Article

Local Linear Approximation Algorithm for Neural Network

Mudong Zeng ^{1,†}, Yujie Liao ^{1,*}, Runze Li ¹ and Agus Sudjianto ²¹ Department of Statistics, Pennsylvania State University, University Park, PA 16802, USA; muz149@psu.edu (M.Z.); rzli@psu.edu (R.L.)² Corporate Model Risk, Wells Fargo Bank, Charlotte, NC 28202, USA; agus.sudjianto@wellsfargo.com

* Correspondence: yxl629@psu.edu

† These authors contributed equally to this work.

Abstract: This paper aims to develop a new training strategy to improve efficiency in estimation of weights and biases in a feedforward neural network (FNN). We propose a local linear approximation (LLA) algorithm, which approximates ReLU with a linear function at the neuron level and estimate the weights and biases of one-hidden-layer neural network iteratively. We further propose the layer-wise optimized adaptive neural network (LOAN), in which we use the LLA to estimate the weights and biases in the LOAN layer by layer adaptively. We compare the performance of the LLA with the commonly-used procedures in machine learning based on seven benchmark data sets. The numerical comparison implies that the proposed algorithm may outperform the existing procedures in terms of both training time and prediction accuracy.

Keywords: local linear approximation; fisher scoring; layerwise optimization

Citation: Zeng, M.; Liao, Y.; Li, R.; Sudjianto, A. Local Linear Approximation Algorithm for Neural Network. *Mathematics* **2022**, *10*, 494. <https://doi.org/10.3390/math10030494>

Academic Editor: Joaquim Fernando Pinto da Costa

Received: 14 December 2021

Accepted: 28 January 2022

Published: 3 February 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Despite that neural networks have many successful applications, there is a long-standing challenge of training feedforward neural network (FNN) models efficiently to strike a balance between training time and prediction performance. Stochastic gradient descent [1–3] is widely used for training neural networks, but it has a slow learning rate and noisy training curves in the beginning of the training epochs.

In order to reduce total training iterations and consequently training time, while achieving high generalization ability, many studies have developed different efficient training algorithms for neural networks, among which linear approximation of the neural network received lots of attentions. Douglas and Meng [4], Singhal and Wu [5] and Stan and Kamen [6] developed linearized least squares using Kalman recursion to update weight parameters. They demonstrated increased training efficiency of this technique compared to gradient descent. However, these methods lacked robustness in terms of model accuracy. Kollias and Anastassiou [7] developed an adaptive least squares algorithm based on the Marquardt-Levenberg optimization algorithm. See Ngia and Sjoberg [8], Kim and Lee [9] and Zhou and Si [10] for other Marquardt-Levenberg based methods. Moreover, Maddox et al. [11], Khan et al. [12] and Mu et al. [13] suggested linearizing neural network with first order Taylor expansion followed by Gaussian process for inference. However, these methods only worked on pre-trained networks but not random initialization [14].

The work in this paper aims to address the problem from a different point of view, and develop a reliable training algorithm called Local Linear Approximation (LLA) Algorithm. In each iteration, LLA uses the first order Taylor expansion to locally approximate activation functions of each neuron. It computes estimated gradient and Hessian matrix, and updates weights and biases in the spirit of Fisher scoring algorithm. Algorithms taking into account Fisher information or Hessian matrix have been proposed via natural gradient descent [15,16]. However, the proposed LLA is different from the existing algorithms. See Section 2.1 for more details. We develop LLA for both regression and

classification problems. The high-level summary of the major contributions of this work are shown below:

(a) We derive a novel adaptive method, LLA, which leverages the ideas of linearizing neural network and iterative Fisher scoring, aiming at reducing training time while achieving higher prediction performance than current baselines. Distinguished from existing methods that linearize neural networks, LLA has a unique initialization procedure.

(b) We investigate the convergence behavior of LLA by establishing a connection between the Fisher scoring algorithm and the iterative regression. The connection implies that LLA and Fisher scoring algorithms share the same rate of convergence. Thus, the LLA has a quadratic convergence rate.

(c) As an extension of the one-hidden-layer LLA, we built a layer-wise optimized adaptive neural network (LOAN) for further improvement. The performance of LLA with multiple hidden layers is examined. The experiments demonstrate that shallow LOAN achieves both stable performance and high prediction accuracy level.

(d) We use several benchmark data sets to illustrate and evaluate our proposed method. The results indicate that LLA has a faster convergence rate and outperforms existing optimization methods and machine learning models in terms of prediction on both training and testing data.

The rest of this paper is organized as follows. In Section 2, we develop the LLA algorithm for regression and classification problems and study the rate of convergence of the proposed LLA algorithm. Section 3 presents numerical comparisons. Conclusion and discussion are given in Section 4.

2. New Method for Constructing Neural Network

According to the universal approximation theorem [17,18], provided that the network has enough hidden units, an FNN with one-hidden-layer and a linear output layer may approximate any Borel measurable functions well (Section 6.4.1 in [19]). It is common to consider a deep neural network for better prediction performance, but in practice one-hidden-layer neural network with a large number of nodes can still achieve high prediction performance [20,21], which is an easy starting point to explain our proposed LLA method.

2.1. Regression Settings

Consider an one-hidden-layer neural network with J_1 nodes:

$$\mathbf{h}^{(1)} = \sigma(W\mathbf{h}^{(0)} + \mathbf{b}) = (\sigma(\mathbf{w}_1^T \mathbf{x} + b_1), \dots, \sigma(\mathbf{w}_{J_1}^T \mathbf{x} + b_{J_1}))^T, \quad (1)$$

where $\mathbf{h}^{(0)} = \mathbf{x}$ is the p -dimensional input vector, W and $\mathbf{b}$ are the weight matrix and the bias vector in the first hidden layer, respectively, $\mathbf{w}_j$ is the j -th row of W corresponding to the j -th node, and b_j is the j -th element of $\mathbf{b}$. Throughout this paper, we set $\sigma(t) = \max\{t, 0\}$, the ReLU activation function. Consider a linear regression model in the output layer

$$y = \beta_0 + \boldsymbol{\beta}^T \mathbf{h}^{(1)} + \varepsilon. \quad (2)$$

For this neural network, the parameters to be optimized are W , a $J_1 \times p$ weight matrix, $\mathbf{b}$, a J_1 -dimensional vector, $\boldsymbol{\beta} = (\beta_1, \dots, \beta_{J_1})^T$, a J_1 dimensional vector and β_0 , a one-dimensional parameter. Denote $\boldsymbol{\theta}_1 = \{\beta_0, \boldsymbol{\beta}\}$, $\boldsymbol{\theta}_2 = \{W, \mathbf{b}\}$ and $\boldsymbol{\theta} = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2\}$. Let $m(\mathbf{x}; \boldsymbol{\theta}) = \beta_0 + \boldsymbol{\beta}^T \mathbf{h}^{(1)}$. Suppose that we have a set of sample $\{\mathbf{x}_i, y_i\}$, $i = 1, \dots, n$. In the literature, the estimation of $\boldsymbol{\theta}$ is to minimize the least squares (LS) function

$$\ell_1(\boldsymbol{\theta}) = \sum_{i=1}^n \{y_i - m(\mathbf{x}_i, \boldsymbol{\theta})\}^2 = \sum_{i=1}^n \{y_i - \beta_0 - \sum_{j=1}^{J_1} \beta_j \sigma(\mathbf{w}_j^T \mathbf{x}_i + b_j)\}^2, \quad (3)$$

which is highly nonlinear in $\boldsymbol{\theta}_2$. Hence, parameter estimation is challenging for deep neural network.

To tackle these problems, we propose an algorithm to minimize (3). Given $\theta_2^{(c)} = \{W^{(c)}, b^{(c)}\}$ in the current step, we propose to approximate $\sigma(x^T w_j + b_j)$ by a linear function based on the first-order Taylor expansion of $\sigma(t)$:

$$\begin{aligned} \sigma(x^T w_j + b_j) &\approx \sigma(x^T w_j^{(c)} + b_j^{(c)}) \\ &+ \{(x^T w_j + b_j) - (x^T w_j^{(c)} + b_j^{(c)})\} \cdot I(x^T w_j^{(c)} + b_j^{(c)} > 0). \end{aligned} \quad (4)$$

We refer (4) to as local linear approximation (LLA) since it is linear approximation with first order Taylor expansion at the neuron level. For any function f , let $\tilde{f}$ be an approximation of f by (4) throughout the paper. Thus, $m(x; \theta)$ is approximated by:

$$\begin{aligned} \tilde{m}(x; \theta, \theta^{(c)}) &= \beta_0 + \sum_{j=1}^{J_1} \beta_j \sigma(x^T w_j^{(c)} + b_j^{(c)}) + \sum_{j=1}^{J_1} \gamma_j I(x^T w_j^{(c)} + b_j^{(c)} > 0) \\ &+ \sum_{j=1}^{J_1} \eta_j^T x I(x^T w_j^{(c)} + b_j^{(c)} > 0) \end{aligned} \quad (5)$$

where $\gamma_j = \beta_j(b_j - b_j^{(c)})$ and $\eta_j = \beta_j(w_j - w_j^{(c)})$. For $j = 1 \dots, J_1$, define

$$z_{1ij} = \sigma(x_i^T w_j^{(c)} + b_j^{(c)}), z_{2ij} = I(x_i^T w_j^{(c)} + b_j^{(c)} > 0), \quad (6)$$

and

$$z_{3ij} = x_i I(x_i^T w_j^{(c)} + b_j^{(c)} > 0). \quad (7)$$

Further define

$$z_{1i} = [z_{1i1}, \dots, z_{1iJ_1}]^T, z_{2i} = [z_{2i1}, \dots, z_{2iJ_1}]^T, z_{3i} = [z_{3i1}, \dots, z_{3iJ_1}]^T, \quad (8)$$

and

$$z_i = [1, z_{1i}^T, z_{2i}^T, z_{3i}^T]^T, \quad (9)$$

which is a $J_1(p+2)$ -dimensional vector. With the aid of approximation (5), the objective function in (3) is approximated by

$$\tilde{\ell}_1(\theta, \theta^{(c)}) = \sum_{i=1}^n \{y_i - \beta_0 - \sum_{j=1}^{J_1} \{\beta_j z_{1ij} + \gamma_j z_{2ij} + \eta_j^T z_{3ij}\}\}^2, \quad (10)$$

which is a least squares function of linear regression with the response y_i and predictors z_i .

