Noniterative Deep Learning: Incorporating Restricted Boltzmann Machine Into Multilayer Random Weight Neural Networks

Xi-Zhao Wang, *Fellow, IEEE,* Tianlun Zhang, and Ran Wang, *Member, IEEE*

***Abstract*—A general deep learning (DL) mechanism for a multiple hidden layer feed-forward neural network contains two parts, i.e., 1) an unsupervised greedy layer-wise training and 2) a supervised fine-tuning which is usually an iterative process. Although this mechanism has been demonstrated in many fields to be able to significantly improve the generalization of neural network, there is no clear evidence to show which one of the two parts plays the essential role for the generalization improvement, resulting in an argument within the DL community. Focusing on this argument, this paper proposes a new DL approach to train multilayer feed-forward neural networks. This approach uses restricted Boltzmann machine (RBM) as the layer-wise training and uses the generalized inverse of a matrix as the supervised fine-tuning. Different from the general deep training mechanism like back-propagation (BP), the proposed approach does not need to iteratively tune the weights, and therefore, has many advantages such as quick training, better generalization, and high understandability, etc. Experimentally, the proposed approach demonstrates an excellent performance in comparison with BP-based DL and the traditional training method for multilayer random weight neural networks. To a great extent, this paper demonstrates that the supervised part plays a more important role than the unsupervised part in DL, which provides some new viewpoints for exploring the essence of DL.**

***Index Terms*—Deep learning (DL), generalized inverse of matrix, random weight neural network (RWNN), supervised learning, training without iteration.**

Manuscript received July 6, 2016; accepted April 20, 2017. Date of publication May 18, 2017; date of current version June 14, 2019. This work was supported in part by the National Natural Science Foundation of China under Grant 71371063, Grant 61170040, Grant 61402460, and Grant 61472257, in part by the Natural Science Foundation of SZU under Grant 2017060, in part by the Basic Research Project of Knowledge Innovation Program in Shenzhen under Grant JCYJ20150324140036825, in part by the Guangdong Provincial Science and Technology Plan Project under Grant 2013B040403005, and in part by the HD Video Research and Development Platform for Intelligent Analysis and Processing in Guangdong Engineering Technology Research Centre of Colleges and Universities under Grant GCZX-A1409. This paper was recommended by Associate Editor F. Sun. *(Corresponding author: Ran Wang.)*

X.-Z. Wang is with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen 518060, China (e-mail: xizhaowang@ieee.org).

T. Zhang is with the Information Science and Technology College, Dalian Maritime University, Dalian 116026, China (e-mail: threekingdomst@163.com).

R. Wang is with the College of Mathematics and Statistics, Shenzhen University, Shenzhen 518060, China (e-mail: wangran@szu.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSMC.2017.2701419

I. Introduction

RECENT years have witnessed successful applications of deep neural networks (DNNs) in image classification [5], [21] and speech recognition [4], [12]. Because of these successful applications, many scholars and engineers in different fields began to study the structure of feed-forward neural networks with multiple hidden layers. Applications of DNNs are being extended to many specific domains to deal with complex classification and recognition problems with big data [35], and the performance is being shown very excellent [3], [20], [22]. Since Hinton started the study on DNN in 2006 [14], one can find from references many deep learning (DL) algorithms now [11], [13], [28]. Basically, two tools have been suggested by Hinton, i.e., RBM and deep belief network (DBN). These two tools have become the most widely used and the most popular terms in DL. A major reason for the successful application of DNNs to different domains is the high capability of feature representation, i.e., DL can extract high quality features that have significant semantic meanings from raw input samples through multiple layers. It is well known that, for classification and recognition problems, feature selection, or extraction has an essentially important impact on the learning process as well as the output results [23], [38], [39].

On the other hand, classifier design is the most fundamental issue for any classification problem. So far, one can find from references a considerable number of learning approaches to classifier design [30], [32], [34], [36], [37]. Among them, back-propagation (BP) and their improved versions are typical training algorithms for feed-forward neural networks. We now focus on a special approach to train feed-forward neural networks, i.e., random assignment of input weights and biases. This training mechanism was first proposed in 1992 by Pao and Takefji [26] and Schmidt *et al.* [27] for single-hidden layer feed-forward neural networks (SLFNs). It was improved in 1994 by Pao *et al.* [25], which pointed out that most linking-weights are not important, thus they are not necessary to be iteratively tuned. Furthermore, the approximation ability of this training method was demonstrated in [17], and the universal approximation theorem of this type of neural networks has been given implicitly in [2].

This type of neural networks and its training methodology were further investigated in 2006 by Huang *et al.* [15], [16] with the new name extreme learning machine (ELM). The essential idea of ELM is identical to Schmidt and Pao's

works [26], [27], but there are some technical handling differences. In ELM, the training consists of two parts. The first part is to randomly assign weights from a predefined interval between input and hidden layers, while the second part is analytically to compute the generalized inverse (GI) of a matrix for weights between hidden and output layers. During the recent decade, one can see rapid growth of studies on this type of neural networks, including their structures [7], [31], [33] and training algorithms [1], [6], [24], [40]. In this paper, we call this type of neural networks as random weight neural network (RWNN) and its training mechanism as random weight assignment (RWA). It summarizes the essence of the works proposed in [2], [15]–[17], and [25]–[27] and aims to highlight the impact of randomness.

It is easy to extend RWA from SLFNs to multiple-hidden layer feed-forward neural networks (MLFNs). That is, weights of all hidden layers are randomly assigned from a predefined interval while weights of output layer are analytically determined by evaluating a GI of matrix or by solving the normal system of linear matrix equations. Initially, the RWA training strategy is to overcome the defect of the extra-high computational complexity when using BP to train an MLFN. Experimentally, RWA shows a much faster training speed than BP while maintains similar generalization ability. However, BP-based DL may acquire the semantic meaning of the learned weights but RWA-based leaning cannot. The reason is that RWA-based learning lacks the iterative tuning of weights and highlights the training efficiency. Thus, BP-based DL can be regarded as an approach to feature extraction but RWA-based learning cannot.

In this paper, a new deep training approach for MLFNs is proposed. For a classification problem, the proposed training approach has two components. The first is an initial extractor of features based on RBM, and the second is a solution of a system of linear matrix equations. It is noteworthy that the proposed approach has no iterative tuning of parameters, thus it is essentially an improved version of RWA-based training method by replacing the randomly assigned weights with RBM-based initial weights while keeping the output layer weights acquired analytically. Experimentally, it shows better generalization ability than the original RWA training and faster speed than BP-based deep training.

The remainder of this paper is organized as follows. Sections II and III give a brief review on the related works. Section IV proposes the new training scheme for MLFNs. Section V conducts experimental comparisons to show the feasibility and effectiveness of the proposed approach. Finally, Section VI concludes this paper.

II. Review on Restricted Boltzmann Machine

In this section, some basic concepts related to RBM are recalled, and related leaning mechanism including the network structure and the training algorithm is introduced.

RBM is a probabilistic graphical model which can be explained by stochastic neural network. An RBM can be used to learn a probability distribution over a set of inputs [12]. Fig. 1 shows the structure of an RBM.

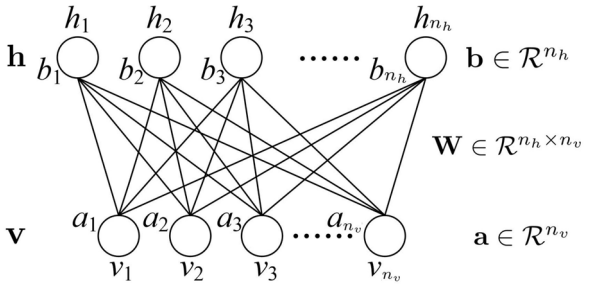


Fig. 1. Structure of an RBM.

