Nucleus Neural Network: A Data-driven Self-organized Architecture

Jia Liu
School of Computer Science and Engineering
Nanjing University of Science and Technology
Nanjing, China
omegaliuj@gmail.com

Haibo He
Department of Electrical, Computer, and Biomedical Engineering
University of Rhode Island
Kingston, RI, USA
haibohe@uri.edu

Maoguo Gong
School of Electronic Engineering
Xidian University
Xi'an, China
gong@ieee.org

Wenhua Zhang
School of Artificial Intelligence
Xidian University
Xi'an, China
zhangwenhua_nuc@163.com

***Abstract*—In this paper, inspired from the nuclei in brain, we propose a nucleus neural network (NNN) and corresponding connecting architecture learning method. In a nucleus, the neurons are not assigned as regular layers, i.e., a neuron may connect to any neurons in the nucleus and the connections are self-organized according to data distribution. This type of architecture gets rid of layer limitation and makes full use of processing capability of each neuron. It is crucial to assign connections between all the neuron pairs. To address the time demanding of objectives in traditional architecture learning methods, we propose an efficient architecture learning model for the nucleus based on the principle that more relevant input and output neuron pair deserves higher connecting density. The new objective measures the information flow through the network architecture without involvement of weights and biases which greatly reduces the computational complexity. We find that this novel architecture is robust to irrelevant components in test data. So we reconstruct a new dataset based on the MNIST dataset where the types of digital backgrounds in training and test sets are different. The new dataset is a great challenge for learners because training data and test data are not only independent but also follow different distributions. Experiments demonstrate that NNN achieves significant improvement over architectures with regular layers on the reconstructed dataset.**

***Index Terms*—neural network, brain-inspired architecture, architecture learning, image classification**

I. Introduction

Mimicking the architecture and mechanism of nature creatures' learning for creating efficient and robust learners is one of the challenges in artificial intelligence research [1]. Artificial neural networks (ANNs) which are inspired form nature neural systems have excellent learning capability in many applications. Specially, neural networks with deep hierarchical layers have achieved many breakthroughs in machine learning, which leads to the great research interests in deep learning [2]. Most existing ANN models are based on the classic hierarchical architectures which are composed of regular layers with no connections between neurons in the same layer or nonadjacent layers as shown in Fig. 1(a), especially those for supervised learning. The learning architecture of human brain is also hierarchical but with multiple nuclei with inner connections, such as lateral geniculate nucleus. The connections in each nucleus is much more complex than that of ANN which simply omits connections in the same layer. Residual network [3] successfully introduces additional connections between nonadjacent layers which leads to deeper network architecture and larger learning capability.

In this paper, inspired from the nuclei in brain where architectures are apparently irregular but subtly organized, we attempt to fully relax the layer limitation and evolve the network architecture freely given a set of neurons. A neuron is able to connect to any neurons as shown in Fig. 1(b) and the connections are organized according to data and tasks. Then a compact network that makes full use of each neuron is constructed and we call it nucleus neural network (NNN). But since we focus on classification problem in this paper, the only limitation is that there are no feed-back connections. To launch NNN, it is crucial to determine the connections between numerous neuron pairs which is a great challenge.

Architecture optimization for ANN has been a research interest for decades with various optimization methods [4]–[8]. The architecture parameters are searched and determined by an objective function. They follow traditional hierarchical

This work was supported in part by the National Nature Science Foundations of China (Grant No. 61906093) and in part by the Nature Science Foundation of Jiangsu Province, China (Grant No. BK20190451).

This work was conducted when Jia Liu was a visiting scholar at the Department of Electrical, Computer, and Biomedical Engineering, University of Rhode Island, Kingston, RI 02881, USA.

