Research on Batch Normalization DBN Classification Method Postprint

Authors: Li Beibei, Song Wei, Dai Xin

Date: 2018-05-02T00:00:00+00:00

Abstract

To address the issue of Deep Belief Networks (DBN) being susceptible to training parameters during the fine-tuning process, a batch-normalized DBN classification method (BNDBN) is proposed. This method first employs DBN for unsupervised learning to obtain high-level representations of the raw data; subsequently, it performs batch normalization on each dimension of the output features from intermediate network layers by introducing scale and shift transformation parameters; the processed features are then fed into a nonlinear transformation activation layer; finally, stochastic gradient descent is utilized to train both the affine transformation parameters and the original network parameters. The BNDBN method reduces the dependence of gradients on parameter scale, effectively mitigates the problem of activation distribution shifts caused by network parameter variations, and enhances training efficiency. To evaluate the effectiveness of the proposed method, experiments were conducted on the MNIST handwritten digit database and the USPS handwritten digit recognition database. Comparative results with Dropout-DBN, DBN, ANN, SVM, and KNN demonstrate that the proposed method achieves significantly improved classification accuracy and exhibits stronger feature extraction capability.

Full Text

Research on Batch Normalization DBN Classification Method

Li Beibei, Song Wei, Dai Xin
(School of Internet of Things Engineering, Jiangnan University, Wuxi, Jiangsu 214122, China)

Abstract: To address the problem that Deep Belief Networks (DBN) are susceptible to training parameters during the fine-tuning process, this paper proposes a Batch Normalization DBN classification method (BNDBN). This

method first utilizes DBN for unsupervised learning to obtain high-level representations of raw data. It then introduces scale and translation transformation parameters to perform batch normalization on each dimension of the output features from intermediate network layers. The processed features are fed into a nonlinear transformation activation layer. Finally, stochastic gradient descent is used to train the affine transformation parameters along with the original network parameters. The BNDBN method reduces the dependence of gradients on parameter scale, effectively solves the problem of activation value distribution changes caused by network parameter variations, and improves training efficiency. To verify the effectiveness of the proposed method, experiments were conducted on the MNIST handwritten digit database and the USPS handwritten digital recognition library. Compared with Dropout-DBN, DBN, ANN, SVM, and KNN, the results demonstrate that the proposed method significantly improves classification accuracy and possesses stronger feature extraction capability.

Keywords: deep belief network; classification; unsupervised learning; scale transformation; translation transformation; batch normalization

0 Introduction

Hinton et al. proposed Deep Belief Networks (DBN) and an unsupervised greedy layer-wise training algorithm in 2006, bringing hope for solving optimization challenges in deep neural networks. DBN is a probabilistic generative model that overcomes training difficulties through "layer-wise initialization." Its ability to process high-dimensional inputs makes it ideal for tasks with inherent high dimensionality. DBN has become one of the most widely used deep learning architectures due to its advantages in automatic feature learning and data dimensionality reduction. DBN has achieved breakthrough progress in speech recognition, image classification, face recognition, and related fields.

Currently, as dataset scales expand and more complex, deeper architectures are proposed, network training has become more challenging, requiring more effective training methods. However, adapting to large datasets alone is insufficient, particularly in deep belief networks where supervised fine-tuning is also a critical stage. Stochastic Gradient Descent (SGD) is one of the most effective methods for fine-tuning deep belief networks, with recent improvements in Adagrad and momentum. Literature [10] proposed an efficient natural gradient parallel training algorithm for deep neural networks applied to image classification, which effectively improved algorithm convergence. Literature [11,12] proposed Dropout and DropConnect algorithms, primarily aiming to introduce sparsity into network models, reduce joint adaptability between neuron nodes, and prevent overfitting. Literature [13] introduced the Dropout algorithm into the DBN fine-tuning process to improve network generalization and discriminative capability. Literature [14-16] used metaheuristic search algorithms and

their variants to fine-tune DBN parameters to obtain near-optimal solutions and avoid local optima.

In the DBN fine-tuning stage, the optimization objective is to minimize the loss function measuring the distance between given labels and network outputs. Although these algorithmic improvements enhance network training effects to varying degrees, they remain sensitive to model hyperparameters and require more careful initialization. Since each network layer' s input is calculated using all lower-layer parameters, small changes in lower-layer parameters may be amplified layer-by-layer as network depth increases. Consequently, the network layer must fit a new distribution to achieve stability, increasing training complexity, reducing training speed, and even causing deeper networks to perform worse than shallow ones.

