Evolving Connections in Group of Neurons for Robust Learning

Jia Liu, *Member, IEEE*, Maoguo Gong, *Senior Member, IEEE*, Liang Xiao, *Member, IEEE*, Wenhua Zhang, and Fang Liu, *Member, IEEE*

***Abstract*—Artificial neural networks inspired from the learning mechanism of the brain have achieved great successes in machine learning, especially those with deep layers. The commonly used neural networks follow the hierarchical multilayer architecture with no connections between nodes in the same layer. In this article, we propose a new group architectures for neural-network learning. In the new architecture, the neurons are assigned irregularly in a group and a neuron may connect to any neurons in the group. The connections are assigned automatically by optimizing a novel connecting structure learning probabilistic model which is established based on the principle that more relevant input and output nodes deserve a denser connection between them. In order to efficiently evolve the connections, we propose to directly model the architecture without involving weights and biases which significantly reduce the computational complexity of the objective function. The model is optimized via an improved particle swarm optimization algorithm. After the architecture is optimized, the connecting weights and biases are then determined and we find the architecture is robust to corruptions. From experiments, the proposed architecture significantly outperforms existing popular architectures on noise-corrupted images when trained only by pure images.**

***Index Terms*—Machine learning, neural-network architecture, particle swarm optimization, probabilistic model.**

I. Introduction

ARTIFICIAL neural networks (ANNs) that mimic the information processing mechanism of nature neural systems are one type of the popular methods dealing with machine-learning problems [1]. There are various types of ANN models dealing with different data and tasks. For instance, the Hopfield neural network [2] is a type of network with feedback connections that is usually used for associative memory [3] and function optimization [4]. Multilayer hierarchical neural networks own an end-to-end architecture with input, output, and multiple hidden layers which are usually used for classification [5] and regression [6]. With the demand for dealing with big data, deep neural networks [7] with more complex architectures have achieved many breakthroughs in numerous applications and for different types of data, different architectures are designed. For example, convolutional neural networks (CNNs) [8] are featured with local receptive field and weight sharing which is suitable to deal with image data [9], [10]. Recurrent neural networks (RNNs) [11] equipped with recurrent layers can well adapt to time-series data [12]–[14].

The architecture of neural networks plays an important role in neural learning. Therefore, architecture design has been a research interest for decades [15], [16]. The architecture indicating the arrange of neurons and assign of connections between pairs of them can be manually designed, such as AlexNet and GoogleNet [17], based on CNN or automatically evolved [18]. The architectures are usually designed for specific purposes. The popular one is to improve the performance in practice. The optimal depth, width in each layer, kernel size (CNN), and even connections evolve with the objective of test accuracy [18]–[20]. Sparsification of network connections could not only increase generalization but also compress the network. Therefore, connecting structure learning is also a popular topic. In [21], redundant connections are removed based on CNN for face recognition. We proposed a multiobjective model for connecting structure learning in [22] with the purpose of well capturing the input distribution by the sparsified structure.

Most of the popular neural networks are based on the hierarchical architecture with multiple layers, including the above-evolved architectures. The layers are defined as a set of neurons with no connections between them. Such architecture perfectly models the nonlinear relationship between input and output neurons and a deeper architecture could learn more complex models [23]. However, since there are no connections between units in the same layer or nonadjacent layers, the neurons are not compactly arranged and the information flow may be blocked. As a consequence, some architectures with skip connections are developed [24]–[26] among which the most successful one is the residual network (ResNet) [24]. ResNet adds shortcut connections between input and output layers of a section of the multilayer network to learn a residual function, that is, $\mathcal{H}(x) - x$ with x and $\mathcal{H}(x)$, respectively, representing

Manuscript received February 7, 2020; revised June 12, 2020; accepted August 25, 2020. Date of publication October 7, 2020; date of current version May 19, 2022. This work was supported in part by the National Nature Science Foundations of China under Grant 61906093, Grant 61802190, and Grant 61871226; in part by the Nature Science Foundation of Jiangsu Province, China, under Grant BK20190451; in part by the Jiangsu Provincial Social Developing Project under Grant BE2018727; and in part by the Fundamental Research Funds for the Central Universities under Grant 30919011279 and Grant 30919011281. This article was recommended by Associate Editor Y. S. Ong. *(Corresponding author: Jia Liu.)*

Jia Liu, Liang Xiao, and Fang Liu are with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China (e-mail: omegaliuj@gmail.com).

Maoguo Gong is with the School of Electronic Engineering, Xidian University, Xi'an 710071, China.

Wenhua Zhang is with the School of Artificial Intelligence, Xidian University, Xi'an 710071, China.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCYB.2020.3022673.

Digital Object Identifier 10.1109/TCYB.2020.3022673

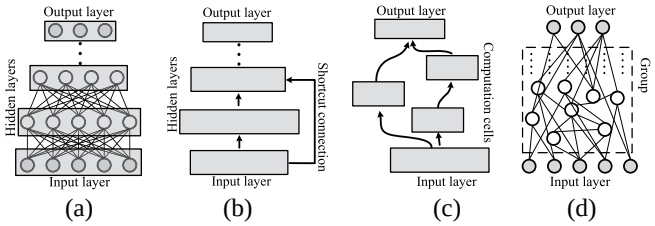


Fig. 1. Different architectures. (a) Hierarchical multilayer architecture. (b) Hierarchical architecture with skip connections. (c) Architecture based on cells. (d) Proposed group architecture.

the input and output of the section of network. A very deep network can be constructed and more complicated models can be learned with these shortcut connections. Recently, there are increasing interests in the architectures based on the block cells [27]–[29]. A cell is a directed acyclic graph consisting of an ordered sequence of nodes [30]. The connections between the cells are evolved based on the practical performance. The evolved architecture further relaxes the layer limitation with more free connections and more compact architecture can be obtained. Inspired from the nucleus in brain, in this article, we attempt a group architecture as shown in Fig. 1 which also exhibits the above-mentioned architectures. It is known that the structure of brain is hierarchical but each layer is much more complex than that in ANN and shows a group architecture. The proposed architecture is composed of the input layer, the output layer, and a group of neurons. In the group, a neuron is possible to connect to any neurons. But since we focus on the classification problem in this article, there are no feedback connections. The new architecture gets rid of the layer limitation and the neurons connect with each other freely which means there are less idle neurons.

However, if all the neurons in the group connect with each other, the number of connections as well as the computational complexity will increase exponentially with more neurons. The group architecture can be taken as multilayer networks with intralayer connections, such as Boltzmann machine [31], RNNs [11], and those with recurrent connections [32]. It is a great challenge to train such architecture and therefore intralayer connections are usually manually assigned, for example, RNN is assigned self-connections and is trained by time sequence. In this article, we try to evolve an appropriate architecture by determining connections between each pair of neurons. But with the large-scale free binary parameters, it is difficult to search the optimal connecting assignation. As mentioned above, it is popular to evolve the network architecture with the practical performance as the objective, for instance, test error for classification problems. However, computing the objective function not only requires high-performance computational devices but also time demanding. For example, by using reinforcement learning, a good architecture is obtained after 1800 GPU days in [27] and 3150 GPU days for evolutionary algorithm in [29]. Some attempts have been proposed to reduce the time demanding, including weight sharing [33], Bayesian optimization [34], and differentiable network architecture search [30]. But they mainly focus on the optimization methods while the complexity of the objective function is still a problem.

In this article, we propose to directly model the connecting structure without the involvement of weights and biases. This will avoid the training of weights and biases which greatly reduces the computational complexity of the objective function. In order to well represent the input and output relationship, we model the architecture based on the principle that more relevant input and output unit pair deserves higher connecting density and establish a probabilistic model to well capture the distribution of observed data. Then, an efficient objective function is derived and an improved binary particle swarm optimizer (BPSO) algorithm [35] is utilized to optimize the objective function. After the optimization of architecture, it is also important to optimize the connecting weights and biases to achieve the final learning purpose. Since the new architecture is different from the multilayer architecture, we propose a probabilistic model to well represent the input distribution and a contrastive divergence algorithm is used to optimize the model. Due to that the new architecture fully frees the architecture limitation and the connections are self-organized according to the input and output relationship, each neuron could play a full role in the information flow. Therefore, more compact architecture can be learned and from the experiments, the proposed architecture could achieve comparable performance with much less neurons and connections compared with multilayer architecture. More important, we find the new architecture is robust to corruptions in the test sets due to its data-driven architecture. The proposed method significantly outperforms existing popular architectures on noise corrupted images. In conclusion, the contributions of this article can be summarized as follows.

1) We attempt a group neural network (GNN) architecture and make it possible to image classification.
2) To optimize the architecture efficiently, we propose to construct the objective function through information flow without the involvement of weights and biases.
3) Learning tricks are proposed to increase the learning efficiency of architecture and parameters.
4) We find a special potential of the new architecture which is robust to corruptions. The potential is significant because the corruptions destroy the identical distribution of independent training and test data.

The remainder of this article is organized as follows. Section II introduces some techniques used in the modeling and optimization process. The detailed modeling and learning methods are described in Section III. In Section IV, we discuss the experimental study and in Section V, conclusion and future work are provided.

II. Preliminaries

In this section, we mainly introduce the related background about the proposed network. The aim of architecture learning for the group architecture is equivalent to removing the redundant connections which is similar to parameter pruning [36] in network compression. So we first review some related work about network pruning. We model the architecture following

probabilistic models and optimize it via BPSO. Then, we introduce neural-network-based probabilistic models and the commonly used particle swarm optimizer (PSO) algorithm.