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

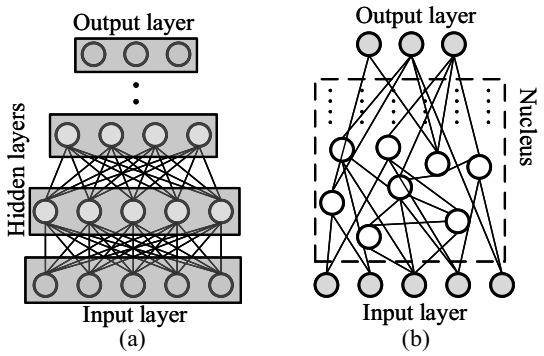

Fig. 1. Difference between multi-layer network and the proposed architecture. (a) Multi-layer network (b) Architecture of NNN

architectures and find the optimal depth, layer width, kernel size (convolutional neural network), and connections between each two layers. For example, NEAT algorithm used in [4] evolves the nodes, connections and weights for a network. In [7] and [8], the sparse connecting structure is focused and redundant connections are removed from a hierarchical compact network. In [5], a complex architecture is evolved with involvement of not only multi-layer configurations but also skip connections. Nowadays, there are increasing interests in architecture learning based on network blocks or cells [6], [9]–[11]. A block or cell is a directed acyclic graph consisting of an ordered sequence of nodes [6].

However, with numerous free binary variables, searching the optimal connecting assignment of NNN is of great difficulty. Most of architecture learning methods search the optimal architecture based on task itself as the objective, for instance, test error for classification. But to compute the objective, it is necessary to train the weights and biases. Therefore, it is computationally demanding even though the searching space in them is not that large. For example, 1800 GPU days of reinforcement learning is required to search a superior architecture in [9] and 3150 GPU days of evolution in [11]. There are also many methods proposed to speed up the learning process, such as weight sharing [12], Bayesian optimisation [13], and differentiable network architecture search [6]. However, most of them focus on reducing the searching space or increasing searching efficiency. In NNN, the optimizer should search in a binary space with over ten thousand dimensions which is much larger than that of most existing architecture learning problems. Moreover, it is difficult to train the weights and biases in NNN by existing parallel devices. It will take even years to obtain a reasonable connecting architecture for NNN via traditional architecture learning methods. As a consequence, we propose a substitutable but more efficient objective function with respect to the architecture. We define a connecting density between a neuron and an input neuron. Then if an input neuron is more relevant to an output neuron according to the mutual information computed from the training data, the connecting density between them should be larger. The objective function is defined based on this principle via the modeling approach of products of experts (PoE) [14] in order to capture the distribution of observed data directly by the architecture without the involvement of weights and biases. This will greatly reduce the computational complexity of objective function and an architecture which is well adapted to input distribution can be learned. A binary particle swarm optimizer (BPSO) algorithm [15] is utilized to optimize the objective function. After the architecture is optimized, the connecting weights and biases are then learned by a novel error driven probabilistic model to well represent the input data.

After training, we find an interesting phenomenon of NNN. Since the architecture is evolved based on the input and output relevance, NNN gives high response to the data where patterns of irrelevant components are never seen by NNN during training. This means that NNN is robust to background changes when trained only by images with pure background in image classification. To highlight the superiority of NNN, we construct a new dataset based on the MNIST digit where the background types of training and test data are different. The training and test sets in the dataset not only are independent but also follow different distributions. The potential application is significant where the learner is expected to learn to infer new patterns like human learning. The new dataset is a great challenge for traditional learners since most of them assume that training and test data are independent but follow identical distribution or follow different distributions but should be not independent.

II. NNN's Architecture Learning

NNN is composed of input layer, output layer and a nucleus as shown in Fig. 1(b). In the nucleus, there are no regular layers and a neuron can connect to any neurons. But for time independent data, feedback connections are unallowed. The goal of architecture learning is to determine the connections between those neurons in the nucleus. It is a popular way to evolve network architectures via an objective function [4], [5]. In order to reduce the computational complexity, we define an objective function via directly modeling the connections without considering the connecting weights and biases. It is established based on the principle that higher relevance leads to higher connecting density so as to learn the relationship between input and output nodes. First the connecting density is defined.