To address the shortcomings of traditional DBN training methods and leverage the advantages of Batch Normalization [17], this paper proposes a Batch Normalization DBN (BNDBN) classification method. This method introduces scale and translation transformation parameters, enabling transformation and reconstruction to restore the feature distribution learned by the original network, thereby providing stable network performance compared to DBN. Experimental analysis on MNIST and USPS handwritten digit recognition databases, comparing with Dropout-DBN, DBN, Artificial Neural Networks (ANN), Support Vector Machines (SVM), and K-Nearest Neighbors (KNN), demonstrates the superior classification accuracy of the proposed method.

1.1 Deep Belief Network

DBN is a probabilistic generative model that can be viewed as a deep network formed by stacking multiple Restricted Boltzmann Machines (RBM) [18]. RBM is a symmetrically connected stochastic neural network consisting of a visible layer v and a hidden layer h, as shown in Figure 1 [Figure 1: see original paper].

RBM is an energy-based model. The energy function for visible and hidden layers is given by:

$$E(v,h|\theta) = -\sum_{i=1}^{n} a_i v_i - \sum_{j=1}^{m} b_j h_j - \sum_{i=1}^{n}\sum_{j=1}^{m} v_i W_{ij} h_j$$

where $\theta = \{W_{ij}, a_i, b_j\}$, W_{ij} is the weight matrix between visible and hidden layers, a_i represents the bias of visible layer nodes, and b_j represents the bias of hidden layer nodes.

The conditional probabilities for visible and hidden layer nodes generated by this model are:

$$P(h|v,\theta) = \frac{1}{Z(\theta)} \exp(-E(v,h|\theta))$$

$$P(v|h,\theta) = \frac{1}{Z(\theta)} \exp(-E(v,h|\theta))$$

When the states of visible layer nodes are given, the activation states of hidden layer nodes are conditionally independent. Therefore, the activation state of the j-th hidden layer node is:

$$P(h_j = 1|v) = \sigma\left(\sum_{i=1}^{n} W_{ij} v_i + b_j\right)$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the nonlinear sigmoid activation function. When the states of hidden layer nodes are given, the activation state of the i-th visible layer node is:

$$P(v_i = 1|h) = \sigma\left(\sum_{j=1}^{m} W_{ij} h_j + a_i\right)$$

The DBN training process includes pre-training and fine-tuning stages. The network is pre-trained based on input data using the Contrastive Divergence (CD-k) algorithm [19] to unsupervisedly train each RBM layer from bottom to top, obtaining corresponding weights and biases, and ultimately acquiring high-level features of the data. Then, supervised learning via Back Propagation (BP) algorithm [20] is used to fine-tune the entire network top-down, enabling the DBN model to fit the input data well. The structure is shown in Figure 2 [Figure 2: see original paper].

1.2 Softmax Classifier

To apply deep belief networks to classification tasks, a classifier must be added to the final layer of the network. To ensure broader applicability, this paper uses the Softmax classifier for classification.

For a training set $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$, where y can take k different values representing k classes. Let $p(y = j|x)$ denote the probability that sample x is classified as class j. For a k-class classifier, the output is a k-dimensional vector (with elements summing to 1):

$$p(y = j|x) = \frac{\exp(\theta_j^T x)}{\sum_{l=1}^{k} \exp(\theta_l^T x)}$$

where θ is a matrix with each row representing the classifier parameters corresponding to a class, totaling k rows. The denominator normalizes the probability distribution so that all probabilities sum to 1. Therefore, the cost function of the Softmax classifier is:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{k} 1\{y^{(i)} = j\} \log \frac{\exp(\theta_j^T x^{(i)})}{\sum_{l=1}^{k} \exp(\theta_l^T x^{(i)})}$$

where $1\{\cdot\}$ is an indicator function, i.e., $1\{\text{true expression}\} = 1$ and $1\{\text{false expression}\} = 0$. This paper uses gradient descent algorithm to minimize the cost function.

2.1 Batch Normalization Algorithm

Traditional regularization algorithms are not continuously differentiable everywhere, and the Batch Normalization (BN) algorithm [17,21] improves upon this. First, traditional regularization algorithms perform joint regularization on input features of each network layer, whereas BN performs independent regularization on each scalar feature of each layer and processes input samples in batches.