Theorem 1. Let $\theta^{(c)}$ be the value of θ at the c -th step. Then

$$\begin{aligned} \theta^{(c+1)} &= \theta^{(c)} + [E\{-\ell_1''(\theta^{(c)})\}]^{-1} \ell_1'\{\theta^{(c)}\} \\ &= \theta^{(c)} + [E\{-\tilde{\ell}_1''(\theta^{(c)})\}]^{-1} \tilde{\ell}_1'\{\theta^{(c)}\}. \end{aligned} \quad (11)$$

The proof of Theorem 1 is given in Appendix A.1 of this paper. Theorem 1 implies that updating θ based on $\tilde{\ell}_1(\theta)$ (the second equation) is actually equivalent to updating θ based on $\ell_1(\theta)$ (the first equation). Let

$$\vartheta = [\beta_0, \beta^T, \gamma^T, \eta_1^T, \dots, \eta_{J_1}^T]^T, \quad (12)$$

where $\gamma = [\gamma_1, \dots, \gamma_{J_1}]^T$. Thus $\tilde{m}(x; \theta, \theta^{(c)}) = z_i^T \vartheta$ is a linear function of ϑ while it is nonlinear in θ . This enables us to update ϑ easily.

Denote the resulting least squares estimates of β_j , γ_j and η_j by $\hat{\beta}_j$, $\hat{\gamma}_j$ and $\hat{\eta}_j$, respectively. By the definition of γ_j and η_j , we can update b_j and w_j as follows

$$b_j^{(c+1)} = b_j^{(c)} + \hat{\gamma}_j / \hat{\beta}_j, \quad \text{and} \quad w_j^{(c+1)} = w_j^{(c)} + \hat{\eta}_j / \hat{\beta}_j, \quad (13)$$

for $j = 1, \dots, J_1$. If $|\hat{\beta}_j|$ is very close to zero, one may simply set $b_j^{(c+1)} = b_j^{(c)}$ and $w_j^{(c+1)} = w_j^{(c)}$. The LLA algorithm is to firstly update $\beta, \gamma, \eta_1, \dots, \eta_{J_1}$ by least squares estimates of Equation (10). Then, the estimates of W and b are obtained by Equation (13). The estimation process is conducted iteratively till convergence. The procedure can be summarized in Algorithm 1.

Motivated by single index models, we design an initialization procedure for W and b with J_1 nodes as follows. Firstly, we fit data $\{x_i, y_i\}$ with a linear regression model and obtain the least squares estimates for the coefficients of x , denoted by $\hat{\alpha}$, and the intercept, denoted by $\hat{\alpha}_0$. Let $\hat{\tau}_j, j = 1, \dots, J_1$ be quantile points of $\hat{\alpha}_0 + x_i^T \hat{\alpha}, i = 1, \dots, n$. We set initial value:

$$b_j^{(0)} = -\hat{\tau}_j \quad \text{and} \quad w_j^{(0)} = \hat{\alpha} \quad (14)$$

for $j = 1, \dots, J_1$. Since the objective function $\ell_1(\theta)$ is nonconvex, numerical minimization algorithm may depend on the initial value. We find that the proposed initialization strategy works reasonably well in practice. We have conducted some numerical comparison between this initialization procedure and the Xavier initialization strategy, which is based on randomization. Based on our simulation study, the proposed initialization procedure has a better and more stable performance than the Xavier initialization strategy.

Algorithm 1 Local Linear Approximation (LLA) Algorithm for Regression

Input: Given data $\{X, y\}$ and the number of nodes J_1

Initialization: Get initial values for $W^{(0)} = [w_1^{(0)}, \dots, w_{J_1}^{(0)}]^T$ and $b = (b_1^{(0)}, \dots, b_{J_1}^{(0)})^T$ in Equation (14)

Output: W and b

- 1: Let $c = 0$.
- 2: **while** the criterion of algorithm convergence is not satisfied **do**
- 3: calculate z_i based on $w_j^{(c)}$ and $b_j^{(c)}$, where z_i is defined in Equations (6)–(9), and obtain the least squares estimates (LSE) $\hat{\beta}_j$ s, $\hat{\gamma}_j$ s and $\hat{\eta}_j$ s by running a linear regression of y_i on covariate z_i according to Equation (10).
- 4: **if** $|\hat{\beta}_j| \geq \epsilon$ **then**
- 5: update the biases and weights by

$$\begin{aligned} b_j^{(c+1)} &\leftarrow b_j^{(c)} + \hat{\gamma}_j / \hat{\beta}_j \quad \text{and} \\ w_j^{(c+1)} &\leftarrow w_j^{(c)} + \hat{\eta}_j / \hat{\beta}_j, \end{aligned}$$

- 6: **else**
 - 7: $b_j^{(c+1)} \leftarrow b_j^{(c)}$ and $w_j^{(c+1)} \leftarrow w_j^{(c)}$
 - 8: **end if**
 - 9: **end while**
 - 10: **return** W and b
-

To understand the convergence behavior of the proposed LLA algorithm, let us briefly describe the Fisher scoring algorithm for a general nonlinear least squares problem. Let

$$\ell_*(\Theta) = \sum_{i=1}^n \{y_i - g(x_i; \Theta)\}^2$$

where $g(\mathbf{x}; \boldsymbol{\Theta})$ is a general nonlinear function with existing finite second-order derivatives with respect to $\boldsymbol{\Theta}$, a general parameter vector. Denote $\ell'_*(\boldsymbol{\Theta})$ and $\ell''_*(\boldsymbol{\Theta})$ to be the gradient and Hessian matrix of $\ell_*(\boldsymbol{\Theta})$, respectively. The Fisher scoring algorithm is to update $\boldsymbol{\Theta}$ by

$$\boldsymbol{\Theta}^{(c+1)} = \boldsymbol{\Theta}^{(c)} + [\hat{E}\{-\ell''_*(\boldsymbol{\Theta}^{(c)})\}]^{-1} \ell'_*(\boldsymbol{\Theta}^{(c)}),$$

where $\hat{E}\{-\ell''_*(\boldsymbol{\Theta})\}$ stands for an estimate of $E\{-\ell''_*(\boldsymbol{\Theta}^{(c)})\}$.

Similar to the proof of Theorem 1, we can rewrite $\boldsymbol{\Theta}^{(c+1)}$ into the following form

$$\begin{aligned} \boldsymbol{\Theta}^{(c+1)} &= \left\{ \sum_{i=1}^n \frac{\partial g(\mathbf{x}_i; \boldsymbol{\Theta}^{(c)})}{\partial \boldsymbol{\Theta}} \frac{\partial g(\mathbf{x}_i; \boldsymbol{\Theta}^{(c)})}{\partial \boldsymbol{\Theta}^T} \right\}^{-1} \\ &\cdot \sum_{i=1}^n \frac{\partial g(\mathbf{x}_i; \boldsymbol{\Theta}^{(c)})}{\partial \boldsymbol{\Theta}} \left[y_i - g(\mathbf{x}_i; \boldsymbol{\Theta}^{(c)}) + \frac{\partial g(\mathbf{x}_i; \boldsymbol{\Theta}^{(c)})}{\partial \boldsymbol{\Theta}^T} \boldsymbol{\Theta}^{(c)} \right], \end{aligned}$$

which is the solution of the least squares problem

$$\sum_{i=1}^n \left[y_i - g(\mathbf{x}_i; \boldsymbol{\Theta}^{(c)}) + \frac{\partial g(\mathbf{x}_i; \boldsymbol{\Theta}^{(c)})}{\partial \boldsymbol{\Theta}^T} \boldsymbol{\Theta}^{(c)} - \frac{\partial g(\mathbf{x}_i; \boldsymbol{\Theta}^{(c)})}{\partial \boldsymbol{\Theta}^T} \boldsymbol{\Theta} \right]^2. \quad (15)$$

We next show that (10) corresponds to (15) for $g(\mathbf{x}_i; \boldsymbol{\theta}_2)$ being $m(\mathbf{x}_i; \boldsymbol{\theta}) = \beta_0 + \sum_{j=1}^{J_1} \beta_j \sigma(\mathbf{w}_j^T \mathbf{x}_i + b_j)$ given $\boldsymbol{\theta}_1$.

Theorem 2. For given $\boldsymbol{\theta}_1$, let $g(\mathbf{x}_i; \boldsymbol{\theta}_2) = m(\mathbf{x}_i; \boldsymbol{\theta})$. Then (10) coincides with

$$\sum_{i=1}^n \left[y_i - g(\mathbf{x}_i; \boldsymbol{\theta}_2^{(c)}) + \frac{\partial g(\mathbf{x}_i; \boldsymbol{\theta}_2^{(c)})}{\partial \boldsymbol{\theta}_2^T} \boldsymbol{\theta}_2^{(c)} - \frac{\partial g(\mathbf{x}_i; \boldsymbol{\theta}_2^{(c)})}{\partial \boldsymbol{\theta}_2^T} \boldsymbol{\theta}_2 \right]^2. \quad (16)$$

The proof of Theorem 2 is given in Appendix A.2 of this paper.

Based on Theorem 2, we may show that the proposed LLA algorithm shares the same convergence rate of the Fisher scoring algorithm. Thus, the convergence rate of the proposed LLA algorithm is quadratic. The Fisher scoring algorithm has been proposed for FNN back in 1990s [22]. Recently, researchers use Fisher information or Hessian matrix information by constructing natural gradient descent [15,16,23]. A natural question which arises here is why the Fisher scoring algorithm is not implemented directly. We have compared the LLA algorithm and the Fisher scoring algorithm via numerical simulation. Our numerical comparison implies that the LLA algorithm is more stable and converges faster than the Fisher scoring algorithm. This may be because $\tilde{m}(\mathbf{x}_i; \boldsymbol{\theta}, \boldsymbol{\theta}^{(c)})$ is linear in $\boldsymbol{\theta}$, and (10) provides a better approximation than (15) for both $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$. In addition, the form of ReLU activation function $\sigma(t)$ implies that its first-order derivative $\sigma'(t) = I(t > 0)$ is bounded, and its second-order derivative is 0 if it exists. The LLA algorithm takes advantages of it since the LLA in (4) is based on $\sigma(t) \approx \sigma(t^{(c)}) + \sigma'(t^{(c)})(t - t^{(c)})$, the first-order Taylor expansion of $\sigma(t)$. Figure 1 presents an illustration of the LLA of ReLU activation function at three different locations.

