symmetry

MDPI

Article

Feedforward Neural Networks with a Hidden Layer Regularization Method

Habtamu Zegeye Alemu, Wei Wu * and Junhong Zhao

School of Mathematical Sciences, Dalian University of Technology, Dalian 116024, China; ahabtamu58@yahoo.com (H.Z.A.); zhaojunhong7510@mail.dlut.edu.cn (J.Z.)

* Correspondence: wuweiw@dlut.edu.cn

Received: 3 September 2018; Accepted: 14 October 2018; Published: 19 October 2018

Abstract: In this paper, we propose a group Lasso regularization term as a hidden layer regularization method for feedforward neural networks. Adding a group Lasso regularization term into the standard error function as a hidden layer regularization term is a fruitful approach to eliminate the redundant or unnecessary hidden layer neurons from the feedforward neural network structure. As a comparison, a popular Lasso regularization method is introduced into standard error function of the network. Our novel hidden layer regularization method can force a group of outgoing weights to become smaller during the training process and can eventually be removed after the training process. This means it can simplify the neural network structure and it minimizes the computational cost. Numerical simulations are provided by using K-fold cross-validation method with $K = 5$ to avoid overtraining and to select the best learning parameters. The numerical results show that our proposed hidden layer regularization method prunes more redundant hidden layer neurons consistently for each benchmark dataset without loss of accuracy. In contrast, the existing Lasso regularization method prunes only the redundant weights of the network, but it cannot prune any redundant hidden layer neurons.

Keywords: sparsity; feedforward neural networks; hidden layer regularization; group lasso; lasso

1. Introduction

Artificial Neural Networks (ANNs) are pretty old ideas to mimic the human brain [1]. ANNs have been intensively studied for many years in the hope of achieving human-like performance in the fields of speech and image recognition [2]. They are designed to solve a variety of problems in the area of pattern recognition, prediction, optimization, associative memory, and control [3]. ANNs are also used for approximation of phenol concentration using computational intelligence methods [4].

The Feedforward Neural Networks (FNNs) are the most fundamental part of ANNs that have been trained by using the connectionist learning procedure called a supervised manner which requires a teacher to specify the desired output vector [5]. An FNN is one part of a multi-layer perceptron (MLP) with unidirectional data flow [6]. In FNNs, the neurons are arranged in the form of layers, namely input, hidden and output layers and there exist the connections between the neurons of one layer to those of the next layer [7]. These connections are repeatedly adjusted to minimize a measure of the difference between the actual network output vector and the desired output vector through the learning process [8]. The FNNs are commonly trained by using the backpropagation (BP) algorithm which uses a gradient descent learning method, also called the steepest descent [9].

Depending on the way the weights are updating, the gradient descent method can be classified into the following three methods: batch gradient descent method, mini-batch gradient descent method and stochastic gradient descent method. In the batch gradient descent method, the weights are updated after all the training examples are shown to the learning algorithm. In the stochastic gradient descent method, the weight parameters are updated using each training example. In a mini-batch gradient

descent method, the training examples are partitioned into small batches, and the weight parameters are updated for each mini-batch example. In this study, we focus only on the batch gradient method.

It is well known that determining the number of input and output neurons naturally depends on the dimension of the problem, but the main problem is determining the optimal number of neurons in the hidden layer that can solve the specific problem [10]. Here, we mean the optimal number of the hidden layer neurons of FNNs is one that is large enough to learn the samples and small enough to perform well on unseen samples. There is no clear standard method to determine the optimal size of the hidden layer neurons for the network to solve a specific problem. However, usually, the number of hidden neurons of the neural network is determined by a trial-and-error method. This will lead to a cost problem in computation. In addition, having too many neurons in the hidden layer may lead to overfitting of the data and poor generalization while having too few neurons in the hidden layer may not provide a network that learns the data [11].

Generally speaking, the constructive and destructive approaches are the two main approaches used in literature for optimizing neural network structure [12]. The first approach, also called the growing method, begins with a minimal neural network structure and adds more hidden neurons only when they are needed to improve the learning capability of the network. The second approach begins with an oversized neural network structure and then prunes redundant hidden layer neurons [13]. A disadvantage of applying the growing method is that the initial neural network with a small number of hidden layer neurons can easily be trapped into local minima and it may need more time to get the optimal number of hidden layer neurons. Therefore, we aimed to find the optimal number of hidden layer neurons by using the pruning method.

Furthermore, depending on the techniques used for pruning, the pruning methods can be further classified into the following methods [14]: regularization (penalty) methods, cross-validation methods, magnitude base methods, evolutionary pruning methods, mutual information, significance based pruning methods, and the sensitivity analysis method. The most popular sensitivity based pruning algorithms are the Optimal Brain Damage [15] and the Optimal Brain Surgeon method [16]. This paper focuses on a pruning technique called the regularization method that mainly addresses the overfitting problem. To do this, we can add the extra regularization terms into the standard error function to sparse the values of weight connections by assuming that sparse neural network models lead to better performance.

Most of the existing regularization methods for FNNs can be further categorized into different L_p regularization methods. Figure 1 indicates a graphical representation of L_p norms with different p-values.

The L_p regularization method is widely applied as a parameter estimation technique to solve the variable selection problem [17]. The most common L_p regularization terms are

$$R(W) = \frac{1}{2}\sum_{i\in W} w_i^2, \qquad \text{“L}_2\text{ regularization”}, \tag{1}$$

$$R(W) = \sum_{i\in W} |w_i|, \qquad \text{“L}_1\text{ regularization”}, \tag{2}$$

$$R(W) = \sum_{i\in W} |w_i|^{1/2}, \qquad \text{“L}_{1/2}\text{ regularization”}, \tag{3}$$

where W is the set of all weights of the neural network and $|\cdot|$ represents the absolute value function. The regularization term in Equation (1) is defined as the 2-norm (squared norm) of the network weights. L_2 regularization does not have the sparsity property, but it has the property of being smooth. The L_1 regularization term leads to an area of the convex optimization problem, which is easy to solve, but it does not give a sufficiently sparse solution [18]. Adding the $L_{1/2}$ regularization term into the

standard error function promotes excess weights to take values close to zero. The $L_{1/2}$ regularization term performs better in the sparsity of weight connections [19–21]. Recently, the $L_{1/2}$ regularization term has been proposed to determine the redundant dimensions of the input data for the multilayer feedforward networks by fixing the number of hidden neurons [22]. The result of this study confirms that the $L_{1/2}$ regularization method produces better performance than L_1 due to its sparsity property.

To sum up, a major drawback of using the L_p regularization terms described above is that they are mainly designed for removing the redundant weights from the neural network, but they cannot remove the redundant or unnecessary hidden neurons of the neural network automatically. This study aimed to investigate the pruning of unnecessary hidden layer neurons of FNNs.

The popular Lasso, least absolute shrinkage and selection operator, regularization method that was originally proposed for estimation of linear models is defined in [23] as

$$\hat{\beta}^{Lasso}(\lambda) = \arg\min \left(\|Y - X\beta\|_2^2 + \lambda \|\beta\|_1 \right), \tag{4}$$

where λ is a regularization parameter, $Y \in \mathbb{R}^m$ is a continuous response, X is an $m \times k$ design matrix, $\beta \in \mathbb{R}^k$ is a vector parameter. Moreover, $\|\cdot\|_2^2$ and $\|\cdot\|_1$ stands for 2-norm (squared norm) and 1-norm, respectively. Lasso tends to produce sparse solutions for network models.