As shown in Fig. 1, the topology of RBM is a two-layer graph. The underlying structure of the network is used to receive the input data, which is called the visible layer represented by $\mathbf{v} = [v_1, v_2, \ldots, v_{n_v}]$, and the upper structure of the network is used to generate a new feature vector, which is called the hidden layer represented by $\mathbf{h} = [h_1, h_2, \ldots .h_{n_h}]$. For the sake of simplicity, this section assumes that the data of the visible and hidden layers in the RBM are subject to Bernoulli distribution. For real-valued attributes, one can refer to [12]. The discrete RBM only takes value 0 or 1, and its training is an unsupervised process during which only the feature vector is needed but the tag data (i.e., the label of a sample) is not required.

Given a training set $\mathbb{S} = \{\mathbf{v}^1, \mathbf{v}^2, \ldots, \mathbf{v}^N\}$ that contains N samples, where $\mathbf{v}^m = [v_1^m, v_2^m, \ldots, v_{n_v}^m], m = 1, \ldots, N$ is the mth training sample. Then, RBM can be considered as an energy model [28]. Given a set of network states $(\mathbf{v}, \mathbf{h})$, RBM network can be corresponding to an energy value through

$$E_\theta(\mathbf{v}, \mathbf{h}) = -\sum_{j=1}^{n_v} a_j v_j - \sum_{i=1}^{n_h} b_i h_i - \sum_{i=1}^{n_h}\sum_{j=1}^{n_v} h_i w_{i,j} v_j \quad (1)$$

where $\mathbf{a} = [a_1, a_2, \ldots, a_{n_v}] \in \mathcal{R}^{n_v}$ is the visible layer bias, $\mathbf{b} = [b_1, b_2, \ldots, b_{n_h}] \in \mathcal{R}^{n_h}$ is the hidden layer bias, and $\mathbf{W} = [w_{i,j}] \in \mathcal{R}^{n_h \times n_v}$ is the weight matrix.

According to (1), the joint probability distribution of a set of visible and hidden states can be obtained by

$$p_\theta(\mathbf{v}, \mathbf{h}) = \frac{1}{Z_\theta} e^{-E_\theta(\mathbf{v}, \mathbf{h})} \quad (2)$$

where $Z_\theta = \sum_{\mathbf{v},\mathbf{h}} e^{-E_\theta(\mathbf{v},\mathbf{h})}$ is the normalization factor. The likelihood function $p(\mathbf{v}|\theta)$ can be derived by using the joint probability distribution function (2) as follows:

$$p(\mathbf{v}|\theta) = \frac{1}{Z_\theta} \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h}|\theta)}. \quad (3)$$

Formulas (1)–(3) form a statistical mechanics model and can be used in the training of RBM. We then introduce the training principle for RBM.

A free energy function is defined as

$$\text{FreeEntropy}(\mathbf{v}) = -\ln \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} \quad (4)$$

and based on (4) we can rewrite (3) as

$$p(\mathbf{v}|\theta) = \frac{e^{-\text{FreeEntropy}(\mathbf{v})}}{Z_\theta}. \quad (5)$$

Taking logarithm for both sides of equality (5) we have

$$\ln p(\mathbf{v}|\theta) = -\text{FreeEntropy}(\mathbf{v}) - \ln Z_\theta. \tag{6}$$

Taking a summation of equality (6) for all vectors $\mathbf{v}$ we obtain

$$\ln \prod_{\mathbf{v}} p(\mathbf{v}|\theta) = -\sum_{\mathbf{v}} \text{FreeEntropy}(\mathbf{v}) - \ln \prod_{\mathbf{v}} Z_\theta. \tag{7}$$

Equality (7) indicates that there exists a negative relationship between the free energy and the likelihood function $\ln \prod_{\mathbf{v}} p(\mathbf{v}|\theta)$ for an RBM. Based on the well-known principle of minimum free energy in physical energy systems, we suppose that the free energy term in (7) attains minimum. It is noted that minimizing the free energy term in (7) is equivalent to maximizing $\ln \prod_{\mathbf{v}} p(\mathbf{v}|\theta)$, i.e., the left side of (7). Thus, the task is to search for a set of parameters which can make $\ln \prod_{\mathbf{v}} p(\mathbf{v}|\theta)$ achieve its maximum. Noting that the logarithm function is monotonically increasing, we only need to find a set of parameters that can make $\prod_{\mathbf{v}} p(\mathbf{v}|\theta)$ attain its maximum. For similarity, we abbreviate $p(\mathbf{v}|\theta)$ as $p(\mathbf{v})$.

Subsequently, we discuss how to adjust the parameters of RBM network according to the principle of maximum likelihood. The likelihood function is represented as

$$\ln L_{\theta,\mathbb{S}} = \ln \prod_{m=1}^{N} p\big(\mathbf{v}^m\big) = \sum_{m=1}^{N} \ln p\big(\mathbf{v}^m\big). \tag{8}$$

The purpose of training RBM is to get the optimal value of parameter θ, that is

$$\theta^* = \underset{\theta}{\text{argmax}}\big(\ln L_{\theta,\mathbb{S}}\big) \tag{9}$$

where θ^* is the optimal value that makes the free energy of RBM system be minimum. The gradient descent technique can be used to find the maximum of $\ln L_{\theta,\mathbb{S}}$ with respect to the parameter θ. Taking the partial derivative of (8) with respect to θ, we have

$$\frac{\partial \ln L_{\theta,\mathbb{S}}}{\partial \theta} = \sum_{m=1}^{N} \frac{\partial \ln p(\mathbf{v}^m)}{\partial \theta}. \tag{10}$$

Replacing parameter θ with $(\mathbf{W}, \mathbf{a}, \mathbf{b})$, we can get the following formulas:

$$\begin{cases} \dfrac{\partial \ln L_{\theta,\mathbb{S}}}{\partial w_{i,j}} = \displaystyle\sum_{m=1}^{N}\left[p(h_i = 1|\mathbf{v}^m)v_j^m - \sum_{\mathbf{v}} p(\mathbf{v})p(h_i = 1|\mathbf{v})v_j\right] \\ \dfrac{\partial \ln L_{\theta,\mathbb{S}}}{\partial a_j} = \displaystyle\sum_{m=1}^{N}\left[v_j^m - \sum_{\mathbf{v}} p(\mathbf{v})v_j\right] \\ \dfrac{\partial \ln L_{\theta,\mathbb{S}}}{\partial b_i} = \displaystyle\sum_{m=1}^{N}\left[p(h_i = 1|\mathbf{v}^m) - \sum_{\mathbf{v}} p(\mathbf{v})p(h_i = 1|\mathbf{v})\right] \end{cases} \tag{11}$$

where $i = 1, 2, \ldots, n_h$ (n_h is the number of hidden layer nodes), $j = 1, 2, \ldots, n_v$ (n_v is the number of visible neurons), and the conditional probability is given by

$$\begin{cases} p(h_i = 1|\mathbf{v}) = \text{sigmoid}\Big(b_i + \sum_{j=1}^{n_v} w_{i,j}v_j\Big) \\ p\big(v_j = 1|\mathbf{h}\big) = \text{sigmoid}\big(a_j + \sum_{i=1}^{n_h} w_{i,j}h_i\big). \end{cases} \tag{12}$$

Algorithm 1: Parameter Updating Algorithm

Input:
Training set $\mathbb{S}$;
Number of iteration ITER;
Learning rate η;
Number of hidden neurons n_h.
Output:
Estimation of parameters $(\mathbf{W}^*, \mathbf{a}^*, \mathbf{b}^*)$.