Figure 1. Illustration of LLA in Equation (4). The LLA algorithm is to approximate $\sigma(t)$ by a local linear function iteratively, where t stands for a general argument and is replaced with $\mathbf{x}^T \mathbf{w}_j + b_j$ in the text. This figure illustrates the LLA of the ReLU activation function at points A, B and C with local linear functions around their neighborhood.

2.2. Classification

2.2.1. Binary Classification

For a classification problem, the output layer is

$$P(y_i = 1 | \mathbf{x}_i) = E(y_i | \mathbf{x}_i) = \frac{\exp\{m(\mathbf{x}_i; \boldsymbol{\theta})\}}{1 + \exp\{m(\mathbf{x}_i; \boldsymbol{\theta})\}},$$

where $m(\mathbf{x}; \boldsymbol{\theta}) = \beta_0 + \boldsymbol{\beta}^T \mathbf{h}^{(1)}$. Thus, training an FNN reduces to estimating $\boldsymbol{\theta}$ by maximizing the log-likelihood function

$$\ell_2(\boldsymbol{\theta}) = \sum_{i=1}^n [y_i m(\mathbf{x}_i; \boldsymbol{\theta}) - b\{m(\mathbf{x}_i; \boldsymbol{\theta})\}], \quad (17)$$

where $b(m) = \log\{1 + \exp(m)\}$. We approximate $\sigma(\mathbf{x}^T \mathbf{w}_j + b_j)$ in the same way as in (4) and approximate $\ell_2(\boldsymbol{\theta})$ similarly by:

$$\tilde{\ell}_2(\boldsymbol{\theta}, \boldsymbol{\theta}^{(c)}) = \sum_{i=1}^n [y_i \tilde{m}(\mathbf{x}_i; \boldsymbol{\theta}, \boldsymbol{\theta}^{(c)}) - b\{\tilde{m}(\mathbf{x}_i; \boldsymbol{\theta}, \boldsymbol{\theta}^{(c)})\}], \quad (18)$$

where $\tilde{m}(\mathbf{x}_i; \boldsymbol{\theta}, \boldsymbol{\theta}^{(c)})$ is defined in Equation (5). Let $\ell'_2(\boldsymbol{\theta})$ and $\ell''_2(\boldsymbol{\theta})$ be the gradient and Hessian matrix of $\ell_2(\boldsymbol{\theta})$, respectively.

According to Theorems 1 and 2 for regression in Section 2.1, extension to classification involves no fundamentally new ideas. Thus, let $\boldsymbol{\theta}^{(c)}$ be the value of $\boldsymbol{\theta}$ at the c -th step, $\boldsymbol{\theta}$ can be updated based on

$$\begin{aligned} \boldsymbol{\theta}^{(c+1)} &= \boldsymbol{\theta}^{(c)} + [E\{-\ell''_2(\boldsymbol{\theta}^{(c)})\}]^{-1} \ell'_2\{\boldsymbol{\theta}^{(c)}\} \\ &= \boldsymbol{\theta}^{(c)} + [E\{-\tilde{\ell}''_2(\boldsymbol{\theta}^{(c)})\}]^{-1} \tilde{\ell}'_2\{\boldsymbol{\theta}^{(c)}\}, \end{aligned}$$

and the convergence rate is quadratic as well.

This implies that we can iteratively update $\boldsymbol{\vartheta}$ using $\tilde{\ell}_2(\boldsymbol{\vartheta})$, in which $\boldsymbol{\vartheta}$ is defined similarly as in (12) and $\tilde{m}(\mathbf{x}; \boldsymbol{\theta}, \boldsymbol{\theta}^{(c)}) = \mathbf{z}_i^T \boldsymbol{\vartheta}$ is a linear function of $\boldsymbol{\vartheta}$. Thus, we can directly apply logistic regression of y_i on $\mathbf{z}_i$ and obtain an estimate of $\boldsymbol{\vartheta}$. To save computational cost, we use one-step iteration rather than a full iteration in the logistic regression. The initial values play an important role in the model performance since the log-likelihood function $\ell_2(\boldsymbol{\theta})$ is nonconcave. Similarly, we propose a method to construct initial values of $\mathbf{W}$ and $\mathbf{b}$ for classification. We fit data using a logistic regression model and obtain the

estimates of the intercept and the coefficients of x , denoted by $\hat{\alpha}_0$ and $\hat{\alpha}$, respectively. Let $\hat{\tau}_j$, $j = 1, \dots, J_1$ be quantile points of $\hat{\alpha}_0 + x_i^T \hat{\alpha}$, $i = 1, \dots, n$. We may set initial values:

$$b_j^{(0)} = -\hat{\tau}_j \text{ and } w_j^{(0)} = \hat{\alpha} \quad (19)$$

for $j = 1, \dots, J_1$. The procedure for updating β_0 , β , b_j and w_j is summarized in Algorithm 2.

Algorithm 2 Local Linear Approximation (LLA) Algorithm for Classification

Input: Given data $\{X, y\}$ and number of nodes J_1 .

Initialization: Get initial values for $W^{(0)} = [w_1^{(0)}, \dots, w_{J_1}^{(0)}]^T$ and $b = (b_1^{(0)}, \dots, b_{J_1}^{(0)})^T$ defined in Equation (19).

Output: W and b

- 1: Let $c = 0$.
- 2: **while** the criterion of algorithm convergence is not satisfied **do**
- 3: calculate z_i based on $w_j^{(c)}$ and $b_j^{(c)}$ where z_i is defined in Equations (6)–(9), and fit a logistic regression of y_i on z_i to obtain $\hat{\beta}_j$ s, $\hat{\gamma}_j$ s and $\hat{\eta}_j$ s according to Equation (18).
- 4: **if** $|\hat{\beta}_j^{(c)}| \geq \epsilon$ **then**
- 5: update the biases and weights by

$$b_j^{(c+1)} \leftarrow b_j^{(c)} + \hat{\gamma}_j / \hat{\beta}_j, \text{ and } w_j^{(c+1)} \leftarrow w_j^{(c)} + \hat{\eta}_j / \hat{\beta}_j,$$

- 6: **else**
 - 7: $b_j^{(c+1)} \leftarrow b_j^{(c)}$ and $w_j^{(c+1)} \leftarrow w_j^{(c)}$
 - 8: **end if**
 - 9: **end while**
 - 10: **return** W and b
-

2.2.2. Extension to Multi-Class Classification

The proposed procedures may be extended to multi-class classification problems. For classification problem with $T + 1$ classes, consider the following model

$$P(y = t | x) = \frac{\exp\{m_t(x; \theta_t)\}}{1 + \sum_{t=1}^T \exp\{m_t(x; \theta_t)\}},$$

and

$$P(y = T + 1 | x) = \frac{1}{1 + \sum_{t=1}^T \exp\{m_t(x; \theta_t)\}},$$

where $m_t(x; \theta_t) = \beta_{t0} + \beta_t^T \sigma(W_t x + b_t)$ with $\theta_t = \{\beta_{t0}, \beta_t, W_t, b_t\}$ for $t = 1, \dots, T$. Let $\theta^* = (\theta_1^T, \dots, \theta_T^T)$. Suppose that $\{x_i, y_i\}$, $i = 1, \dots, n$, is a set of samples from the model above. Denote $y_{ti} = I(y_i = t)$. Then, the log-likelihood function is

$$\ell_2(\theta^*) = \sum_{i=1}^n \left(\sum_{t=1}^T y_{ti} m_t(x_i; \theta_t) - \log \left[1 + \sum_{t=1}^T \exp\{m_t(x_i; \theta_t)\} \right] \right). \quad (20)$$

Note that for one-hidden-layer FNN, $\sigma(W_t x + b_t) = \{\sigma(x^T w_{t1} + b_{t1}), \dots, \sigma(x^T w_{tJ_1} + b_{tJ_1})\}^T$, where $W_t = (w_{t1}, \dots, w_{tJ_1})^T$ is the $J_1 \times p$ weight matrix and $b_t = (b_{t1}, \dots, b_{tJ_1})^T$ is the bias vector for class t . We may apply the same linear approximation in (4) to $\sigma(x^T w_{tj} + b_{tj})$ for all t . This results in a linear approximation $\tilde{m}_t(x; \theta_t, \theta_t^{(c)})$ of $m_t(x; \theta_t)$. Specifically, let

$$z_{t1ij} = \sigma(x_i^T w_{tj}^{(c)} + b_{tj}^{(c)}), \quad z_{t2ij} = I(x_i^T w_{tj}^{(c)} + b_{tj}^{(c)} > 0),$$

and

$$z_{t3ij} = \mathbf{x}_i I(\mathbf{x}_i^T \mathbf{w}_{tj}^{(c)} + b_{tj}^{(c)} > 0).$$

Then

$$\tilde{m}_t(x; \boldsymbol{\theta}_t, \boldsymbol{\theta}_t^{(c)}) = \beta_{t0} + \sum_{j=1}^{J_1} \{\beta_{tj} z_{t1ij} + \gamma_{tj} z_{t2ij} + \boldsymbol{\eta}_{tj}^T \mathbf{z}_{t3ij}\}$$

where $\gamma_{tj} = \beta_{tj}(b_{tj} - b_{tj}^{(c)})$ and $\boldsymbol{\eta}_{tj} = \beta_{tj}(\mathbf{w}_{tj} - \mathbf{w}_{tj}^{(c)})$. That is,

$$\tilde{m}_{th1}(x_i; \boldsymbol{\theta}_t, \boldsymbol{\theta}_t^{(c)}) = \mathbf{z}_{ti}^T \boldsymbol{\vartheta}_t$$

is a linear function of $\boldsymbol{\vartheta}_t$, where $\boldsymbol{\vartheta}_t = (\beta_{t0}, \boldsymbol{\beta}_t^T, \boldsymbol{\gamma}_t^T, \boldsymbol{\eta}_{t1}^T, \dots, \boldsymbol{\eta}_{tJ_1}^T)$ with $\boldsymbol{\beta}_t = (\beta_{t1}, \dots, \beta_{tJ_1})$, and $\boldsymbol{\gamma}_t = (\gamma_{t1}, \dots, \gamma_{tJ_1})^T$.