A. Network Pruning

Network pruning removes redundant connections from a densely connected network to achieve better generalization and meanwhile lower computational and spatial complexity. In some methods, the redundant connections are removed via pruning criteria. The intuitive way is based on the amplitude of the weights [37]. In [21], CNN was sparsified via the correlation between two neurons inspired from Hebbian learning. But those methods determine the connections after training, and the architecture should be tuned iteratively which leads to multiple training of network parameters. Evolutionary algorithms are popular to evolve spare connections [22], [38], [39]. But for deep networks, those methods suffer from the large computational burden as introduced above. Moreover, existing network pruning methods are based on the multilayer network. In this article, we design a novel architecture learning method for the group architecture.

B. Probabilistic Models

The probabilistic model plays an important role in the development of deep learning [40]. The breakthrough from shallow networks to deep networks is the deep belief net (DBN) [41] proposed by Hinton in 2006 which was later successfully applied to dimension reduction [42]. In DBN, the restricted Boltzmann machine (RBM) [43], which is a type of probabilistic model, is used to pretrain the deep network layer by layer. This mitigates the gradient vanishing in the backpropagation (BP) algorithm. RBM learns to represent the input data by capturing the distribution of them. Then by layer-wise training, the abstract features can be extracted. This novel training paradigm has led to the research interests in unsupervised representation learning [40], and various feature learning models have been proposed based on RBM [44]–[46].

In this article, we model the architecture based on the Gibbs distribution

$$P(x;\theta)=\frac{\exp[-E(x,\theta)]}{\sum_{\tilde{x}}\exp\left[-E(\tilde{x},\theta)\right]} \tag{1}$$

where x is the observed data in the dataset and θ is the parameter set, including connecting weights and biases in neural models but in this article, θ is replaced by connecting assignment. The denominator is the partition function where $\tilde{x}$ denotes the data in the entire data space that x belongs to. It is defined to guarantee that the probability sum over the entire space is 1. For the model that fits the distribution of input data well, it should give high probability to the observed data and waste as little probability as possible on the remaining data in the entire data space. Then, the model can well capture the distribution of the observed data. We also establish such a model to optimize the connecting weights and biases after the optimization of architecture in order to mitigate the uneven influence of the output layer to input layers due to the irregular architecture.

The probabilistic model can be solved by maximizing the log likelihood via the gradient descent

$$-\frac{\partial\log p(x;\theta)}{\partial\theta}=\frac{\partial E(x,\theta)}{\partial\theta}-\sum_{\tilde{x}}p(\tilde{x};\theta)\frac{\partial E(\tilde{x},\theta)}{\partial\theta} \tag{2}$$

where the second term is the expectation of the gradient for data in the entire data space and can be estimated by alternant Gibbs sampling. Hinton proposed an efficient optimization method which is called the contrastive divergence algorithm [47]. With the contrastive divergence, RBM can be efficiently trained which reveals the learning capability of deep neural networks.

Probabilistic models capture the distribution of observed data by modeling them with trainable parameters. Therefore, in this article, we take the architecture as the trainable parameter and propose a probabilistic model to evolve the data adaptive architecture. On the one hand, the architecture can be evolved based on the model without the involvement of weights and biases which greatly reduces the computational complexity. On the other hand, the learned compact architecture can well capture the data distribution and follow the information flow between input and output neurons. With the model, we utilize the PSO algorithm to evolve the optimal architecture.

C. PSO Algorithm

PSO is a population-based heuristic optimization method inspired from the foraging behavior of flying birds [48]–[51]. Population-based methods are popular to optimize discrete problems, such as structure learning [22], community detection [52], and sparse representation [53]. In PSO, given a population with n individuals (solutions), each individual is taken as a particle and it adjusts its flying according to its own and companies' experiences

$$v_i^{G+1}=\omega v_i^G+c_1\text{rand}[0,1]\cdot\left(p_i^{\text{best}}-x_i^G\right)+c_2\text{rand}[0,1]\cdot\left(g^{\text{best}}-x_i^G\right) \tag{3}$$

where rand$[0, 1]$ is a randomly generated value between 0 and 1. c_1 and c_2 are user-defined parameters that controls the importance of the three terms. x_i^G denotes the ith particle in the population at the Gth generation. v_i denotes the flying direction and velocity of x_i and the position of the ith particle in the $G+1$th generation is obtained as $x_i^{G+1}=x_i^G+v_i^{G+1}$. p_i^{best} indicates the personal best solution found by the ith particle evaluated by a fitness function $f(x_i)$, that is, $p_i^{\text{best}}=x_i^g$ with $x_i^g=\arg\max\{f(x_i^1),f(x_i^2),\ldots,f(x_i^G)\}$. g^{best} is the global best solution found by all the particles in the population, that is, $g^{\text{best}}=p_i^{\text{best}}$ with $p_i^{\text{best}}=\arg\max\{f(p_1^{\text{best}}),f(p_2^{\text{best}}),\ldots,f(p_n^{\text{best}})\}$. The entire process of PSO is summarized in Algorithm 1.

In this article, the optimized variable is the architecture which is a binary matrix indicating the connecting assignment between each pair of neurons. Therefore, we use a BPSO [35] algorithm to optimize the connecting assignment. In BPSO, v

Algorithm 1 Process of PSO

Initialization:

Initialize the population $\{x_1^0, x_2^0, \ldots, x_n^0\}$ randomly and generate $\{p_1^{best}, p_2^{best}, ..., p_n^{best}\}$ and g^{best}.

G=0.

Repeat:

Compute v_i^{G+1} for each individual according to Eq. (3).

Update each individual $x_i^{G+1} = x_i^G + v_i^{G+1}$.

Update p_i^{best} for each individual and g^{best}.

$G = G + 1$.

Stop Criterion: Stop the iteration process when the number of iterations exceeds maximum value.

is generated according to (3) and the position of the ith particle is obtained by a probability matrix

$$S_i^G = \frac{1}{1 + \exp\left(-v_i^G\right)} \tag{4}$$

with each component in the matrix denoting the probability being connected. Then, x_i^G is obtained via sampling.

To further improve the efficiency of BPSO, in this article, we propose a novel multiscale BPSO with an increasing optimization scale. BPSO is first used to search in a low scale space and the space is enlarged and the population is initialized according to the optimized population in the lower scale space. This decreases the computational complexity of BPSO in early stages and improves the learning efficiency.

III. Methodology

To achieve the classification task, the compact group architecture should be trained by instances twice, one for learning the connecting architecture and one for connecting weights and biases. This process is different from most traditional architecture learning methods where parameter learning is executed numerous times to compute the objective function for evolving suitable architecture. Therefore, the proposed process is computational efficiency. Meanwhile, the architecture learned could well represent the input and output relationship.

A. Architecture Learning

The group architecture is shown in Fig. 1(d) where all the connections should be assigned by the learning method given a set of neurons. In this article, we propose to directly model the architecture based on the input and output relationship with the principle that more relevance between input and output units deserves higher connecting density. Here, the connecting density plays a crucial role which determines the connecting keynote. We first define the connecting density.

1) Connecting Density: The connecting architecture of network indicates the information flow from input to output. If an input neuron is more relevant to an output neuron, more information should be transferred. This means that wider and shallower information path should be constructed. Therefore, the connecting density should be defined according to both number of connections and depth. Here, the connecting density $\rho_{ij}(\varphi)$ between a neuron i and an input neuron j is defined as follows:

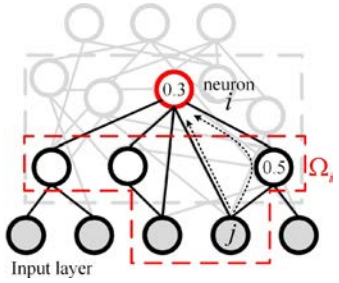


Fig. 2. Connecting density and its computing process. Values in the units are connecting density between this neuron and the input neuron j. For example, there are five connections connect with the neuron i among which one connection directly connects to the neuron j. Therefore, the full information of neuron j can be transformed to i, that is, the connecting density 1 is directly transmitted. One connection indirectly connects to neuron j and the connecting density 0.5 is transmitted to neuron i. Then, the connecting density between neurons i and j is obtained by (5): $(1 + 0.5)/5 = 0.3$.

$$\rho_{ij}(\varphi) = \frac{\sum_{k \in \Omega_i} \rho_{kj}(\varphi)}{M(\Omega_i)} \tag{5}$$

where φ is the connecting matrix with each component indicates the connecting status between each pair of neurons (1 for connected and 0 for unconnected). Ω_i denotes the set of lower neurons that directly connect to neuron i and $M(\Omega_i)$ denotes the number of such neurons. The connecting density can be propagated from the input neurons with that between an input neuron and itself being 1. Fig. 2 explains the above relationships where the number in the neuron is the connecting density between the neuron and input neuron j.

The connecting density can represent the information decay through the path. If a neuron directly connects to the input neuron, the information could be fully transferred to it. But since there are information from other neurons, the original information is mixed. Therefore, high density means a shallower path with less information from other neurons. For multilayer networks, the connecting density between different input and output neurons are equal. Since the relevance between different input and output neurons is not equal, in this article, the connecting density determined by the architecture is expected to well represent the input and output relationship which is evaluated by the mutual information between them.