A. Connecting Density

To evaluate the architecture, we should consider not only depth but also width of the information flow between an input and an output neurons. Therefore, we define a propagated connecting density with the initial density between an input neuron and itself being 1. Then the density $\mathcal{D}_{ij}(\varphi)$ between a neuron i in the nucleus and an input neuron j can be computed as follows:

$$\mathcal{D}_{ij}(\varphi) = \frac{\sum_{k\in\Omega_i} \mathcal{D}_{kj}(\varphi)}{N_{\Omega_i}} \tag{1}$$

where φ denotes the architecture of the network which is a binary matrix indicating the connecting status between each

pair of neurons with 1 denoting connected and 0 unconnected. Ω_i denotes the set of lower neurons that directly connect to i and N_{Ω_i} is the number of neurons in the set. It means the average connecting density from the lower neural nodes and can be simply read as the residue of input information. The connecting density starts from the input neurons, propagates through the connections, and finally that between input and output neurons is obtained. The connecting density represents a information path between an input and an output neurons. From Eq. (1), higher density means wider and shallower path with more processing nodes and less information degression. If the path is deeper, the information of this input neuron will be combined with information from other neurons. Therefore, the density will be lower. The information path should well adapt to the relevance between input and output neurons. Based on this principle, we then construct the model of each output neuron.

B. Modeling Single Output Neuron

Here we utilize the normalized pointwise mutual information (NPMI) [16] to represent the relevance between input and output neurons for each data. Suppose an input neuron j and an output neuron i, given a pair of input and output neurons' status $\{x_j, y_i\}$ with input vector x and referenced output vector y, the NPMI between them $\mathbb{N}(x_j; y_i)$ can be computed by:

$$\mathbb{N}(x_j; y_i) = \frac{\mathbb{I}(x_j; y_i)}{\mathbb{H}(x_j, y_i)} = \log \frac{P(x_j, y_i)}{P(x_j)P(y_i)} / \log \frac{1}{P(x_j, y_i)} \tag{2}$$

where $\mathbb{I}$ denotes the pointwise mutual information (PMI) and $\mathbb{H}$ denotes the self-information. $P(\cdot)$ denotes the probability which is counted from the dataset. The NPMI normalizes PMI into an interval of $[-1, 1]$ with -1 denoting the two values will never occur together, 0 independence, and 1 complete co-occurrence. Because the probability of a negative valued $\mathbb{N}(x_j; y_i)$ is very small and can be omitted, we define the relevance between x_j and y_i as $\mathbb{R}(x_j, y_i) = \max(\mathbb{N}(x_j; y_i), 0)$ [17] to avoid unexpected overflow.

The connecting density should follow the relevance for a better information flow of input and output neurons. Therefore, the possibility that an output neuron and its corresponding information path can well represent the input vector is measured by the cosine distance between relevance and connecting density:

$$p_i(x, y_i; \varphi) = \cos(\mathcal{D}_i(\varphi), \mathbb{R}(y_i)) \tag{3}$$

where $\mathcal{D}_i(\varphi)$ and $\mathbb{R}(y_i)$ respectively denotes the connecting density and relevance vectors with n dimensions, i.e., $\mathcal{D}_i(\varphi) = [\mathcal{D}_{i1}(\varphi), \mathcal{D}_{i2}(\varphi), ..., \mathcal{D}_{in}(\varphi)]^T$ and $\mathbb{R}(y_i) = [\mathbb{R}(x_1, y_i), \mathbb{R}(x_2, y_i), ..., \mathbb{R}(x_n, y_i)]^T$ where n is the number of input neurons. For an architecture that best captures the relevance between an output neuron and input neurons, the possibility, i.e., cosine distance will be close to 1. While if there is no enough connecting density for transforming the information, the possibility will be close to 0.

