A Novel Algorithm for Input Weights of Single-Hidden-Layer Neural Networks (Postprint)

Authors: Liu Jinpeng, Tián Dàgāng

Date: 2018-10-11T00:00:00+00:00

Abstract

To address the problem of random initialization of input weights and hidden layer thresholds in traditional Extreme Learning Machines, an output value back-allocation algorithm is proposed. The algorithm, based on the traditional Extreme Learning Machine, obtains optimal output value allocation coefficients through optimization methods and determines network input parameters using the least squares method. Experiments applying the proposed algorithm to commonly used datasets and comparisons with other improved Extreme Learning Machine algorithms demonstrate its good learning and generalization capabilities and simple network structures, thereby proving the algorithm's effectiveness.

Full Text

A New Algorithm for Input Weights of Single Hidden Layer Neural Networks

Liu Jinpeng, Tian Dagang
Business School, University of Shanghai for Science & Technology, Shanghai 200093, China

Abstract: To address the problem of randomly assigned input weights and hidden layer thresholds in traditional extreme learning machines, this paper proposes a back distribution algorithm for output values. Building upon the traditional extreme learning machine framework, the algorithm obtains optimal output value distribution coefficients through optimization methods and determines network input parameters using the least squares method. Experiments on commonly used datasets demonstrate that the proposed algorithm achieves good learning and generalization performance compared with other improved extreme learning machine algorithms, while obtaining simpler network structures, thereby proving its effectiveness.

Keywords: extreme learning machine; single-hidden layer neural networks; optimization; output back distribution

0 Introduction

Extreme learning machine (ELM), first proposed by Huang et al. in 2006, represents a novel single hidden layer feedforward neural network model whose computational speed is thousands of times faster than traditional feedforward neural networks while maintaining good generalization capability [1]. The core idea of ELM involves randomly generating network input weights and hidden layer thresholds, then determining output weights through the least squares method. However, due to the randomness of hidden layer parameters (input weights and hidden layer thresholds), it is difficult to guarantee optimal results [2,3].

To find optimal network parameters, Zhu et al. proposed evolutionary extreme learning machine (E-ELM), where input weights and hidden layer thresholds are optimized using differential evolution methods while output weights are calculated using the Moore-Penrose (M-P) generalized inverse. This algorithm offers fast learning speed but relatively poor generalization ability [4]. Huang et al. introduced incremental extreme learning machine (I-ELM), which effectively improves network learning speed but yields slightly inferior testing accuracy [5,6]. Rong et al. proposed pruning extreme learning machine (P-ELM), which can achieve good testing accuracy and simple network structures, but requires relatively long learning time [7]. Emilio et al. presented Bayesian extreme learning machine (BELM), which enhances network generalization capability, though the random parameter issue remains to be improved [8]. Han et al. employed hybrid learning algorithms to overcome ELM limitations, using improved particle swarm optimization extreme learning machine (PSO-ELM) to select input weights and hidden layer thresholds, achieving good network generalization capability but with complex model structures [9].

Overall, discussions on ELM have primarily focused on two aspects: how to determine the connection weights between the input layer and hidden layer, and how to determine the number of hidden layer nodes. This paper specifically addresses the problem of randomly selected connection weights between the input layer and hidden layer (input weights) and hidden layer thresholds in ELM, proposing a new algorithm for determining single hidden layer neural network weights—the output value backward distribution algorithm—which uses optimization methods to determine input weights. Numerical experiments demonstrate that this algorithm can solve the problem of random parameter initialization and achieve good generalization capability.

1 Basic Theory

1.1 Single Hidden Layer Feedforward Neural Networks

A single hidden layer feedforward neural network consists of an input layer, a single hidden layer, and an output layer [10]. Nodes in adjacent layers are fully connected through connection weights. Such a network can be represented by the following function:

$$f(X) = \sum v_i \varphi \left(\sum w_{ij} x_j + b_i \right) + v_0 = \sum v_i \varphi (W_i \cdot X + b_i)$$

where $X = (x_1, x_2, x_3, \dots, x_n)^T$ denotes the n-dimensional input vector, $W_i = (w_{i1}, w_{i2}, w_{i3}, \dots, w_{in})^T$ represents the connection weights between the input layer and the i-th hidden layer node (i.e., input weights) whose magnitude indicates the strength of connection between nodes, b_i is the threshold of the i-th hidden layer node (i.e., hidden layer threshold), v_i denotes the connection weight from the i-th hidden layer node to the output layer (i.e., output weight), $\varphi(\cdot)$ represents the hidden layer activation function (Sigmoid function is used in this paper), $W_i \cdot X$ denotes the inner product of W_i and X, and $f(X)$ is the network output.