2) Mutual Information: In order to well represent each input data, we use the pointwise mutual information to evaluate the relationship between input and output neurons for each data. Given a pair of values (x_j, y_i) imposed on the jth input neuron and ith output neuron, the pointwise mutual information is computed by

$$\mathbb{I}\left(x_j; y_i\right) = \log \frac{P\left(x_j, y_i\right)}{P(x_i)P\left(y_j\right)} \tag{6}$$

where $\mathbb{I}(\cdot)$ is the pointwise mutual information and $P(\cdot)$ denotes the probability which is estimated by all data in the dataset. For image data, the gray level is discrete and the probability can be directly counted. Continuous data can be discretized by even thresholds and then the probability can be counted. Pointwise mutual information is a measure of how much the actual probability of a particular co-occurrence

of events $p(x; y)$ differs from what we would expect it to be on the basis of the probabilities of the individual events and the assumption of independence $p(x)p(y)$ [54]. Since the mutual information is the expectation of the pointwise mutual information, we use the pointwise one for each data instead of only the expectation to capture the distribution.

Since the value of connecting density defined above is between 0 and 1, we further normalize the pointwise mutual information by [54]

$$\mathbb{N}(x_j; y_i) = \frac{\mathbb{I}(x_j; y_i)}{\mathbb{H}(x_j, y_i)} = \frac{\mathbb{I}(x_j; y_i)}{-\log P(x_j, y_i)} \tag{7}$$

where $\mathbb{H}(x_j, y_i)$ is the joint amount of information. However, the normalized mutual information is in the range of $[-1, 1]$. But the probability of negative values is small and can be omitted [55]. Therefore, we restrain the negative values to be 0, $\mathbb{R}(x_j; y_i) = \max[\mathbb{N}(x_j; y_i), 0]$. Then, the evaluation of the relationship between input and output components for each data is obtained. With the connecting density between each pair of input and output units and the corresponding relevance of each data, the probabilistic model is established.

3) Probabilistic Model: As introduced above, the probabilistic model is established based on the principle that larger relevance deserves higher density. Then, an energy describing such a principle is first defined as the cosine distance

$$\mathbb{E}(x, y; \varphi) = \cos\big[\rho(\varphi), \mathbb{R}(x, y)\big] \tag{8}$$

where $\rho(\varphi)$ and $\mathbb{R}(x, y)$, respectively, denote the vectors containing connecting densities and mutual information between all pairs of input and output neurons. If the architecture that can well follow the input and output relationship, the energy of it will be low. However, only minimizing the energy will lead to low energy for all the possible data. For example, when all the input neurons have equivalent relevance to output neurons, the connecting densities are equal which leads to uncertainty of depth. Therefore, a probabilistic model is established based on the energy to learn the distribution of observed data

$$p(x, y; \varphi) = \frac{\exp\big[-\mathbb{E}(x, y; \varphi)\big]}{\sum_{(\tilde{x}, \tilde{y})} \exp\big[-\mathbb{E}(\tilde{x}, \tilde{y}; \varphi)\big]}. \tag{9}$$

Similar to RBM and other probabilistic models, here $(\tilde{x}, \tilde{y})$ denotes one of all the possible input and output data in the entire data space. For learning the data-driven architecture, the model should assign more probability to observed data, that is, training data and as less probability as possible to all the other data, that is, fantasy data. However, different from PoE, the trainable parameter in this model is the architecture φ which is a binary matrix. It is scarcely possible to use the gradient method to derive the optimal architecture. For discrete variance, it is popular to evolve the optimal value by population-based methods [52], [56]. As a consequence, a BPSO algorithm is utilized in this article. It is necessary to derive a computable objective function.

4) Objective Function: Due to the unreachable fantasy data in (9), it is difficult to estimate the probability directly. Optimizing the model is to increase the numerator and decrease the denominator. It is easy to compute the numerator given the architecture and observed data. Fortunately, even though the denominator is difficult to be directly estimated, the value of it is only determined by the architecture since it is the sum over all the possible data. Therefore, we can design an alternative computable function to replace the denominator. Here, the new function is defined as the overall squared connecting density for all input and output pairs

$$\mathbb{D}(\varphi) = \sum_{i,j} \big\| \rho_{ij}(\varphi) \big\|^2. \tag{10}$$

This function denotes the overall representation of the architecture for all the data and larger value of it leads to larger energy over all the possible data. But this function may not follow the linear relationship with the denominator in (9). As a consequence, we add the function to the denominator as a new term and the objective function is obtained

$$\max J(\varphi) = \sum_{(x,y)\in\Psi} \exp\big[-\mathbb{E}(x, y; \varphi)\big] - \lambda \mathbb{D}(\varphi) \tag{11}$$

where Ψ is the dataset that contains all the observed data. λ is a user-defined parameter that controls the importance of the two terms. Optimizing this objective will increase the numerator and decrease the denominator of the probabilistic model in (9). Therefore, it is expected to achieve the architecture that follows the principle. Then with the computable objective function, an improved BPSO algorithm is designed. To compute the objective, only one forward of the connecting density is needed compared with numerous forward and backward process to train the parameters in traditional objectives. Thus, the computational complexity of this objective is much lower.

It deserves to be noted that the objective is not the performance in classification which is different from most architecture learning methods. The proposed objective function evaluates the information flow through the architecture. In general, an architecture that cannot guarantee information flow between input and output neurons would not perform well in practice. Therefore, optimizing the proposed objective function will increase the classification performance as well. It is a substitute but it makes it possible to optimize the complex architecture due to its low computational complexity. Some existing compressing methods also use this way, such as the parameter pruning methods as introduced above.

5) Improved BPSO Algorithm for Optimization: The PSO algorithm is an efficient global optimization method and is widely used in many problems as introduced above. In this article, a BPSO is utilized to optimize the architecture. However, the high-dimensional searching space is still a great challenge for BPSO in this problem. For example, if there are 200 neurons in the group, the dimension of the variable is over 40 000. Therefore, we propose a multiscale BPSO (MS-BPSO) to improve the learning efficiency.

In the nature evolution theory, the genetic information in lower creatures is less than that in higher creatures. The genetic information increases with the process of evolution. Inspired from this process, in this problem, the high-dimensional variable can be evolved from the results of a lower dimensional variable. One stage of the multiscale process is exhibited in Fig. 3 which includes two steps, that is, split and evolving. In MS-BPSO, 2^k neurons are binded as one cell at the first stage.

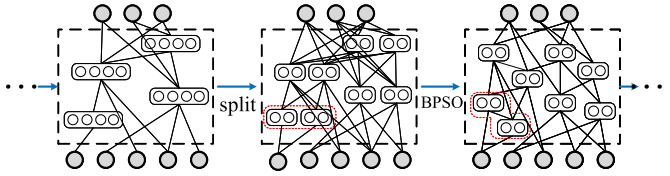


Fig. 3. Exhibition of one stage of MS-BPSO. In each stage, one neuron splits into two neurons and they share the same connecting configuration. Then, BPSO is utilized with the result in the last stage as the initialized architecture.

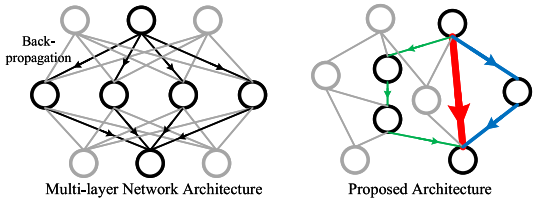


Fig. 4. Illustration of the difference between the multilayer architecture and the proposed architecture in terms of the BP algorithm.

Then, BPSO is utilized to search in the low-scale searching space. After convergence, each cell spilts into two cells containing 2^{k-1} neurons. The two cells share the same connecting configuration. With such architectures as initialized population, BPSO is utilized again. After k stages, each cell contains only one neuron. Then, the best architecture searched by BPSO is the final result.

Given an N-dimensional binary variable, at the first stage, the searched variable has $N/2^k$ dimensions which greatly reduces the searching space and also the complexity of the objective function. Even though the final searching space is not reduced, the initialized population has been evolved into a good position. Therefore, with the same number of generations, MS-BPSO is more efficient. After the architecture is optimized, it is important to optimize the connecting weights and biases in order to achieve the learning task.

B. Network Parameter Learning

The widely used parameter learning method for hierarchical neural networks is the BP algorithm [8]. In the BP algorithm, the gradients of parameters in each layer are computed from the output errors backpropagated from the output layer. In this architecture, the BP algorithm can also be used since there is no feedback connections. The errors of each neuron in the group can be backpropagated from the output layer. But since there are no regular layers in the group, some connections may not be fully trained by the BP algorithm. Fig. 4 illustrates the difference between hierarchical and proposed group architecture in terms of the BP algorithm. In hierarchical architecture, since there are no connections in the same layer, the errors can be evenly backpropagated to the input layer as shown in Fig. 4. However, in the proposed architecture, there are many paths with different depths for backpropagating errors. As shown in Fig. 4, the errors could be fully backpropagated through the red connection. Then, the weights on the red connection can be adequately trained. But since BP only focuses on the output loss, with the less influence of weights of deep connections on output units, the training of lower blue and green connections is restrained by the red connection. Therefore, we propose a probability model to give consideration to input distribution.