C. Modeling the Whole Network

The whole network can be modeled by combining the models of output neurons. For better modeling the high-dimensional space of input data, in this paper, we combine the neurons by multiplying them together and renormalizing as in PoE. PoE has the advantage that they can produce much sharper distributions [14]. In NNN, the whole network model is formulated as follows:

$$p(x, y; \varphi) = \frac{\prod_i p_i(x, y_i; \varphi)}{\sum_{(\chi,\gamma)} \prod_i p_i(\chi, \gamma_i; \varphi)} \tag{4}$$

where (χ, γ) denotes one of all the possible data in the whole data space which (x, y) belongs to. The denominator is the renormalization term used to guarantee that the probability sum over the whole data space is 1. The model follows PoE where simple models $p_i(x, y_i; \varphi)$ are multiplied and renormalized to construct a complex model $p(x, y; \varphi)$. Optimizing this model means to increasing the representation capability of observed data while decreasing that of all the other data. Then the architecture φ will well capture the distribution of observed data.

However, optimizing this model is of great difficulty due to the unreachable fantasy data in the whole data space. In PoE [14], the model is optimized by using the gradient of a log-likelihood. Gibbs sampling is used to estimate the gradient expectation of the whole data space. However, in this model, the decision variable is the architecture φ which is binary. Such a problem is suitable to be solved by evolutionary algorithms or population based methods [8], [18]. A computable objective function is necessary to be derived.

D. Objective Function

Fortunately, the value of the denominator in Eq. (4) depends only on the architecture φ. Optimizing this model amounts to maximizing the numerator wile minimizing the denominator. Therefore, the denominator can be replaced by a computable function with respect to φ and taken as a new term. Here we use the L2-norm of connecting density vector of each output neuron:

$$\rho(\varphi) = \sum_i \|\mathcal{D}_i(\varphi)\|_2 \tag{5}$$

The denominator in Eq. (4) aims to represent the whole data space by connecting density. Since the whole data space contains all the possible cases, connecting density controls the representation ability of the architecture for all the possible data. Therefore, minimizing $\rho(\varphi)$ will decrease the denominator in Eq. (4).

Since there is no evidence to suggest the linear relationship between $\rho(\varphi)$ and the denominator in Eq. (4), the new objective function is reconstructed by adding the two terms, i.e., numerator in Eq. (4) and Eq. (5):

$$\max J(\varphi) = \sum_{(x,y)\in\mathbf{D}} \prod_i p_i(x, y_i; \varphi) - \lambda\rho(\varphi) \tag{6}$$

where $\mathbf{D}$ is the training set and λ is a user defined parameter that controls the importance of the two terms. In this objective

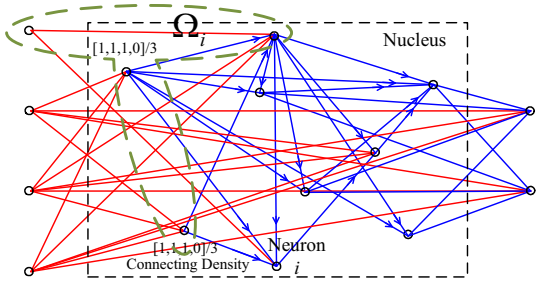

Fig. 2. A toy example of learned architecture from a simulated dataset. Red lines represents the connections from input neurons.

function, the denominator in Eq. (4) is replaced by a simplified function but can achieve similar effectiveness to that of Eq. (4). Maximizing this function amounts to assigning more probability to the observed data (increasing numerator) while restraining the probability of all the other data (decreasing denominator). Then the architecture will capture the distribution of observed data and well represent the input and output relationship of the data in dataset $\mathbf{D}$. Finally we utilize a BPSO algorithm [15] which is a popular optimization method in wide applications [19], [20] to optimize the objective function.

E. Toy Example of Learned Architecture

To better explain the whole story, we provide a toy example of the learned architecture from a simulated dataset. The dataset is constructed as a classification problem. There are 4 attributes in the input data with the first 3 attributes following the joint Gaussian distribution and the last attribute follows the uniform distribution. In the learning process, we set 9 neurons in the nucleus and Fig. 2 shows the learned architecture. It is intuitive that the first 3 attributes contribute more to the classification, and therefore more connections are connected to them. They also directly connect to the output neurons. The last attribute has no contributes to the output neurons, the depth of it is over 3 layers. The learned architecture follows the principle in appearance which demonstrates the effect of the objective function and learning method.

