. In contrast, this test focuses on utilizing only the minimum number of neurons necessary to compare the performances of different methods. As shown in Figure 6.5(a), each discontinuous segment can, in theory, be approximated using just two neurons. An effective approximation would place a pair of neurons for each discontinuous line, with the proximity of the corresponding breaking lines serving as a measure of the quality of the approximation. Consequently, we opt for using only 4 neurons in this example. For Adam method,$\alpha_1 = 0.01$, $\alpha_f = 0.8$ and $T = 2000$ and in KFRA, $\gamma = 0.005$.

As shown in Figure 6.5(c), the SgGN loss curve not only decays at a faster rate than those of the other three methods but also reaches the final training loss in approximately 20 iterations. Table 6.4 compares the least squares loss values after 142 iterations for the second-order methods and 10,000 iterations for Adam. Given the integration mesh size $h = 0.01$, there exists a theoretical lower bound on the proximity of the breaking lines, thus imposing a limit on the minimal achievable loss value. Even so, SgGN attains a loss magnitude of 10^{-3}, lower than the 10^{-2} magnitudes demonstrated by the other methods. Furthermore, as displayed in Figure 6.5. SgGN (see Figure 6.5(h)) accurately positioned all the four breaking lines to capture the discontinuities. In contrast, the other three methods (see Figures 6.5(i) and 6.5(k)) captured only one side of the discontinuity lines.

6.4. Two-dimensional function in $\hat{\mathcal{M}}_n(\Omega)$. For the previous three problems, the target functions do not reside within the defined network function space $\hat{\mathcal{M}}_n(\Omega)$. Consequently, the resultant loss function does not converge to zero. This non-zero convergence precludes a definitive



(a) Exact Solution (b) Initial Breaking Lines (c) Loss Curve

(d) SgGN u_n (e) BFGS u_n (f) KFRA u_n (g) Adam u_n

(h) SgGN Trained Breaking Lines (i) BFGS Trained Breaking Lines (j) KFRA Trained Breaking Lines (k) Adam Trained Breaking Lines

Fig. 6.5: Two-dimensional piece-wise constant function approximation: target function, initial breaking lines, optimization loss curves and approximation results using four optimization methods.

Table 6.4: Comparison for a two-dimensional piece-wise constant function.

Method	SgGN		BFGS		KFRA	Adam
Iteration	**9**	142	100	142	142	10,000
$\mathcal{J}_{m,\mu}$	8.82E-2	**3.16E-3**	9.20E-2	8.92E-2	9.40E-2	9.23E-2

405 determination of the optimal approximation for each target function within the specified function
406 space. To better assess the performance of parameter optimization, we introduce an artificial func-

tion from $\hat{\mathcal{M}}_n(\Omega)$ with randomly selected optimal parameters $\mathbf{c}^*$ and $\mathbf{r}^*$ that

$$u(\mathbf{x}) = \sum_{i=1}^{N} c_i^* \phi_i(\mathbf{x}; \mathbf{r}_i^*) + \alpha_0^*, \tag{6.1}$$

where N is number of neurons. In this case, the known optimal parameters $\mathbf{c}^*$ and $\mathbf{r}^*$ allow us to directly evaluate the performance of different optimization methods by tracking the movement of the breaking lines toward these optimal values. When too many neurons are provided, any initial positions of the breaking lines would naturally be close to some of the optimal ones. Moreover, with uniform initialization, some of the initial positions will be inherently close to the optimal solutions. To focus on the movement of the breaking lines, we opt for initialization along only horizontal and vertical axes, and we limit the number of neurons to 5. For Adam method, we have $\alpha_1 = 0.1$, $\alpha_f = 0.5$, $T = 2000$ for horizontal initialization, and $\alpha_1 = 0.1$, $\alpha_f = 0.8$, $T = 3000$ for vertical initialization respectively. In KFRA, $\gamma = 0.005$.

Table 6.5: Comparison for a two-dimensional piece-wise linear function with horizontal initial breaking lines (VI) and vertical initial breaking lines (VI).