Replacing $m_t(x_i; \boldsymbol{\theta}_t)$ with $\tilde{m}_t(x_i; \boldsymbol{\theta}_t, \boldsymbol{\theta}_t^{(c)})$, we may update $\boldsymbol{\theta}_t$ easily by using similar strategy described in Section 2.2. Thus, Algorithms 2 can be used to iteratively update $\beta_{t0}, \boldsymbol{\beta}_t, W_t$ and $\mathbf{b}_t$. Similarly, we may extend the LLA proposed in Section 2.2 to FNN for multi-class classification problem.

2.3. Layer-Wise Optimized Adaptive Neural Network

In practice, one-hidden-layer neural network with a large number of nodes may approximate regression functions well and lead to good prediction performance in many applications. Sometimes, it is still of interest to examine the performance of deep neural network for further improvement. We consider regression problem as an example in this subsection, and propose a layer-wise optimized adaptive neural network (LOAN) as an extension of one-hidden layer LLA. The LLA may be extended to estimate the weights and biases in the LOAN layer by layer adaptively.

Consider an L -hidden layer FNN:

$$\mathbf{h}^{(l)} = \sigma_l(W_l \mathbf{h}^{(l-1)} + \mathbf{b}_l) \quad (21)$$

for $l = 1, \dots, L$ with $\mathbf{h}^{(0)} = \mathbf{x}$, the p -dimensional input vector, where $\sigma_l(\cdot)$ s are activation functions, applied to each component of output from the previous layer. W_l and $\mathbf{b}_l$ are the weight matrix and the bias vector in the l -th hidden layer. Consider a linear regression model in the output layer

$$y = \beta_0 + \boldsymbol{\beta}^T \mathbf{h}^{(L)} + \varepsilon. \quad (22)$$

Denote $f(x; \boldsymbol{\theta}) = \beta_0 + \boldsymbol{\beta}^T \mathbf{h}^{(L)}$, where $\boldsymbol{\theta} = \{\beta_0, \boldsymbol{\beta}, W_1, \dots, W_L, \mathbf{b}_1, \dots, \mathbf{b}_L\}$.

The back-propagation with gradient decent algorithm is to seek optimal weights and biases in all layers of FNN via minimizing $\sum_{i=1}^n \{y_i - f(x_i, \boldsymbol{\theta})\}$. This leads to a high-dimensional, nonconvex minimization problem. In this subsection, we propose LOAN for deep FNN. Layerwise construction of neural network has been proposed in the literature. Contrastive divergence [24] trains Boltzmann machines through approximate likelihood gradient with sampling Gibbs chain [25–28]. The decoupled neural interface [29] updates weight matrix using synthetic gradient, a gradient approximation, estimated based on local information and trains each layer asynchronously. More recent work on this topic include Chen et al. [30], Shin [31], and Song et al. [32].

The idea of LOAN can be summarized as follows. We start with $\mathbf{h}^{(0)} = \mathbf{x}$, fit the data with one-hidden-layer network as described in Section 2.1, and obtain estimates $\hat{W}$ and $\hat{\mathbf{b}}$ of W and $\mathbf{b}$. We set $W_1 = \hat{W}$ and $\mathbf{b}_1 = \hat{\mathbf{b}}$ for the weight matrix and bias vector in FNN. Then we define $\tilde{x}_i = \mathbf{h}_i^{(1)} = \sigma(W_1 \mathbf{h}_i^{(0)} + \mathbf{b}_1)$. We fit data $\{\tilde{x}_i, y_i\}$ with one-hidden-layer network, and obtain $\hat{W}$ and $\hat{\mathbf{b}}$ by the proposed LLA. Then we set $W_2 = \hat{W}$ and $\mathbf{b}_2 = \hat{\mathbf{b}}$. Therefore, by running one-hidden-layer network on data $\{\tilde{x}_i, y_i\}$ with $\tilde{x}_i = \mathbf{h}_i^{(l-1)}$, we construct a LOAN by estimating W_l s and $\mathbf{b}_l$ s. This procedure can be summarized in Algorithm 3.

Algorithm 3 Layerwise Optimized Adaptive Network (LOAN)

Input: LOAN structure $[J_1, \dots, J_L]$ with J_l being the number of nodes for the l -th layer.

Initialization: Obtain the estimates of W_1 and b_1 in the first hidden layer by fitting the data $\{x_i, y_i\}$ with one-hidden-layer LLA in Algorithm 1.

Set $h_i^{(1)} = \sigma(W_1 x_i + b_1)$.

- 1: **for** $l = 1, \dots, L - 1$ **do**
- 2: set $\tilde{x}_i = h^{(l)}$,
- 3: obtain estimated W_{l+1} and b_{l+1} by fitting the data $\{\tilde{x}_i, y_i\}$ with one-hidden-layer LLA in Algorithm 1,
- 4: set $h_i^{(l+1)} = \sigma(W_{l+1} \tilde{x}_i + b_{l+1})$.
- 5: **end for**

As described in Algorithm 3, the proposed LOAN is distinguished from the existing layer-wise deep neural networks [28,30–32] in that LOAN builds the deep FNN layer by layer rather than treating the objective function as a function of weights and biases in all layers simultaneously.

3. Numerical Comparison

In this section, we compare the performance of the proposed LLA with commonly-used procedures in machine learning. To be specific, we shall compare LLA algorithm with Adam and Adamx [33]. Furthermore, we will compare the proposed LLA model with the following popular machine learning models (regressors or classifiers).

1. MLP multi-layer perceptron [34,35] (python scikit-learn version 0.22.1)
2. XGBoost [36] (python version 1.2.0)
3. GBM gradient boosting algorithm [35,37] (python scikit-learn version 0.22.1)
4. AdaBoost [35,38] (python scikit-learn version 0.22.1),
5. RF random forest [35,39] (python scikit-learn version 0.22.1),

Codes for the numerical studies can be found at <https://github.com/zengmudong/Local-Linear-Approximation-Algorithm-for-Neural-Network>, accessed on 15 January 2022.

3.1. Regression Problems

Firstly, we compare the performance of LLA with commonly-used procedures for regression problems based on four benchmark data sets. The stability constant ϵ in the Algorithm 1 is set to be 10^{-3} in this subsection.

(1) Airfoil (AF, for short) data, which consists of $n = 1503$ observations with $p = 5$ predictors, can be downloaded from <https://archive.ics.uci.edu/ml/datasets/Airfoil+Self-Noise> (accessed on 15 January 2022). The response is the scaled sound pressure level. The explanatory variables include frequency, angle of attack, chord length, free-stream velocity, suction side displacement thickness.

(2) Bikesharing (BS, for short) data, which consists of $n = 17,379$ observations with $p = 8$ predictors and can be downloaded from <https://archive.ics.uci.edu/ml/datasets/bike+sharing+dataset> (accessed on 15 January 2022). The response is the log-transformed count. The explanatory variables include normalized hour, normalized temperature, normalized humidity, normalized wind speed, season, holiday, weekday, and weather situation.

(3) California house price (CHP, for short) data [40], which consists of $n = 20,640$ observations and $p = 8$ predictors, can be downloaded from Carnegie-Mellon *StatLib* repository (<http://lib.stat.cmu.edu/datasets/> (accessed on 15 January 2022)). The response variable is the median house value. The explanatory variables include longitude, latitude, housing median age, medium income, population, total rooms, total bedrooms, and households.

(4) Parkinson (PK, for short) data, which consists of $n = 5874$ observations and $p = 19$ predictors, can be downloaded from <https://archive.ics.uci.edu/ml/datasets/Parkinsons+Telemonitoring> (accessed on 15 January 2022). The response variable is the total UPDRS

scores, and the explanatory variables include subject age, subject gender, time interval from baseline recruitment date, and 16 biomedical voice measures.

In the numerical comparison, we first standardize the response and predictors so that their sample means and variances equal 0 and 1, respectively. We split each data set into 80% training data and 20% testing data with seed $0, \dots, 29$. Thus, we may obtain 30 MSEs for testing data and 30 MSEs for training data for each procedure. We report the sample mean and standard deviation (std) of the 30 MSEs for each procedure and data set. In addition to MSE, other measures such as $\sum_{i=1}^N |y_i - \hat{y}_i| / \sum_{i=1}^N |y_i - \bar{y}_i|$ have also been considered. The patterns for other measures are similar to those with MSE.

3.1.1. Impact of the Number of Nodes for Regression Problems

As illustrated in Figure 2, we investigate the performance of LLA with different number of nodes for testing and training data sets.

Figure 2. Box plots of 30 MSEs of one-hidden-layer LLA with $J = 30, 35, \dots, 100$ nodes for training and testing data. The left panel is for the AF data set and the right panel is for the CHP data set.

The left and right panels of Figure 2 depict the box-plots of MSEs for AF and CHP data sets, respectively. For CHP data, the MSE of training data decreases as the number of nodes increases, while the MSE of testing data decreases firstly and then remains at the same level as the number of nodes increases. This is expected since as the number of nodes increases, the corresponding neural network may be overfitted, which may lead to an increase of prediction error. For AF data set, the MSE of training and testing data decreases at first, but then increases as the number of nodes increases, reaching its minimum when the number of nodes is around 60. The difference in the pattern of training MSE between AF and CHP data in Figure 2 might be due to different sample sizes of AF data ($n = 1503$) and CHP data ($n = 20,640$).

Experiments of these two data sets suggest that LLA yields the best prediction performance when the number of nodes in one-hidden-layer neural network is 60. Thus, we set 60 as the default number of nodes for LLA regressor.

3.1.2. Comparison with Stochastic Gradient Algorithms

In this subsection, we compare the performance of LLA with Adam and Adamax [33], two popular stochastic gradient algorithms.

We consider default settings for these three algorithms. The default number of nodes in LLA is 60 by default, as discussed in Section 3.1.1. For the Adam algorithm, the default step size is $\alpha = 0.001$, exponential decay rates for the moment estimates are $\beta_1 = 0.9$, $\beta_2 = 0.999$, the constant for numerical stability is $\epsilon = 10^{-8}$, and moving averages are initialized as

zero vectors. The default settings for Adamax algorithm are the same as those of Adam, except that the step size is set to be $\alpha = 0.002$.