The connecting density of some neurons and Ω_i of a neuron i are shown in Fig. 2. All the initial connecting densities of input neurons are 1. As shown in the figure, the connecting density between a neuron and input neurons is represented as a vector. Then they propagate through the whole network and $\mathcal{D}_i(\varphi)$ for each output neuron is obtained.

III. NNN's Parameter Learning

After the architecture is learned, the network parameters, i.e., the weights on each connection and bias on each neuron, have also to be learned to achieve the function of classification or other tasks. Back-propagation [21] is the common method that trains multi-layer neural networks. In multi-layer network, the errors can be back-propagated layer by layer and then the gradient of parameters can be obtained by the back-propagated errors. However, in NNN, there are no regular layers which leads to irregular depth. Even though the errors can also back-propagate from output layer, the different depth leads to different error decay. Then, the difference leads to uneven impact of reference output on input neurons. Therefore, some weights may not be well trained by directly using back-propagation.

A. Probabilistic Model

In the parameter learning process, we should not only consider the errors in the output layer, but also consider the representation capability for input data. As a consequence, similar to restricted Boltzmann machine (RBM) [24], we construct a probabilistic model with the errors in the output layer as the energy:

$$p(x|y;\theta) = \frac{\exp(-E(x,y;\theta))}{\sum_{\chi}\exp(-E(\chi,y;\theta))} \tag{7}$$

where $E(x,y;\theta)$ is the square error between output of the network $f_\theta(x)$ and reference output y, i.e., $E(x;\theta) = \|y - f_\theta(x)\|_2^2$ with θ being the network parameter set. Similar to the architecture model, χ denotes all the possible data in the whole data space. This model denotes a parameterized probability density that captures the distribution of observed data given a reference output.

B. Optimization

This model can be solved via the gradient of log-likelihood:

$$\begin{aligned}\triangle\theta =& \frac{\partial \log p(x|y;\theta)}{\partial\theta}\\ =& -\frac{\partial E(x)}{\partial\theta} + \sum_{\chi}[\frac{\exp(-E(\chi))}{\sum_{\chi}\exp(-E(\chi))}\frac{\partial E(\chi)}{\partial\theta}]\\ =& -\frac{\partial E(x)}{\partial\theta} + \mathbb{E}_{p(\chi|y;\theta)}\frac{\partial E(\chi)}{\partial\theta}\end{aligned} \tag{8}$$

where $\mathbb{E}$ denotes the expectation. The first term in Eq. (8) is easy to compute with the back-propagation algorithm. For the second term, we use the data $\hat{x}$ sampled from the distribution $p(x|y;\theta)$ to estimate the expectation of gradients. However, different from the simple architecture of RBM [24], for NNN, it is difficult and even impossible to compute the probability of each input neuron's statues. Therefore, we propose to solve the problem in reverse.

The sampled data is more likely to locate in the high density region of the distribution. Therefore, we can drive a sample to move to the high density region by gradient, i.e., $\triangle\hat{x} = \partial \log p(\hat{x}|y;\theta)/\partial\hat{x}$. Fortunately, during the sampling process, the network parameter set θ is fixed which leads to the dominator in Eq. (7) being a constant. Then the updating gradient is computed as follows:

$$\triangle\hat{x} = \frac{\partial \log\exp(-E(\hat{x}))}{\partial\hat{x}} = -\frac{\partial E(\hat{x})}{\partial\hat{x}} \tag{9}$$

The gradient is computed by the error back-propagated from the output layer. Then a random generated data can be driven to the high density region by using gradient descent. After several iterations, the sampled data $\hat{x}$ is obtained which is used

to estimate the gradient expectation of fantasy data. Then the gradient of network parameters is computed by:

$$\triangle\theta = -\frac{\partial E(x)}{\partial \theta} + \frac{\partial E(\hat{x})}{\partial \theta} \tag{10}$$

The whole parameter learning process is summarised in Algorithm 1.