An extension of Lasso, group Lasso was originally used to solve linear regression problems and it is one of the most popular regularization method for variable selection [24,25]. For a given training set that consists of M input-output pairs $f(x_i, y_i)_{1 \le i \le M}$, the following optimization problem with group Lasso regularization term was used in [26] to sparse the network with any L numbers of layers that consists of N_l neurons numbers each of which is encoded by parameters $\theta_l^n = [w_l^n, b_l^n]$, w_l^n is a linear operator acting on the layer's input and b_l^n is a bias, where these parameters form the parameter set $\Theta = \{\theta_l\}_{1 \le l \le L}$, with $\theta = \{\theta_l^n\}_{1 \le n \le N_l}$,

$$\min_{\Theta} \left\{ \frac{1}{M} \sum_{i=1}^{M} \ell(y_i, f(x_i, \Theta)) + \sum_{l=1}^{L} \lambda_l \sqrt{p_l} \sum_{n=1}^{N_l} \|\theta_l^n\|_2 \right\}, \tag{5}$$

where $\ell(.)$ is a loss function that compares the network prediction with the ground-truth output, such as the logistic loss for classification or the square loss for regression, p_l is the size of parameters grouped together in layer l and λ_l is the regularization coefficient. The regularization parameters λ_l are scaled with group size $\sqrt{p_l}$ to regularize larger groups in (5). Here, tuning different regularization parameters λ_l for each groups in each layer is considered as one disadvantage. However, by rescaling the groups, we can simplify the cost function in Equation (5) into

$$\min_{\Theta} \left\{ \frac{1}{M} \sum_{i=1}^{M} \ell(y_i, f(x_i, \Theta)) + \lambda \sum_{l=1}^{L} \sum_{n=1}^{N_l} \|\theta_l^n\|_2 \right\}. \tag{6}$$

Now, one can use Equation (6) that is simplified from Equation (5) to sparse the neural network structure by penalizing each group in each layer with the same regularization parameter λ. Particularly, it is important to prune the redundant neurons from the input and hidden layers.

Hence, developing an automated hidden layer regularization method by using the idea of Equation (6), which can find out a small, necessary, and sufficient number of neurons in the hidden layer of FNNs without an additional retraining process is our primary motivation. To achieve this, there are two approaches. The first approach is considering only the norm of the total entering weights to each hidden layer neurons. The second approach is considering only the norm of the total outgoing weights from each hidden layer neurons. In this paper, we propose a group Lasso regularization method by using the second approach. Here, our goal is shrinking the total outgoing weights from unnecessary or redundant neurons of the hidden layer to zero without loss of accuracy.

Furthermore, we conduct experiments by using the benchmark datasets to compare our proposed hidden layer regularization method with the standard batch gradient method without

any regularization term and the popular Lasso regularization method. The numerical results demonstrate the effectiveness of our proposed hidden layer regularization method on both sparsing and generalization ability.

The rest of this paper is organized as follows: in Section 2, Materials and Methods are described. In Section 3, the results are presented. In Section 4, we discuss in detail the numerical results. Finally, we conclude this paper with some remarks in Section 5.

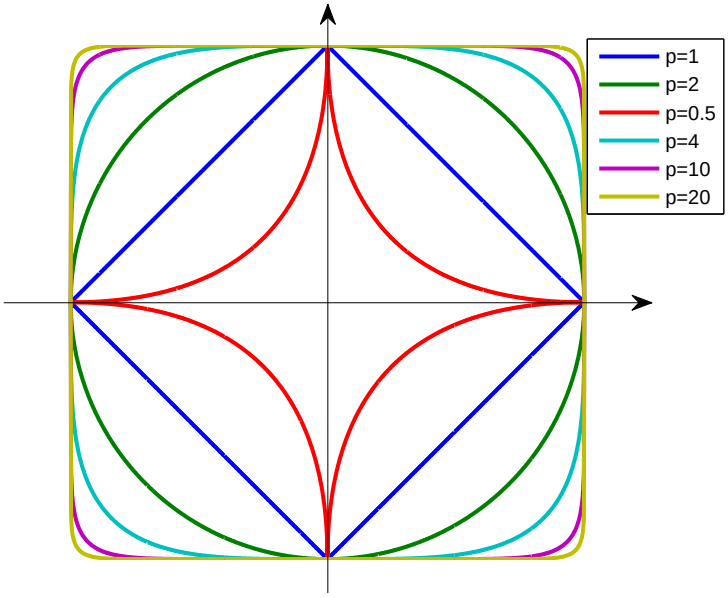


Figure 1. The properties of L_p norms.

2. Materials and Methods

2.1. Neural Network Structure and Batch Gradient Method without Regularization Term

We consider an FNN with one hidden layer of the network structure $p+1-q+1-r$, consisting of $p+1$ number of neurons in the input layer, $q+1$ number of neurons in the hidden layer and r number of neurons in the output layer (including bias neurons in the input and hidden layers) (see Figure 2). Here, the extra neuron added into both input and hidden layers represents the bias neuron with output value +1, and it allows us to shift the activation function to the left or right, which is very important for successful network training process. We note that this fully connected neural network structure has $q(p+1)+r(q+1)$ total number of weight connections. To perform computations for our given neural network structure, we represent the weight connections of the neural network in the form of matrices and vectors and apply matrix-vector operations. Let $w_j = (w_{j1}, w_{j2}, w_{j3}, \ldots, w_{ji}, \ldots, w_{jp}, w_{j(p+1)})^T \in \mathbb{R}^{p+1}$ be the weight vector connecting the input neurons and the hidden layer neuron $j (j = 1, 2, 3, \ldots, q)$ and $\tilde{w}_k = (\tilde{w}_{k1}, \tilde{w}_{k2}, \tilde{w}_{k3}, \ldots, \tilde{w}_{kj}, \ldots, \tilde{w}_{kq}, \tilde{w}_{k(q+1)})^T \in \mathbb{R}^{q+1}$ be the weight vector connecting the hidden layer neurons and the output neuron $k (k = 1, 2, 3, \ldots, r)$, where the symbol "$T$" represents the transpose. We can also represent in matrix form as shown below:

$$w = (w_1, w_2, w_3, \ldots, w_j, \ldots, w_q)^T \in \mathbb{R}^{q\times(p+1)},$$
$$\tilde{w} = (\tilde{w}_1, \tilde{w}_2, \tilde{w}_3, \ldots, \tilde{w}_k, \ldots, \tilde{w}_r)^T \in \mathbb{R}^{r\times(q+1)}.$$

To make our presentation simple, we can rewrite all the weight parameters in a compact form as follows: $W = (\tilde{w}_1^T, \tilde{w}_2^T, \tilde{w}_3^T, \ldots, \tilde{w}_k^T, \ldots, \tilde{w}_r^T, w_1^T, w_2^T, w_3^T, \ldots, w_j^T, \ldots, w_q^T)^T \in \mathbb{R}^{r(q+1)+q(p+1)}$. Let $g : \mathbb{R} \to \mathbb{R}$ be a given activation function for hidden layer neurons and output neurons which is typically

but not necessarily a sigmoid function that squashes (limits) the outputs of the summation neurons. For any $x = (x_1, x_2, x_3, \ldots, x_j, \ldots, x_q)^T \in \mathbb{R}^q$, we introduce the vector-valued function

$$G : \mathbb{R}^q \to \mathbb{R}^{q+1}, G(x) = (g(x_1), g(x_2), g(x_3), \ldots, g(x_j), \ldots, g(x_q), +1)^T, \tag{7}$$

where component $+1$ represents the output value from bias neuron. Let $\{\xi^m, o^m\}_{m=1}^M \subset \mathbb{R}^{P+1} \times \mathbb{R}^r$ be a set of training examples, where $\xi^m = (\xi_1^m, \xi_2^m, \xi_3^m, \ldots, \xi_i^m, \ldots, \xi_p^m, +1)^T \in \mathbb{R}^{p+1}$ and $o^m \in \mathbb{R}^r$ are the input and the corresponding one-hot encoded ideal output of the mth sample, respectively. Then, for each input ξ^m, the actual output vector of the hidden layer neurons is $G(w\xi^m)$ and finally the actual output to the network at output neuron k is:

$$y_k^m = g(\tilde{w}_k \cdot G(w\xi^m)). \tag{8}$$

Consequently, the actual output vector to the network at output layer is: $y^m = (y_1^m, y_2^m, y_3^m, \ldots y_k^m, \ldots, y_r^m)^T \in \mathbb{R}^r$.