Since the output neuron activation function is linear, the network output weight vector $V = (v_1, v_2, \dots, v_L)^T$ can be solved using the least squares method, which is how ELM operates. However, W and b are not easily obtained, a problem addressed in references [6][7][8]. This paper presents a new algorithm to determine hidden layer parameters (W, b).

1.2 Extreme Learning Machine Algorithm

The basic steps of extreme learning machine are as follows:

a) Randomly select hidden layer parameters (network input weights W_i and hidden layer thresholds b_i);

b) Calculate the hidden layer output matrix H based on hidden layer parameters;

c) Solve for output weights V.

As evident from above, a major challenge in ELM is the selection problem of W and b.

2 Output Value Backward Distribution Algorithm Based on ELM

Consider arbitrary N samples (X_j, Y_j), $j = 1, 2, \ldots, N$, where $X_j = (x_{j1}, x_{j2}, \ldots, x_{jn})^T$ is the n-dimensional input vector and $Y = (y_1, y_2, \ldots, y_N)^T$ represents the N sample outputs. Let the number of hidden layer nodes be L, the hidden layer activation function be the Sigmoid function, $V = (v_1, v_2, \ldots, v_L)^T$ be the output weights, W_i denote the connection weights between the input vector and the i-th hidden layer node (i.e., input weights), and b_i be the threshold of the i-th hidden layer node (i.e., hidden layer threshold).

The sample learning problem can be expressed as:

$$f(X_k) = \sum v_i \varphi(W_i^T \cdot X_k + b_i) + v_0 \approx y_k, \quad k = 1, 2, \cdots, N$$

where selecting appropriate output weights v_i, input weights W_i, and hidden layer thresholds b_i minimizes the error between sample outputs y_k and network outputs.

Since the network output function is linear, equation (2) constitutes a linear system of equations about $V = (v_1, v_2, \ldots, v_L)^T$. Given input weights and hidden layer thresholds, output weights can be obtained by solving this linear system.

Let the distribution coefficient vector be $\alpha = (\alpha_1, \alpha_2, \cdots, \alpha_L)$ with $0 \leq \alpha_i \leq 1$, $i = 1, 2, \ldots, L$. Let the output of hidden layer node i be y_{α_i}, i.e.:

$$\varphi(W_i^T \cdot X + b_i) = y_{\alpha_i}$$

Then:

$$W_i^T \cdot X + b_i = \varphi^{-1}(y_{\alpha_i})$$

Since the selected activation function is the Sigmoid function with range $(0, 1)$, to ensure $\varphi^{-1}(y_{\alpha_i})$ is well-defined, sample outputs y must be transformed to lie within $(0, 1)$.

Substituting samples into equation (4) yields the following system:

$$\begin{cases} \varphi(W_1^T \cdot X_1 + b_1) = y_1 \alpha_i \\ \varphi(W_1^T \cdot X_2 + b_1) = y_2 \alpha_i \\ \vdots \\ \varphi(W_1^T \cdot X_N + b_1) = y_N \alpha_i \end{cases}$$

From equation (5), we obtain:

$$\begin{cases} W_i^T \cdot X_1 + b_i = \varphi^{-1}(y_1\alpha_i) \\ W_i^T \cdot X_2 + b_i = \varphi^{-1}(y_2\alpha_i) \\ \vdots \\ W_i^T \cdot X_N + b_i = \varphi^{-1}(y_N\alpha_i) \end{cases}$$