Second, because each network layer' s input changes with training parameters, the changed input cannot fully express the original input features. For a d-dimensional input data x, the BN algorithm introduces scale transformation parameter γ and translation transformation parameter β. Through transformation and reconstruction, data can also fall into nonlinear distributions, thereby maintaining model expressiveness. After parameter training and learning, when $\gamma = \sqrt{\text{Var}[x]}$ (i.e., the standard deviation of input) and $\beta = E[x]$ (i.e., the expectation of input), the original feature distribution that the network should learn can be completely restored.

The BN algorithm performs batch normalization on each dimension of the input. The formula is as follows:

For a certain layer with d-dimensional input x, each batch sample set is $B = \{x_1, \ldots, x_m\}$, and the network has l hidden layers. Each layer normalizes each dimension of input $x^{(k)}$:

$$\mu_B^{(k)} = \frac{1}{m}\sum_{i=1}^{m} x_i^{(k)}$$

$$\sigma_B^{2(k)} = \frac{1}{m}\sum_{i=1}^{m} (x_i^{(k)} - \mu_B^{(k)})^2$$

$$\hat{x}_i^{(k)} = \frac{x_i^{(k)} - \mu_B^{(k)}}{\sqrt{\sigma_B^{2(k)} + \epsilon}}$$

where $x_i^{(k)}$ represents the k-th dimension of input x_i, $\mu_B^{(k)}$ denotes the mean of sample set B, and $\sigma_B^{2(k)}$ denotes the variance of sample set B.

Scale and translation transformation parameters are used to maintain model expressiveness. The transformed formula is:

$$y_i^{(k)} = \gamma^{(k)} \hat{x}_i^{(k)} + \beta^{(k)} \equiv \mathrm{BN}_{\gamma^{(k)}, \beta^{(k)}}(x_i^{(k)})$$

where $y_i^{(k)}$ is the output after batch normalization processing, which is then fed into the next nonlinear transformation activation layer. Parameters $\gamma^{(k)}$ and $\beta^{(k)}$ are trained together with the original network parameters using stochastic gradient descent to compute gradients and iteratively update them.

2.2 Batch Normalization DBN Structure

During traditional DBN fine-tuning, the input distribution of each layer changes with parameters, increasing network training complexity. If the distribution of nonlinear inputs can be maintained in a more stable state during training, the DBN optimization process will be less prone to problems. Therefore, this paper introduces the batch normalization algorithm during the fine-tuning stage. The BNDBN structure is shown in Figure 3 [Figure 3: see original paper].

In Figure 3(b), a batch normalization layer (BN layer) is introduced to process input features before feeding them into the activation function layer. The BNDBN method can reduce the dependence of gradients on parameter magnitude or initial values. When traditional methods iteratively update network parameters, excessively high learning rates can cause problems such as gradient vanishing and falling into local minima. The BNDBN method introduces scale transformation parameter γ and translation transformation parameter β to perform batch normalization on each dimension of output features from each hidden layer.

The final output calculation for each hidden layer of the DBN network is as follows:

$$z^{(l+1)} = f(\mathrm{BN}(W^{(l+1)} y^{(l)} + b^{(l+1)})) = f(\gamma^{(l+1)} \hat{x}^{(l+1)} + \beta^{(l+1)})$$

where f is the sigmoid function, and weights W and biases b are the layer parameters to be learned. The DBN processed by batch normalization requires

backpropagation to compute the cost function gradient while also computing the gradients for the affine transformation parameters introduced in the batch normalization algorithm.

The batch-normalized formula (12) can be replaced with:

$$\mathrm{BN}(Wu+b) = \gamma\hat{x} + \beta$$

where the transformation acts separately on each dimension of input u. After adding scale transformation with parameter γ to the transformed input, the partial derivatives of batch normalization are:

$$\frac{\partial \mathrm{BN}(Wu+b)}{\partial W} = \frac{\partial \mathrm{BN}(Wu+b)}{\partial (Wu)} \cdot \frac{\partial (Wu)}{\partial W}$$

$$\frac{\partial \mathrm{BN}(Wu+b)}{\partial W} = \gamma \cdot \frac{\partial \hat{x}}{\partial W}$$

From equations (14) and (15), we can see that adding scale transformation with parameter γ to a network layer does not affect gradient propagation. Additionally, larger weights result in smaller gradients after scale transformation, enabling the introduced batch normalization algorithm to maintain stability during parameter training.