Then, the standard mean square error function to the network without any regularization term is defined as

$$\begin{aligned} \tilde{E}(W) &= \frac{1}{2M} \sum_{m=1}^{M} \sum_{k=1}^{r} (o_k^m - y_k^m)^2 \\ &= \frac{1}{2M} \sum_{m=1}^{M} \sum_{k=1}^{r} (o_k^m - g(\tilde{w}_k \cdot G(w\xi^m)))^2 \\ &= \sum_{m=1}^{M} \sum_{k=1}^{r} g_{mk}(\tilde{w}_k \cdot G(w\xi^m)), \end{aligned} \tag{9}$$

where

$$g_{mk}(t) = \frac{1}{2M}(o_k^m - g(t))^2$$

is a composite function and its derivative is

$$g'_{mk}(t) \equiv \frac{-1}{M}(o_k^m - g(t))g'(t), \quad t \in \mathbb{R}, m = 1, 2, \ldots, M.$$

Then,

$$\tilde{E}_{\tilde{w}_{kj}}(W) = \sum_{m=1}^{M} g'_{mk}(\tilde{w}_k \cdot G(w\xi^m))\, g(w_j\xi^m), \tag{10a}$$

$$\tilde{E}_{w_{ji}}(W) = \sum_{m=1}^{M} \sum_{k=1}^{r} g'_{mk}(\tilde{w}_k \cdot G(w\xi^m))\, \tilde{w}_{kj} g'(w_j\xi^m)\xi_i^m, \tag{10b}$$

for $i = 1, 2, 3, \ldots, p, p+1$; $j = 1, 2, 3, \ldots, q, q+1$ and $k = 1, 2, 3, \ldots, r$. The gradient of the error function defined in Equation (9) with respect to the weight vector W is also defined as

$$\tilde{E}_W(W) = (\tilde{E}_{\tilde{w}_1}^T, \tilde{E}_{\tilde{w}_2}^T, \tilde{E}_{\tilde{w}_3}^T, \ldots, \tilde{E}_{\tilde{w}_k}^T, \ldots, \tilde{E}_{\tilde{w}_r}^T, \tilde{E}_{w_1}^T, \tilde{E}_{w_2}^T, \tilde{E}_{w_3}^T, \ldots, \tilde{E}_{w_j}^T, \ldots, \tilde{E}_{w_q}^T)^T \in \mathbb{R}^{r(q+1)+q(p+1)}, \tag{11}$$

where

$$\begin{aligned}
\tilde{E}^T_{\tilde{w}_1} &= \left(\tilde{E}_{\tilde{w}_{11}}(W), \tilde{E}_{\tilde{w}_{12}}(W), \tilde{E}_{\tilde{w}_{13}}(W), \ldots, \tilde{E}_{\tilde{w}_{1j}}(W), \ldots, \tilde{E}_{\tilde{w}_{1q}}(W), \tilde{E}_{\tilde{w}_{1(q+1)}}(W)\right), \\
\tilde{E}^T_{\tilde{w}_2} &= \left(\tilde{E}_{\tilde{w}_{21}}(W), \tilde{E}_{\tilde{w}_{22}}(W), \tilde{E}_{\tilde{w}_{23}}(W), \ldots, \tilde{E}_{\tilde{w}_{2j}}(W), \ldots, \tilde{E}_{\tilde{w}_{2q}}(W), \tilde{E}_{\tilde{w}_{2(q+1)}}(W)\right), \\
&\vdots \\
\tilde{E}^T_{\tilde{w}_r} &= \left(\tilde{E}_{\tilde{w}_{r1}}(W), \tilde{E}_{\tilde{w}_{r2}}(W), \tilde{E}_{\tilde{w}_{r3}}(W), \ldots \tilde{E}_{\tilde{w}_{rj}}(W), \ldots, \tilde{E}_{\tilde{w}_{rq}}(W), \tilde{E}_{\tilde{w}_{r(q+1)}}(W)\right), \\
\tilde{E}^T_{w_1} &= \left(\tilde{E}_{w_{11}}(W), \tilde{E}_{w_{12}}(W), \tilde{E}_{w_{13}}(W), \ldots, \tilde{E}_{w_{1i}}(W), \ldots, \tilde{E}_{w_{1p}}(W), \tilde{E}_{w_{1(p+1)}}(W)\right), \\
\tilde{E}^T_{w_2} &= \left(\tilde{E}_{w_{21}}(W), \tilde{E}_{w_{22}}(W), \tilde{E}_{w_{23}}(W), \ldots, \tilde{E}_{w_{2i}}(W), \ldots, \tilde{E}_{w_{2p}}(W), \tilde{E}_{w_{2(p+1)}}(W)\right), \\
&\vdots \\
\tilde{E}^T_{w_q} &= \left(\tilde{E}_{w_{q1}}(W), \tilde{E}_{w_{q2}}(W), \tilde{E}_{w_{q3}}(W), \ldots, \tilde{E}_{w_{qi}}(W), \ldots, \tilde{E}_{w_{qp}}(W), \tilde{E}_{w_{q(p+1)}}(W)\right).
\end{aligned}$$

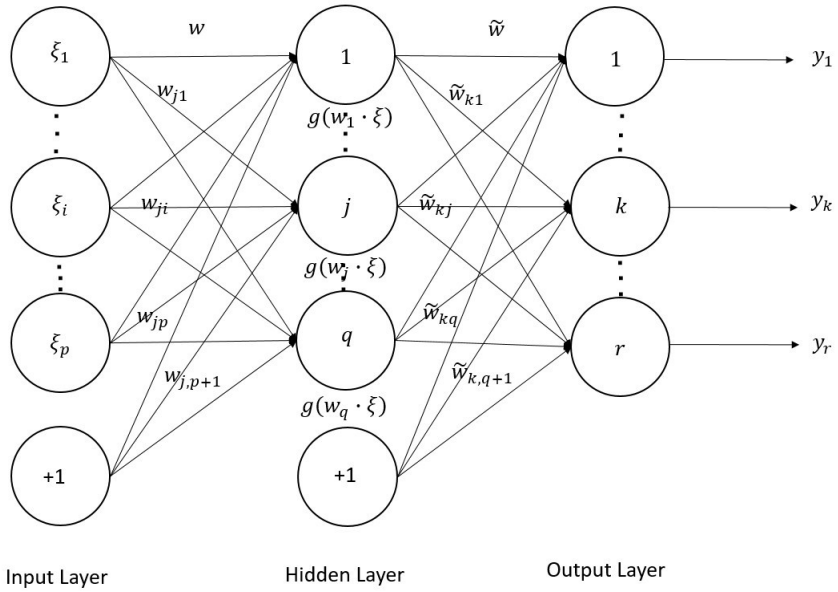


Figure 2. Feedforward neural network with one hidden layer and multiple neurons at the output layer.

2.2. A Batch Gradient Method with Hidden Layer Regularization Terms

The network complexity is usually measured in terms of the number of free parameters, i.e., by the total number of weights in the network. By adding any regularization terms into the standard error function we can limit the growth of the weights during network training process. In the next section we construct the proposed hidden layer regularization terms that can remove unnecessary or redundant hidden layer neurons of FNNs.

2.2.1. Batch Gradient Method with Lasso Regularization Term

We first describe the popular Lasso regularization method. Here, our main concerned is to penalize each weight connections of the neural network by using the Lasso regularization term. The new error function $E(W)$ with Lasso hidden layer regularization term is defined as follows:

$$E(W) = \tilde{E}(W) + \lambda \|W\|_1, \tag{12}$$

where $\tilde{E}(W)$ is the standard error function defined in Section 2.1, $\|W\|_1 = \sum_{i \in W} \mid w_i \mid$, W is the set of all weights and λ is the regularization parameter that prevents the network weights from growing too large by penalizing each weight during network training.