The network parameters should not only reduce the output loss on the training data but also well represent the input data. Therefore, a probabilistic model is established with the output loss as the energy

$$p(x|y;\theta) = \frac{\exp\left[-E(x,y;\theta)\right]}{\sum_{\tilde{x}} \exp\left[-E(\tilde{x},y;\theta)\right]} \tag{12}$$

where θ is the network parameter set, including the connecting weights and biases. $E(x,y;\theta)$ denotes the energy and here the output loss is used as the energy, that is, $E(x,y;\theta) = \|y - f_\theta(x)\|_2^2$ where $f_\theta(x)$ denotes the output of the network given input of x. Following the routine of probabilistic models, $\tilde{x}$ denotes one of all the possible data in the data space. For a model that well represents the input data, it should achieve less training loss than other fantasy data. Since the network parameters are continuous, it can be optimized by maximizing the log likelihood via gradient ascent

$$\begin{aligned}\Delta\theta &= \frac{\partial \log p(x|y;\theta)}{\partial\theta} \\ &= -\frac{\partial E(x|y;\theta)}{\partial\theta} + \sum_{\tilde{x}} p(\tilde{x}|y;\theta)\frac{\partial E(\tilde{x}|y;\theta)}{\partial\theta}.\end{aligned} \tag{13}$$

The first term is the gradient of θ in terms of training loss which can be easily computed by the BP algorithm. The second term is the gradient expectation of all the possible data which is difficult to estimate. In RBM and PoE, the expectation is estimated by Gibbs sampling as introduced above. In Gibbs sampling, the status probability of the input data is derived according to the model, and then the value of the input data is sampled from the probability. Since RBM and PoE are based on the single-layer network, it is convenient to derive the probability. In the proposed architecture, it is quite difficult to directly derive the probability with the complex connections. As a consequence, in this article, we propose a reversed sampling method.

It is intuitive that the sampled data are more likely to locate at the high density region of the model. Therefore, the sampled data can be obtained by increasing the probability density with the sampled data as the variable

$$\max_{\hat{x}} \log p\left(\hat{x}|y;\theta\right) \tag{14}$$

where $\hat{x}$ is the sampled data. Then, we derive the gradient of the log likelihood

$$\begin{aligned}\Delta\hat{x} &= \frac{\partial \log p\left(\hat{x}|y;\theta\right)}{\partial\hat{x}} \\ &= \frac{\partial - E\left(\hat{x}|y;\theta\right)}{\partial\hat{x}} - \frac{\partial \log \sum_{\tilde{x}} \exp\left[-E(\tilde{x}|y;\theta)\right]}{\partial\hat{x}}.\end{aligned} \tag{15}$$

Fortunately, the denominator in (12) is not the function of $\hat{x}$. Therefore, the second term in (15) is 0. Then, the first term can be obtained by the errors backpropagated from the output layer. With the sampled $\hat{x}$, the second term in (13) can be computed and the parameters can be updated with the gradient.

However, to estimate the gradient expectation, numerous iterations are necessary to obtain the optimal value of the sampled data. Moreover, several sampled data should be generated to estimate the expectation more accurately. One iteration needs a forward and a backward. Then, to compute the gradient of network parameters, numerous forward and backward processes are necessary which is significantly time consuming. Therefore, we consider a contrastive divergence algorithm [47] to optimize the model. Maximizing the probability in (12) is equivalent to minimizing the Kullback–Leibler (KL) divergence $Q||\hat{Q}^{\infty}$ between sampled distribution $\hat{Q}^{\infty}$ (via numerous iterations) and observed distribution Q which means that the model can well capture the distribution of observed data. Instead of optimizing the KL divergence, the contrastive divergence algorithm minimizes the difference between two KL divergences $\min(Q||\hat{Q}^{\infty} - \hat{Q}^{1}||\hat{Q}^{\infty})$ where $\hat{Q}^{1}$ denotes the distribution that is one step closer to $\hat{Q}^{\infty}$ than Q. Minimizing the contrastive divergence indicates that $Q = \hat{Q}^{1}$ which indirectly indicates that $Q = \hat{Q}^{\infty}$. Then, the KL divergence will be minimized. The updating gradient of network parameters is computed by

$$\begin{aligned}\Delta\theta &= \frac{\partial\left(Q\middle\|\hat{Q}^{\infty} - \hat{Q}^{1}\middle\|\hat{Q}^{\infty}\right)}{\partial\theta} \\ &= -\left\langle\frac{\partial E(x,y;\theta)}{\partial\theta}\right\rangle_{Q} + \left\langle\frac{\partial E\left(\hat{x},y;\theta\right)}{\partial\theta}\right\rangle_{\hat{Q}^{1}} \\ &\quad + \frac{\partial\hat{Q}^{1}}{\partial\theta}\frac{\partial\left(\hat{Q}^{1}\middle\|\hat{Q}^{\infty}\right)}{\partial\hat{Q}^{1}} \end{aligned} \tag{16}$$

where $\langle\cdot\rangle_{Q}$ denotes the expectation of the gradient following the distribution of Q. The last term is problematic to compute, but extensive simulations show that it can safely be ignored because it is small and seldom opposes the resultant of the other two terms [47]. Then, the first term is the gradient of the observed data and the second term is that of the data that is one step closer to sampled distribution. $\hat{x}$ can be obtained by one iteration of gradient descent which can result in a higher probability than that of x. As a consequence, to compute the updating gradient of network parameters, three times forward and backward processes are necessary in each iteration (one for sampling). This significantly reduces the computational complexity. The overall process is exhibited in Algorithm 2.

The new parameter learning method may mitigate the problem we analyze above. When the probability model is optimized, the sampled distribution should follow the distribution of the observed data. Therefore, if the lower blue and green connections cannot be well trained, the input status cannot be accurately sampled due to the direct influence of lower connections. During the learning process, the reconstructed data $\hat{x}$ is also obtained by using the BP algorithm and the gradient of each weight is the subtraction between gradients computed by observed data and reconstructed data. Then, if the weight in a connection is not well trained, the reconstructed status of its input unit is distinct with the observed input status. Then the amplitude of the gradient will be large to modify the weight. Otherwise, the amplitude will be close to 0. Moreover, the reconstructed data is generated by adding the gradient backpropagated from the output layer. This gradient can be taken as a disturbance that modifies the observed data to achieve lower loss. The parameter learning aims to restrain the reconstructed data which can be taken as an adversarial training [57].

Algorithm 2 Parameter Learning Process of the Proposed Architecture

Initialization: initialize the connecting weights and biases randomly.

Repeat:

FOR each data x in the dataset:

Sampling: Generate $\hat{x} = x + \Delta\hat{x}$ according to the gradient in Eq. (15).

Computing gradients: Compute the gradient $\partial E(x)/\partial\theta$ with the observed data x and the gradient $\partial E(\hat{x})/\partial\theta$ with the sampled reconstructed data $\hat{x}$.

Updating: Update the network parameter with the gradient in Eq. (16).

Stop Criterion: stop the iteration process when the training error is lower than a threshold or the number of iterations exceeds maximum value.

Algorithm 3 Learning Process of the Entire System

Set Hyperparameters: k, c_1, c_2, number of neurons, λ, number of maximum iterations.

Architecture Learning:

Initialize the population of binded connecting matrix φ randomly.

Repeat:

Optimization with PSO in **Algorithm 1**.

$k - 1$.

Stop when $k = 1$ and output optimized φ.

Parameter Learning:

Construct information flow on φ.

Optimize the connecting weights and biases according to **Algorithm 2**.

It deserves to be noted that compared with the basic BP algorithm which is composed of one forward and one backward processes in each iteration, the proposed method needs two forward and two backward processes in each iteration. One for reconstruction and the gradient $\partial E(x)/\partial\theta$, and another for the gradient $\partial E(\hat{x})/\partial\theta$. Therefore, the computational time is approximately twice that of the BP algorithm with the same number of iterations. In conclusion, the learning process of the entire system is summarized in Algorithm 3.

C. Toy Example

To better describe the entire story of architecture learning and parameter learning. We construct a simple dataset to illustrate the trained architecture and sampled data. First, a simple simulated dataset for classification is constructed. There are two classes and three attributes in each data as shown in Fig. 5. Among the three attributes, the first two attributes follow the Gaussian distribution with different parameters for different

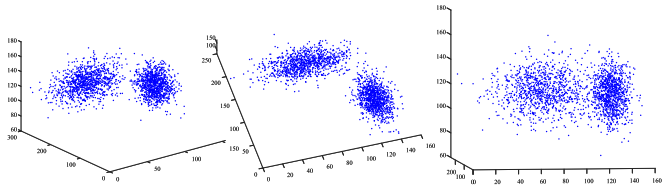


Fig. 5. Simple simulated dataset with three attributes. Blue point denotes each data and the distribution of them is illustrated by different view angles. The third attribute indicated by the vertical axis has no contribution to the classification.

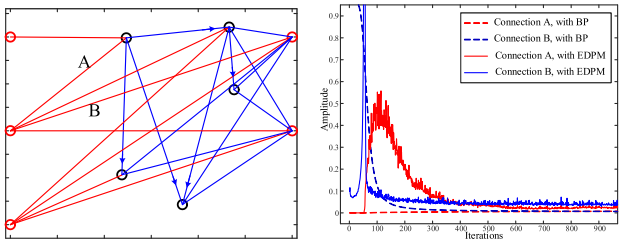


Fig. 6. (a) Connecting architecture learned by the simulated dataset. Red circles denote the neurons in the input and output layers. Connections connecting to input neurons are marked by the red lines. (b) Gradient amplitude of the two connections A and B during training with BP and the proposed EDPM, respectively.

classes. The third attribute follows the Gaussian distribution with the same parameter for all data which means that the third attribute has no contribution to the classification. First, the architecture learning is executed and the learned architecture is illustrated in Fig. 6(a).