1 Initialize the weight matrix $\mathbf{W}$, the visible layer bias $\mathbf{a}$ and the hidden layer bias $\mathbf{b}$ based on $\mathbb{S}$ and n_h;
2 Let $\mathbf{W}^1 = \mathbf{W}$, $\mathbf{a}^1 = \mathbf{a}$, $\mathbf{b}^1 = \mathbf{b}$;
3 **for** $t = 1$ *to* ITER **do**
4 Compute the gradient $(\Delta\mathbf{W}^t, \Delta\mathbf{a}^t, \Delta\mathbf{b}^t)$ using (13) of CD-1;
5 Update parameters by

$$\begin{cases} \mathbf{W}^{t+1} = \mathbf{W}^t + \eta\Big(\frac{1}{N}\Delta\mathbf{W}^t\Big) \\ \mathbf{a}^{t+1} = \mathbf{a}^t + \eta\Big(\frac{1}{N}\Delta\mathbf{a}^t\Big) \\ \mathbf{b}^{t+1} = \mathbf{b}^t + \eta\Big(\frac{1}{N}\Delta\mathbf{b}^t\Big); \end{cases} \tag{14}$$

6 **end**
7 Let $\mathbf{W}^* = \mathbf{W}^{\text{ITER}+1}$, $\mathbf{a}^* = \mathbf{a}^{\text{ITER}+1}$, $\mathbf{b}^* = \mathbf{b}^{\text{ITER}+1}$;
8 **return** $(\mathbf{W}^*, \mathbf{a}^*, \mathbf{b}^*)$.

It is still unable to update the parameters based on these gradient formulas, since the complexity of computing $\Sigma_{\mathbf{v}}$ is very high, i.e., $O(2^{n_v+n_h})$. An efficient approximation method is the CD algorithm proposed by Hinton [10]. The gradient formulas included in CD algorithm are listed as follows:

$$\begin{cases} \dfrac{\partial \ln L_{\theta,\mathbb{S}}}{\partial w_{i,j}} \approx \displaystyle\sum_{m=1}^{N}\Big[p(h_i = 1|\mathbf{v}^{(m,0)})v_j^{(m,0)} \\ \qquad\qquad - p\big(h_i = 1|\mathbf{v}^{(m,k)}\big)v_j^{(m,k)}\Big] \\ \dfrac{\partial \ln L_{\theta,\mathbb{S}}}{\partial a_j} \approx \displaystyle\sum_{m=1}^{N}\Big[v_j^{(m,0)} - v_j^{(m,k)}\Big] \\ \dfrac{\partial \ln L_{\theta,\mathbb{S}}}{\partial b_i} \approx \displaystyle\sum_{m=1}^{N}\Big[p\big(h_i = 1|\mathbf{v}^{(m,0)}\big) - p\big(h_i = 1|\mathbf{v}^{(m,k)}\big)\Big] \end{cases} \tag{13}$$

where k is the number of sampling times in CD algorithm, which is usually equal to 1, and 0 represents the starting point of sampling. The CD algorithm with $k = 1$ is called CD-1 [12].

Finally, we can use formula (13) to update parameters $(\mathbf{W}, \mathbf{a}, \mathbf{b})$, and the detailed steps are given in Algorithm 1.

III. Random Weight Neural Networks

As aforementioned, RWNN is a feed-forward neural network trained by the RWA mechanism. It is noteworthy that the name RWNN refers to the network itself but the name RWA represents the training algorithm for RWNN. The essence of RWA is that the training process does not include iterative tuning of weight parameters. Essentially, RWA has two steps: 1) the weights of hidden layer nodes are randomly selected from a given interval and 2) the weights of output layer nodes are obtained analytically.

Given a training set $\mathbb{X} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N} \subset \mathcal{R}^L \times \{1, \ldots, C\}$, where $\mathbf{x}_i = [x_{i1}, x_{i2}, \ldots, x_{iL}]$ is the ith training sample, $\mathbf{y}_i = [y_{i1}, y_{i2}, \ldots, y_{iC}]$ is the label vector of $\mathbf{x}_i$, N is the number of samples, L is the number of features, and C is the

Algorithm 2: RWA: SLFNs Training Algorithm for Classification Problems (Fig. 2)

Input:
Training set $\mathbb{X} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N} \in \mathcal{R}^L \times \{1, \ldots, C\}$;
Tag matrix $\mathbf{T}_{N\times C} = [\mathbf{y}_1^{\mathrm{T}}, \mathbf{y}_2^{\mathrm{T}}, \ldots, \mathbf{y}_N^{\mathrm{T}}]^{\mathrm{T}}$;
Interval $[a, b]$, where a and b are real numbers ($a < b$).
Output:
Output function $f(\mathbf{x})$.

1 Generate the hidden layer weight matrix $\mathbf{A}_{L\times D}$ and bias matrix $\mathbf{B}_{N\times D}$ by randomly selecting real numbers from $[a, b]$, where D is the dimensionality of the hidden layer;

2 Compute the hidden layer output matrix through the following formula (15)

$$\mathbf{H}_{N\times D} = h(\mathbf{X}) = \text{sigmoid}(\mathbf{XA} + \mathbf{B}), \tag{15}$$

where

$$\mathbf{X}_{N\times L} = \begin{bmatrix} x_{11} & x_{12} & \ldots & x_{1L} \\ x_{21} & x_{22} & \ldots & x_{2L} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & \ldots & x_{NL} \end{bmatrix}, \tag{16}$$

$$\mathbf{A}_{L\times D} = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1D} \\ a_{21} & a_{22} & \ldots & a_{2D} \\ \vdots & \vdots & \ddots & \vdots \\ a_{L1} & a_{L2} & \ldots & a_{LD} \end{bmatrix}, \tag{17}$$

$$\mathbf{B}_{N\times D} = \begin{bmatrix} b_{11} & b_{12} & \ldots & b_{1D} \\ b_{21} & b_{22} & \ldots & b_{2D} \\ \vdots & \vdots & \ddots & \vdots \\ b_{N1} & b_{N2} & \ldots & b_{ND} \end{bmatrix}, \tag{18}$$

and $\mathbf{H}_{ij} = \text{sigmoid}(b_{ij} + \sum_{k=1}^{L} x_{ik}a_{kj})$, $i = 1, \ldots, N$, $j = 1, \ldots, D$;

3 Compute the weight matrix of the output layer nodes

$$\beta_{D\times C} = \left(\mathbf{H}^{\mathrm{T}}\mathbf{H}\right)^{-1}\mathbf{H}^{\mathrm{T}}\mathbf{T}; \tag{19}$$

4 The output function is determined as

$$f(\mathbf{x}) = h(\mathbf{x})\beta; \tag{20}$$

5 **return** $f(\mathbf{x})$.

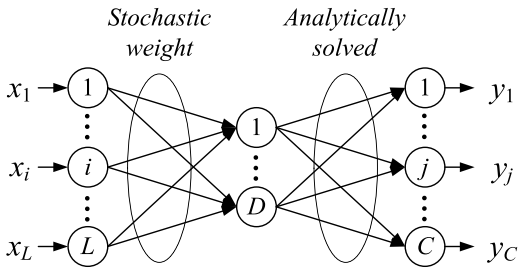


Fig. 2. Single hidden layer RWNN.

number of classes. The RWA training algorithm used in this paper obtains the weights of output layer nodes by solving the normal system of linear matrix equations [26], [27]. The RWA training method is given in Algorithm 2, and the overall structure is shown in Fig. 2.

We have several remarks on the RWA training algorithm.