Algorithm 1 Parameter Learning Process of NNN

Initialization: initialize the connecting weights and biases randomly.

Repeat:

FOR each batch in the dataset:

Sampling: Obtain the sampled data batch $\hat{x}$ by gradient descent with the gradient in Eq. (9) from random sample batches.

Updating: Compute the gradients $\frac{\partial E(x)}{\partial \theta}$ and $\frac{\partial E(\hat{x})}{\partial \theta}$ and generate the updating gradient of θ as Eq. (10). Update θ.

end FOR

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

This new model and learning process can relieve the uneven impact of output neurons. The updating gradient is the difference between gradients of observed data and sampled data. For a deep path, there is more error decay from the output layer. Then the components of deep paths in a sampled data is more random. Thus the gradient difference between observed data and sampled data will be larger. While for a shallow path, the gradient difference will be smaller. Then the uneven impact will be offset by the uneven difference.

C. Toy Example and Potential Application

Similarly, we provide a toy example to explain the observed distribution and sampled distribution by the network. We train the network architecture and parameters by a simulated dataset. There are 3 attributes in each data. The first two attributes follow the joint Gaussian distribution and the last attribute is a constant. The distribution of the simulated data is shown by the blue points in Fig. 3. There are two classes and different classes follow Gaussian distribution with different parameters.

In Fig. 3, the distribution sponsored by the network is shown by the red points. With the randomly initialized network, the sampled data $\hat{x}$ are randomly distributed. During the learning process, the network begins to capture the distribution of observed data. After learning, the learned network can well capture the distribution of the first two attributes in the observed data. Because the first two attributes play important roles in decision, the network assigns more connecting density for them. Then the network can learn to follow the distribution of them by connecting weights and biases. The last dimension has no contributions to classification and thus that of the sampled data is approximately randomly distributed after learning.

This novel phenomenon demonstrates NNN is robust to irrelevant components, i.e., the background. The sampled distribution represents the energy driven response of the network to different data. That means, the samples at the higher density region have lower energy, i.e., output difference from reference. Therefore, NNN assigns high response to the data indicated by red points in Fig. 3. In practical, there are more complex data and some attributes are not always background. But since the connecting density is also not binary, with the generative model derived objective function, the architecture and the learned network parameters are expected to well capture the distribution of input data. As a consequence, in this paper, we construct a novel dataset to test NNN.

IV. Reconstructed Dataset and Experiments

In supervised learning, it is usually supposed that the training dataset and test dataset are independent and identically distributed. Therefore, large scale labeled dataset coving most cases in practical applications is necessary to train a robust model. Transfer learning [22] poses this problem and solves it by transferring the knowledge from labeled data in source domain to unlabeled data in target domain. But transfer learning methods suppose that source domain and target domain are not independent. They estimate the distribution bias with the help of target data. From the toy example, NNN shows significant potential applications. Therefore a new dataset is constructed based on the MNIST dataset to verify the superiority of NNN over traditional learners.

A. Dataset

In the new dataset, we train the learners by the training set in the MNIST dataset[1] and apply the trained learner to digits with various backgrounds[2]. Fig. 4 shows some images in the training set and test set. MNIST dataset is a collection of handwritten digits (0-9). However, in practice, there are also digits on various backgrounds. Then some derivatives of MNIST dataset are created, including MNIST digits with random image background (bg-img) and random noise background (bg-rand) [23] which will be used as testing data in the new dataset. Traditional learners learn to recognize digits on various backgrounds relying on a large scale labeled dataset containing most cases (supervised learning) or the test cases are available (transfer learning) during learning process. But a smarter learner is expected to recognize digits on various backgrounds after learned only from the digits on the blackboard. The inference capability seems more important for a learner. Therefore, the new problem is significant in real applications but a great challenge for most existing learners.