The goal of the network training is to get W^* such that

$$W^* = \arg\min \left\{ \tilde{E}(W) + \lambda \|W\|_1 \right\}. \tag{13}$$

The gradients of the error function in Equation (12) with respect to w_{ji} and $\tilde{w}_{kj}$ are expressed, respectively as

$$E_{w_{ji}}(W) = \tilde{E}_{w_{ji}}(W) + \lambda \text{sign}(w_{ji}), \tag{14a}$$
$$E_{\tilde{w}_{kj}}(W) = \tilde{E}_{\tilde{w}_{kj}}(W) + \lambda \text{sign}(\tilde{w}_{kj}), \tag{14b}$$

where the "sign" is signum function of a real number x that can be defined as follows:

$$\text{sign}(x) = \begin{cases} -1 & \text{if} \quad x < 0, \\ 0 & \text{if} \quad x = 0, \\ 1 & \text{if} \quad x > 0. \end{cases} \tag{15}$$

The corresponding increments of the weights w_{ji}^n and $\tilde{w}_{kj}^n$ are respectively defined as follows:

$$\triangle w_{ji}^n = -\eta \left[\tilde{E}_{w_{ji}^n}(W) + \lambda \text{sign}(w_{ji}^n) \right], \tag{16a}$$
$$\triangle \tilde{w}_{kj}^n = -\eta \left[\tilde{E}_{\tilde{w}_{kj}^n}(W) + \lambda \text{sign}(\tilde{w}_{kj}^n) \right], \tag{16b}$$

where η is a positive parameter that controls the speed of learning process.

In the above equations, the minus sign indicates that the learning is taking place in the opposite direction to the gradient descent, which is the steepest descent. Starting with initial value W^0, the weight vector W^n is updated iteratively by using the following learning algorithm:

$$W^{n+1} = W^n + \triangle W^n. \tag{17}$$

In component form, the weights are updated iteratively by

$$w_{ji}^{n+1} = w_{ji}^n + \triangle w_{ji}^n, \tag{18a}$$
$$\tilde{w}_{kj}^{n+1} = \tilde{w}_{kj}^n + \triangle \tilde{w}_{kj}^n. \tag{18b}$$

2.2.2. Batch Gradient Method with Group Lasso Regularization Term

Now, let us formulate our proposed hidden layer regularization method. This is done by penalizing only the norm of total outgoing weights from each hidden layer neuron. Its new error function is formulated as

$$E(W) = \tilde{E}(W) + \lambda \sum_{j=1}^{q+1} \left\| \tilde{w}_j \right\|_2, \tag{19}$$

where $\left\|\tilde{w}_j\right\|_2 = \left(\tilde{w}_{1j}^2 + \tilde{w}_{2j}^2 + \ldots + \tilde{w}_{kj}^2 + \ldots + \tilde{w}_{rj}^2\right)^{\frac{1}{2}}$ is the 2- norm (not squared norm) of outgoing weight vector from the j-th hidden layer neuron, and λ is the regularization parameter. Similarly, the goal of the network training by using group Lasso is also to get W^* such that

$$W^* = \arg\min \left\{ \tilde{E}(W) + \lambda \sum_{j=1}^{q+1} \left\|\tilde{w}_j\right\|_2 \right\}. \tag{20}$$

Adding the extra group Lasso regularization term into the standard error function plays a great role to shrink all the outgoing weights of a redundant or unnecessary hidden layer neuron j of FNNs. Hence, any neuron j of a hidden layer with the norm of its outgoing weights is zero or near to zero should be automatically removed without degradation of the neural network accuracy.

The corresponding gradients of the error function defined in Equation (19) with respect to w_{ji} and $\tilde{w}_{kj}$ are expressed, respectively, as

$$E_{w_{ji}}(W) = \tilde{E}_{w_{ji}}(W), \tag{21a}$$

$$E_{\tilde{w}_{kj}}(W) = \tilde{E}_{\tilde{w}_{kj}}(W) + \frac{\lambda \tilde{w}_{kj}}{\left\|\tilde{w}_j\right\|_2}. \tag{21b}$$

Thus,

$$\triangle w_{ji}^n = -\eta \tilde{E}_{w_{ji}^n}(W), \tag{22a}$$

$$\triangle \tilde{w}_{kj}^n = -\eta \left[\tilde{E}_{\tilde{w}_{kj}^n}(W) + \frac{\lambda \tilde{w}_{kj}^n}{\left\|\tilde{w}_j^n\right\|_2} \right]. \tag{22b}$$

By starting with initial weight vector W^0, the weight vector W^n are updated iteratively by using the learning algorithm formulated by the Equation (17). Similarly, each weight component is also updated iteratively by using Equations (18a) and (18b).

2.3. Datasets

Four benchmark datasets from the UCI machine learning repository [27] are chosen for numerical simulation purposes and their detail properties are described in Table 1.

The first dataset is an iris dataset, which is one of the well-known benchmark datasets in machine learning. It contains three sets of flower types (Iris Setosa, Iris Versicolour, and Iris Virginica) and four dimensions (sepal length in cm, sepal width in cm, petal length in cm, petal width in cm).

The second one is the zoo dataset which is one of the seven class classification datasets. It contains 101 examples, each with 17 Boolean-valued attributes. Our task is to classify animals into 7 categories based on given information.

The third dataset is the seeds dataset, which is one of the multiclass classification datasets. It contains 210 examples, each with seven features. Our task is to classify varieties of wheat into three classes (i.e., Kama, Rosa and Canadian).

The fourth dataset that we use is ionosphere dataset. This radar dataset was collected by a system in Goose Bay, Labrador. This system consists of a phased array of 16 high-frequency antennas with a total transmitted power on the order of 6.4 kilowatts. The targets were free electrons in the ionosphere. "Good" radar returns are those showing evidence of some type of structure in the ionosphere. "Bad" returns are those that do not; their signals pass through the ionosphere. This data has 351 instances and 35 attributes. All 34 predictor attributes are continuous, and the 35th attribute is either "Good" or "Bad". This is a binary classification task.

Table 1. Properties of benchmark datasets.

Datasets	No. of Examples	No. of Attributes	No. of Class
Iris	150	4	3
Zoo	101	17	7
Seeds	210	7	3
Ionosphere	351	34	2

2.3.1. *K*-fold Cross-Validation Method

To avoid overtraining, we use the *K*-fold cross-validation method [28]. The parameter K represents the fold size. First, we shuffled the original dataset, $X = \{\xi^m, o^m\}_{m=1}^{M} \subset \mathbb{R}^{P+1} \times \mathbb{R}^r$, randomly. As shown in Figure 3 the shuffled dataset is divided into K roughly equal-sized sub datasets, $X_1, X_2, X_3, \ldots, X_k, \ldots, X_K$, where $X_k = \{\xi^m, o^m\}_{m=1}^{M'} \subset \mathbb{R}^{P+1} \times \mathbb{R}^r$ and $M' = \frac{M}{K}$ is the size of each fold. One of the K sets is used as the validation set for validating the trained network while the remaining $K-1$ datasets are used as training set. Hence the training set, $X_{train} = X_1, X_2, X_3, \ldots, X_{k-1}, X_{k+1}, \ldots, X_K$ and the validation set, $X_{validation} = X_k$.

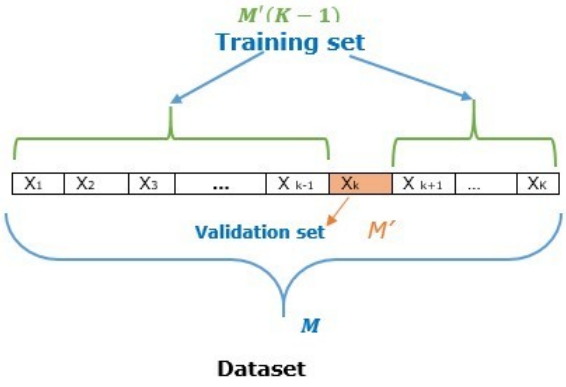


Figure 3. *K*-fold cross-validation method.

The cross-validation process is repeated K times (K folds). The advantage of applying K-fold is that each fold is used exactly once as the validation set. Finally, the K results are averaged to obtain a single average accuracy result.