For brevity, denote:

$$\tilde{W}_i = \begin{pmatrix} W_i \\ b_i \end{pmatrix}, \quad \tilde{X}_j = \begin{pmatrix} X_j \\ 1 \end{pmatrix}$$

where $\tilde{W}_i$ is an $(n+1)$-dimensional vector, $i = 1, 2, \ldots, L$, and $\tilde{X}_j$ is an $(n+1)$-dimensional vector, $j = 1, 2, \ldots, N$. Then:

$$\tilde{W} = \begin{pmatrix} \tilde{W}_1 & \tilde{W}_2 & \cdots & \tilde{W}_L \end{pmatrix} = \begin{pmatrix} W_1 & W_2 & \cdots & W_L \\ b_1 & b_2 & \cdots & b_L \end{pmatrix}$$

$$\tilde{X} = \begin{pmatrix} \tilde{X}_1 & \tilde{X}_2 & \cdots & \tilde{X}_N \end{pmatrix} = \begin{pmatrix} X_1 & X_2 & \cdots & X_N \\ 1 & 1 & \cdots & 1 \end{pmatrix}$$

where $\tilde{W}$ is an $(n+1) \times L$ matrix and $\tilde{X}$ is an $(n+1) \times N$ matrix. The hidden layer neuron output matrix H can be expressed as:

$$H = \begin{pmatrix} \varphi(\tilde{W}_1^T \cdot \tilde{X}_1) & \varphi(\tilde{W}_L^T \cdot \tilde{X}_1) \\ \varphi(\tilde{W}_1^T \cdot \tilde{X}_2) & \varphi(\tilde{W}_L^T \cdot \tilde{X}_2) \\ \vdots & \vdots \\ \varphi(\tilde{W}_1^T \cdot \tilde{X}_N) & \varphi(\tilde{W}_L^T \cdot \tilde{X}_N) \end{pmatrix}_{N \times (L+1)} = \varphi(\tilde{X}^T \tilde{W})$$

In the coefficient matrix, when $A = (a_{ij})$, $\varphi(A)$ denotes the matrix $(\varphi(a_{ij}))$.

Thus, equation system (6) can be simplified as:

$$\tilde{X}^T \tilde{W}_i = \varphi^{-1}(Y\alpha_i)$$

where $\varphi^{-1}(Y\alpha_i) = (\varphi^{-1}(y_1\alpha_i), \varphi^{-1}(y_2\alpha_i), \cdots, \varphi^{-1}(y_N\alpha_i))^T$. Therefore, the least squares solution in equation (7) is:

$$\tilde{W}_i = (\tilde{X}^T)^\dagger \varphi^{-1}(Y\alpha_i) = (\tilde{X}\tilde{X}^T)^{-1}\tilde{X}\varphi^{-1}(Y\alpha_i), \quad i = 1, 2, \cdots, L$$

Equation (8) expresses the solution containing the output value distribution coefficient vector α. To determine α, we minimize the following objective function under constraints:

$$\min_{\alpha} \left\| \sum v_i \varphi \left(\tilde{X}^T (\tilde{X}^T)^{\dagger} \varphi^{-1}(Y\alpha_i) \right) - Y \right\|$$

where the L_2 norm is used. Thus, the problem of solving input weights and hidden layer thresholds is transformed into an optimization problem involving the output value distribution coefficient vector α:

$$\begin{aligned} &\min_{\alpha} \left\| \sum v_i \varphi \left(\tilde{X}^T (\tilde{X}^T)^{\dagger} \varphi^{-1}(Y\alpha_i) \right) - Y \right\| \\ &\text{s.t.} \quad 0 \le \alpha_i \le 1 \end{aligned}$$

The optimal α can be obtained from equation (9). After determining α, $\tilde{W}_i$ is calculated according to equation (8), and V is then determined using the ELM algorithm:

$$V = H^{\dagger} Y = (H^T H)^{-1} H^T Y$$

Let $e(V, \alpha) = \left\| \sum v_i \varphi \left(\tilde{X}^T (\tilde{X}^T)^{\dagger} \varphi^{-1}(Y\alpha_i) \right) - Y \right\|$. The basic steps of the output value backward distribution algorithm are:

Input: X, Y, L, ε
Output: W, V

a) Randomly select output weights $V = (v_1, v_2, \dots, v_L)^T$ where $v_i \neq 0$;

b) Solve for the optimal output value distribution coefficient vector $\alpha = (\alpha_1, \alpha_2, \cdots, \alpha_L)^T$ using equation (9);

c) Calculate input weights and hidden layer thresholds $\tilde{W}_i$ using equation (8);

d) Determine new output weights V using equation (10);

e) If $e(V, \alpha) < \varepsilon$, terminate the algorithm; otherwise return to step b).