1) RWA is not a new algorithm. It briefly summarizes the works of [2], [15]–[17], and [25]–[27] regarding RWA in a training process of feed-forward neural networks. The rename of previous works is nothing but to clarify some confusions existing in topic.
2) It has been experimentally noted that both parameters a and b in Algorithm 2 are sensitive to the training process. One can see that a significant difference exists with respect to the performance of RWA training for two sets of parameters $[a_1, b_1]$ and $[a_2, b_2]$. This indicates that the parametric interval $[a, b]$ has a significant impact on RWA training, and therefore, it leads to some difficulties for users to select this interval in real problems.
3) The training strategy of RWA was earliest proposed in [26] and [27], but in the subsequent decade it did not attract much interest of scholars investigating artificial neural networks. Until 2006 when the name ELM appears in [15] and [16], this training strategy receives extensive and intensive studies in terms of its universal approximation ability, the extremely fast training speed, the good generalization ability, and its applications in various domains. According to [29], the solution of an SLFN trained by the RWA mechanism is represented as

$$\beta = \left(\frac{I}{\lambda} + \mathbf{H}^{\mathrm{T}}\mathbf{H}\right)^{-1}\mathbf{H}^{\mathrm{T}}\mathbf{T} \tag{21}$$

in which a positive value (I/λ) is added to the diagonal of $\mathbf{H}^{\mathrm{T}}\mathbf{H}$ in the calculation of the output weights to adjust the singularity of matrix. We take the regularizing factor λ in (21) as a sufficiently large number, then, (21) is approximately identical to (19).
4) The regularizing factor λ plays a role of controlling both the amount of learned weights and the singularity of matrix $\mathbf{H}$. Since it is proven that the matrix $\mathbf{H}$ is of full rank with probability 1 in [8] and therefore singularity of matrix $\mathbf{H}$ is no longer a problem.
5) RWA can be easily extended to train an MLFN. We call this training algorithm RWA1, and describe it in Algorithm 3. The framework of RWA1 is shown in Fig. 3(a).

It is necessary to point out that the essence of RWA1 is the random assignment of weights for all hidden layer nodes. This scheme determines the weight parameters without iterative tuning by using certain strategies.

Regarding the approximation ability of RWNNs, one can find the detailed derivation, theoretical validation, and different explanations from [2] and [15]–[17]. Here, we will no longer review these approximation theorems.

IV. Proposed Training Scheme

This section will propose a new deep training scheme for MLFNs. The central idea is to improve the RWA1 training algorithm (i.e., Algorithm 3) by replacing the random assignment of weights with RBM-based weight initialization for all hidden layer nodes. The determination of weights for output layer in this scheme is identical to that in RWA1.

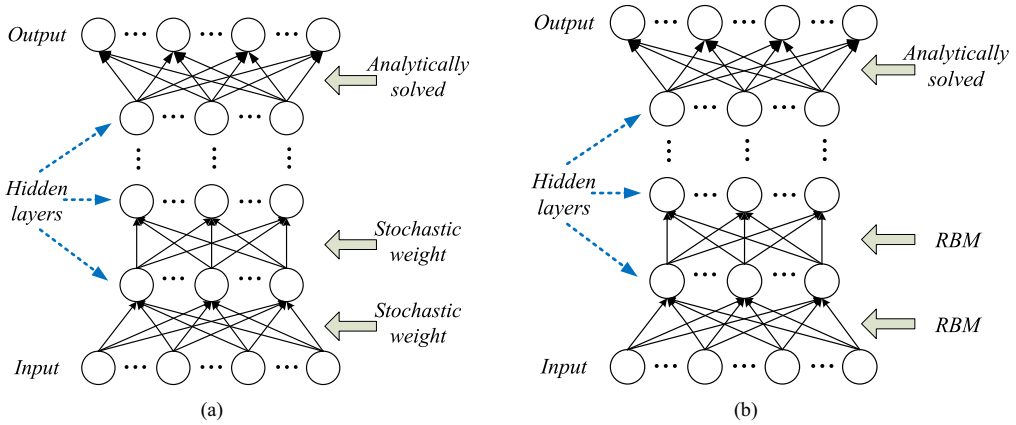


Fig. 3. Multiple hidden layer RWNNs with different training mechanisms. (a) RWA1. (b) RBM-GI.

Algorithm 3: RWA1: MLFNs Training Algorithm for Classification Problems [Fig. 3(a)]

Input:
Training set $\mathbb{X} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N} \in \mathcal{R}^L \times \{1, \ldots, C\}$;
Tag matrix $\mathbf{T}_{N\times C} = [\mathbf{y}_1^{\mathrm{T}}, \mathbf{y}_2^{\mathrm{T}}, \ldots, \mathbf{y}_N^{\mathrm{T}}]^{\mathrm{T}}$;
Interval $[a, b]$, where a and b are real numbers ($a < b$);
The number of hidden layers n.
Output:
Output function $f(\mathbf{x})$.

Get the input feature matrix $\mathbf{X}_1$ by (16);
for $i = 1$ *to* n **do**
  Randomly generate weight matrix $\mathbf{A}_i$ and bias matrix $\mathbf{B}_i$ for the i-th hidden layer;
  Compute random features for the i-th hidden layer using (15)

$$\mathbf{H}_i = h(\mathbf{X}_i) = \text{sigmoid}(\mathbf{X}_i\mathbf{A}_i + \mathbf{B}_i);$$

  Let $\mathbf{X}_{i+1} = \mathbf{H}_i$;
end
Calculate the weight matrix β of the output layer

$$\beta = \left(\mathbf{H}_n^{\mathrm{T}}\mathbf{H}_n\right)^{-1}\mathbf{H}_n^{\mathrm{T}}\mathbf{T};$$

The output function is determined as

$$f(\mathbf{x}) = h(\mathbf{x})\beta;$$

return $f(\mathbf{x})$.

A. Proposed Learning Framework

In the initial DBN learning framework introduced by Hinton *et al.* [13], RBM plays a role of weight parameter initialization. For a given feed-forward neural network with multiple hidden layers, RBM is used to pretrain the weight parameters for all hidden layer nodes and then all weight parameters for both hidden layers and output layer are iteratively tuned according to the BP algorithm. The iteration ends until the predefined stopping criterion is satisfied. This training framework has been acknowledged as a typical DL mechanism, which demonstrates advantages in many areas. The key advantages include the high generalization ability and semantic meaning of feature combination based on the learned weights. However, it also suffers from several disadvantages, i.e., the high computational complexity and the low convergence rate of the tuning process. In order to overcome these disadvantages, a new training scheme is proposed in this section. In the new scheme, RBM is still used to acquire the weight parameters for all hidden layer nodes. Afterward, the pretrained parameters result in a system of linear matrix equations, which is solved by evaluating the GI of the matrix. This training scheme is briefly denoted as RBM-GI. Basically, RBM-GI can be viewed as an improved version of RWA1, which is achieved through replacing the random assignment of weights with RBM-based weight initialization for all hidden layer nodes. The structure of RBM-GI is shown in Fig. 3(b), and the training process is shown in Fig. 4.

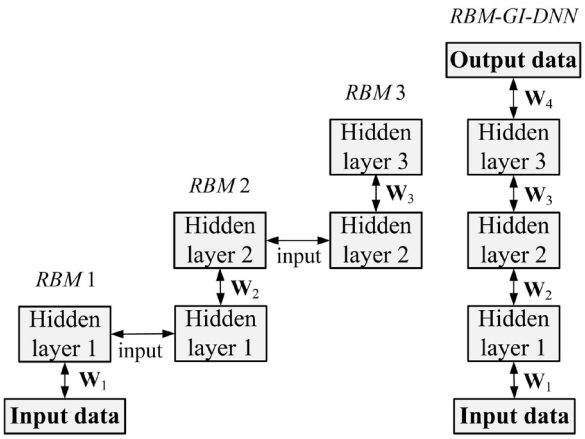


Fig. 4. Training process.

As shown in Fig. 4, we initially train the first RBM. Then, we use the output of the first RBM as the input of the second RBM to be trained. This process, i.e., the output of the ith RBM as the input of the $(i + 1)$th RBM, repeats until all weights of hidden layer nodes are determined. For a classification problem, the proposed RBM-GI is described as Algorithm 4.

Algorithm 4 can be viewed as two parts. The first part, i.e., steps 1–7, represents a process of unsupervised learning for training RBM parameters while the second part, i.e., steps 8–9, gives a process of supervised learning for estimating

Algorithm 4: RBM-GI [Fig. 3(b)]

Input:
Training set $\mathbb{X} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N} \in \mathcal{R}^L \times \{1, \ldots, C\}$;
Tag matrix $\mathbf{T}_{N\times C} = [\mathbf{y}_1^{\mathrm{T}}, \mathbf{y}_2^{\mathrm{T}}, \ldots, \mathbf{y}_N^{\mathrm{T}}]^{\mathrm{T}}$;
The number of hidden layers n;
The n-dimensional vector $[m_1, m_2, \ldots, m_n]$ where $m_j, j = 1, 2, \ldots, n$ is the number of nodes in the j-th hidden layer.