2.3 Batch Normalization DBN Training Process

The specific training process for batch normalization DBN is as follows:

a) Data preprocessing: Convert images to grayscale and normalize grayscale values to [0,1].

b) Construct the DBN network using RBMs, initialize weight matrices and biases for visible and hidden layers, and set learning rate, iteration count, and Mini-Batch size.

c) Feed preprocessed data into the network input layer. Use bottom-up unsupervised learning for pre-training, employing equation (4) to compute hidden layer node activation states and equation (5) to compute visible layer node activation states for each RBM. Repeat this process, compute partial derivatives of the log probability, and finally update the weight matrix and biases in parameter space using:

$$\Delta W_{ij} = \epsilon(\langle v_i h_j \rangle_{\mathrm{data}} - \langle v_i h_j \rangle_{\mathrm{recon}})$$

$$\Delta a_i = \epsilon(\langle v_i \rangle_{\text{data}} - \langle v_i \rangle_{\text{recon}})$$

$$\Delta b_j = \epsilon(\langle h_j \rangle_{\text{data}} - \langle h_j \rangle_{\text{recon}})$$

where ϵ is the learning rate; ΔW_{ij} is the weight update value; Δa_i and Δb_j are bias update values; $\langle \cdot \rangle_{\text{data}}$ denotes expectation over data; $\langle v_i h_j \rangle_{\text{data}}$ represents the product of visible layer node v_i and hidden layer node h_j before reconstruction; and $\langle v_i h_j \rangle_{\text{recon}}$ represents the value after reconstruction, reflecting the distribution of the reconstruction model.

d) Further optimize the BNDBN network. Use parameter values obtained from pre-training (weights and biases) as initial values for this stage. Perform batch normalization on network inputs using equations (6)-(9) and feed them into activation layers. Then add a Softmax classifier at the top of the network and use Mini-Batch SGD to minimize $J(\theta)$. Compute the cost function gradient and the affine transformation parameters γ and β from batch normalization. The derivative formulas for γ and β use the chain rule:

$$\frac{\partial J}{\partial \gamma^{(k)}} = \sum_{i=1}^{m} \frac{\partial J}{\partial y_i^{(k)}} \cdot \frac{\partial y_i^{(k)}}{\partial \gamma^{(k)}} = \sum_{i=1}^{m} \frac{\partial J}{\partial y_i^{(k)}} \cdot \hat{x}_i^{(k)}$$

$$\frac{\partial J}{\partial \beta^{(k)}} = \sum_{i=1}^{m} \frac{\partial J}{\partial y_i^{(k)}} \cdot \frac{\partial y_i^{(k)}}{\partial \beta^{(k)}} = \sum_{i=1}^{m} \frac{\partial J}{\partial y_i^{(k)}}$$

Then update γ and β according to gradient descent formulas.

e) Test the trained network with test data. In the batch normalization layer, use the expected value of standard deviation and expected value of mean from the training stage to process test data:

$$E[x] \leftarrow E_B[\mu_B]$$

$$\text{Var}[x] \leftarrow \frac{m}{m-1} E_B[\sigma_B^2]$$

where $E_B[\mu_B]$ represents the expected value of means across all batches, and $E_B[\sigma_B^2]$ represents the unbiased estimate of standard deviation for each batch.

3 Experimental Results and Analysis

3.1 Experimental Datasets

To verify the effectiveness of the proposed algorithm, experiments were conducted on the MNIST and USPS handwritten digit datasets. The MNIST handwritten digit dataset includes 60,000 training samples and 10,000 test samples composed of Arabic digits 0-9, with each image being 28×28 pixels. From this, 6,000 samples were randomly selected as the training set and 2,000 as the test set. The USPS dataset is a U.S. Postal Service handwritten digit recognition library containing 9,298 handwritten digit images, all 16×16 pixel grayscale values. The grayscale values were normalized, with 7,000 selected as training data and 4,000 as test data.

3.2.1 Parameter Analysis

Experiments were run on a Windows 7 operating system using MATLAB R2008b development environment. Experimental parameters were selected as ideal values after extensive testing. Excluding the BN layer, the network node configuration was 784-100-100-10 (four layers total), mini-batch size was set to 100, momentum parameter to 0.9, and initial iteration count to 100.

To test the impact of learning rate on BNDBN, the learning rate was varied within the interval [0.0005, 0.05] while other parameters remained fixed. Comparative experiments were conducted on MNIST and USPS datasets, as shown in Figure 4 [Figure 4: see original paper] and Figure 5 [Figure 5: see original paper]. Observing Figure 4 reveals that the BNDBN algorithm demonstrates excellent classification performance. The classification error rate curves of BNDBN are consistently below those of Dropout-DBN and DBN, with classification error rates under 6%. Moreover, the optimal classification error rate of the proposed algorithm is 0.6% lower than that of Dropout-DBN, indicating better stability and enabling the BNDBN model to effectively extract primary data features.