Figure 3 depicts the MSEs of training and testing data versus the number of epochs for each algorithm. From Figure 3, the MSEs of LLA algorithm become stable quickly after a few epochs. Compared with the Adam and the Adamax, the LLA algorithm needs fewer epochs to achieve stability. When the MSEs become stable, the LLA has the least MSEs among these three algorithms. This is expected because for a nonlinear least squares problem, the LLA algorithm utilizes both gradient and Hessian of the objective function in the minimization problem. In contrast, the Adam and Adamax algorithms are stochastic gradient descent algorithms, which only take advantage of the gradient of the objective function. Thus, they converge comparatively slowly.

Figure 3. MSE of LLA, Adam and Adamax algorithms versus the number of epochs.

3.1.3. Performance of LLA in Regression Settings

Next, we compare the performance of LLA with other commonly used procedures. All hyperparameters are set by default. However, the performance of procedures such as MLP might be further improved by extensively tuning their hyper-parameters. Table 1 depicts the values of 100 times of the sample mean and standard deviation of the 30 MSEs of testing data (testing MSE for short) as well as the 30 MSEs of training data (training MSE for short). Table 1 implies that XGBoost performs better than other two boosting methods: AdaBoost and GBM, and the LLA performs well in terms of prediction accuracy. Specifically, the LLA has the lowest testing MSEs for AF and BS data sets. For CHP, the LLA has slightly greater testing MSE than XGBoost, while XGBoost has a deeper tree by default. For AF, BS, and CHP data sets, the difference between testing MSE and training MSE is small for LLA. This implies that the LLA is less likely to be over-fitting. As illustrated in Figure 4, for AF, BS, and CHP data sets, the prediction MSE of MLP and GBM are likely to have large variation for different data splittings and thus more potential outliers. LLA prediction has relatively small standard deviation of MSEs and fewer points outside the interquartile range (IQR) of boxplots, compared to other procedures. This implies that LLA is likely to obtain consistent results and has a fairly stable performance. For PK data, LLA outperforms MLP in terms of both prediction accuracy and standard deviation of MSEs. However, XGBoost and RF have the best performance. This motivates us to examine the data further.

Table 1. Comparison with commonly-used regressors with default hyper-parameter values.

Methods	AF 100× Mean (Std)	BS 100× Mean (Std)	CHP 100× Mean (Std)	PK 100× Mean (Std)
MSE for Testing Data				
LLA	4.3317 (0.5895)	6.7291 (0.2957)	18.5801 (0.6955)	22.8113 (5.4049)
MLP	25.4913 (3.4600)	9.9173 (2.0827)	24.1495 (1.0754)	31.4595 (2.3832)
XGBoost	5.1042 (0.7245)	7.8751 (0.3137)	16.4491 (0.6905)	3.7880 (0.4386)
AdaBoost	31.7242 (2.6197)	27.7419 (0.7384)	56.3744 (5.3267)	60.9171 (1.8358)
GBM	14.7338 (1.5846)	16.5968 (0.5592)	21.2866 (0.7254)	21.0340 (0.7685)
RF	6.5254 (0.7346)	10.2843 (0.4672)	21.2067 (0.9217)	2.8041 (0.4835)
MSE for Training Data				
LLA	2.3213 (0.1634)	5.3265 (0.0828)	15.7216 (0.1275)	11.7173 (0.8164)
MLP	24.5284 (0.0268)	7.7488 (0.7496)	23.7055 (1.1570)	23.7447 (1.3502)
XGBoost	0.2623 (0.0268)	4.3713 (0.0946)	5.7634 (0.1456)	0.3115 (0.0264)
AdaBoost	29.2881 (1.1124)	27.2715 (0.6059)	55.8631 (5.5463)	60.4119 (1.1948)
GBM	11.2280 (0.6303)	15.8681 (0.1972)	19.5493 (0.1864)	19.0881 (0.7516)
RF	0.9366 (0.0388)	1.4413 (0.0179)	3.8432 (0.0766)	0.3897 (0.0356)

Figure 4. Comparison with commonly-used procedures with default hyper-parameter value for regression problems.

By doing exploratory data analysis of PK data, we find that several of the 19 predictors in PK data are highly correlated. For example, the correlation between Shimmer and Shimmer(dB) is 0.9923. The correlation between Jitter(%) and Jitter(RAP) is 0.9841. The poor performance of the LLA may be due to the collinearity of predictors since the linear regression is applied for updating the weights and biases. Thus, we conduct a principal component analysis (PCA), and the first 6, 9, and 13 principal components (PC) can explain 95%, 99%, and 99.9% of the variance. We further apply all procedures to PK data with predictors being set to PCs. Table 2 depicts the MSEs for PK data with PC predictors. Compared with Table 1, the performance of XGBoost and RF becomes poor. Tables 1 and 2 imply that the performance of RF and XGBoost are not robust under linear transformation of predictors. The performance of MLP is quite stable across different numbers of PC variables. The performance of the LLA improves significantly via using PCs.

Table 2. Comparison of the impact of PCA on LLA and existing procedures based on PK data.

Methods	6 PCs 100×Mean (Std)	9 PCs 100×Mean (Std)	13 PCs 100×Mean (Std)	All (19) PCs 100×Mean (Std)
MSE for Testing Data				
LLA	23.1102 (2.1758)	9.3551 (0.9400)	10.2220 (1.1961)	22.2247 (3.0623)
MLP	41.7293 (2.6673)	34.7943 (2.3511)	30.6869 (2.1018)	30.6617 (2.9831)
XGBoost	50.8823 (3.2462)	47.1367 (2.7843)	42.9631 (2.6408)	44.4751 (2.1000)
AdaBoost	81.2930 (2.6109)	78.9303 (2.7339)	76.2363 (2.2727)	76.5351 (2.5682)
GBM	67.0713 (2.3618)	62.4927 (2.2980)	59.1466 (1.8239)	59.7928 (1.9610)
RF	41.2433 (2.8571)	39.6862 (2.6231)	37.7282 (2.5413)	39.9511 (2.5392)
MSE for Training Data				
LLA	17.0868 (1.2977)	5.8743 (0.3081)	5.8837 (0.5092)	11.7854 (0.8391)
MLP	37.1915 (1.8439)	27.9013 (0.9417)	22.0252 (0.9036)	20.5078 (0.9063)
XGBoost	10.0009 (0.7694)	6.6056 (0.4240)	4.0770 (0.3162)	2.9568 (0.2952)
AdaBoost	79.1353 (2.3975)	76.3020 (2.1195)	73.5840 (2.0781)	73.5073 (2.3712)
GBM	59.1133 (1.5346)	53.7506 (1.1314)	49.5714 (0.9834)	49.4500 (0.9870)
RF	5.9341 (0.2602)	5.6795 (0.1998)	5.3804 (0.2432)	5.6393 (0.2185)

3.2. Classification

In this section, we compare the performance of LLA classifier with other commonly-used procedures based on the following three data sets.

(1) Ringnorm (RN, for short) data consists of $n = 7400$ observations with $p = 20$ predictors and can be downloaded from <https://www.cs.toronto.edu/~dave/data/ringnorm/desc.html> (accessed on 15 January 2022). This data set has been analyzed in [41]. Each class is drawn from a multivariate normal distribution. Breiman [41] reports the theoretical expected misclassification rate as 1.3%.

(2) Magic gamma (MG, for short) data consists of $n = 19,020$ observations with $p = 11$ predictors and can be downloaded from <https://archive.ics.uci.edu/ml/datasets/MAGIC+Gamma+Telescope> (accessed on 15 January 2022). The data are Monte Carlo generated to simulate registration of high energy gamma particles in a ground-based atmospheric Cherenkov gamma telescope using the imaging technique [42]. The binary response is whether the event is a signal or a background event. The explanatory variables include fLength (major axis of ellipse), fLength (minor axis of ellipse), fSize (10-log of sum of content of all pixels), fSize (ratio of sum of two highest pixels over fSize), fConc1 (ratio of highest pixel over fSize), fAsym (distance from highest pixel to center projected onto major axis), etc.

(3) Mammographic mass (MM, for short) data consists of $n = 800$ observations with missing values deleted and $p = 4$ predictors. It can be downloaded from <https://archive.ics.uci.edu/ml/datasets/Mammographic+Mass> (accessed on 15 January 2022). The response is severity (benign or malignant). The explanatory variables include age, shape, margin, and density.

We first standardize the continuous predictors so that their sample means and variances equal 0 and 1, respectively. We split each data set into 80% training data and 20% testing data with seed $0, \dots, 29$. Thus, we may obtain 30 AUC-ROC (AUC-ROC is the area under the receiver operating characteristic curve, which plots two parameters true positive rate and false positive rate. It provides an aggregate measure of performance of a classifier across all possible classification thresholds.) (AUC for short hereafter) scores for testing data and 30 AUC scores for training data for each procedure. We report the sample mean and standard deviation (std) of the 30 AUC scores for each procedure and data set.

3.2.1. Impact of the Number of Nodes for Classification Problems

We firstly investigate the impact of the number of nodes on the performance of LLA classifier in terms of AUCs for testing and training data sets. The number of nodes ranges from 15 to 70 in the LLA.

Figure 5 depicts the box-plots of AUCs for RN and MM data, respectively. For RN data set, the AUCs of training and testing data remain close to 1 across different number of nodes, and reach the maximum at approximately 30. As shown in Figure 5, when the number of nodes is small, LLA algorithm has a stable performance. As the number of nodes increases, it is likely to incur overfitting problem, especially for RN data set, leading to estimates with larger variance. For MM data, the AUCs of training data first increase and then keep constant when the number of nodes is chosen to be from 50 to 70. For the testing data, when the number of nodes varies from 15 to 35, the AUCs are close with each other. As the number of nodes increases, the AUCs for the testing data decrease and keep the same at approximately 0.85. When the number of nodes is too large, overfitting issue is likely to happen. As illustrated from these two data sets, when the number of nodes is set to be 30, the corresponding LLA classifier yields good performance in terms of prediction. Therefore, we set 30 as the default value in our numerical experiments for classification problems.

Figure 5. Box plots of 30 AUCs of LLA algorithm with nodes being set to $J = 15, 20, \dots, 70$ for training and testing data. The top panel is the AUCs of RN data set, and the bottom panel is the AUCs of MM data set.