In the experiments, 60000 training digits in the MNIST dataset are used to train the classifiers. Then 50000 digits with bg-img and 50000 digits with bg-rand are used to test the classifiers. It deserves to be noted that training and test processes are independent, i.e., the digits in test set have no influence on the training process. To my knowledge, few

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

[2] available at http://www.iro.umontreal.ca/~lisa/icml2007

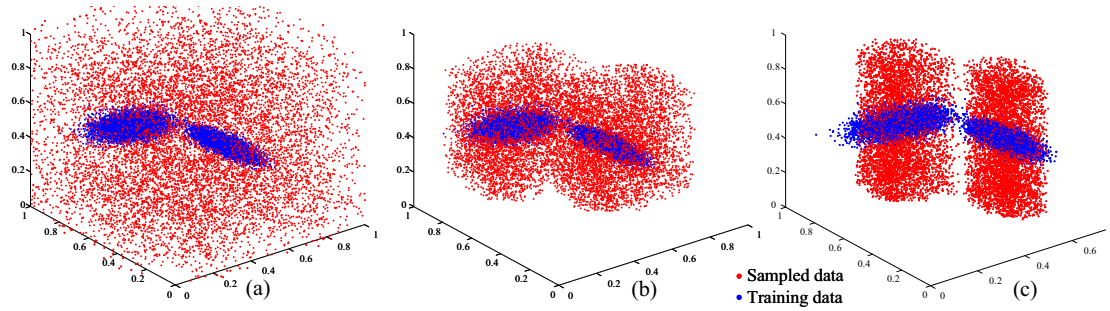

Fig. 3. Distribution of observed data and sampled data in different learning stages. The blue points denote the observed data and red ones are sampled data. (a) Sampled distribution by randomly initialized network. (b) Sampled distribution by the network during the learning process. (c) Sampled distribution by the network after training.

Fig. 4. Some images in training and test datasets. (a) Training set with pure background. (b) Test set with random image patches as background. (c) Test set with random noise as background.

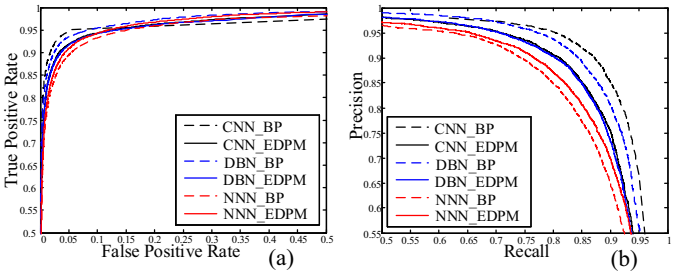

Fig. 5. ROC and PR curves of the test result on digits in MNIST test set. (a) ROC curves (b) PR curves

methods could well solve this problem. We set the number of neurons in the nucleus to be 200 and $\lambda = 1$. We compare the proposed NNN with architectures of DBN and CNN. There are many architecture learning methods as introduced above. But they follow the multi-layer architectures including DBN and CNN. Moreover, they optimize the architectures with the objective of minimizing test error which is estimated by the training data. The test set follows different distribution from training set which leads to the wrong estimation of test error in traditional architecture learning methods. They cannot learn additional gains except for a lower error in traditional machine learning problems. In this paper, the architecture is totally new without baseline architectures. Therefore, other architecture learning methods are not compared because they will achieve similar performance to their baselines due to the same objective of them. We set the scale of DBN as "784-500-300-10" [24] and CNN as LeNet in [25]. In BPSO, we set 100000 as the maximum number of generations. All the three architectures are trained by both back-propagation (BP) and the proposed error driven probabilistic model (EDPM). The experimental results are evaluated by receiver operating characteristic (ROC) curve, precision-recall (PR) curve, area under ROC curve (AUC), average precision (AP), and the classification accuracy (CA). Since the irregular architecture is difficult to be accelerated GPU parallel computation, all the algorithms are conducted by visual studio 2017 on CPU (Intel I7 3.7GHz) and the code of NNN is available at GitHub[3].