For a given set of learning parameters $\{W, \eta, \lambda, K\}$, we compute the training accuracy for each $k = 1, 2, 3, \ldots, K$ as follows:

$$Training Accuracy(k) = \frac{1}{M'(K-1)} \sum_{m=1}^{M'(K-1)} I(\arg\max_{k=1,2,\ldots,r} y^m, \arg\max_{k=1,2,\ldots,r} o^m), \tag{23}$$

where, the indicative function $I(a,b) = \begin{cases} 1, & \text{if} \quad a = b, \\ 0, & \text{if} \quad a \neq b, \end{cases}$

and y^m and o^m are predicted vector of all output layer neurons and a one-hot encoded ideal output of X_{train}, respectively. The corresponding testing accuracy is computed as

$$Testing Accuracy(k) = \frac{1}{M'} \sum_{m=1}^{M'} I(\arg\max_{k=1,2,\ldots,r} y^m, \arg\max_{k=1,2,\ldots,r} o^m). \tag{24}$$

Similarly, y^m and o^m are predicted vectors of all output layer neurons and a one-hot encoded ideal output of $X_{validation}$. Finally, the average training and testing accuracy of all K results are computed respectively as

$$TrainingAccuracy = \frac{1}{K}\sum_{k=1}^{K} TrainingAccuracy(k), \tag{25}$$

$$TestingAccuracy = \frac{1}{K}\sum_{k=1}^{K} TestingAccuracy(k). \tag{26}$$

All numerical simulations were run on a MacBook Air laptop designed by apple in California assembled in China with the processor running at 1.6 GHz Intel Core i5, Memory 8 GB 1600 NHz DDR3 and Graphics Intel HD Graphics 6000 1536 MB of RAM by using MATLAB R2014a (MathWorks, Inc., Natick, MA, USA).

2.3.2. Data Normalization

In the data mining area, data preprocessing is a crucial technique to clean the data before we are using it for the neural network training process. The normalization is needed to improve the performance of numerical computation and obtain better neural network output results by avoiding the influence of one attribute over another. A min-max normalizing method [29] is one of the most common data normalizing methods and it is defined as

$$z_i = \left(\frac{x_i - \min(x)}{\max(x) - \min(x)}\right)(\max_{new} - \min_{new}) + \min_{new}, \tag{27}$$

where $x = [x_1, x_2, x_3, \ldots, x_n]^T$ is original data values of a single feature of dataset, x_i is i-th original data value and z_i is i-th normalized data value. The $\max_{new}$ and $\min_{new}$ represent the maximum range and minimum range for normalized dataset, respectively. Thus, each of the original benchmark dataset is normalized in a range of 0 to 1. All class attributes of the benchmark dataset are also encoded by one-hot representation.

2.3.3. Activation Function

A pure linear (purelin) activation function with the range $(-\infty, \infty)$ is the most common activation function for the output layer in MATLAB. Herein, after we normalized our datasets, we choose the logistic sigmoid function as a transfer function for all neurons in the hidden layer and output layer and it is expressed as

$$\text{logsig(x)} = \frac{1}{1 + \exp(-\text{x})}. \tag{28}$$

The logistic sigmoid activation function has the range $(0, 1)$. This function can work well for classification tasks by giving the output results at the output layer in the range 0 to 1.

2.4. Hidden Neuron Selection Criterion

To determine whether a neuron j in the hidden layer will survive or remove after training, the strategy that we used as a neuron selection criterion is just computing the norm of total outgoing weights from the neuron j. In literature, there is no standard threshold value to remove unnecessary weight connections and redundant neurons from the initially assumed neural network structure. According to [20,30], the sparsity of the learning algorithm was measured by using the number of weights whose absolute values are $\leq$0.0099 and $\leq$0.01, respectively. In this study, we have arbitrary chosen 0.00099 as a threshold value which is more less than the existing thresholds in literature.

Training algorithm and heuristics for FNN parameter selection

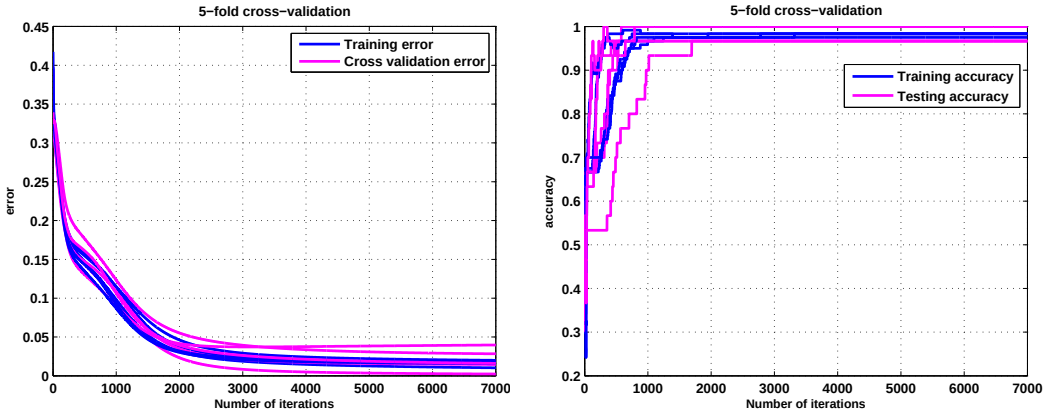
1. Use a K-fold cross-validation method to split the dataset into a training set and validation set.
2. Pick large fully connected FNNs with structure $p+1-q+1-r$, where $p+1$ is neuron number in the input layer, $q+1$ is neuron number in the hidden layer, and r is neuron number in the output layer (including bias neuron). The number of the input layer and output layer neurons are set equal to the numbers of attributes and classes for each dataset, respectively. Likewise, the number of hidden layer neurons initially is set randomly in a way that it needs to be remarkably bigger than the numbers of attributes and classes.
3. Randomly initialize network weights w and $\tilde{w}$ in $[-0.5, 0.5]$.
4. For each $k = 1, 2, 3, \ldots, K$ train the network using a training set with the standard batch gradient method without any regularization term (i.e., $\lambda = 0$) as a function learning rate η and pick the best η that gives the best learning.
5. Use the best learning rate η obtained from step 3 and start the training process further by increasing the regularization coefficient λ up to too many numbers of hidden layer neurons are removed, and the accuracy of the network is degraded.
6. Compute the norm of the total outgoing weights from each neuron j in the hidden layer; if the norm of total outgoing weights from neuron j in the hidden layer is less or equal to the threshold value, then remove the neuron j from the network else the neuron j will survive.
7. Compute the training accuracy using Equation (23).
8. Evaluate the trained network using a validation set.
9. Compute the testing accuracy using Equation (24).
10. Compute the average training and average testing accuracy of the overall results of K using Equations (25) and (26), respectively.
11. Compute the average number of redundant or unnecessary hidden layer neurons.
12. Select the best regularization parameter λ that gives the best average results.

3. Results

In this section, we present the numerical results of our proposed batch gradient method with a group Lasso regularization term (BGGLasso) compared to the existing learning methods (i.e., standard batch gradient method without any regularization term (BG) and batch gradient method with a Lasso regularization term (BGGLasso)) using benchmark datasets. Accuracy and sparsity are our major metrics to compare the performance of our proposed method with existing learning methods. As shown in Tables 2–6, the sparsity of the proposed hidden layer regularization method is measured by using the average number of pruned weights (AVGNPWs) and the average number of pruned hidden layer neurons (AVGNPHNs). Similarly, the average accuracy of the neural network for each datasets is measured by using average training accuracy (Training Acc. (%)) and average testing accuracy (Testing Acc. (%)). For each benchmark dataset, the 5-fold cross-validation method is employed to obtain numerical results. We also find out the best learning rate η by using the baseline BG (batch gradient method without any regularization term).