The algorithm is clearly convergent. In fact, $e(V_0, \alpha_0) \ge e(V_1, \alpha_0) \ge e(V_1, \alpha_1) \ge e(V_2, \alpha_1) \ge \cdots \ge 0$.

The proposed output value backward distribution algorithm retains the advantages of ELM since V is obtained through least squares, while W is calculated via α using $\tilde{W}_i = (\tilde{X}^T)^{\dagger} \varphi^{-1}(Y\alpha_i) = (\tilde{X}\tilde{X}^T)^{-1} \tilde{X} \varphi^{-1}(Y\alpha_i)$, where α is obtained through optimization algorithms. This approach is expected to yield better results than random selection. Moreover, if any $\alpha_i = 0$ appears in the computation results, the corresponding hidden layer node can be removed, giving the algorithm the ability to select hidden layer nodes.

Equation (9) does not guarantee a convex optimization problem, but computational performance is satisfactory, particularly for classification problems, though the underlying reasons require further investigation.

3 Experiments

3.1 Experimental Data

All experimental datasets used in this paper are sourced from the UCI Machine Learning Repository [13], which contains real-world data commonly used in machine learning research. This paper selects the following datasets: Iris, Wine, Pima Indians Diabetes, Wisconsin Breast Cancer, Heart, Balance Scale, Haberman' s Survival, and User Knowledge Modeling. Test samples are randomly selected as shown in Table 1 .

Table 1. Basic Attributes of Datasets

Dataset Name	Attributes	Classes
Iris	4	3
Wine	13	3
Pima I.D	8	2
Wisconsin B.C	9	2
Heart	13	2
Balance Scale	4	3
Haberman' s S	3	2
User K.M	5	4

3.2 Experimental Process and Results

a) Iris dataset. This example classifies samples into three categories corresponding to 1, 2, and 3. The activation function in the output value backward distribution algorithm is the Sigmoid function, requiring transformation of the desired output to (0, 1) using:

$$y_0 = \frac{y - \min(y)}{\max(y) - \min(y)}$$

where y_0 is the transformed desired output and y is the original sample desired output.

Applying the proposed algorithm, we construct a single hidden layer network with three hidden nodes (i.e., $L = 3$ in equation (2)). With randomly assigned output weights and substituting training and test samples into the algorithm, the final network input weights are:

$$W = \begin{pmatrix} -0.0780 & -0.0515 & -0.1784 \\ -0.1784 & -0.1784 & -0.1784 \end{pmatrix}$$

Hidden layer thresholds $b = (-27.4835, -12.9238, -12.9245)^T$, and final network output weights $V = (1.1625 \times 10^{12}, 2.2334 \times 10^3, -2.2333 \times 10^3)^T$.

b) Wine dataset. This example classifies samples into 3 categories. Before experiments, categories are transformed using equation (11). A single hidden layer network with 20 hidden nodes is initially constructed. Through learning with the proposed algorithm, $\alpha_1, \alpha_4, \alpha_9, \alpha_{10}, \alpha_{13}, \alpha_{14}, \alpha_{15}, \alpha_{16}, \alpha_{17}, \alpha_{18}$ all become zero, allowing removal of corresponding hidden nodes to obtain a network with 10 hidden nodes. Network input weights are:

$$W = \begin{pmatrix} -0.0094 & 8.4085 \times 10^{-5} & -0.0269 & \cdots & -0.0028 \\ 1.4270 \times 10^{-5} & -0.0243 & -0.0178 & \cdots & -0.0113 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -2.7344 \times 10^{-5} & -0.0028 & 4.3188 \times 10^{-4} & \cdots & -0.0113 \end{pmatrix}$$