We set five neurons in the group and the architecture is evolved according to the input and output relationship. It is obvious that the first two neurons in the input layer are assigned more connections and they also directly connect to the output neurons. The third attribute has no contribution to the output layer, therefore, only one connection is assigned for it. This demonstrates that the learned architecture follows the principle that higher correlation leads to larger connecting density. The learned architecture is expected to learn the input and output relationship. After architecture learning, parameter learning is implemented to learn the classification function. To exhibit the effect of the proposed error-driven probabilistic model (EDPM), we select two connections marked as A and B in Fig. 6(a) and plot their gradient amplitude during training as shown in Fig. 6(b). The connection B directly connects to the output unit, therefore, the gradient amplitude is large at the early stage of training which can modify the weight rapidly. As a consequence, there is a peak along the curve of connection B either trained by BP or EDPM. However, for BP, the maturity of connection B will restrain the training of connection A due to the same destination of them but a longer way for A. As a consequence, the gradient amplitude of A is restrained to be much smaller and the weight is approximately random when trained by BP. But this phenomenon can be mitigated by EDPM since the update gradient is the difference between two differential coefficients. As explained above, the gradient is difficult to be restrained by mature connections. As a consequence, the gradient amplitude is also large as the red line shows.

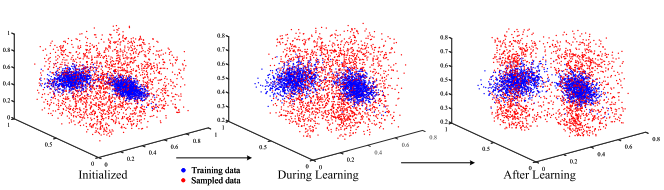


Fig. 7. Training data and sampled data. The sampled data indicated by red points are sampled based on the proposed sampling method with the network parameters. The sampled data during different stages of the learning process are exhibited.

The sampled data during the learning process are exhibited in Fig. 7. The initialized network has no capability to represent the input data, therefore, the sampled data are randomly distributed as shown in Fig. 7. During the training process, the distribution of the sampled data begin to converge to that of the training data. This means that network tends to capture the input distribution. However, after training, data generated from the first two attributes well follow that of distribution in the training data. Since the third attribute has no contribution to the classification, the sampled data from this attribute cannot converge to the training distribution. This interesting phenomenon demonstrates that the network could highlight the components that are relevant to the classification while omitting irrelevant components. This means that the new network is robust to background variance and corruptions. A detailed evaluation of the proposed network is implemented in the next section.

IV. Experimental Study

In this section, we focus on the classification problem and evaluate the proposed GNN and corresponding learning method with more complex datasets, that is, MNIST, MNIST with random background, and CIFAR-10 datasets. Moreover, CIFAR-10 images corrupted by salt–pepper noise and Gaussian noise are used to test the robust learning of GNN. First, the improved MS-BPSO algorithm is verified by comparison with the baseline. Then, the traditional architectures, including DBN and CNN, are compared with GNN on the three datasets. After that, state-of-the-art image classification methods are compared to highlight the special superiority of GNN and its problems. Finally, the robustness to hyperparameters is tested. Above all, the datasets are introduced.

A. Datasets

The MNIST dataset is a collection of images of handwritten digits from 0 to 9. The size of each image is 28×28. In the MNIST dataset, there are 60 000 training images and 10 000 test images. These digits are collected with the pure background, but in practice, there are digits under different backgrounds, therefore, we use a MNIST derived dataset (MNIST bg-rand) where digits are on background with random noise [58]. The training and test digits in this dataset are 12 000 and 50 000, respectively. CIFAR-10 is an image dataset containing 60 000 color images of 32×32 pixels from ten

Fig. 8. Illustration of images in three datasets. (a) MNIST dataset. (b) MNIST bg-rand dataset. (c) CIFAR-10 dataset. (d) CIFAR-10 images corrupted by salt–pepper noise. (e) CIFAR-10 images corrupted by Gaussian noise.

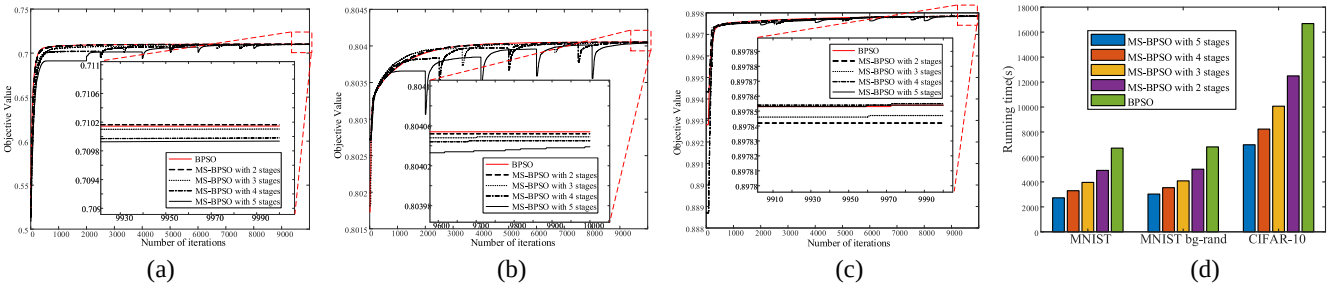


Fig. 9. Convergence curves and computational time of different BPSO on the three datasets. (a) Curves on the MNIST dataset. (b) Curves on the MNIST bg-rand dataset. (c) Curves on the CIFAR-10 dataset. (d) Computational time (s).

classes. The ten classes include airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck, which are completely mutually exclusive. The images in this dataset are divided into 50 000 for training and 10 000 for test. Some images in the three datasets are illustrated in Fig. 8. It can be found that in MNIST bg-rand and CIFAR-10 datasets, the backgrounds are more complex than that in the MNIST dataset. Moreover, it is even difficult for human to recognize the objects when they are corrupted by noise severely.

B. Evaluation of Learning Process

In order to increase the converging speed of BPSO, we propose to evolve the population from low-scale data space. In this section, we verify the efficiency of MS-BPSO by comparing it with the BPSO. Meanwhile, a different number of stages, that is, the values of k are set to be $k = \{5, 4, 3, 2\}$, respectively. The overall number of generations for BPSO and MS-BPSO is 10 000 and we set the same number of generations for different stages, for example, 2000 generations for each stage when $k = 5$. For MNIST and MNIST bg-rand datasets, we set 256 neurons in the group and for the CIFAR-10 dataset, there are 512 neurons. The convergence curves which exhibit the objective values in each generation are illustrated in Fig. 9. For more clearly illustration, we zoom part of the curves in as surrounded by the red dashed boxes.

It can be found that at the end of evolving, BPSO and MS-BPSO achieve almost the same performance. In MS-BPSO, there is a leap between each two stages due to the extend of the architecture. However, the objective value increases significantly after the leap. This demonstrates that the performance degression caused by neuron split can be rapidly recovered. On the other hand, the curves demonstrate that the architecture evolved by bundled neurons provides a better initialization than the randomly initialized architectures in BPSO. From the amplified curves, MS-BPSO outperforms BPSO on MNIST and CIFAR-10 datasets with appropriate number of stages.

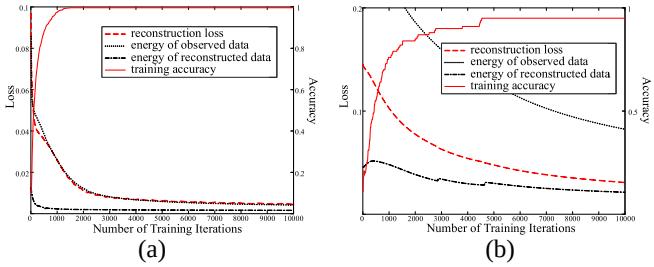


Fig. 10. Reconstruction loss, energy of observed data, energy of reconstructed data, and training accuracy during the learning process. (a) MNIST dataset. (b) CIFAR-10 dataset.

But most MS-BPSO cannot outperform BPSO while the performance gap is not large. From the above analysis, MS-BPSO is superior in terms of efficiency. Therefore, we plot the running time bars for BPSO and MS-BPSO on the three datasets as shown in Fig. 9(d).

Since MS-BPSO evolves from low-scale populations, the running time at early stages can be greatly reduced. Therefore, the computational time of MS-BPSO is significantly less than that of BPSO and more stages leads to less running time. When $k = 5$, the computational time is even approximately half of that of BPSO. Therefore, MS-BPSO achieves the almost equivalent performance with much less running time compared with BPSO. In the following experiments, we set $k = 5$.

Then, we show some variables during the parameter learning process in Fig. 10, including the reconstruction loss, that is, $\|x - \hat{x}\|^2$, the energy of observed data, that is, $E(x)$, the energy of reconstructed data, that is, $E(\hat{x})$, and training error. During the learning process, the energy of reconstructed data is always lower than that of observed data because reconstructed data is more close to the low region of the energy field. With the iterations, the energies of two types of data become closer and the distance between x and $\hat{x}$ becomes smaller. After 5000 iterations, the learning begins to converge and the training accuracy stops to increase.