3.1. The Iris Results

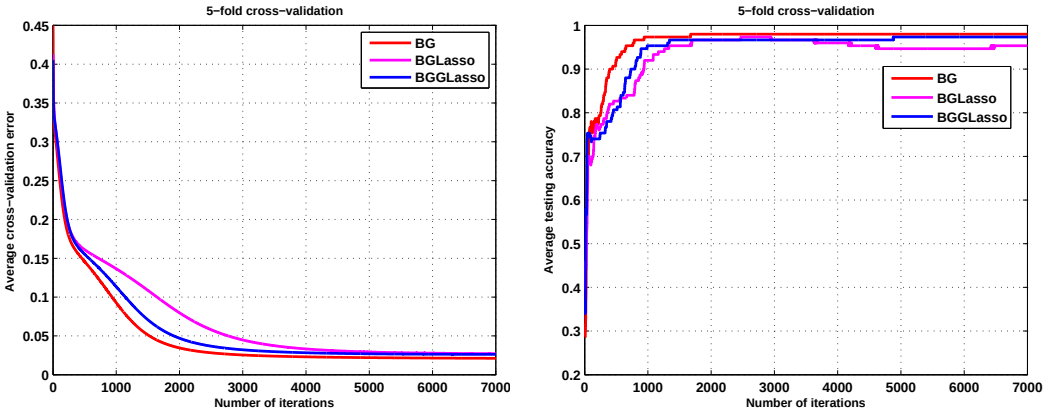
The numerical results in Figure 4 represents all 5-fold results obtained by using standard BG only. We started the training process with the oversized initial network structure, namely $5-16-3$ (including bias neurons in the input and hidden layers). The network was trained until the prespecified maximum number of iteration epochs of 7000 is met. We select the best learning rate $\eta = 0.050$. In Figure 4a, we demonstrate all the training errors with their corresponding cross validation errors of all 5-folds. Similarly, in Figure 4b, all of the training accuracies with their corresponding testing accuracies of all 5-folds are demonstrated.



(**a**) (**b**)

Figure 4. Best learning curves for iris dataset only with standard BG by using $\eta = 0.050$: (**a**) error and (**b**) accuracy.

After this, for the sake of comparison and simplicity, we only present the best learning curves using validation sets in terms of average results. The best learning curves for average cross-validation error results and average testing accuracy of our proposed BGGLasso learning method with BG and BGLasso learning methods by using the iris dataset are displayed in Figure 5a,b, respectively. In Table 2, the average sparsity result and average accuracy result with the best learning rate η and the best regularization parameter λ over 5-fold-cross validation for the iris dataset are recorded.



(**a**) (**b**)

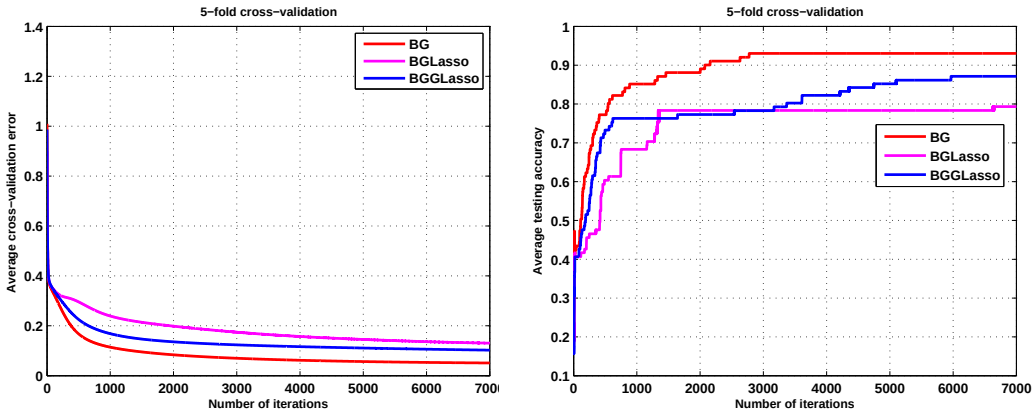
Figure 5. Best learning curves for iris dataset by using BG, BGLasso and BGGLasso regularization methods with $\eta = 0.050, \lambda = 0.010$: (**a**) average cross-validation error and (**b**) average testing accuracy.

Table 2. Comparison of best average results for the iris dataset.

Methods	λ	η	**Sparsity**		$\|E_W(W)\|$	**Accuracy**	
			AVGNPWs	**AVGNPHNs**		**Training Acc. (%)**	**Testing Acc. (%)**
BG	0.000	0.050	0.000	0.000	0.010	98.500	98.000
BGLasso	0.010	0.050	**53.200**	0.000	0.060	97.800	93.300
BGGLasso	0.010	0.050	18.400	**6.000**	**0.020**	98.300	97.300

3.2. The Zoo Results

To check the sparsity ability of the group Lasso and Lasso regularization terms using the zoo dataset, we randomly started with big neural network structure $18 - 26 - 7$ (the number of input neurons, the number of hidden layer neurons, and the number of output layer neurons), including the bias neurons in the input and hidden layers. For zoo dataset, the selected best learning rate η and the best regularization parameter λ are 0.040 and 0.030, respectively. The maximum number of training iteration is 7000. The best learning curves for the zoo dataset with average cross-validation error result and average testing accuracy result are shown in Figure 6a,b. Table 3 summarizes the average results with best learning parameters.



(**a**) (**b**)

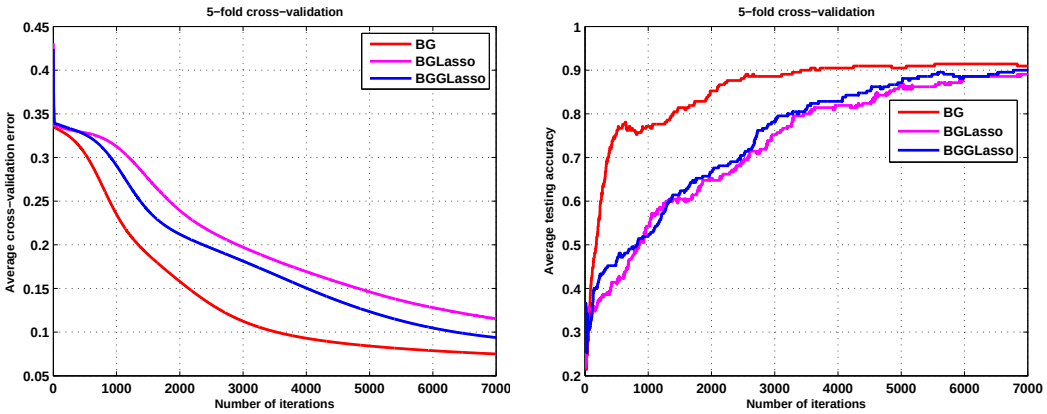
Figure 6. Best learning curves for zoo dataset by using BG, BGLasso and BGGLasso regularization methods with $\eta = 0.040, \lambda = 0.030$: (**a**) average cross-validation error and (**b**) average testing accuracy.

Table 3. Comparison of best average results for zoo dataset.

Methods	λ	η	**Sparsity**		$\|E_W(W)\|$	**Accuracy**	
			AVGNPWs	**AVGNPHNs**		**Training Acc .(%)**	**Testing Acc. (%)**
BG	0.000	0.040	0.000	0.000	0.0200	99.800	93.100
BGLasso	0.030	0.040	**314.400**	0.000	0.200	86.600	79.300
BGGLasso	0.030	0.040	47.800	**6.800**	**0.030**	93.600	87.100

3.3. The Seed Results

Similarly, to test the sparsity ability of the group Lasso and Lasso regularization terms using the seeds dataset, we started from a large network structure $8 - 16 - 3$ (including bias neurons in the input and hidden layers), and the maximum number of training epochs is 7000. The best learning rate η and the best regularization parameter λ for the seeds dataset are 0.080 and 0.009, respectively. The best learning curves for seeds dataset with average cross-validation error results and average testing accuracy results are shown in Figure 7a,b.

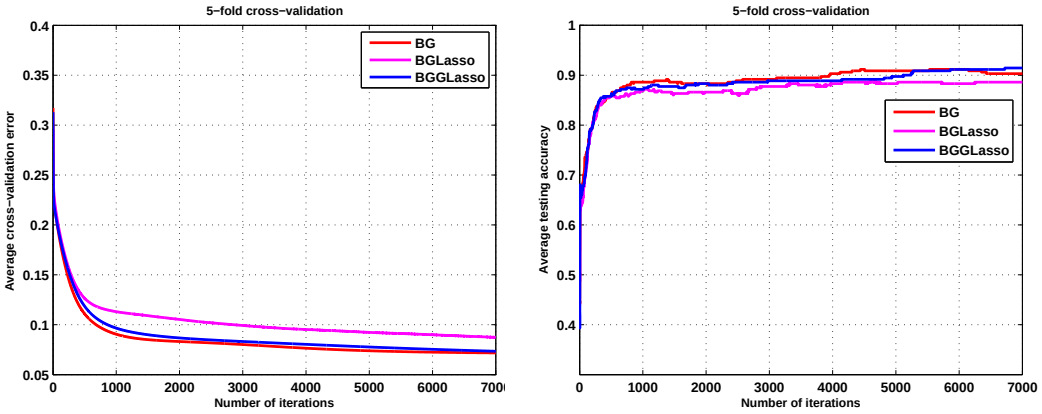


(**a**) (**b**)

Figure 7. Best learning curves for seeds dataset by using BG, BGLasso and BGGLasso regularization methods with $\eta = 0.080, \lambda = 0.009$: (**a**) average cross-validation error and (**b**) average testing accuracy.