Output:
Output function.

1 Get the input feature matrix $\mathbf{X}$ by (16), let $\mathbf{H}_0 = \mathbf{X}$;
2 Take $\mathbf{H}_0$ as the input, call Algorithm 1 to learn the parameters between the input layer and the first hidden layer to get the weights and bias $(\mathbf{W}_1, \mathbf{b}_1)$;
3 **for** $i = 1$ *to* n **do**
4 Get the output matrix of the i-h hidden layer

$$\mathbf{H}_i = \text{sigmoid}\left(\mathbf{W}_i\mathbf{H}_{i-1} + \mathbf{b}_i\right);$$

5 Take $\mathbf{H}_i$ as the input, call Algorithm 1 to learn the parameters between the ith hidden layer and the $(i+1)$th hidden layer to get $(\mathbf{W}_{i+1}, \mathbf{b}_{i+1}, \mathbf{b}_i')$, where $\mathbf{b}_i'$ is called the reconstruction bias of $\mathbf{b}_i$;
6 **end**
7 Calculate the weight matrix β of the output layer

$$\beta = \left(\mathbf{H}_n^{\mathrm{T}}\mathbf{H}_n\right)^{-1}\mathbf{H}_n^{\mathrm{T}}\mathbf{T};$$

8 The output function is given by

$$\begin{aligned}\mathbf{H}_1 &= \text{sigmoid}(\mathbf{W}_1\mathbf{H}_0 + \mathbf{b}_1)\\ \mathbf{H}_2 &= \text{sigmoid}(\mathbf{W}_2\mathbf{H}_1 + \mathbf{b}_2)\\ &\vdots\\ \text{Output} &= \beta\mathbf{H}_n.\end{aligned}$$

parameters of the output layer by solving the GI of matrix. It is noteworthy that there is no iterative tuning of parameters.

B. Remarks on RBM-GI

Basically, the essences of RBM-GI are two-fold. The first is the RBM weight initialization, which was first introduced into DL by Hinton *et al.* [13]. The second is the RWA mechanism [26], [27], which proves that most connecting weights are not necessarily to be iteratively tuned for a fully connected feed-forward neural network. Viewing the essence of DL proposed by Hinton *et al.* [12], we can find that the key parts are also twofold, i.e., 1) the RBM-based weight initialization and 2) the BP-based weight tuning. One interesting question is: which part is more important? To answer this question, we can conduct an experimental comparison between random assignment weight initialization and RBM weight initialization on different datasets by using BP algorithm. Empirical studies show that, regarding the convergence rate of BP algorithm and the impossibility of dropping into the local minimum, RBM weight initialization is superior to random assignment weight initialization, which inspires us to introduce the RBM-based weight initialization into the RWA training. One explanation for RBM being superior to random assignment is that RBM pretraining can make the tuning start from a better point approaching to the global optimum. These analyses indicate that BP is the tuning algorithm body but RBM can help BP speed-up the convergence rate and help BP avoid dropping into local minimum. Even if RBM is not used, BP still can work. Thus, in the proposed RBM-GI, GI is the training algorithm body and RBM is used to improve the performance of GI.

A significant merit of DL [13] is that the acquired weights can lead to feature combination, which indicates explicitly semantic meanings in specific domains [19]. It is proposed in [9] that the combination of DBN and ELM can well handle the problem of speech emotion recognition. Another interesting question is whether or not the RBM has a key impact on the semantic feature extraction. Some scholars stated that the well-tuned-weight leads to the semantic feature combination when the network is convergent to the global minimum, and RBM parameter initialization is only to improve the algorithm convergence performance but has no impact on the semantic feature extraction. Other scholars stated that the global minimum is difficult to be achieved for a feed-forward neural network with many hidden layers, since it is a very complicated optimization problem with a considerable number of variables, and the corresponding training data is generally insufficient, thus RBM initialization possibly leads to a suboptimal minimum with semantic feature combination. The idea of RBM-GI is partially consistent to the latter standpoint.

There is a viewpoint that, with the increase of the amount of data, the role of pretraining appears to be unimportant. This makes the learning ability of RBM to the raw data not be fully reflected. It implicitly stated that the weight initialization will be unimportant for training an MLFN if the training data is vast. Our numerical experiments do not support this viewpoint, and the weight initialization based on RBM demonstrates its powerful impact on DNN training without iteration.

Basically, the proposed RBM-GI approach connects an unsupervised leaning mechanism (i.e., RBM) with a supervised learning methodology (i.e., solving a system of linear matrix equations through evaluating the GI of a matrix). It is different from conventional DL methods in two ways. The first difference is that RBM-GI uses RBM to initially train the weight parameters of the network and these parameters do not need to be readjusted. The second difference is that RBM-GI acquires the output weights by computing a GI rather than iteratively adjust, which has much lower computational complexity in comparison with BP-like algorithms.

V. Experimental Comparison

This section will conduct experimental comparisons to show the feasibility and effectiveness of the proposed RBM-GI algorithm.

A. Experiment on UCI Benchmark Datasts

In this section, we conduct experiments on various UCI benchmark datasets.

1) Data Preparation: We select 16 representative classification datasets from UCI Machine Learning Repository,

TABLE I
SELECTED UCI BENCHMARK DATASETS FOR PERFORMANCE COMPARISON

ID	Dataset	#Example	#Attribute	#Class
1	Optical Recognition of Handwritten Digits	5,620	64	10
2	Pen-Based Recognition of Handwritten Digits	10,992	16	10
3	Waveform Database Generator (1)	5,000	21	3
4	Waveform Database Generator (2)	5,000	40	3
5	USPS	9,298	256	10
6	Landsat Satellite	6,435	36	7
7	Letter Recognition	20,000	16	26
8	MAGIC Gamma Telescope	19,020	10	2
9	Wall-Following Robot Navigation	5,456	24	4
10	Satlog (Shuttle)	43,500	9	7
11	EEG Eye State	14,980	14	2
12	Firm-Teacher Clave-Direction	10,800	19	2
13	Skin Segmentation	245,057	3	2
14	Balance Scale	625	4(20)	3
15	Mushroom	8,124	22(125)	2
16	Connect-4	67,557	42(126)	3

Note: For datasets *Balance Scale*, *Mushroom*, and *Connect-4*, the number outside bracket is the number of original symbolic attributes and the number in bracket is the number of numerical attributes after transformation.

which focus on various learning fields. These datasets contain both symbolic attributes and numerical attributes, which will be used to verify the priority of RBM-GI. The detailed information of these datasets is shown in Table I.

Datasets 1–13 are of numerical attributes and datasets 14–16 are of symbolic attributes. Since neural networks cannot directly deal with the symbolic data, we need to transfer the symbolic data to numerical data. For example, suppose that the symbolic attribute F takes values a, b, and c. We transform each symbolic attribute value as an numerical attribute, finally, a, b, and c will become three numerical attributes taking value 0 or 1. Since one symbolic attribute will be replaced by multiple numerical attributes, the final numerical dataset may have an obvious increase of attribute number.

2) Experimental Design: Three algorithms are listed in the following for performance comparison.

1) *RWA1:* This algorithm randomly assigns the weights of all hidden layer nodes and analytically solve the inverse of a matrix as parameters of output layer (the details are in Section III).
2) *DBN:* This is the initial DL algorithm proposed by Hinton *et al.* [13]. It uses RBM to initialize the weights for all layers and then uses BP to iteratively tune these weight parameters until the algorithm converges or the error attains a predefined threshold.
3) *RBM-GI:* This algorithm uses RBM to determine the weights of all hidden layer nodes and uses the same methods as in RWA1 for computing the output layer parameters (the details are in Section IV-A).