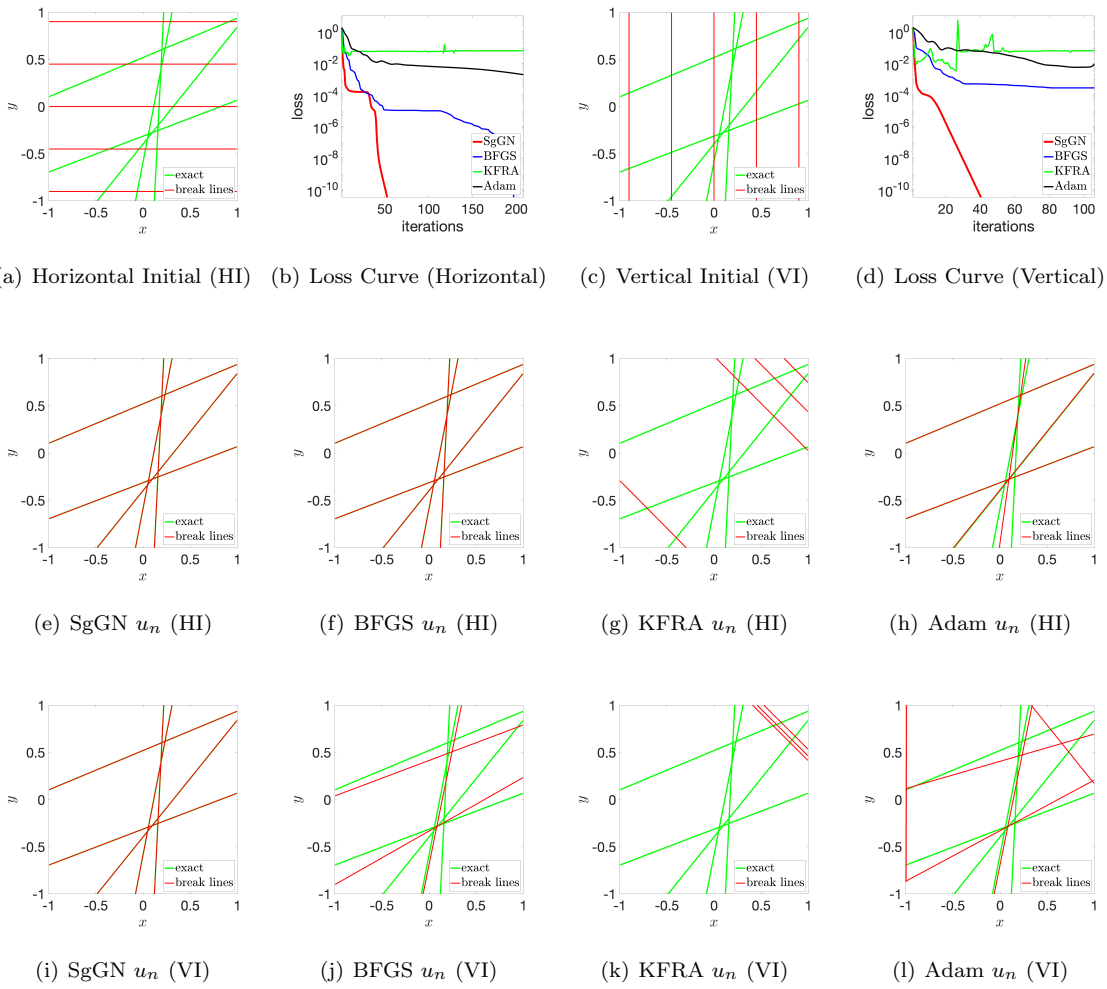
Method	SgGN		BFGS		KFRA	Adam
Iteration	**99**	207	204	207	207	10,000
$\mathcal{J}_{m,\mu}$ (HI)	6.28E-22	**6.68E-27**	7.50E-22	7.50E-22	6.12E-2	1.17E-5
Iteration	**4**	105	30	105	105	10,000
$\mathcal{J}_{m,\mu}$ (VI)	2.35E-4	**4.34E-26**	5.21E-4	2.71E-4	5.56E-2	2.15E-4

The loss decay curves are depicted in Figures 6.6(b) and 6.6(d). SgGN reaches the magnitude 10^{-10} within just 50 iterations while the other three methods decay more slowly, requiring a greater number of iterations to reach their final training loss. Table 6.5 compares the least squares loss values when initialized horizontally and vertically, respectively. Notably, SgGN excels by achieving a near-zero loss value in both scenarios, a level of accuracy significantly higher than what the other methods managed to accomplish. Further examination of the approximation results, as shown in Figure 6.6, reveals more nuanced insights. SgGN accurately moves all five breaking lines to their optimal positions under both horizontal and vertical initializations (see Figures 6.6(e) and 6.6(i)). On the other hand, while the other methods (Figures 6.6(f), 6.6(h), 6.6(j) and 6.6(l)) either perform well only with horizontal initializations or fail altogether to correctly position all the breaking lines.