C. Comparison With Traditional Architectures

For MNIST and MNIST bg-rand datasets, we compare the proposed GNN with DBN and CNN. We set the architecture of DBN as "784-500-300-10" [41] and CNN as LeNet [8] which has two convolutional layers (with 6 and 16 feature maps obtained by 5×5 convolution kernels, respectively) and three full connecting layers (120-84-10). We set 256 neurons in the group and $\lambda = 1$.

1) Experiments on GNN Trained by the MNIST Dataset: We first train DBN, CNN, and the proposed GNN by the training set in the MNIST dataset. Moreover, since the objective of architecture learning is not based on test accuracy, we also compare the randomly initialized GNN and dense GNN (all neurons are connected with each other) to demonstrate the effectiveness of the architecture learning model. All the learning rates, including BP and sampling process, are set to be 0.01. The three architectures are trained by both BP algorithm and the proposed EDPM. After training, we first implement the trained models to test set in the MNIST dataset. The classification accuracy is listed in Table I.

From the classification for MNIST digits, CNN with the BP algorithm achieves the best performance due to the specially designed architecture for images. EDPM cannot outperform BP on architectures of DBN and CNN. Because EDPM is a probabilistic model which means that the model gives high response to observed data while restraining unseen data. Therefore, some images in the test set will be excluded by the model. However, for GNN, EDPM mitigates the problem when BP is applied to this architecture. As a consequence, EDPM greatly improves the performance of GNN. Even though GNN cannot outperform CNN, it outperforms DBN with less neurons and connections. In GNN, there are 256 neurons in the group and 186 465 connections after training. While in DBN, there are 800 hidden units and 545 000 connections and even more in CNN. This means that GNN could learn a more compact architecture due to that a neuron can process information from all possible neurons instead of neurons from the last layer. Consequently, with less neurons and connections, GNN outperforms DBN on this problem. In the MNIST dataset, the digits are on pure background, but there are many digits on various backgrounds in practice. Therefore, with the trained model by training set in the MNIST dataset, digits with the background of random noise are tested. The classification accuracy is also listed in Table I.

It is obvious that DBN and CNN trained by digits with the pure background cannot well deal with digits with other backgrounds. Because the background pixels in MNIST digits are 0 and then the backpropagated error cannot influence the connections that connect to background pixels. Therefore, after training, the weights on the connections that connect the background pixels is approximately random. Then with a nonzero background pixel, it will propagate to the output layer and influence the output. Moreover, CNN achieves the worst performance because the kernels are shared for both foreground and background pixels. The change of background will influence the output a lot. Since the architecture of GNN is evolved based on the input and output relevance, it is more robust to background changes and it significantly outperforms DBN and CNN on this test set. This is significant in machine learning because this learning paradigm contradicts the basic hypothesis in machine learning that training and test data follow the identical distribution. Here, training and test data not only follow different distribution but also are independent. This means that GNN can infer new instances from limited examples. Then, we train the three architectures by the training set in the MNIST bg-rand dataset.

2) Experiments on GNN Trained by the MNIST bg-rand Dataset: After trained by the digits with the background of random noise, the test accuracy on the test set in the MNIST bg-rand dataset is listed in Table I. Similarly, CNN achieves the best performance in this problem. However, this time EDPM outperforms BP for both DBN and CNN. Because the background is random noise, which has no regular pattern for EDPM to model. Then, the sampled background pixels are random. Thus, the model focuses more on the foreground. While for BP, the uncertainty of background pixels leads to randomized connecting weights. Therefore, the EDPM could reduce the influence of background pixels. GNN cannot outperform CNN but outperforms DBN with less processing units and parameters which demonstrates the processing efficiency of each neuron in the group.

Similarly, we also apply the trained architectures to digits with pure background and the test accuracy is listed in Table I. We find that models trained by digits with random background could well classify the digits with pure background. However, due to the data-driven architecture of GNN, it achieves the best performance. The above experiments demonstrate that GNN is a compact architecture that can fully use the processing capability of each neuron in the group. Moreover, GNN is robust to background changes and significantly outperforms traditional architectures when the backgrounds of training and test sets are different.

Among the architectures, the randomly initialized GNN achieves the worst result. Because the connections are randomly assigned, the architecture cannot guarantee the information flow between input and output neurons. Therefore, it cannot perform well in classification and the use of EDPM degrades the performance.

3) Experiments on the CIFAR-10 Dataset: For the CIFAR-10 dataset, we set 512 neurons in the group and $\lambda = 1$. We set the architecture of DBN as "3072-1024-512-10" and CNN with two convolutional layers (both 64 feature maps obtained by 5×5 convolution kernel) and three full connecting layers (384-192-10). The classification accuracy is listed in Table II. Similarly, CNN achieves the best performance with BP due to its specially designed architecture for the image. However, compared with DBN, GNN greatly improves the performance because of the learned architecture based on input and output relationship.

As shown in Fig. 8(c), the CIFAR-10 dataset contains objects with various appearance on different backgrounds. The pooling layer in CNN could remove these redundant information and the learned architecture in GNN could reduce the impact of background. But GNN achieves the equivalent performance to CNN with 512 neurons and 1 217 843 connections which are much less than 1536 hidden neurons

TABLE I
CLASSIFICATION ACCURACY (%) OF DIFFERENT ARCHITECTURES TRAINED BY TRAINING SETS IN MNIST AND MNIST BG-RAND DATASETS ON TEST SETS IN THE TWO DATASETS

Training Set	Test Set	DBN		CNN		Dense GNN		Random GNN		GNN	
		BP	EDPM	BP	EDPM	BP	EDPM	BP	EDPM	BP	EDPM
MNIST	MNIST	97.09	94.22	**99.14**	94.15	96.23	97.59	86.87	86.96	94.21	98.62
	MNIST bg-rand	30.27	9.84	21.60	16.18	31.44	33.07	31.93	33.05	32.90	**67.38**
MNIST bg-rand	MNIST bg-rand	89.94	90.26	93.45	**93.71**	91.57	84.52	63.06	58.26	90.34	91.99
	MNIST	75.61	78.51	64.30	69.46	66.90	52.07	46.48	35.18	85.73	**85.93**

TABLE II
TEST ACCURACY (%) ON THE CIFAR-10 DATASET

Methods	CIFAR		CIFAR (salt-pepper noise)		CIFAR (Gaussian noise)	
	BP	EDPM	BP	EDPM	BP	EDPM
DBN	57.25	55.74	35.99	34.98	35.40	35.13
CNN	**76.85**	69.50	55.36	45.12	40.64	44.71
GNN	72.15	72.71	44.78	**56.45**	40.87	**50.58**

TABLE III
TEST ACCURACY (MEAN±STD) OF DIFFERENT METHODS ON THE MNIST DATASET

Test Digits	ResNet	DARTS	GNN
MNIST	99.34±0.04	**99.67±0.04**	98.52±0.08
MNIST bg-rand	43.18±9.74	42.50±7.76	**66.86±2.80**
Training time (h)	0.57	12.75	3.84+1.53

TABLE IV
TEST ACCURACY (MEAN±STD) OF DIFFERENT METHODS ON THE CIFAR DATASET

Test Images	ResNet	DARTS	GNN
CIFAR	90.75±0.16	**96.58±0.18**	76.68±0.83
CIFAR (Gaussian noise)	44.80±1.05	41.36±4.10	**51.28±1.12**
CIFAR (salt-pepper noise)	50.30±0.87	47.63±4.95	**56.26±0.50**
Training time (h)	2.92	63.62	12.27+11.55

and 3 675 136 connections in DBN. As a consequence, GNN is a compact architecture that takes full use of the processing capability of each neuron. The superiority of GNN is its robustness. Therefore, we continue to apply the trained architectures to test sets corrupted by noise and the classification accuracy is listed in Table II. Note that the accuracy of corrupted test sets is obtained by models trained by original images where the distribution is not disturbed by noise. Then GNN can significantly outperform CNN because some corrupted pixels can be avoided by GNN which may influence CNN severely. The above experiments demonstrate the superiority of the proposed architecture over traditional hierarchical neural-network architectures. We find that GNN not only learns a compact architecture but also can learn from compact datasets. GNN is more robust to corruptions than traditional architectures which means less training data can learn a robust model. Next, we compare GNN with state-of-the-art methods.

D. Comparison With State-of-the-Art Methods

Here, we compare GNN with ResNet-20 [24] and DARTS [30]. ResNet is an architecture with skip connections which is different from traditional architectures but has achieved excellent performance in image classification and other learning problems. Here, we use ResNet-20 with 20 layers for image classification. DARTS is an evolved architecture with block cells as introduced above. Then we train those methods with MNIST digits and CIFAR-10 images and test the trained model with MNIST bg-rand digits and noise corrupted CIFAR-10 images. Note that the noise changes the distribution of training data in test data which violates the hypothesis of independent and identical distribution in machine learning. This is a great challenge for learning models. The classification accuracy is listed in Tables III and IV, respectively. The results are in mean±std over ten independent runs.