3.2.2. Comparison with Stochastic Gradient Algorithms

We then examine the convergence rate of the LLA algorithm in comparison with Adam and Adamax, two well-known stochastic gradient algorithms for neural networks. To make

a fair comparison among the LLA, Adam and Adamax, we set $J_1 = 30$ in one-hidden-layer network for all these three algorithms. The default settings are used for the three algorithms. Specifically, the default step size for Adam is $\alpha = 0.001$, the exponential decay rates for the moment estimates $\beta_1 = 0.9$, $\beta_2 = 0.999$. The constant for numerical stability is set to be $\epsilon = 10^{-8}$. And the moving averages are initialized to be zero vectors. For Adamax, the step size is set to be $\alpha = 0.002$.

Figure 6 depicts the AUC scores of training and testing data versus the number of epochs for each algorithm. As seen from Figure 6, the AUC curves of the LLA algorithm increase and then become stable within 20 epochs for both RN and MG data sets. Both Adam and Adamax need many more epochs to have stable AUC values. For example, both Adam and Adamax need 150 epochs to have stable AUC values for RN data. This implies that the LLA converges faster than Adam and Adamax, consistent with our theoretical analysis. The LLA also performs better than the Adam and Adamax in terms of AUC score. This can be clearly seen from the AUC plots for the MG data set.

Figure 6. AUCs of LLA, Adam and Adamax algorithms versus the number of epochs.

3.2.3. Performance of LLA in Classification Problems

We next compare the performance of the LLA classifier with other procedures with default settings. The number of nodes in the LLA for RN, MM and MG data sets are 30. The number of max epochs is set to be 100. Table 3 depicts the values of 100 times of the sample mean and standard deviations of the 30 AUC scores for the testing data (testing AUC for short) as well as those for the training data (training AUC for short). Table 3 implies that the LLA performs the best for RN and MM data sets in terms of AUC scores, followed by MLP. Figure 7 compares the AUC scores of different methods more explicitly. As shown in Figure 7, all procedures have small standard deviation for the testing data. However, for the training data, RF has a large standard deviation for the RN data while XGBoost, GBM, and Adaboost have more fluctuations for the MM data. The LLA has small standard deviations for all data sets, illustrating that the LLA retains stability. The differences between the testing AUCs and training AUCs are small for the LLA classifier, which shows that it is less likely for the LLA classifier to be over-fitting. For MG data, the RF performs the best, followed by XGBoost. This motivates us to examine the data further by considering multi-layer network, which will be discussed in Section 3.4 in details.

Table 3. Comparison with commonly-used classifiers with default hyper-parameters value.

Methods	RN	MM	MG
	100 × Mean (Std)	100 × Mean (Std)	100 × Mean (Std)
AUC Scores for Testing Data			
LLA	99.8007 (0.0750)	86.4075 (2.3068)	90.7903 (0.6989)
MLP	99.7777 (0.0856)	86.3424 (2.1636)	93.0479 (0.3572)
XGBoost	99.7641 (0.0723)	82.8425 (2.4301)	93.4792 (0.4691)
AdaBoost	99.4795 (0.1202)	85.6732 (2.3398)	89.4802 (0.5192)
GBM	99.6859 (0.0793)	85.2137 (2.5059)	92.4214 (0.5086)
RF	99.5045 (0.1249)	83.2001 (1.9674)	93.5235 (0.4453)
AUC Scores for Training Data			
LLA	99.9705 (0.0072)	89.3458 (0.5252)	90.9545 (0.3217)
MLP	100.0000 (0.0000)	89.4176 (0.4736)	94.1618 (0.1429)
XGBoost	100.0000 (0.0000)	96.3674 (0.2275)	99.4716 (0.0675)
AdaBoost	99.8049 (0.0199)	88.7022 (0.5184)	90.4314 (0.1327)
GBM	99.9389 (0.0097)	92.7635 (0.4055)	93.8208 (0.0975)
RF	100.0000 (0.0000)	97.3740 (0.2280)	100.0000 (0.0000)

Figure 7. Box plots of 30 AUCs for the LLA and the five commonly-used procedures with default hyper-parameter values for classification problems.

3.3. Analysis of Computational Time

In this section, we compare the computational time of LLA with other commonly used procedures. As shown in Figure 3, LLA converges faster than ADAM algorithm in terms of training epochs. Each training epoch of LLA takes longer time than Adam and Adamax, though. To make fair comparison on computational time, we use the ‘tensorflow.keras.Sequential’ to build one-hidden-layer neural networks with Adam and Adamax optimizers. Similar to the Monte Carlo experiment settings used for Tables 1 and 3, we randomly split the data into two parts (80% for training and 20% for testing), and repeat the process for 30 times to calculate the mean and standard deviation. For regression problems, models contain 60 nodes (by default) in the hidden layer and are forced to run 1000 epochs. For classification problems, models contain 30 nodes (by default) in the hidden layer and are forced to run 150 epochs. The experiments are run on a personal computer (3.1 GHz Intel Core i5 processor) with Python 3.8 and TensorFlow 2.4.0. Table 4 shows the sample

means along with standard deviations (in parenthesis) of testing MSE (100 times of the actual values), and training time (in seconds). The results show that although LLA takes longer time in training 1000/150 epochs, it can achieve higher accuracy in both regression and classification models. Moreover, as shown in Figures 3 and 6, LLA converges in fewer epochs than Adam or Adamax, demonstrating that if early stopping is applied, computational time of LLA is comparable to the time of Adam and Adamax. Consider AF data set as an example. If LLA stops at 220 epochs, its average and standard deviation of training time will be 55.3 and 9.2 s, meaning that the computational time of LLA is close to Adam's and Adamax's. For classification problems, LLA can also converge in fewer epochs than Adam or Adamax as shown in Figure 6.

Table 4. Training time and accuracy (testing MSE/AUC for regression/classification problems) of LLA, Adam, and Adamax. For AF and CHP (regression) data sets, the training epoch is fixed at 1000. For RN and MG (classification) data sets, the training epoch is fixed at 150. MSE(std) and AUC(std) in this table denote 100 times of the average(standard deviation) of MSE/AUC values over epochs. Time is measured in seconds.

Data		LLA	Adam	Adamax
AF	MSE (std)	4.6 (0.6)	25.2 (4.0)	24.3 (3.2)
	Time (std)	207.3 (12.6)	58.8 (5.1)	49.3 (5.2)
CHP	MSE (std)	18.8 (0.7)	24.9 (1.2)	23.8 (0.1)
	Time (std)	998.9 (190.1)	407.3 (80.6)	397.2 (77.4)
RN	AUC (std)	99.8 (0.1)	99.6 (0.2)	99.6 (0.0)
	Time (std)	101.2 (1.7)	26.8 (1.4)	24.1 (0.9)
MG	AUC (std)	91.8 (0.7)	91.9 (0.0)	91.7 (0.1)
	Time (std)	122.8 (2.3)	68.0 (2.9)	59.1 (3.9)

3.4. Performance of Deep LOAN

In this subsection, we examine the performance of the LOAN with multiple-hidden layers. Since there is no option to set different depth levels for AdaBoost, we exclude it for this comparison. We set the MLP, XGBoost, GBM, and RF with the same number of layer or depth as L , the number of hidden layers used in the LLA. Other parameters of the MLP, XGBoost, GBM and RF are set to be the default values. For regression problem, we apply these four procedures and LLA to CHP data. For classification problem, we investigate MG data in details.

For regression problems, Table 5 depicts the sample mean and standard deviation of the 30 training and testing data splittings for $L = 1, 2$ and 3, based on CHP data set. The number of nodes in the one-hidden-layer LOAN is set to be 60. The number of nodes in the LOAN with two hidden layer is set to be 60 and 10 in the first and second layer, respectively. The number of nodes in the LOAN with three hidden layers is set to be 60, 10, and 50 in the first, second and third layer, respectively. From Table 5, we can see that both the LOAN and MLP have smaller MSE as L increases, but there is no dramatic improvement, while XGBoost, GBM, and RF seem to have more improvement as L increases. Table 5 indicates that LOAN performs the best when $L = 1, 2$ and 3 in terms of mean and standard deviation of both training and testing MSEs.

For classification problems, Table 6 depicts the sample means and standard deviations of AUC scores for different values of L based on MG data set. For LOAN with different number of layers, 30 nodes is chosen for each layer. Compared with GBM and RF, LLA, MLP, and XGBoost perform better. As the number of layer increases, the improvement of LOAN is dramatic. Although LLA does not perform the best when only one hidden layer is used, it achieves the highest AUC scores when L is set to be 2 and 3. The standard deviations for LOAN continue to decrease as L increases for both testing and training data, illustrating that deep LLA leads to a more stable performance in terms of prediction for this data set.

These experiments demonstrate that shallow LOAN achieves both stable performance and high prediction accuracy level compared to commonly-used deep learning models.

Table 5. Comparison of the number of hidden layers in LOAN and existing regressors based on CHP data.

Methods	$L = 1$	$L = 2$	$L = 3$
	$100 \times \text{Mean (Std)}$	$100 \times \text{Mean (Std)}$	$100 \times \text{Mean (Std)}$
MSE for Testing Data			
LOAN	18.580 (0.696)	18.016 (0.674)	17.524 (0.653)
MLP	22.538 (0.967)	20.627 (0.896)	20.232 (0.876)
XGBoost	28.537 (0.777)	20.796 (0.661)	18.340 (0.652)
GBM	35.532 (0.869)	24.591 (0.745)	21.287 (0.726)
RF	66.745 (1.496)	53.697 (1.151)	43.940 (1.096)
MSE for Training Data			
LOAN	15.722 (0.128)	14.365 (0.243)	12.555 (0.157)
MLP	21.272 (0.656)	18.353 (0.472)	16.737 (0.543)
XGBoost	27.909 (0.183)	19.150 (0.242)	15.127 (0.167)
GBM	35.161 (0.196)	23.762 (0.210)	19.549 (0.186)
RF	66.707 (0.609)	53.538 (0.333)	43.483 (0.329)

Table 6. Comparison of the number of hidden layers in LOAN and existing classifiers based on MG data.