6.5. Data science application. Lastly, we test an application of shallow network in data science. The task is to predict the age of abalones using physical measurements, drawing data from the UCI dataset [19]. Since there is no prior knowledge about network structure setting, we test two shallow networks where the numbers of neurons are 40 and 80 respectively. The parameters for Adam in the two tests are $\alpha_1 = 0.1$, $\alpha_f = 0.8$, $T = 1500$, and in KFRA, $\gamma = 1$.

Table 6.6: Comparison for the data science problem.

Method		SgGN		BFGS		KFRA	Adam
40 neurons	Iteration	**35**	200	50	200	200	10,000
	$\mathcal{J}_{m,\mu}$	1.90	1.90	2.09	**1.89**	2.59	2.02
80 neurons	Iteration	**9**	200	40	200	200	10,000
	$\mathcal{J}_{m,\mu}$	2.00	**1.70**	2.14	2.02	2.59	1.93



(a) Horizontal Initial (HI) (b) Loss Curve (Horizontal) (c) Vertical Initial (VI) (d) Loss Curve (Vertical)

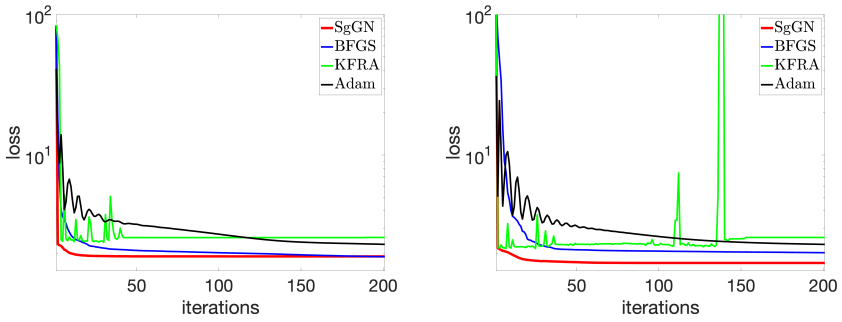
(e) SgGN u_n (HI) (f) BFGS u_n (HI) (g) KFRA u_n (HI) (h) Adam u_n (HI)

(i) SgGN u_n (VI) (j) BFGS u_n (VI) (k) KFRA u_n (VI) (l) Adam u_n (VI)

Fig. 6.6: Two-dimensional piece-wise linear function approximation: target function, initial breaking lines, optimization loss curves and approximation results using the optimization methods with horizontal initial (HI) and vertical initial (VI) breaking lines.

The loss decay curve for SgGN reaches a magnitude close to the final loss just within just 35 iterations(see Figure 6.7). In contrast, the loss curves for the other three methods decay more slowly, requiring a greater number of iterations to to reach their final training loss. Table 6.6 compares the least squares loss values after 200 iterations for the second-order methods and 10,000 iterations for Adam, with number of neurons 40 and 80 respectively. Both SgGN and BFGS have similar performance in the scenario with 40 neurons. However, SgGN demonstrates a better least squares loss than the other three methods with 80 neurons. This unequivocally attests to SgGN's superiority in terms of both the accuracy and effectiveness for this data science application.

7. Conclusions and Discussions. Newton's method in optimization is a second-order iterative method for numerically solving optimization problems with general objective functions. One



(a) 40 neurons (b) 80 neurons

Fig. 6.7: Data science application: optimization loss curves using the optimization methods.

of its variants, BFGS has been successfully applied to NN-based machine learning applications. For nonlinear least-squares problems, one may use methods of Gauss-Newton type that exploit the quadratic form of the objective function. The structured-guided Gauss-Newton (SgGN) method introduced in this paper is an iterative method for solving nonlinear least-squares problems using shallow ReLU NN as a model. In addition to the least squares structure aspect, SgGN method effectively makes use of the structure of the network. Guided by both structure types, the method has some attractive features. One feature is the guarantee of the positive definiteness of the mass and layer Gauss-Newton matrices without the need of extra shifting like in usual Gauss-Newton methods.

Another feature is the rapid convergence in practice. the SgGN method was tested for several one and two dimensional least-squares problems which are difficult for commonly used training algorithms in machine learning such as BFGS and Adam. The loss curves for all the test problems clearly show the superior convergence of SgGN. SgGN often out-performs those methods by a large margin. This conclusion is further strengthened by examining the ability and effectiveness of the methods in moving the breaking hyper-planes (breaking points for one dimension and breaking lines for two dimensions).