The purpose of conducting this experiment is to compare the performance of RWA1, DBN, and RBM-GI for training an MLFN. We want to find their respective advantages and disadvantages, which may provide users some helpful guidelines to select an algorithm for training this type of neural networks.

It is noteworthy that datasets *Optical Recognition of Handwritten Digits*, *Pen-Based Recognition of Handwritten Digits*, *USPS*, and *Landsat Satellite* have their standard partitions of training set and testing set. As for the other datasets, our general scheme is to randomly select 90% of the samples as the training set and the remaining 10% as the testing set.

The number of hidden layers for the above three models is set to be 3. For each dataset, we tune the number of hidden nodes in the three layers for the proposed RBM-GI method, and select the network structure that can achieve the highest validation accuracy. Since we want to compare the performances of the methods under the same conditions, for fair comparison, the same network structure determined by RBM-GI is adopted for RWA1 and DBN. The detailed structural information have been listed in the last column of Table II. Moreover, the interval $[a, b]$ is set as $[0, 0.1]$ for RWA1, and the maximum iteration number in DBN is set as 2000. The three algorithms are implemented in MATLAB under the hardware environment with Core i7-3632QM CPU, 4GB RAM and 64-bit windows 7 operating system.

For each dataset, our experiment will write down three index values, i.e., training accuracy, training time, and testing accuracy. The experimental results are listed in Table II.

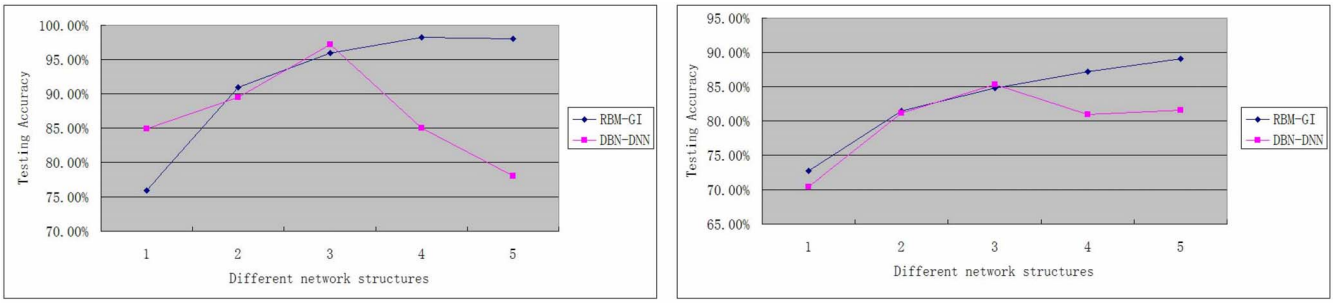
3) Result Analysis: For fixed number of hidden layers and number of each hidden layer nodes, from Table II we can view that DBN has the highest accuracy on a couple of datasets but has the longest training time on all datasets in comparison with the other two algorithms. RWA1 has the shortest training time but has the lowest prediction accuracy on all datasets. The training time of RBM-GI is slightly longer than that of RWA1, but is far less than DBN. We highlight that RBM-GI has the testing accuracy worse than DBN on some datasets but better than DBN on other datasets. It is hard to say which one is better for RBM-GI and DBN with respect to the testing accuracy. In this way, RBM-GI can replace DBN for reducing the computational complexity when tackling a classification problem for big data.

It is noteworthy that the above experiments are conducted with three fixed hidden layers. Each hidden layer has the same number of nodes for the three algorithms. One question is whether the number of nodes for each hidden layer has a critical impact on the testing accuracy. To answer this question we conduct an additional experiment. Let the number of hidden layer nodes vary, we observe the accuracy change. The hidden structure is set as 10-10-10, 50-50-50, 100-100-100, 500-500-500, 1000-1000-1000. The accuracy change with the network

TABLE II
COMPARATIVE RESULTS ON THE SELECTED UCI BENCHMARK DATASETS

ID	RWA1			DBN			RBM-GI			Hidden Structure
	Testing Accuracy (%)	Training Accuracy (%)	Training Time (s)	Testing Accuracy (%)	Training Accuracy (%)	Training Time (s)	Testing Accuracy (%)	Training Accuracy (%)	Training Time (s)	
1	95.99	98.67	0.6552	95.72	99.40	469.68	**96.83**	98.48	7.64	100-100-200
2	88.19	94.13	0.5304	**97.15**	98.20	5,484.90	95.94	98.19	30.87	100-100-100
3	83.40	88.80	1.1388	**86.40**	87.12	220.90	84.00	89.16	146.28	200-300-300
4	86.20	88.16	1.3572	**88.60**	87.47	270.69	88.20	88.78	146.72	200-300-300
5	91.98	96.86	2.1684	**94.67**	98.37	1,980.60	91.38	96.05	176.05	256-256-300
6	74.10	75.13	0.3900	**85.30**	84.60	11,055.00	84.75	86.79	4.57	100-100-100
7	73.30	74.80	1.2324	62.05	64.10	3,406.80	**74.45**	76.04	159.65	100-100-100
8	78.23	77.96	1.1388	81.29	85.90	751.40	**83.86**	83.32	153.49	100-100-100
9	83.14	86.47	0.8268	**87.10**	91.43	4,906.00	84.91	87.63	97.84	100-100-200
10	95.23	95.63	0.9984	**96.03**	96.53	15,481.00	91.55	92.67	125.69	50-50-50
11	57.20	54.89	0.6708	60.40	57.20	4,074.90	**64.33**	64.14	79.03	50-50-100
12	**99.91**	99.97	0.3120	99.10	100.00	170.78	99.81	99.61	47.08	50-50-50
13	96.64	96.57	3.6816	**98.01**	97.93	2,877.60	97.25	97.14	203.72	10-10-30
14	81.33	91.05	0.1248	**100.00**	100.00	29.29	85.82	92.17	1.20	100-100-100
15	**100.00**	99.89	0.8736	**100.00**	100.00	155.64	**100.00**	99.97	148.93	150-150-150
16	75.67	76.58	5.1168	**83.85**	84.27	14,462.00	76.06	76.44	17.94	150-150-150

Note: For each dataset, the highest testing accuracy is in bold face.



(a) (b)

Fig. 5. Testing accuracy on 5-hidden layer structures. (a) Pen-Based Recognition of Handwritten Digits. (b) Landsat Satellite.

structure on datasets *pen-based recognition of handwritten digits* and *landsat satellite* is shown in Fig. 5, where the horizontal coordinate represents the five structures and the vertical coordinate denotes the testing accuracy. Basically, the results show that the performance of RBM-GI is being improved with the increase of hidden layer node numbers.

From Fig. 5 we can see that, with the number of hidden nodes increasing, RBM-GI has an improved performance but DBN has not. One speculated reason is that DBN may not converge to its global optimum due to the insufficient number of samples. Besides, the number of hidden layer nodes has a critical impact on the testing accuracy of RBM-GI. This point has been confirmed by a comparison between DBN and ELM-autoencoder in [18] where ELM-autoencoder has a structure more complex than DBN. Since the training complexity of ELM-autoencoder is much lower than DBN, this type of comparison between two neural networks with different numbers of hidden layer nodes is still regarded as fair.

Furthermore, RBM-GI has a training time much less than DBN. For example, regrading the five structures in Fig. 5(a), the training time is 1.8096 s, 6.924 s, 30.8726 s, 74.3501 s, and 205.2973 s for RBM-GI, and is 200.71 s, 861.172 s, 5484.9 s, 27 267 s, and 154 820 s for DBN, respectively. It is observed that, with the increase of hidden layer number, the magnitude of time-increasing for DBN is much faster than that for RBM-GI. In this situation, the convergence rate of DBN is very slow, or the iteration does not converge. Comparatively, the training complexity of RBM-GI is insensitive to the increase of hidden layer nodes number, and therefore, RBM-GI implies a great potential in handling big data classification problems.