Methods	$L = 1$	$L = 2$	$L = 3$
	$100 \times \text{Mean (Std)}$	$100 \times \text{Mean (Std)}$	$100 \times \text{Mean (Std)}$
AUC for Testing Data			
LLA	90.7903 (0.6989)	93.8978 (0.3857)	94.3385 (0.3556)
MLP	92.6980 (0.3915)	93.0171 (0.4185)	92.8342 (0.5058)
XGBoost	89.5843 (0.6247)	92.4452 (0.4989)	93.2186 (0.4459)
GBM	87.8336 (0.6711)	91.4000 (0.5723)	92.4214 (0.5087)
RF	82.2163 (1.1715)	86.5378 (0.6972)	87.6666 (0.6656)
AUC for Training Data			
LLA	90.9545 (0.3217)	95.2309 (0.1458)	96.9006 (0.1088)
MLP	93.3201 (0.1632)	94.7166 (0.1804)	95.6310 (0.2325)
XGBoost	90.2653 (0.1400)	93.8429 (0.1111)	95.6249 (0.0967)
GBM	88.2751 (0.1475)	92.2619 (0.1215)	93.8208 (0.0975)
RF	82.4167 (0.6175)	86.8213 (0.2831)	88.0429 (0.1949)

4. Conclusions and Discussion

In this paper, we develop the LLA algorithm in an FNN for both regression and classification problems, and show that the LLA has a quadratic convergence rate. Our comparison based on benchmark data sets indicates that the LLA can greatly improve training efficiency and outperforms commonly-used procedures in both training time and prediction accuracy. This algorithm is generally applicable, and can be extended to deep neural networks via layer-wise or back-propagation construction. Exciting future work could further extend the proposed method in different studies such as survival data analysis. Parallel computing algorithm can also be developed for the proposed method.

Author Contributions: R.L. and A.S. proposed the topic and framework. M.Z. and Y.L. implemented the algorithm and conducted real data analysis. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data analyzed in Sections 3.1 and 3.2 are publicly available.

Acknowledgments: The authors are grateful to the reviewers for constructive comments, which lead to a significant improvement of this work. The views expressed in the paper are those of the authors and do not represent the views of Wells Fargo.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

This appendix consists of the proofs of Theorems 1 and 2.

Appendix A.1. Proof of Theorem 1

Proof. Recall $m(\mathbf{x}_i; \boldsymbol{\theta}) = \beta_0 + \sum_{j=1}^{J_1} \beta_j \sigma(\mathbf{x}_i^T \mathbf{w}_j + b_j)$. Thus, the corresponding least squares problem is

$$\ell_1(\boldsymbol{\theta}) = \sum_{i=1}^n \{y_i - m(\mathbf{x}_i; \boldsymbol{\theta})\}^2.$$

The Fisher scoring algorithm is to update

$$\boldsymbol{\theta}^{(c+1)} = \boldsymbol{\theta}^{(c)} + \left[\mathbb{E}\{-\ell_1''(\boldsymbol{\theta}^{(c)})\} \right]^{-1} \ell_1'(\boldsymbol{\theta}^{(c)}).$$

The gradient and Hessian matrix of the least squares function can be derived

$$\begin{aligned} \ell_1'(\boldsymbol{\theta}) &= -2 \sum_{i=1}^n \frac{\partial m(\mathbf{x}_i; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \{y_i - m(\mathbf{x}_i; \boldsymbol{\theta})\} \\ \ell_1''(\boldsymbol{\theta}) &= -2 \sum_{i=1}^n \left[\frac{\partial^2 m(\mathbf{x}_i; \boldsymbol{\theta})}{\partial \boldsymbol{\theta} \partial \boldsymbol{\theta}^T} \{y_i - m(\mathbf{x}_i; \boldsymbol{\theta})\} - \frac{\partial m(\mathbf{x}_i; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \frac{\partial m(\mathbf{x}_i; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}^T} \right]. \end{aligned}$$

Thus,

$$\mathbb{E}\{-\ell_1''(\boldsymbol{\theta})\} = -2 \sum_{i=1}^n \mathbb{E}\left[\frac{\partial m(\mathbf{x}_i; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \frac{\partial m(\mathbf{x}_i; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}^T} \right],$$

and a natural estimate of $\mathbb{E}\{-\ell_1''(\boldsymbol{\theta})\}$ is

$$\hat{\mathbb{E}}\{-\ell_1''(\boldsymbol{\theta})\} = -2 \sum_{i=1}^n \frac{\partial m(\mathbf{x}_i; \boldsymbol{\theta}^{(c)})}{\partial \boldsymbol{\theta}} \frac{\partial m(\mathbf{x}_i; \boldsymbol{\theta}^{(c)})}{\partial \boldsymbol{\theta}^T}$$

Then

$$\boldsymbol{\theta}^{(c+1)} = \boldsymbol{\theta}^{(c)} + \left[\sum_{i=1}^n \frac{\partial m(\mathbf{x}_i; \boldsymbol{\theta}^{(c)})}{\partial \boldsymbol{\theta}} \frac{\partial m(\mathbf{x}_i; \boldsymbol{\theta}^{(c)})}{\partial \boldsymbol{\theta}^T} \right]^{-1} \cdot \left[\sum_{i=1}^n \frac{\partial m(\mathbf{x}_i; \boldsymbol{\theta}^{(c)})}{\partial \boldsymbol{\theta}} \{y_i - m(\mathbf{x}_i; \boldsymbol{\theta}^{(c)})\} \right]. \quad (\text{A1})$$

By using Fisher scoring algorithm, we do not need to evaluate $\partial^2 m(\mathbf{x}_i; \boldsymbol{\theta}) / \partial \boldsymbol{\theta} \partial \boldsymbol{\theta}^T$, which typically is hard to calculate. The LLA is to approximate $m(\mathbf{x}_i; \boldsymbol{\theta})$ by

$$\begin{aligned} \tilde{m}(\mathbf{x}_i; \boldsymbol{\theta}, \boldsymbol{\theta}^{(c)}) &= \beta_0 + \sum_{j=1}^{J_1} \beta_j \sigma(\mathbf{x}_i^T \mathbf{w}_j^{(c)} + b_j^{(c)}) \\ &\quad + \sum_{j=1}^{J_1} \beta_j (b_j - b_j^{(c)}) I(\mathbf{x}_i^T \mathbf{w}_j^{(c)} + b_j^{(c)} > 0) \\ &\quad + \sum_{j=1}^{J_1} \beta_j (\mathbf{w}_j - \mathbf{w}_j^{(c)})^T \mathbf{x}_i I(\mathbf{x}_i^T \mathbf{w}_j^{(c)} + b_j^{(c)} > 0). \end{aligned}$$

Note that

$$\begin{aligned}\frac{\partial m(x_i; \boldsymbol{\theta})}{\partial \beta_0} \Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(c)}} &= 1, \\ \frac{\partial m(x_i; \boldsymbol{\theta})}{\partial \beta_j} \Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(c)}} &= \sigma(\mathbf{x}_i^T \mathbf{w}_j^{(c)} + b_j^{(c)}) \\ \frac{\partial m(x_i; \boldsymbol{\theta})}{\partial b_j} \Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(c)}} &= \beta_j^{(c)} I\{\mathbf{x}_i^T \mathbf{w}_j^{(c)} + b_j^{(c)} > 0\}, \\ \frac{\partial m(x_i; \boldsymbol{\theta})}{\partial w_j} \Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(c)}} &= \beta_j^{(c)} \mathbf{x}_i I\{\mathbf{x}_i^T \mathbf{w}_j^{(c)} + b_j^{(c)} > 0\}\end{aligned}$$

and

$$\begin{aligned}\frac{\partial \tilde{m}(x_i; \boldsymbol{\theta}, \boldsymbol{\theta}^{(c)})}{\partial \beta_0} \Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(c)}} &= 1, \\ \frac{\partial \tilde{m}(x_i; \boldsymbol{\theta}, \boldsymbol{\theta}^{(c)})}{\partial \beta_j} \Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(c)}} &= \sigma(\mathbf{x}_i^T \mathbf{w}_j^{(c)} + b_j^{(c)}), \\ \frac{\partial \tilde{m}(x_i; \boldsymbol{\theta}, \boldsymbol{\theta}^{(c)})}{\partial b_j} \Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(c)}} &= \beta_j^{(c)} I\{\mathbf{x}_i^T \mathbf{w}_j^{(c)} + b_j^{(c)} > 0\}, \\ \frac{\partial \tilde{m}(x_i; \boldsymbol{\theta}, \boldsymbol{\theta}^{(c)})}{\partial w_j} \Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(c)}} &= \beta_j^{(c)} \mathbf{x}_i I\{\mathbf{x}_i^T \mathbf{w}_j^{(c)} + b_j^{(c)} > 0\}\end{aligned}$$

Thus,

$$\begin{aligned}\frac{\partial m(x_i; \boldsymbol{\theta})}{\partial \beta_0} \Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(c)}} &= \frac{\partial \tilde{m}(x_i; \boldsymbol{\theta}, \boldsymbol{\theta}^{(c)})}{\partial \beta_0} \Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(c)}} \\ \frac{\partial m(x_i; \boldsymbol{\theta})}{\partial \beta_j} \Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(c)}} &= \frac{\partial \tilde{m}(x_i; \boldsymbol{\theta}, \boldsymbol{\theta}^{(c)})}{\partial \beta_j} \Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(c)}} \\ \frac{\partial m(x_i; \boldsymbol{\theta})}{\partial b_j} \Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(c)}} &= \frac{\partial \tilde{m}(x_i; \boldsymbol{\theta}, \boldsymbol{\theta}^{(c)})}{\partial b_j} \Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(c)}} \\ \frac{\partial m(x_i; \boldsymbol{\theta})}{\partial w_j} \Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(c)}} &= \frac{\partial \tilde{m}(x_i; \boldsymbol{\theta}, \boldsymbol{\theta}^{(c)})}{\partial w_j} \Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(c)}}\end{aligned}$$

Therefore, the Fisher scoring algorithm for the least squares function based on the local linear approximation

$$\tilde{\ell}_1(\boldsymbol{\theta}) = \sum_{i=1}^n \left\{ y_i - \tilde{m}(x_i; \boldsymbol{\theta}, \boldsymbol{\theta}^{(c)}) \right\}^2$$

has the same updated $\boldsymbol{\theta}^{(c+1)}$ as given in (A1). This completes the proof of Theorem 1. $\square$