For traditional classification problems, ResNet and DARTS achieve excellent performance due to their large modeling capability for complex distributions and finely tuned hyperparameters. The performance of GNN is poor compared with them but outperforms DBN which is also not specially designed for images in architecture. Because it is difficult to accelerate the learning process of GNN via existing parallel computation platforms, the scale of GNN is small and the construction of GNN is more like traditional multilayer perceptions. There are no complex network components, such as convolution, pooling, and other learning tricks as ResNet and DARTS. However, when it goes to noise corrupted images, ResNet and DARTS fail to classify them accurately. Because ResNet and DARTS follow the hypothesis that the distributions of training and test data are independent and identical. It is intuitive that a model with more modeling parameters can better model the entire complex distribution with enough training data and well-designed architecture, objective, and optimization method. But when the distribution of training data is corrupted by noise, they cannot infer the unknown corrupted distribution. GNN is able to infer unknown distributions as in Fig. 7. Therefore, with corrupted test data, it outperforms the ResNet and DARTS. The training times of the three methods are also listed in the tables. For GNN, the training times include those of architecture and parameter. Note that ResNet and DARTS are accelerated by GPU parallel computation and GNN is implemented serially on CPU due to its irregular architecture. DARTS takes days to obtain an excellent architecture. But since the objective of architecture search is training error. The architecture cannot adapt to noise corrupted test data.

E. Experiments on Hyperparameters

There are some user-defined hyperparameters in GNN which influence the performance of GNN. In this section, we

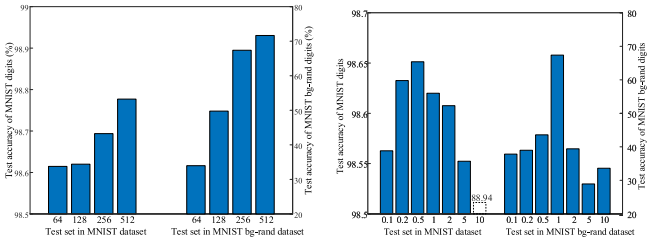


Fig. 11. Test accuracy of GNN with different hyperparameter values on test sets of MNIST and MNIST bg-rand. (a) Number of neurons. (b) λ.

test the robustness of GNN on the number of neurons and value of λ. First, we set number of neurons in the group to be $\{64, 128, 256, 512\}$ and train the architecture and parameter with the MNIST dataset. We set $\lambda = 1$ and the test accuracy on test sets of MNIST and MNIST bg-rand is illustrated in Fig. 11(a) where the horizontal axis denotes the number of neurons and vertical axis denotes the test accuracy.

It is obvious that better performance is obtained with more processing units. But for test data in the MNIST dataset, the accuracy distinction is within 0.2%. This means that even with 64 neurons, the performance is not degraded a lot compared to GNN with 512 neurons on the MNIST test set. For traditional classification problems, GNN is robust to the number of neurons. However, when GNN is applied to classifying digits with random background, more neurons could achieve significant improvement. Because for traditional classification problems, training set and test set follow identical distribution. Then the background pixels which are with the value of 0 in the test set will not disturb the accuracy of the trained model. Since there are pixels that are not always background, for a digit with background of random noise (nonzero values), more neurons are necessary to learn the complex relationship between input and output neurons. A more complex distribution is constructed by more neurons. However, with more neurons, the searching space will increase exponentially. Since it is difficult to accelerate GNN via existing GPU computation platforms, such as Tensorflow[1] which are developed for deep architectures based on regular multilayer prototype, more neurons pose a giant computational burden. It is time demanding to train such a large-scale network. Even though, the 512 neurons are still less than the 800 hidden neurons in DBN.

Then, we set the number of neurons to be 256 and set $\lambda = \{0.1, 0.2, 0.5, 1, 2, 5, 10\}$ to train GNN. The test error on the test sets of MNIST and MNIST bg-rand datasets are illustrated in Fig. 11(b) where the dashed bar denotes that the test accuracy is much less than the minimal accuracy exhibited. Similarly, the performance on the MNIST dataset is not sensitive to values of λ. Except $\lambda = 10$, the performance difference is within 0.1% while with the difference of λ being 4.9. Because λ controls the importance of improving the representation capability for visible data while decreasing that of all the other data. Therefore, larger λ means to evolve the architecture with less entire connecting density which may also reduce the representation capability for observed data. Since digits in the MNIST dataset is not complex, the GNN is robust to λ. However, when the trained GNN is applied to digits with the background of noise, the λ should be carefully selected. GNN achieves much better performance when $\lambda = 1$. Because smaller λ leads to most input and output neuron pairs with large connecting density. Then the background cannot be well restrained. While with larger λ, the architecture is the incapability to well represent the digits. Therefore, when $\lambda = 10$ both digits with pure and noise backgrounds cannot be well recognized.

V. Conclusion

In this article, we propose a novel compact group architecture for neural network (GNN) and its corresponding connecting and parameter learning methods. This new architecture is different from the multilayer networks which are composed of regular layers. The new architecture is composed of input, output layers, and a group of neurons. There are no connections between neurons in the input or output layers but all the other neurons are connected freely. Therefore, it is crucial to determine the connections between the neurons since it is unpractical to connect all the neurons. However, the objective of traditional architecture learning methods is time demanding due to the involvement of connecting weights and biases. With such a large-scale searching space, it is even impossible to obtain a reasonable architecture for the group network. As a consequence, we propose a novel architecture learning model based on the principle that higher relevance between input and output neurons desires larger connecting density. A probabilistic model is established to make the architecture well capture the distribution of data. Moreover, an efficient objective function is derived from the model and an improved binary particle swarm optimization algorithm is used to optimize the objective. After assigning the connections, it is also important to train the connecting weights and biases. However, due to the special irregular architecture, the parameters cannot be well trained by directly using the BP algorithm. Consequently, we establish a probabilistic model and corresponding optimization method. After training, we find the proposed architecture is robust to background variance and corruptions. Experiments on optimization, classification, and hyperparameters demonstrate the superiority of the proposed architecture.

However, the defects are also obvious. Even though GNN outperforms large-scale architectures on the reconstructed dataset, due to the irregular architecture, the training process is not accelerated via GPU parallel computation which leads to the large computational burden when it is applied to large-scale datasets. The experiments show the potential of large-scale GNN. Considering the prospect of GNN, in future work, we will accelerate the learning process via GPU parallel computation (directly code based on CUDA[2] and CUDNN[3] instead of existing platforms) and use more efficient evolving strategies, such as knowledge transfer [59].

[1]https://tensorflow.google.cn/

[2]https://developer.nvidia.com/cuda-downloads

[3]https://developer.nvidia.com/cudnn

REFERENCES

[1] D. Livingstone, *Artificial Neural Networks: Methods and Applications*. Totowa, NJ, USA: Humana Press, 2008.

[2] Y. Pu, Z. Yi, and J. Zhou, "Fractional Hopfield neural networks: Fractional dynamic associative recurrent neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 10, pp. 2319–2333, Oct. 2017.

[3] J. Liu, M. Gong, and H. He, "Deep associative neural network for associative memory based on unsupervised representation learning," *Neural Netw.*, vol. 113, pp. 41–53, May 2019.

[4] C. Li, X. Yu, T. Huang, G. Chen, and X. He, "A generalized Hopfield network for nonsmooth constrained convex optimization: Lie derivative approach," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 2, pp. 308–321, Feb. 2016.

[5] A. Esteva *et al.*, "Dermatologist-level classification of skin cancer with deep neural networks," *Nature*, vol. 542, no. 7639, pp. 115–118, 2017.

[6] R. Hu, S. Wen, Z. Zeng, and T. Huang, "A short-term power load forecasting model based on the generalized regression neural network with decreasing step fruit fly optimization algorithm," *Neurocomputing*, vol. 221, pp. 24–31, Jan. 2017.

[7] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[8] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[10] Y. Wei *et al.*, "Cross-modal retrieval with CNN visual features: A new baseline," *IEEE Trans. Cybern.*, vol. 47, no. 2, pp. 449–460, Feb. 2017.

[11] Y. Du, W. Wang, and L. Wang, "Hierarchical recurrent neural network for skeleton based action recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Boston, MA, USA, 2015, pp. 1110–1118.

[12] J.-T. Chien and Y.-C. Ku, "Bayesian recurrent neural network for language modeling," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 2, pp. 361–374, Feb. 2016.

[13] X. Chen *et al.*, "Efficient training and evaluation of recurrent neural network language models for automatic speech recognition," *IEEE/ACM Trans. Audio, Speech, Language Process.*, vol. 24, no. 11, pp. 2146–2157, Nov. 2016.

[14] T. Zhang, W. Zheng, Z. Cui, Y. Zong, and Y. Li, "Spatial–temporal recurrent neural network for emotion recognition," *IEEE Trans. Cybern.*, vol. 49, no. 3, pp. 839–847, Mar. 2019.

[15] G. F. Miller, P. M. Todd, and S. U. Hegde, "Designing neural networks using genetic algorithms," in *Proc. Int. Conf. Genet. Algorithms*, 1989, pp. 379–384.

[16] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evol. Comput.*, vol. 10, no. 2, pp. 99–127, 2002.

[17] P. Ballester and R. M. Araujo, "On the performance of GoogLeNet and alexnet applied to sketches," in *Proc. AAAI Conf. Artif. Intell.*, 2016, pp. 1124–1128.

[18] E. Real *et al.*, "Large-scale evolution of image classifiers," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 2902–2911.

[19] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, no. 1, pp. 281–305, 2012.