Figure 5 shows the classification error rate comparison of the three algorithms with varying learning rates on the USPS dataset. Notably, BNDBN' s classification error rate is lower than other algorithms. Further increasing the learning rate, while initially slowing training, yields better classification results, proving that the proposed BNDBN enhances network classification performance while preventing algorithm divergence.

Traditional DBN' s greedy unsupervised training method can effectively extract data features, with weights and biases of each layer positioned optimally, making further training with SGD more likely to converge to the optimum. Therefore, the training effectiveness of the previous stage affects the fine-tuning stage and ultimately impacts classification performance. To test the optimal iteration count required for BNDBN in the pre-training stage, comparative experiments were conducted, as shown in Figure 6 [Figure 6: see original paper] and Figure 7 [Figure 7: see original paper].

As shown in Figures 6 and 7, the iteration count for training each RBM has a certain impact on overall network performance in both MNIST and USPS, with BNDBN classification results consistently higher than DBN. In MNIST, BNDBN achieves optimal performance at 600 iterations with 95.20% accuracy, while Dropout-DBN and DBN achieve their best results at 93.7% and 93.15%, respectively. For the USPS dataset, BNDBN reaches 97.70% classification accuracy at 300 iterations, representing improvements of 0.87% and 1.67% over the best accuracies of Dropout-DBN and DBN, respectively. Overall, comparisons with other algorithms further demonstrate the superior classification performance of the proposed algorithm.

To verify the performance of BNDBN with deeper networks, the number of hidden layers was gradually increased while other parameters remained fixed, with each hidden layer containing 100 nodes, up to a maximum of 10 layers. Classification accuracy comparisons on MNIST and USPS are shown in Figure 8 [Figure 8: see original paper] and Figure 9 [Figure 9: see original paper].

Observations from Figures 8 and 9 show that classification accuracy gradually decreases as the number of hidden layers increases initially. On the MNIST dataset, BNDBN achieves optimal classification accuracy with 2 hidden layers. On the USPS dataset, the best classification accuracy occurs with 5 hidden layers. For the Dropout-DBN algorithm, results become relatively poor when the number of hidden layers exceeds 7. Overall, the proposed algorithm consistently outperforms both DBN and Dropout-DBN.

To further validate the effectiveness of the proposed algorithm, BNDBN was compared with DBN, Dropout-DBN, and other widely-used classification algorithms including BP, SVM, KNN, and DBN on MNIST and USPS datasets. The comparison of classification accuracies is presented in Table 1 , which lists the optimal classification accuracy for each algorithm. In the ANN algorithm, hidden layer nodes were set to 100; SVM used a polynomial kernel function; KNN used Euclidean distance with $K = 10$.

Table 1: Classification Accuracy Comparison of Six Algorithms (%)

Dataset	BNDBN	Dropout-DBN	DBN	ANN	SVM	KNN
MNIST	95.20	93.70	93.15	91.80	92.45	90.65
USPS	97.70	96.83	96.03	94.57	95.20	93.85

Table 1 shows that DBN classification results are significantly higher than ANN on both MNIST and USPS, demonstrating DBN' s effectiveness in image classification tasks. While Dropout-DBN improves DBN' s overfitting problem to some extent, it remains affected by parameters, resulting in lower classification accuracy. The proposed BNDBN algorithm achieves optimal classification accuracy among all five comparison algorithms, indicating that it addresses DBN' s shortcomings and improves classification performance.

3.2.3 Training Time Comparison Analysis

Table 2 presents the training time comparison of BNDBN, Dropout-DBN, DBN, BP, SVM, and KNN algorithms on MNIST and USPS datasets. Due to the deep architecture of deep neural networks and high computational complexity, training time is longer compared to traditional classification algorithms like SVM and KNN, though classification accuracy is higher. BNDBN requires the least training time among DBN and Dropout-DBN, achieving approximately 1.5% and 2% higher best classification accuracy than Dropout-DBN and DBN, respectively, in less time. This demonstrates that the proposed BNDBN algorithm offers further improvements in training time, accelerates convergence speed, and provides superior classification performance.