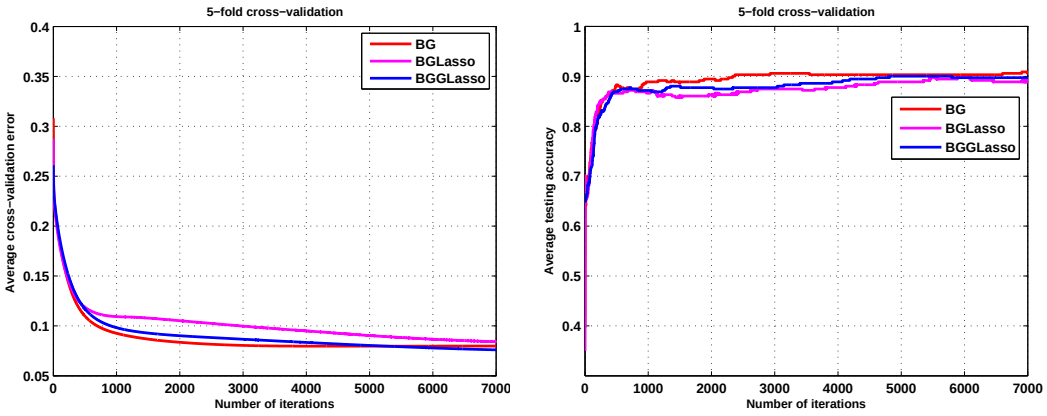
3.4. The Ionosphere Results

Here, we started with two different initial network structures ($35-41-2$ and $35-50-2$) (each network includes bias neurons in the input and hidden layers), and both took 7000 training epochs. We trained these two different network structures with the same learning parameters and the selected best learning rate η and λ are 0.050 and 0.007, respectively. The best learning curves for cross-validation error and average testing accuracy of the first network structure are shown in Figure 8a,b, respectively. Similarly, the best learning curves for average cross-validation error and average testing accuracy of the second network structure are also shown in Figure 9a,b, respectively.



(**a**) (**b**)

Figure 8. Best learning curves for ionosphere dataset by using BG, BGLasso and BGGLasso regularization methods with $\eta = 0.050, \lambda = 0.007$ and network structure $35-41-2$: (**a**) average cross-validation error and (**b**) average testing accuracy.



(**a**) (**b**)

Figure 9. Best learning curves for ionosphere dataset by using BG, BGLasso and BGGLasso regularization methods with $\eta = 0.050, \lambda = 0.007$ and network structure $35 - 50 - 2$: (**a**) average cross-validation error and (**b**) average testing accuracy.

4. Discussion

The main goal of this study is to prune the redundant or unnecessary hidden layer neurons of the FNNs. In this respect, the regularization terms are often introduced into the error function and have shown to be efficient to improve the generalization performance and decrease the magnitude of the network weights [31]. In particular, L_p regularizations are used to regularize the sum of the norm of the weights during training. Lasso [23] is one of the most popular L_p regularization terms that is used to remove the redundant weights. However, Lasso regularization is mainly introduced for removing the redundant weights, and a neuron can be removed only if all of its outgoing weights have been close to zero. As shown in Tables 2–6, the batch gradient method with Lasso regularization (BGLasso) can find more redundant weights, but it cannot find any redundant hidden layer neurons.

Group Lasso [26] was used for imposing the sparsity on group level to eliminate the redundant neurons of the network. As shown in Tables 2–6, the batch gradient method with group Lasso regularization term (BGGLasso) can identify unnecessary or redundant hidden layer neurons. The average number of pruned hidden layer neurons (AVGNPHNs) by BGGLasso is higher for each dataset. In these tables, the average norm of the gradient of the error function $\|E_W(W)\|$ for our proposed learning method is also smaller than BGLasso. This tells us that the BGGLasso converges better than BGLasso. Tables 4 and 6 are the results of ionosphere dataset using the same parameters except that the initial number of hidden layer neurons are different (i.e., $35 - 41 - 2$ and $35 - 50 - 2$), respectively. Here, we confirm that the results are not significantly different.

Table 4. Comparison of best average results for seeds dataset.

Methods	λ	η	**Sparsity**		$\|E_W(W)\|$	**Accuracy**	
			AVGNPWs	**AVGNPHNs**		**Training Acc. (%)**	**Testing Acc. (%)**
BG	0.000	0.080	0.000	0.000	0.0200	92.500	91.400
BGLasso	0.009	0.080	**101.600**	0.000	0.060	89.500	89.100
BGGLasso	0.009	0.080	24.600	**8.200**	**0.040**	91.700	90.900

Table 5. Comparison of best average results for ionosphere dataset with network structure $35-41-2$.

Methods	λ	η	Sparsity		$\|E_W(W)\|$	Accuracy	
			AVGNPWs	AVGNPHNs		Training Acc. (%)	Testing Acc. (%)
BG	0.000	0.050	0.000	0.000	0.020	98.800	92.000
BGLasso	0.007	0.050	**1327.800**	0.000	0.090	94.000	88.900
BGGLasso	0.007	0.050	45.600	**22.800**	**0.040**	97.100	91.400

Table 6. Comparison of best average results for ionosphere dataset with network structure $35-50-2$.

Methods	λ	η	Sparsity		$\|E_W(W)\|$	Accuracy	
			AVGNPWs	AVGNPHNs		Training Acc. (%)	Testing Acc. (%)
BG	0.000	0.050	0.000	0.000	0.020	98.000	90.900
BGLasso	0.007	0.050	**1646.400**	0.000	0.100	94.500	89.800
BGGLasso	0.007	0.050	60.400	**30.200**	**0.050**	96.900	90.000

Moreover, Figures 5a, 6a, 7a, 8a and 9a, display comparison results of the average cross-validation error obtained by different hidden layer regularization methods for iris, zoo, seeds, and ionosphere datasets, respectively. The x-axis represents the maximum number of iterations and the y-axis represents the average cross-validation error of every iteration. As we can see from these figures, BGGLasso is monotonically decreasing and converges more quickly with a smaller cross-validation error compared to the popular BGLasso regularization method. Similarly, Figures 5b, 6b, 7b, 8b and 9b depict the average testing accuracy results of hidden layer regularization methods for iris, zoo, seeds and ionosphere datasets, respectively. In each learning curves of these figures, the x-axis represents the maximum number of iterations and y-axis represents the average testing accuracy. From these learning curves, we can see that BGGLasso always has better classification performance on the validation sets as compared to BGLasso regularization method.

As seen from the above discussion, we find that our proposed BGGLasso regularization method outperforms the existing BGLasso regularization method in all numerical results. The importance of applying BGGLasso regularization method does not only result in more sparsity in hidden layer neurons but also achieves a much better test accuracy results than BGLasso. From the results of BGGLasso in Tables 2–6, the number of the redundant weights and the number of the redundant hidden layer neurons are proportional. This phenomenon indicates that the batch gradient method with a group Lasso regularization term has limitations on removing weight connections from surviving hidden layer neurons. All of our numerical results are obtained by applying our proposed method using one hidden layer of FNNs. One can extend our proposed approach for sparsification of FNNs that contains any number of hidden layers.