[20] R. Jozefowicz, W. Zaremba, and I. Sutskever, "An empirical exploration of recurrent network architectures," in *Proc. 32nd Int. Conf. Mach. Learn.*, 2015, pp. 2342–2350.

[21] Y. Sun, X. Wang, and X. Tang, "Sparsifying neural network connections for face recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Las Vegas, NV, USA, 2016, pp. 4856–4864.

[22] L. Jia, M. Gong, Q. Miao, X. Wang, and L. Hao, "Structure learning for deep neural networks based on multiobjective optimization," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 6, pp. 2450–2463, Jun. 2018.

[23] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Training very deep networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 2377–2385.

[24] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Las Vegas, NV, USA, 2016, pp. 770–778.

[25] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Cision Pattern Recognit.*, Boston, MA, USA, 2015, pp. 1–9.

[26] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu, "Deeply-supervised nets," in *Proc. 18th Int. Conf. Artif. Intell. Stat.*, 2015, pp. 562–570.

[27] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Salt Lake City, UT, USA, 2018, pp. 8697–8710.

[28] C. Liu *et al.*, "Progressive neural architecture search," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 19–34.

[29] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proc. AAAI Conf. Artif. Intell.*, 2019, pp. 4780–4789.

[30] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *Proc. Int. Conf. Learn. Represent.*, 2019, p. 13. [Online]. Available: https://openreview.net/group?id=ICLR.cc/2019/Conference#accepted-oral-papers

[31] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, "A learning algorithm for Boltzmann machines," *Cogn. Sci.*, vol. 9, no. 1, pp. 147–169, 1985.

[32] M. Liang, X. Hu, and B. Zhang, "Convolutional neural networks with intra-layer recurrent connections for scene labeling," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 937–945.

[33] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, "Efficient architecture search by network transformation," in *Proc. AAAI Conf. Artif. Intell.*, 2018, pp. 2787–2794.

[34] K. Kandasamy, W. Neiswanger, J. Schneider, B. Poczos, and E. P. Xing, "Neural architecture search with Bayesian optimisation and optimal transport," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 2016–2025.

[35] A. A. Esmin, R. A. Coelho, and S. Matwin, "A review on particle swarm optimization algorithm and its variants to clustering high-dimensional data," *Artif. Intell. Rev.*, vol. 44, no. 1, pp. 23–45, 2015.

[36] Y. L. Cun, J. S. Denker, and S. A. Solla, *Optimal Brain Damage*. San Francisco, CA, USA: Morgan Kaufmann, 1990, pp. 598–605.

[37] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2015, pp. 1135–1143.

[38] X. Yao, "Evolving artificial neural networks," *Proc. IEEE*, vol. 87, no. 9, pp. 1423–1447, Sep. 1999.

[39] F. H. F. Leung, H. K. Lam, S. H. Ling, and P. K. S. Tam, "Tuning of the structure and parameters of a neural network using an improved genetic algorithm," *IEEE Trans. Neural Netw.*, vol. 14, no. 1, pp. 79–88, Jan. 2003.

[40] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 8, pp. 1798–1828, Aug. 2013.

[41] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, no. 7, pp. 1527–1554, 2006.

[42] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[43] A. Fischer and C. Igel, "An introduction to restricted Boltzmann machines," in *Proc. Iberoamerican Congr. Pattern Recognit.*, 2012, pp. 14–36.

[44] G. B. Huang, H. Lee, and E. Learned-Miller, "Learning hierarchical representations for face verification with convolutional deep belief networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Providence, RI, USA, 2012, pp. 2518–2525.

[45] N.-N. Ji, J.-S. Zhang, and C.-X. Zhang, "A sparse-response deep belief network based on rate distortion theory," *Pattern Recognit.*, vol. 47, no. 9, pp. 3179–3191, 2014.

[46] M. Gong, J. Liu, H. Li, Q. Cai, and L. Su, "A multiobjective sparse feature learning model for deep neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 12, pp. 3263–3277, Dec. 2015.

[47] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural Comput.*, vol. 14, no. 8, pp. 1771–1800, 2002.

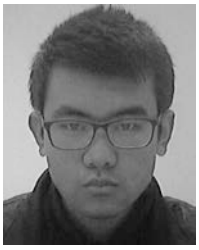
[48] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. IEEE Int. Conf. Neural Netw.*, Perth, WA, Australia, 1995, pp. 1942–1948.

[49] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *Proc. IEEE Int. Conf. Evol. Comput. World Congr. Comput. Intell.*, Anchorage, AK, USA, 1998, pp. 69–73.

[50] M.-P. Song and G.-C. Gu, "Research on particle swarm optimization: A review," in *Proc. IEEE Int. Conf. Mach. Learn. Cybern.*, vol. 4. Shanghai, China, 2004, pp. 2236–2241.

[51] K. Mistry, L. Zhang, S. C. Neoh, C. P. Lim, and B. Fielding, "A micro-GA embedded PSO feature selection approach to intelligent

facial emotion recognition," *IEEE Trans. Cybern.*, vol. 47, no. 6, pp. 1496–1509, Jun. 2017.
[52] M. Gong, Q. Cai, X. Chen, and L. Ma, "Complex network clustering by multiobjective discrete particle swarm optimization based on decomposition," *IEEE Trans. Evol. Comput.*, vol. 18, no. 1, pp. 82–97, Feb. 2014.
[53] L. Li, X. Yao, R. Stolkin, M. Gong, and S. He, "An evolutionary multiobjective approach to sparse reconstruction," *IEEE Trans. Evol. Comput.*, vol. 18, no. 6, pp. 827–845, Dec. 2014.
[54] G. Bouma, "Normalized (pointwise) mutual information in collocation extraction," in *Proc. Int. Conf. German Soc. Comput. Linguistics (GSCL)*, 2009, pp. 31–40.
[55] K. W. Church and P. Hanks, "Word association norms, mutual information, and lexicography," *Comput. Linguistics*, vol. 16, no. 1, pp. 22–29, 1990.
[56] X. Yu *et al.*, "Set-based discrete particle swarm optimization based on decomposition for permutation-based multiobjective combinatorial optimization problems," *IEEE Trans. Cybern.*, vol. 48, no. 7, pp. 2139–2153, Jul. 2018.
[57] B. Biggio and F. Roli, "Wild patterns: Ten years after the rise of adversarial machine learning," *Pattern Recognit.*, vol. 84, pp. 317–331, Dec. 2018.
[58] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P. A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *J. Mach. Learn. Res.*, vol. 11, no. 12, pp. 3371–3408, 2010.
[59] K. K. Bali, Y. Ong, A. Gupta, and P. S. Tan, "Multifactorial evolutionary algorithm with online transfer parameter estimation: MFEA-II," *IEEE Trans. Evol. Comput.*, vol. 24, no. 1, pp. 69–83, Feb. 2020.



Jia Liu (Member, IEEE) received the B.S. and Ph.D. degrees in electronic engineering from Xidian University, Xi'an, China, in 2013 and 2018, respectively.

He is currently an Associate Professor with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China. His current research interests include computational intelligence and image understanding.

Maoguo Gong (Senior Member, IEEE) received the B.S. degree (First Class Hons.) in electronic engineering and the Ph.D. degree in electronic science and technology from Xidian University, Xi'an, China, in 2003 and 2009, respectively.

Since 2006, he has been a Teacher with Xidian University. From 2008 to 2010, he was promoted as an Associate Professor and as a Full Professor, respectively, both with exceptional admission. His research interests are in the areas of computational intelligence with applications to optimization, learning, data mining, and image understanding.

Dr. Gong received the Prestigious National Program for the support of Top-Notch Young Professionals from the Central Organization Department of China, the Excellent Young Scientist Foundation from the National Natural Science Foundation of China, and the New Century Excellent Talent in University from the Ministry of Education of China. He is an Associate Editor of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION and the IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS.

Liang Xiao (Member, IEEE) received the B.S. degree in applied mathematics and the Ph.D. degree in computer science from the Nanjing University of Science and Technology (NJUST), Nanjing, China, in 1999 and 2004, respectively.

From 2006 to 2008, he was a Postdoctoral Research Fellow with the Pattern Recognition Laboratory, NJUST. From 2009 to 2010, he was a Postdoctoral Fellow with Rensselaer Polytechnic Institute, Troy, NY, USA. Since 2013, he has been the Deputy Director of the Jiangsu Key Laboratory of Spectral Imaging Intelligent Perception, NJUST. Since 2014, he has been the Vice-Director of the Key Laboratory of Intelligent Perception and Systems for High-Dimensional Information of Ministry of Education, NJUST, where he is currently a Professor with the School of Computer Science and Engineering. His fields of interests include remote sensing image processing, image modeling, computer vision, machine learning, and pattern recognition.

Wenhua Zhang received the B.S. degree in communication engineering from the North University of China, Taiyuan, China, in 2015. She is currently pursuing the Ph.D. degree in artificial intelligence from Xidian University, Xi'an, China.

Her research interests include machine learning and image processing.

Fang Liu (Member, IEEE) received the B.S. degree in information and computing science from Henan University, Kaifeng, China, in 2012, and the Ph.D. degree in intelligent information processing from Xidian University, Xi'an, China, in 2018.

She is a Lecturer with the Nanjing University of Science and Technology, Nanjing, China. Her research interests include deep learning, object detection, polarimetric SAR image classification, and change detection.