Finally, experimental results confirm the crucial impact of RMB weight initialization in the proposed method. The DBN model proposed by Hinton *et al.* [13], [14] is to produce a multilayer generative model by layer-wise pretraining. The training process uses greedy algorithm to stack a plurality of RBMs to produce many nonlinear feature extractors, which can be used to effectively learn complex statistical structure in the data. Using RBM to initialize parameters of each layer in the network can avoid shortcomings of gradient dispersion that often occurs in global tuning. The performance of RMB weight initialization plus BP fine tuning is much better than that of the traditional BP algorithm in the process of DNN training on normal sized datasets, but is not competitive on big data. RMB weight initialization in this model plays a different role in comparison with RBM-GI.

B. Experiment on MNIST Handwritten Dataset

In this section, we conduct experiment on the MNIST handwritten digits recognition dataset.[1] This dataset contains

[1] http://yann.lecun.com/exdb/mnist/

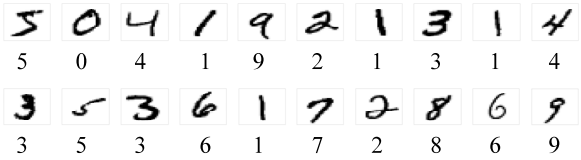


Fig. 6. Sample images in MNIST dataset.

TABLE III
PERFORMANCE COMPARISON ON MNIST DATASET

Method	Training Accuracy (%)	Testing Accuracy (%)	Training Time (s)
RWA1	99.28	97.76	1339.47
DBN	100.00	98.75	47230.00
RBM-GI	99.30	97.58	1373.99

60 000 training samples and 10 000 testing samples. Each sample is a 28×28-pixel gray-level image representing a handwritten digit from classes "0"–"9." Some sample images in this dataset are demonstrated in Fig. 6. We use the $28 \times 28 = 784$ raw pixels as the input features, and compare the performances of RWA1, DBN, and the proposed RBM-GI. The optimal network structure for DBN on this dataset is set as 500-500-2000 as reported in [13]. Besides, empirical studies show that a larger number of hidden layers can improve the performance of RWA1 and RBM-GI on this dataset. Thus, we apply four hidden layers and tune the network structure as 700-500-500-10000 for RWA1 and RBM-GI. The comparative results are demonstrated in Table III. It can be observed that the proposed RBM-GI has achieved similar results with RWA1. Compared with DBN, although its testing accuracy is slightly lower, the execution time is much faster. Besides, according to the results reported in existing works, the state-of-the-art methods for this dataset, i.e., MLP-BP [29] and deep forest [41], have achieved testing accuracy of 97.39% and 96.80%, respectively, which are very close to the proposed RBM-GI.

C. Experiment on ORL Face Recognition Dataset

In this section, we conduct experiment on the ORL face recognition dataset.[2] This dataset is composed of 400 face images from 40 persons. Each person has ten images with different face expressions from different angles. The sample images for two persons in this dataset are demonstrated in Fig. 7. We randomly select nine images as the training samples for each person, and use the rest image as the testing sample. We resize each face image into 32×32 pixels, and use the $32 \times 32 = 1024$ pixel values as the input features. Empirical studies show that on this sataset, a smaller number of hidden layers is suitable for DBN, and a larger number of hidden layers can improve the performance of RWA1 and RBM-GI. Finally, the network structure is tuned as 2500-2500-10000 for DBN, and 2500-2500-2500-10000 for RWA1 and RBM-GI. The comparative results are demonstrated in Table IV. On this dataset, RBM-GI and RWA1 have obvious advantage over DBN regarding both accuracy and execution time.

[2]http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html

Fig. 7. Sample images in ORL dataset.

TABLE IV
PERFORMANCE COMPARISON ON ORL DATASET

Method	Training Accuracy (%)	Testing Accuracy (%)	Training Time (s)
RWA1	100.00	97.50	12.00
DBN	94.50	85.00	7628.00
RBM-GI	100.00	97.50	80.49

VI. CONCLUSION

The general criterion for training an MLFN is the minimization of error between computed outputs and expected outputs. DL tries to achieve this minimization by iteratively tuning the weight parameters based on gradient descent technique. This paper proposes the RBM-GI approach, with the idea that the minimization is unnecessarily to be achieved by iterative technique but can be achieved by noniterative learning method. Due to the much lower training complexity and good generalization ability, the random assignment mechanism for training an MLFN can replace the iterative tuning mechanism for big data classification problems. It is highlighted that, as a noniterative training technique, the heuristic assignment of weight parameters (such as the RBM and the Auto-encoder) can improve the training efficiency and testing accuracy significantly.

REFERENCES

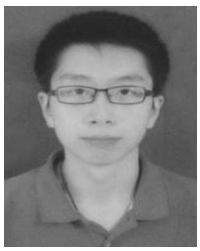
[1] S. Balasundaram and D. Gupta, "On optimization based extreme learning machine in primal for regression and classification by functional iterative method," *Int. J. Mach. Learn. Cybern.*, vol. 7, no. 5, pp. 707–728, 2016.
[2] D. S. Broomhead and D. Lowe, "Multivariable functional interpolation and adaptive networks," *Complex Syst.*, vol. 2, no. 2, pp. 321–355, 1998.
[3] K.-C. Chan, C.-K. Koh, and C. S. G. Lee, "An automatic design of factors in a human-pose estimation system using neural networks," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 46, no. 7, pp. 875–887, Jul. 2016.
[4] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition," *IEEE Audio, Speech, Language Process.*, vol. 20, no. 1, pp. 30–42, Jan. 2012.
[5] C. C. Dan, U. Meier, L. M. Gambardella, and J. Schmidhuber, "Convolutional neural network committees for handwritten character classification," in *Proc. Int. Conf. Document Anal. Recognit.*, 2011, pp. 1135–1139.
[6] S. Ding, N. Zhang, J. Zhang, X. Xu, and Z. Shi, "Unsupervised extreme learning machine with representational features," *Int. J. Mach. Learn. Cybern.*, vol. 8, no. 2, pp. 587–595, 2017.
[7] A. M. Fu, C. Dong, and L. S. Wang, "An experimental study on stability and generalization of extreme learning machines," *Int. J. Mach. Learn. Cybern.*, vol. 6, no. 1, pp. 129–135, 2015.
[8] A.-M. Fu, X.-Z. Wang, Y.-L. He, and L.-S. Wang, "A study on residence error of training an extreme learning machine and its application to evolutionary algorithms," *Neurocomputing*, vol. 146, no. 1, pp. 75–82, 2014.
[9] K. Han, D. Yu, and I. Tashev, "Speech emotion recognition using deep neural network and extreme learning machine," in *Proc. INTERSPEECH*, Singapore, 2014, pp. 223–227.
[10] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural Comput.*, vol. 14, no. 8, pp. 1771–1800, 2002.