Appendix A.2. Proof of Theorem 2

Proof. By definition of z_{1ij} , it follows that

$$\begin{aligned}g(x_i, \boldsymbol{\theta}_2^{(c)}) &= f(x_i; \boldsymbol{\theta}_1, \boldsymbol{\theta}_2^{(c)}) \\ &= \beta_0 + \sum_{j=1}^{J_1} \beta_j \sigma(\mathbf{x}_i^T \mathbf{w}_j^{(c)} + b_j^{(c)}) \\ &= \beta_0 + \sum_{j=1}^{J_1} \beta_j z_{1ij}.\end{aligned}$$

Furthermore, it follows that for $j = 1, \dots, J_1$,

$$\frac{\partial g(x_i; \theta_2^{(c)})}{\partial b_j} = \beta_j I(x_i^T w_j^{(c)} + b_j^{(c)} > 0) = \beta_j z_{2ij}$$

$$\frac{\partial g(x_i; \theta_2^{(c)})}{\partial w_j} = \beta_j x_i I(x_i^T w_j^{(c)} + b_j^{(c)} > 0) = \beta_j z_{3ij}$$

Then

$$\frac{\partial g(x_i; \theta_2^{(c)})}{\partial b_j} (b_j - b_j^{(c)}) = \beta_j (b_j - b_j^{(c)}) z_{2ij} = \gamma_j z_{2ij}$$

by definitions of γ_j and z_{2ij} . Similarly, we have

$$\frac{\partial g(x_i; \theta_2^{(c)})}{\partial w_j^T} (w_j - w_j^{(c)}) = \beta_j (w_j - w_j^{(c)}) z_{3ij} = \eta_j^T z_{3ij}$$

by definitions of η_j and z_{3ij} . Thus, (16) equals

$$\sum_{i=1}^n \{y_i - \beta_0 - \sum_{j=1}^{J_1} \beta_j z_{1ij} - \sum_{j=1}^{J_1} \gamma_j z_{2ij} - \sum_{j=1}^{J_1} \eta_j^T z_{3ij}\}^2$$

which is (10) exactly. This completes the proof. $\square$

References

1. Gardner, W.A. Learning characteristics of stochastic-gradient-descent algorithms: A general study, analysis, and critique. *Signal Process.* **1984**, *6*, 113–133. [\[CrossRef\]](#)
2. Oymak, S.; Soltanolkotabi, M. Overparameterized nonlinear learning: Gradient descent takes the shortest path? In Proceedings of the 36th International Conference on Machine Learning (ICML), PMLR, Long Beach, CA, USA, 9–15 June 2019; Volume 97, pp. 4951–4960.
3. Ali, A.; Dobriban, E.; Tibshirani, R. The implicit regularization of stochastic gradient flow for least squares. In Proceedings of the 37th International Conference on Machine Learning (ICML), PMLR, Virtual, 13–18 July 2020; pp. 233–244.
4. Douglas, S.; Meng, T.Y. Linearized least-squares training of multilayer feedforward neural networks. In Proceedings of the IJCNN-91-Seattle International Joint Conference on Neural Networks, Seattle, WA, USA, 8–12 July 1991; Volume 1, pp. 307–312. [\[CrossRef\]](#)
5. Singhal, S.; Wu, L. Training multilayer perceptrons with the extended kalman algorithm. *Adv. Neural Inf. Process. Syst.* **1988**, *1*, 133–140.
6. Stan, O.; Kamen, E. A local linearized least squares algorithm for training feedforward neural networks. *IEEE Trans. Neural Netw.* **2000**, *11*, 487–495. [\[CrossRef\]](#) [\[PubMed\]](#)
7. Kollias, S.; Anastassiou, D. An adaptive least squares algorithm for the efficient training of artificial neural networks. *IEEE Trans. Circuits Syst.* **1989**, *36*, 1092–1101. [\[CrossRef\]](#)
8. Ngia, L.S.; Sjöberg, J. Efficient training of neural nets for nonlinear adaptive filtering using a recursive Levenberg-Marquardt algorithm. *IEEE Trans. Signal Process.* **2000**, *48*, 1915–1927. [\[CrossRef\]](#)
9. Kim, C.T.; Lee, J.J. Training two-layered feedforward networks with variable projection method. *IEEE Trans. Neural Netw.* **2008**, *19*, 371–375.
10. Zhou, G.; Si, J. Advanced neural-network training algorithm with reduced complexity based on Jacobian deficiency. *IEEE Trans. Neural Netw.* **1998**, *9*, 448–453. [\[CrossRef\]](#) [\[PubMed\]](#)
11. Maddox, W.; Tang, S.; Moreno, P.; Wilson, A.G.; Damianou, A. Fast adaptation with linearized neural networks. In Proceedings of the 24th International Conference on Artificial Intelligence and Statistics, PMLR, Virtual, 13–15 April 2021; Volume 130, pp. 2737–2745.
12. Khan, M.E.; Immer, A.; Abedi, E.; Korzepa, M. Approximate inference turns deep networks into Gaussian processes. *arXiv* **2019**, arXiv:1906.01930.
13. Mu, F.; Liang, Y.; Li, Y. Gradients as features for deep representation learning. In Proceedings of the 8th International Conference on Learning Representations (ICLR), Addis Ababa, Ethiopia, 26–30 April 2020.
14. Jacot, A.; Gabriel, F.; Hongler, C. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in Neural Information Processing Systems*; Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2018; Volume 31, pp. 8571–8580.

15. Martens, J. Deep learning via hessian-free optimization. In Proceedings of the 27th International Conference on Machine Learning (ICML), Haifa, Israel, 21–24 June 2010; pp. 735–742.
16. Kunstner, F.; Balles, L.; Hennig, P. Limitations of the empirical fisher approximation for natural gradient descent. *arXiv* **2020**, arXiv:1905.12558v3.
17. Hornik, K.; Stinchcombe, M.; White, H. Multilayer feedforward networks are universal approximators. *Neural Netw.* **1989**, *2*, 359–366. [[CrossRef](#)]
18. Cybenko, G. Approximation by superpositions of a sigmoidal function. *Math. Control. Signals Syst.* **1989**, *2*, 303–314. [[CrossRef](#)]
19. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press: Cambridge, UK, 2016; Volume 1.
20. Huang, G.B.; Zhu, Q.Y.; Siew, C.K. Extreme learning machine: Theory and applications. *Neurocomputing* **2006**, *70*, 489–501. [[CrossRef](#)]
21. Yu, D.; Deng, L. Efficient and effective algorithms for training single-hidden-layer neural networks. *Pattern Recognit. Lett.* **2012**, *33*, 554–558. [[CrossRef](#)]
22. Kurita, T. Iterative weighted least squares algorithms for neural networks classifiers. *New Gener. Comput.* **1994**, *12*, 375–394. [[CrossRef](#)]
23. Martens, J. New Insights and Perspectives on the Natural Gradient Method. *J. Mach. Learn. Res.* **2020**, *21*, 1–76.
24. Hinton, G.E. Training products of experts by minimizing contrastive divergence. *Neural Comput.* **2002**, *14*, 1771–1800. [[CrossRef](#)]
25. Bengio, Y.; Lamblin, P.; Popovici, D.; Larochelle, H. Greedy layer-wise training of deep networks. *Adv. Neural Inf. Process. Syst.* **2007**, *19*, 153.
26. Bengio, Y.; Delalleau, O. Justifying and generalizing contrastive divergence. *Neural Comput.* **2009**, *21*, 1601–1621. [[CrossRef](#)]
27. Hinton, G.E. Learning multiple layers of representation. *Trends Cogn. Sci.* **2007**, *11*, 428–434. [[CrossRef](#)]
28. Salakhutdinov, R.; Hinton, G. Deep boltzmann machines. In Proceedings of the 12th International Conference on Artificial Intelligence and Statistics, PMLR, Clearwater Beach, FL, USA, 16–18 April 2009; Volume 5, pp. 448–455.
29. Jaderberg, M.; Czarnecki, W.M.; Osindero, S.; Vinyals, O.; Graves, A.; Silver, D.; Kavukcuoglu, K. Decoupled neural interfaces using synthetic gradients. In Proceedings of the 34th International Conference on Machine Learning, PMLR, Sydney, Australia, 6–11 August 2017; Volume 70, pp. 1627–1635.
30. Chen, S.; Wang, W.; Pan, S.J. Deep neural network quantization via layer-wise optimization using limited training data. In Proceedings of the AAAI Conference on Artificial Intelligence, Honolulu, HI, USA, 27 January–1 February 2019; Volume 33, pp. 3329–3336.
31. Shin, Y. Effects of depth, width, and initialization: A convergence analysis of layer-wise training for deep linear neural networks. *arXiv* **2019**, arXiv:1910.05874.
32. Song, Y.; Meng, C.; Liao, R.; Ermon, S. Nonlinear equation solving: A faster alternative to feedforward computation. *arXiv* **2020**, arXiv:2002.03629.
33. Kingma, D.P.; Ba, J. Adam: A Method for Stochastic Optimization. *arXiv* **2017**, arXiv:1412.6980v9.
34. Pal, S.K.; Mitra, S. Multilayer perceptron, fuzzy sets, classification. *IEEE Trans. Neural Netw.* **1992**, *3*, 683–697. [[CrossRef](#)] [[PubMed](#)]
35. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. Scikit-learn: Machine learning in Python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830.
36. Chen, T.; Guestrin, C. Xgboost: A scalable tree boosting system. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 13–17 August 2016; pp. 785–794.
37. Friedman, J.H. Greedy function approximation: A gradient boosting machine. *Ann. Stat.* **2001**, *29*, 1189–1232. [[CrossRef](#)]
38. Freund, Y.; Schapire, R.E. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.* **1997**, *55*, 119–139. [[CrossRef](#)]
39. Breiman, L. Random forests. *Mach. Learn.* **2001**, *45*, 5–32. [[CrossRef](#)]
40. Pace, R.K.; Barry, R. Sparse spatial autoregressions. *Stat. Probab. Lett.* **1997**, *33*, 291–297. [[CrossRef](#)]
41. Breiman, L. *Bias, Variance, and Arcing Classifiers*; Technical Report, 460; Statistics Department, University of California: Berkeley, CA, USA, 1996.
42. Heck, D.; Knapp, J.; Capdevielle, J.; Schatz, G.; Thouw, T. CORSIKA: A Monte Carlo code to simulate extensive air showers. *Rep. Fzka* **1998**, *6019*, 5–8.