Hidden layer thresholds $b = (-0.0390, 0.0102, -0.0471, -0.0094, 0.1210, 0.0812, 0.0835, -0.0508, -0.0234, 0.0754$ and network output weights $V = (7.0693 \times 10^5, -1.9033 \times 10^{10}, 8.0851 \times 10^{10}, 6.6569 \times 10^{10}, -8.4713 \times 10^5, -2.1497 \times 10^{10}, 1.2798 \times 10^{10}, -5.6410 \times 10^{10})^T$.

c) Pima Indians Diabetes dataset. After transforming the sample desired outputs using equation (11), single hidden layer networks with 4, 2, and 7 hidden nodes are constructed for learning. Experimental results are shown in Table 2 . Due to space limitations, network input weights, hidden layer thresholds, and output weights are omitted.

d) Wisconsin Breast Cancer dataset. This dataset contains two classes (benign and malignant) corresponding to 1 and 0. A single hidden layer network with 1 hidden node is constructed, yielding network input weights $W = (-1.4687 \times 10^{-10}, 6.5666 \times 10^{-10}, 1.6532 \times 10^{-9}, -9.6924 \times 10^{-9}, -36.9432 \times 10^{-9}, 21.4134 \times 10^{-9}, -107.7062 \times 10^{-9})$, hidden layer threshold $b = 40.6548$, and network output weights $V = 1.0269$.

e) Heart dataset. This dataset contains 13 attributes as sample inputs, with sample outputs representing class membership (classes 1 and 2). After transformation using equation (11), a single hidden layer network with 2 hidden nodes is constructed, yielding network input weights:

$$W = \begin{pmatrix} -0.0041 & 4.6387 \times 10^{-4} & -0.0022 & \cdots & 0.7403 \\ -0.0709 & -0.0389 & 0.7403 & \cdots & 0.4878 \end{pmatrix}$$

Hidden layer thresholds $b = (-24.6254, -3.4921)^T$, and output weights $V = (2.5763 \times 10^{10}, 0.4878)^T$.

f) Balance Scale, Haberman's Survival, and User Knowledge Modeling datasets. Balance Scale is a multi-class dataset with desired outputs -1, 0, and 1; Haberman' s Survival is binary classification with outputs 1 and 2; User

Knowledge Modeling is multi-class with outputs 1, 2, 3, and 4. All undergo transformation using equation (11).

Network outputs y are rounded to obtain y^* and compared with desired outputs, producing the experimental results shown in Table 2 .

Table 2. Experimental Results on Datasets

Dataset	RMSE	TestRMSE	Training Accuracy	Testing Accuracy	Node Count
Pima I.D	0.19	0.20	77.32%	79.57%	2
Wisconsin B.C	0.10	0.12	97.89%	95.79%	1
Heart	0.15	0.16	86.67%	84.44%	2
Balance Scale	0.12	0.14	90.59%	88.24%	3
Haberman's S	0.18	0.20	73.04%	78.43%	2
User K.M	0.08	0.10	94.19%	98.62%	4

For the Iris, Wine, Pima Indians Diabetes, and Wisconsin Breast Cancer examples, the proposed algorithm is compared with several improved ELM algorithms from literature: OP-ELM [14], H-ELM [15], traditional ELM [15], SBELM [16], and PSO-ELM [17], as shown in Table 3 .

Table 3. Comparison of Classification Results

Dataset	Algorithm	Accuracy	Nodes
Iris	OP-ELM	98.33%	N1=N2=20; N3=200
	SBELM	98.33%	2.4×3
	Proposed	**98.33%**	**3**
Wine	OP-ELM	90.7%	N1=N2=20; N3=500
	SBELM	99.41%	3.3×3
	Proposed	**98.3%**	**10**
Pima I.D	OP-ELM	80.47%	N1=N2=10; N3=200
	SBELM	76.95%	-
	PSO-ELM	76.38%	-
	Proposed	**78.66%**	**2**
Wisconsin B.C	OP-ELM	95.6%	-
	SBELM	97.22%	-
	Proposed	**95.79%**	**1**

Comparison shows that the proposed algorithm achieves comparable accuracy

to the aforementioned algorithms. Except for the Wine dataset where node counts differ slightly from SBELM, the proposed algorithm requires the fewest hidden nodes for all other datasets. According to statistical learning theory, networks with fewer nodes have a higher probability of better generalization capability.