Table 2: Training Time Comparison of Different Algorithms (minutes)

Dataset	BNDBN	Dropout-DBN	DBN	ANN	SVM	KNN
MNIST	45.2	52.3	48.7	38.5	12.3	8.7
USPS	28.6	34.2	31.5	24.3	9.8	6.2

4 Conclusion

This paper proposes a Batch Normalization DBN classification method and applies it to classification and recognition. The paper first introduces the structure of Deep Belief Networks (DBN), then details the batch normalization DBN classification method and its training process. This method not only leverages DBN' s excellent feature extraction capability but also incorporates the advantages of batch normalization to solve problems existing in traditional DBN training methods. Finally, comparative experiments on MNIST and USPS datasets demonstrate the proposed algorithm' s superior classification performance. Future work will continue to focus on network parameter optimization to further enhance the algorithm' s feature extraction capability.

References

[1] Hinton G E, Osindero S, Teh Y W. A fast learning algorithm for deep belief nets [J]. Neural Computation, 2006, 18 (7): 1527-1554.

[2] Li Hui. Research and Application of Deep Generative Model Learning Algorithms [D]. Beijing: University of Chinese Academy of Sciences, 2015.

[3] Hinton G E, Salakhutdinov R R. Reducing the dimensionality of data with neural networks [J]. Science, 2006, 313 (5786): 504-507.

[4] Zhang J, Tao Z Y. Recognition of speech based on deep learning [J]. Electronic Design Engineering, 2015, 23 (18): 72-73.

[5] Liu F, Jiao L, Hou B, et al. POL-SAR image classification based on Wishart DBN and local spatial information [J]. IEEE Trans on Geoscience & Remote Sensing, 2016, 54 (6): 3292-3308.

[6] Lin Miaozhen. Research on Face Recognition Based on Deep Learning [D]. Dalian: Dalian University of Technology, 2015.

[7] Bottou L. Large-Scale Machine Learning with Stochastic Gradient Descent [C]// Proc of COMPSTAT. Physica-Verlag, 2010: 177-186.

[8] Duchi J, Hazan E, Singer Y. Adaptive subgradient methods for online learning and stochastic optimization [J]. Journal of Machine Learning Research, 2011, 12 (7): 257-269.

[9] Sutskever I, Martens J, Dahl G, et al. On the importance of initialization and momentum in deep learning [C]// Proc of International Conference on Machine Learning. 2013: III-1139.

[10] Povey D, Zhang X, Khudanpur S. Parallel training of deep neural networks with natural gradient and parameter averaging [J]. Eprint Arxiv: 1410. 7455, 2014: 124-145.

[11] Srivastava N, Hinton G, Krizhevsky A, et al. Dropout: a simple way to prevent neural networks from overfitting [J]. Journal of Machine Learning Research, 2014, 15 (1): 1929-1958.

[12] Wan L, Zeiler M D, Zhang S, et al. Regularization of neural networks using drop connect [C]// Proc of International Conference on Machine Learning. 2013: 1058-1066.

[13] Wang Zhongmin, Wang Xi, Song Hui. Mobile user behavior recognition method based on random Dropout deep belief network [J]. Computer Application Research, 2017, 34 (12).

[14] Papa J P, Scheirer W, Cox D D. Fine-tuning deep belief networks using harmony search [J]. Applied Soft Computing, 2015, 46 (C): 875-885.

[15] Rosa G, Papa J, Costa K, et al. Learning parameters in deep belief networks through firefly algorithm [M]// Artificial Neural Networks in Pattern Recognition. [S. l.]: Springer International Publishing, 2016: 138-149.

[16] Rodrigues D, Yang X S, Papa J P. Fine-tuning deep belief networks using cuckoo search [M]// Bio-Inspired Computation and Applications in Image Processing. 2016: 47–59.

[17] Ioffe S, Szegedy C. Batch normalization: accelerating deep network training by reducing internal covariate shift [J]. Computer Science, 2015: 448-456.

[18] Fischer A, Igel C. An introduction to restricted Boltzmann machines [M]// Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications. Berlin: Springer, 2012: 14-36.

[19] Hinton G E. Training products of experts by minimizing contrastive divergence [J]. Neural Computation, 2002, 14 (8): 1771-1800.

[20] Vogl T P, Mangis J K, Rigler A K, et al. Accelerating the convergence of the back-propagation method [J]. Biological Cybernetics, 1988, 59 (4): 257-263.

[21] Cooijmans T, Ballas N, Laurent C, et al. Recurrent batch normalization [J]. arXiv preprint arXiv: 1603. 09025, 2016.

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