[11] G. E. Hinton, *A Practical Guide to Training Restricted Boltzmann Machines* (LNCS 7700). Heidelberg, Germany: Springer, 2012, pp. 599–619.
[12] G. E. Hinton *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, Nov. 2012.
[13] G. E. Hinton, S. Osindero, and Y.-W. The, "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, no. 7, pp. 1527–1554, 2006.
[14] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.
[15] G.-B. Huang, L. Chen, and C. K. Siew, "Universal approximation using incremental constructive feedforward networks with random hidden nodes," *IEEE Trans. Neural Netw.*, vol. 17, no. 4, pp. 879–892, Jul. 2006.
[16] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: Theory and applications," *Neurocomputing*, vol. 70, nos. 1–3, pp. 489–501, 2006.
[17] B. Igelnik and Y.-H. Pao, "Stochastic choice of basis functions in adaptive function approximation and the functional-link net," *IEEE Trans. Neural Netw.*, vol. 6, no. 6, pp. 1320–1329, Nov. 1995.
[18] L. L. C. Kasun, H. Zhou, G.-B. Huang, and C. M. Vong, "Representational learning with extreme learning machines for big data," *IEEE Intell. Syst.*, vol. 28, no. 6, pp. 31–34, Nov./Dec. 2013.
[19] H. Lee, P. T. Pham, Y. Largman, and A. Y. Ng, "Unsupervised feature learning for audio classification using convolutional deep belief networks," in *Proc. NIPS*, Vancouver, BC, Canada, 2009, pp. 1096–1104.
[20] D. Li, *Deep Learning for Signal and Information Processing*. Boston, MA, USA: Now, 2013.
[21] S. Li, M.-C. Lee, and C.-M. Pun, "Complex Zernike moments features for shape-based image retrieval," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 39, no. 1, pp. 227–237, Jan. 2009.
[22] T. Li, S. Duan, J. Liu, L. Wang, and T. Huang, "A spintronic memristor-based neural network with radial basis function for robotic manipulator control implementation," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 46, no. 4, pp. 582–588, Apr. 2016.
[23] C.-L. Liu, W.-H. Hsaio, C.-H. Lee, and H.-C. Chi, "An HMM-based algorithm for content ranking and coherence-feature extraction," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 43, no. 2, pp. 440–450, Mar. 2013.
[24] P. Liu, Y. Huang, L. Meng, S. Gong, and G. Zhang, "Two-stage extreme learning machine for high-dimensional data," *Int. J. Mach. Learn. Cybern.*, vol. 7, no. 5, pp. 765–772, 2016.
[25] Y.-H. Pao, G.-H. Park, and D. J. Sobajic, "Learning and generalization characteristics of the random vector functional-link net," *Neurocomputing*, vol. 6, no. 2, pp. 163–180, 1994.
[26] Y.-H. Pao and Y. Takefji, "Functional-link net computing: Theory, system architecture, and functionalities," *IEEE Comput. J.*, vol. 25, no. 5, pp. 76–79, May 1992.
[27] W. F. Schmidt, M. A. Kraaijveld, and R. P. W. Duin, "Feedforward neural networks with random weights," in *Proc. 11th IAPR Int. Conf. Pattern Recognit. Conf. B Pattern Recognit. Methodol. Syst.*, vol. 2. The Hague, The Netherlands, 1992, pp. 1–4.
[28] P. Smolensky, *Information Processing in Dynamical System: Foundations of Harmony Theory*. Cambridge, MA, USA: MIT Press, 1986.
[29] J. Tang, C. Deng, and G.-B. Huang, "Extreme learning machine for multilayer perceptron," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 4, pp. 809–821, Apr. 2015.
[30] R. Wang, C.-Y. Chow, and S. Kwong, "Ambiguity-based multiclass active learning," *IEEE Trans. Fuzzy Syst.*, vol. 24, no. 1, pp. 242–248, Feb. 2016.
[31] R. Wang, Y.-L. He, C.-Y. Chow, F.-F. Ou, and J. Zhang, "Learning ELM-tree from big data based on uncertainty reduction," *Fuzzy Sets Syst.*, vol. 258, pp. 79–100, Jan. 2015.
[32] R. Wang, S. Kwong, D. Chen, and J. Cao, "A vector-valued support vector machine model for multiclass problem," *Inf. Sci.*, vol. 235, pp. 174–194, Jun. 2013.
[33] R. Wang, S. Kwong, and X. Wang, "A study on random weights between input and hidden layers in extreme learning machine," *Soft Comput.*, vol. 16, no. 9, pp. 1465–1475, 2012.
[34] R. Wang, S. Kwong, X.-Z. Wang, and Q.-S. Jiang, "Segment based decision tree induction with continuous valued attributes," *IEEE Trans. Cybern.*, vol. 45, no. 7, pp. 1262–1275, Jul. 2015.
[35] X.-Z. Wang, "Learning from big data with uncertainty—Editorial," *J. Intell. Fuzzy Syst.*, vol. 28, no. 5, pp. 2329–2330, 2015.
[36] X.-Z. Wang, R. A. R. Aamir, and A.-M. Fu, "Fuzziness based sample categorization for classifier performance improvement," *J. Intell. Fuzzy Syst.*, vol. 29, no. 3, pp. 1185–1196, 2015.
[37] X.-Z. Wang, R. Wang, H.-M. Feng, and H.-C. Wang, "A new approach to classifier fusion based on upper integral," *IEEE Trans. Cybern.*, vol. 44, no. 5, pp. 620–635, May 2014.
[38] Q. Wu, Z. Wang, F. Deng, Z. Chi, and D. D. Feng, "Realistic human action recognition with multimodal feature selection and fusion," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 43, no. 4, pp. 875–885, Jul. 2013.
[39] C. Xiao and W. A. Chaovalitwongse, "Optimization models for feature selection of decomposed nearest neighbor," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 46, no. 2, pp. 177–184, Feb. 2016.
[40] J. Zhang, S. Ding, N. Zhang, and Z. Shi, "Incremental extreme learning machine based on deep feature embedded," *Int. J. Mach. Learn. Cybern.*, vol. 7, no. 1, pp. 111–120, 2016.
[41] Z.-H. Zhou and J. Feng, "Deep forest: Towards an alternative to deep neural networks," unpublished paper, 2017. [Online]. Available: https://arxiv.org/abs/1702.08835

Xi-Zhao Wang (M'03–SM'04–F'12) received the Doctoral degree in computer science from the Harbin Institute of Technology, Harbin, China, in 1998.

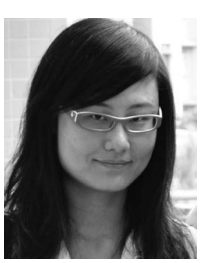
From 2001 to 2014, he has been a Full Professor and the Dean of the College of Mathematics and Computer Science, Hebei University, Hebei, China. From 1998 to 2001, he was a Research Fellow with the Department of Computing, Hong Kong Polytechnic University, Hong Kong. Since 2014, he has been a Full Professor with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China. His current research interests include supervised and unsupervised learning, active learning, reinforcement learning, manifold learning, transfer learning, unstructured learning, uncertainty, fuzzy sets and systems, fuzzy measures and integrals, rough set, and learning from big data.

Dr. Wang was a recipient of many awards from the IEEE International Conference on Systems, Man, and Cybernetics (SMC) Society. He is a member of the Board of Governors of the IEEE SMC in 2005, from 2007 to 2009, and from 2012 to 2014, the Chair of the Technical Committee on Computational Intelligence of the IEEE SMC, and a Distinguished Lecturer of the IEEE SMC. He was the Program Co-Chair of the IEEE SMC 2009 and 2010. He is the Editor-in-Chief of the *International Journal of Machine Learning and Cybernetics*. He is also an Associate Editor of the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS-PART B: CYBERNETICS, *Information Sciences Journal*, and the *International Journal of Pattern Recognition and Artificial Intelligence*.



Tianlun Zhang received the bachelor's degree in information management and information system from the Industrial and Commercial College, Hebei University, Hebei, China, in 2014, and the M.Sc. degree in software engineering from the College of Mathematics and Information Science, Hebei University, Hebei, in 2016. He is currently pursuing the Ph.D. degree with the Information Science and Technology College, Dalian Maritime University, Dalian, China.

His current research interests include machine learning and pattern recognition.



Ran Wang (S'09–M'14) received the B.Eng. degree in computer science from the College of Information Science and Technology, Beijing Forestry University, Beijing, China, in 2009, and the Ph.D. degree from the Department of Computer Science, City University of Hong Kong, Hong Kong, in 2014.

From 2014 to 2016, she was a Post-Doctoral Researcher with the Department of Computer Science, City University of Hong Kong. She is currently an Assistant Professor with the College of Mathematics and Statistics, Shenzhen University, Shenzhen, China. Her current research interests include pattern recognition, machine learning, and fuzzy sets and fuzzy logic and their related applications.