All experimental data are classification datasets, primarily because the proposed algorithm is an output value backward distribution algorithm. Following this approach, when samples represent classification problems, output values consist of only a few discrete numbers, making them easier to distribute using several α_k values. Consequently, the algorithm performs well on classification problems but yields poor fitting results for regression datasets with continuous output values.

---

4 Conclusion

This paper proposes an output value backward distribution algorithm to determine input weights and hidden layer thresholds for single hidden layer neural networks. Input weights are determined through the distribution coefficient vector α, which is obtained via optimization methods. Experimental results demonstrate that the proposed algorithm can achieve simple single hidden layer network structures while maintaining good learning quality.

References

[1] Huang GuangBin, Zhu QinYu, Siew C K. Extreme learning machine: theory and applications [J]. Neurocomputing, 2006, 70 (1-3): 489-501.

[2] Niu Peifeng, Ma Yunpeng, Liu Weiyan, et al. Improvement and application of extreme learning machine algorithm [J]. Journal of Yanshan University, 2015 (2): 127-132.

[3] Lu Haifeng, Wei Wei, Yang Mengyue. Receptive field class-constrained extreme learning machine [J/OL]. Application Research of Computers, 2018, 35 (10). http://www.arocmag.com/article/02-2018-10-009.html.

[4] Zhu Qinyu, Qin A K, Suganthan P N, et al. Evolutionary extreme learning machine [J]. Pattern Recognition, 2005, 38 (10): 1759-1763.

[5] Huang Guangbin, Chen Lei, Siew C. K. Universal approximation using incremental constructive feedforward networks with random hidden nodes [J]. IEEE Trans on Neural Networks, 2006, 17 (4): 879-892.

[6] Huang Guangbin, Chen Lei. Convex incremental extreme learning machine [J]. Neurocomputing, 2007, 70 (16-18): 3056-3062.

[7] Rong H J, Ong Y S, Tan A H, et al. A fast pruned-extreme learning machine for classification problem [J]. Neurocomputing, 2008, 72: 359-366.

[8] Soria Olivas E, Gomez Sanchis J, Martin J D, et al. BELM: Bayesian extreme learning machine [J]. IEEE Trans on Neural Networks, 2011, 22 (3): 405-410.

[9] Han F, Yao H F, Ling Q H. An improved evolutionary extreme learning machine based on particle swarm optimization [J]. Neurocomputing, 2013, 116: 87-93.

[10] Hagan M T, Demuth H B. Neural Network Design [M]. Beijing: China Machine Press, 2002: 8-15.

[11] Huang Gao, Huang Guangbin, Song Shiji, et al. Trends in extreme learning machines [J]. Neural Networks, 2015, 61 (C): 32-48.

[12] Serre D. Matrices: Theory and Applications [M]. Springer, New York, 2002: 18-25.

[13] UCI Machine Learning Repository [DB/OL]. http://archive.ics.uci.edu/ml/datasets.html.

[14] Miche Y, Sorjamaa A, Bas P, et al. OP-ELM: Optimally Pruned Extreme Learning Machine [J]. IEEE Trans on Neural Networks, 2010, 21 (1): 158-162.

[15] Tang Jiexiong, Deng Chenwei, Huang Guangbin. Extreme Learning Machine for Multilayer Perceptron [J]. IEEE Trans on Neural Networks and Learning Systems, 2016, 27 (4): 809-821.

[16] Luo Jiahua, Vong C M, Wong P K. Sparse Bayesian Extreme Learning Machine for Multi-classification [J]. IEEE Trans on Neural Networks and Learning Systems, 2014, 25 (4): 836-843.

[17] Gu Xiujun. A Study of Extreme Learning Machine Based on Particle Swarm Optimization And Its Application [D]. Zhenjiang: Jiangsu University, 2013.

Note: Figure translations are in progress. See original paper for figures.