Each iteration of the SgGN requires linear solvers to approximately invert the mass matrix $\mathcal{A}(\mathbf{r}^{(k)})$ and the layer Gauss-Newton matrix $\mathcal{H}(\mathbf{r}^{(k)})$ for the linear and nonlinear parameters, respectively. While both the matrices are symmetric and positive definite, they are nevertheless ill-conditioned. In the numerical experiments reported in this paper, the truncated SVD is used as the linear solver, albeit at a significant computational cost. More efficient linear solvers will be investigated in an upcoming paper [7].

REFERENCES

[1] M. Ainsworth and Y. Shin, *Active neuron least squares: a training method for multivariate rectified neural networks*, SIAM Journal on Scientific Computing, 44 (2022), pp. A1807–C366.

[2] A. Botev, H. Ritter, and D. Barber, *Practical Gauss-Newton optimisation for deep learning*, in Int'l. Conf. Mach. Learning, PMLR, 2017, pp. 557–565.

[3] L. Bottou, F. E. Curtis, and J. Nocedal, *Optimization methods for large-scale machine learning*, SIAM Review, 60 (2018), p. 223–311.

[4] C. G. Broyden, *The convergence of a class of double-rank minimization algorithms 1. general considerations*, IMA J. Appl. Math., 6 (1970), pp. 76–90.

[5] Z. Cai, J. Chen, and M. Liu, *Least-squares ReLU neural network (LSNN) method for linear advection-reaction equation*, J. Comput. Phys., (443 (2021) 110514).

[6] Z. Cai, J. Choi, and M. Liu, *Least-squares neural network (LSNN) method for linear advection-reaction equation: general discontinuous interface*, arXiv:2301.06156v3[math.NA], (2023).
[7] Z. Cai, T. Ding, M. Liu, X. Liu, and J. Xia, *Mass and Gauss-Newton matrices for shallow ReLU neural network*, manuscript.
[8] Z. Cai and M. Liu, *Self-adaptive ReLU neural network method in least-squares data fitting*, Handbook of Research on Adaptive Artificial Intelligence, (submitted).
[9] J. E. Dennis Jr and R. B. Schnabel, *Numerical methods for unconstrained optimization and nonlinear equations*, SIAM, 1996.
[10] R. Fletcher, *A new approach to variable metric algorithms*, Computer J., 13 (1970), pp. 317–322.
[11] C. Gambella, B. Ghaddar, and J. Naoum-Sawaya, *Optimization problems for machine learning: a survey*, arXiv:1901.05331 [math.OC], (2019).
[12] D. Goldfarb, *A family of variable-metric methods derived by variational means*, Math. Comp., 24 (1970), pp. 23–26.
[13] D. P. Kingma and J. Ba, *ADAM: A method for stochastic optimization*, in Int'l Conf. Representation Learning, San Diego, 2015; arXiv preprint arXiv:1412.6980.
[14] K. Levenberg, *A method for the solution of certain non-linear problems in least squares*, Quart. Appl. Math., 2 (1944), pp. 164–168.
[15] M. Liu, Z. Cai, and J. Chen, *Adaptive two-layer ReLU neural network: I. best least-squares approximation*, Comput. Math. Appl., 113 (2022), pp. 34–44.
[16] D. W. Marquardt, *An algorithm for least-squares estimation of nonlinear parameters*, J. SIAM, 11 (1963), pp. 431–441.
[17] J. Martens and R. Grosse, *Optimizing neural networks with Kronecker-factored approximate curvature*, in Int'l. Conf. Mach. Learning, PMLR, 2015, pp. 2408–2417.
[18] The MathWorks Inc., *Deep Learning Toolbox*, 2023.
[19] W. Nash, T. Sellers, S. Talbot, A. Cawthorn, and W. Ford, *Abalone*. UCI Machine Learning Repository, 1995. DOI: https://doi.org/10.24432/C55C7W.
[20] J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer, 2006.
[21] J. M. Ortega and W. C. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables*, SIAM, 2000.
[22] D. F. Shanno, *Conditioning of quasi-Newton methods for function minimization*, Math. Comp., 24 (1970), pp. 647–656.
[23] J. Shen, Y. Wang, and J. Xia, *Fast structured direct spectral methods for differential equations with variable coefficients, i. the one-dimensional case*, SIAM J. Sci. Comput., 38 (2016), pp. A28–A54.
[24] S. Sun, Z. Cao, H. Zhu, and J. Zhao, *A survey of optimization methods from a machine learning perspective*, IEEE Trans. on Cybernetics, 50 (2020), pp. 3668 – 3681.