5. Conclusions

We have developed an efficient approach to eliminate the redundant or unnecessary hidden layer neurons from a feedforward neural network. To this end, we have proposed a group Lasso regularization method by considering only the outgoing weights from each hidden layer neuron. This method can easily help us to identify that the number of redundant or unnecessary hidden layer neurons with the norm of outgoing weight connections is less than the predefined threshold value. Not only does our proposed method identify the redundant hidden neurons, but it also yields a sparse network. The numerical simulations show that our proposed method outperforms the existing learning methods on the sparsity of hidden layer neurons and the results are consistent for each benchmark dataset. One key advantage of our proposed hidden layer regularization method is that it can sparse the redundant hidden neurons. By contrast, one disadvantage of our proposed regularization method is that it cannot sparse any redundant weights from the surviving hidden layer neurons. To fill this gap, in the future, we have intended to develop more regularization methods that can ultimately solve

this problem. Furthermore, we plan to test our proposed hidden layer regularization method with big datasets by using deep neural networks.

Author Contributions: H.Z.A. developed the mathematical model, carried out the numerical simulations and wrote the manuscript; W.W. advised on developing the learning algorithms and supervised the work; J.Z. contributed to analysis of the results.

Acknowledgments: This work is partially supported by the Natural Science Foundation of China: 61473059, 11401076 and 61473328; the Fundamental Research Funds for the Central Universities: DUT13-RC(3)068, DUT18JC02 and DUT17LK46; and the Dalian High Level Talent Innovation Support Program: 2015R057.

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

References

1. Haykin, S. *Neural Networks: A Comprehensive Foundation*; Prentice Hall PTR: Upper Saddle River, NJ, USA, 1994.
2. Lippmann, R. An introduction to computing with neural nets. *IEEE ASSP Mag.* **1987**, *4*, 4–22. [CrossRef]
3. Jain, A.K.; Mao, J.; Mohiuddin, K.M. Artificial neural networks: A tutorial. *Computer* **1996**, *29*, 31–44. [CrossRef]
4. Plawiak, P.; Tadeusiewicz, R. Approximation of phenol concentration using novel hybrid computational intelligence methods. *Int. J. Appl. Math. Comput. Sci.* **2014**, *24*, 165–181. [CrossRef]
5. Hinton, G.E. Connectionist learning procedures. *Artif. Intell.* **1989**, *40*, 185–234. [CrossRef]
6. Pławiak, P.; Rzecki, K. Approximation of phenol concentration using computational intelligence methods based on signals from the metal-oxide sensor array. *IEEE Sens. J.* **2015**, *15*, 1770–1783.
7. Plagianakos, V.P.; Sotiropoulos, D.G.; Vrahatis, M.N. An Improved Backpropagation Method with Adaptive Learning Rate. In Procceedings of the 2nd International Conference on: Circuits, Systems and Computers, Iraeus, Greece, 26–28 October 1998.
8. Rumelhart, D.E.; Hinton, G.E.; Williams, R.J. Learning representations by back-propagating errors. *Nature* **1986**, *323*, 533. [CrossRef]
9. Wilson, D.R.; Martinez, T.R. The general inefficiency of batch training for gradient descent learning. *Neural Netw.* **2003**, *16*, 1429–1451. [CrossRef]
10. Sietsma, J.; Dow, R.J. Neural net pruning-why and how. In Procceedings of the IEEE International Conference on Neural Networks, ISan Diego, CA, USA, 24–27 July 1988; Volume 1, pp. 325–333.
11. Setiono, R. A penalty-function approach for pruning feedforward neural networks. *Neural Comput.* **1997**, *9*, 185–204. [CrossRef] [PubMed]
12. Aran, O.; Yildiz, O.T.; Alpaydin, E. An Incremental Framework Based on Cross-Validation for Estimating the Architecture of a Multilayer Perceptron. *IJPRAI* **2009**, *23*, 159–190, doi:10.1142/S0218001409007132. [CrossRef]
13. Augasta, M.G.; Kathirvalavakumar, T. A Novel Pruning Algorithm for Optimizing Feedforward Neural Network of Classification Problems. *Neural Process. Lett.* **2011**, *34*, 241–258, doi:10.1007/s11063-011-9196-7. [CrossRef]
14. Augasta, M.G.; Kathirvalavakumar, T. Pruning algorithms of neural networks,a comparative study. *Cent. Eur. J. Comput. Sci.* **2013**, *3*, 105–115. [CrossRef]
15. LeCun, Y.; Denker, J.S.; Solla, S.A. Optimal brain damage. *Adv. Neural Inf. Process. Syst.* **1990**, *2*, 598–605.
16. Hassibi, B.; Stork, D.G.; Wolff, G.J. Optimal brain surgeon and general network pruning. In Proceedings of the IEEE International Conference on Neural Networks, San Francisco, CA, USA, 28 March–1 April 1993; pp. 293–299.
17. Chang, X.; Xu, Z.; Zhang, H.; Wang, J.; Liang, Y. Robust regularization theory based on $Lq(0 < q < 1)$ regularization: The asymptotic distribution and variable selection consistence of solutions. *Sci. Sin. Math.* **2010**, *40*, 985–998.
18. Xu, Z.; Zhang, H.; Wang, Y.; Chang, X.; Liang, Y. $L_{\frac{1}{2}}$ Regularizer. *Sci. China Inf. Sci.* **2010**, *53*, 1159–1169. [CrossRef]
19. Wu, W.; Fan, Q.; Zurada, J.M.; Wang, J.; Yang, D.; Liu, Y. Batch gradient method with smoothing $L_{\frac{1}{2}}$ regularization for training of feedforward neural networks. *Neural Netw.* **2014**, *50*, 72–78. [CrossRef] [PubMed]

20. Liu, Y.; Li, Z.; Yang, D.; Mohamed, K.S.; Wang, J.; Wu, W. Convergence of batch gradient learning algorithm with smoothing $L_{\frac{1}{2}}$ regularization for Sigma–Pi–Sigma neural networks. *Neurocomputing* **2015**, *151*, 333–341. [CrossRef]
21. Fan, Q.; Wu, W.; Zurada, J.M. Convergence of batch gradient learning with smoothing regularization and adaptive momentum for neural networks. *SpringerPlus* **2016**, *5*, 295. [CrossRef] [PubMed]
22. Li, F.; Zurada, J.M.; Liu, Y.; Wu, W. Input Layer Regularization of Multilayer Feedforward Neural Networks. *IEEE Access* **2017**, *5*, 10979–10985. [CrossRef]
23. Tibshirani, R. Regression shrinkage and selection via the lasso. *J. R. Stat. Soc. Ser. B (Methodol.)* **1996**, *58*, 267–288.
24. Yuan, M.; Lin, Y. Model selection and estimation in regression with grouped variables. *J. R. Stat. Soc. Ser. B (Stat. Methodol.)* **2006**, *68*, 49–67. [CrossRef]
25. Meier, L.; Van De Geer, S.; Bühlmann, P. The group lasso for logistic regression. *J. R. Stat. Soc. Ser. B (Stat. Methodol.)* **2008**, *70*, 53–71. [CrossRef]
26. Alvarez, J.M.; Salzmann, M. Learning the number of neurons in deep networks. In *Advances in Neural Information Processing Systems, Proceedings of the Annual Conference on Neural Information Processing Systems, Barcelona, Spain, 5–10 December 2016*; Neural Information Processing Systems Foundation, Inc.: La Jolla, CA, USA, 2016; pp. 2270–2278.
27. Dua, D.; Taniskidou, E.K. UCI Machine Learning Repository. University of California, Irvine, School of Information and Computer Sciences. Available online: http://archive.ics.uci.edu/ml (accessed on 14 October 2018).
28. Rodriguez, J.D.; Perez, A.; Lozano, J.A. Sensitivity analysis of k-fold cross validation in prediction error estimation. *IEEE Trans. Pattern Anal. Mach. Intell.* **2010**, *32*, 569–575. [CrossRef] [PubMed]
29. Han, J.; Pei, J.; Kamber, M. *Data Mining: Concepts and Techniques*; Elsevier: New York, NY, USA, 2011.
30. Zhang, H.; Tang, Y.; Liu, X. Batch gradient training method with smoothing L_0 regularization for feedforward neural networks. *Neural Comput. Appl.* **2015**, *26*, 383–390, doi:10.1007/s00521-014-1730-x. [CrossRef]
31. Reed, R. Pruning algorithms—A survey. *IEEE Trans. Neural Netw.* **1993**, *4*, 740–747, doi:10.1109/72.248452. [CrossRef] [PubMed]