[3]https://github.com/liusiqinqinqin/nucleus-neural-network

B. Experimental Results

First we demonstrate the learning capability of NNN on traditional classification problem where both training and test sets are from MNIST dataset. The ROC and PR curves of each classifier is shown in Fig. 5. It can be found that NNN cannot outperform traditional classifiers in this problem. The values of AUC, AP, and CA are listed in Table I (test set of MNIST). CNN achieves the best performance from the criteria due to its special architecture. Although NNN achieves the lowest classification accuracy, the accuracy is close to that of DBN. It deserves to be noted that there are only 200 neurons in the nucleus and 146686 connections in the whole network after training. In DBN, there are 800 hidden units and 545000 connections and much more in CNN. Therefore, NNN achieves equivalent performance to that of DBN with less processing units and parameters. This demonstrates that NNN is a compact architecture and can make full use of each neuron. This is due to that a connection can connect to any neurons instead of only to the neurons in the next layer. However, since the architecture of NNN is not regularly assigned, it is difficult to accelerate the parameter learning process parallelly via existing computing platforms such as Tensorflow. It is a great training burden with more neurons. The superiority of NNN is its robustness to irrelevant components in input data, i.e., background in images. Therefore, we continue to use the trained classifiers to classify the digits with various backgrounds.

The ROC and PR curves of results on the two test sets, i.e.,

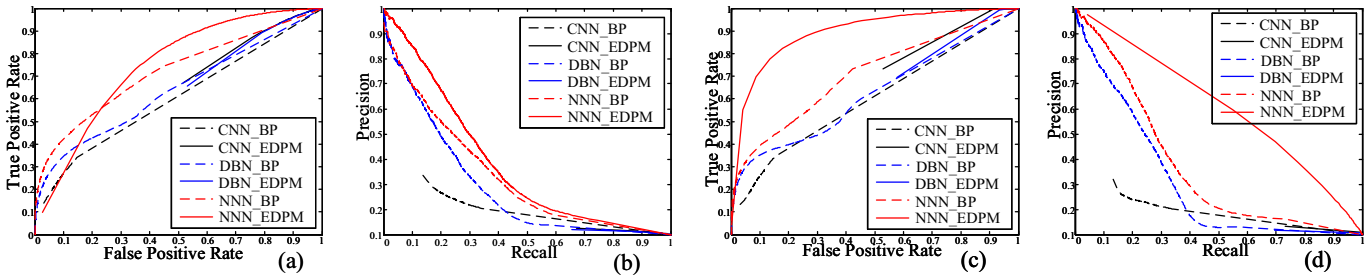

Fig. 6. ROC and PR curves of the test result on digits with currupted background. (a) ROC curves on test data with bg-img (b) PR curves on test data with bg-img (c) ROC curves on test data with bg-rand (d) PR curves on test data with bg-rand

MNIST digits with bg-img and bg-rand, are shown in Fig. 6. It is obvious that the curves of NNN cover all the other curves. Traditional architectures are exhausted to deal with such a problem. Because they are composed of hierarchical layers, input neurons have approximately equal impact on the output neurons from architecture. During the back-propagation process, the output errors will not influence the weights of background pixels a lot due to the weak contribution of them. Therefore, back-propagation could not weaken the influence of background changes on output. Then when images in training and test set are with different backgrounds, the classification results will be different a lot. In NNN, the less relevant neurons will be directed into deeper and narrower paths because they have less direct impact on the output neurons. Therefore, NNN is more robust and greatly outperforms the traditional architectures.

Then the AUC, AP, and CA values of the two test sets are listed in Table I (test sets of bg-img and bg-rand, respectively). It can be found that CNN achieves worst performance in this problem. Because in CNN, the parameters in local convolutional kernels are shared, the background and foreground use the same kernels. Therefore, the influence of background neurons on the output neurons will be larger compared with the architectures of DBN and NNN. The proposed parameter learning method decreases the performance of hierarchical architectures because the model increases the energy, i.e., the training error of other data and decreases the energy of observed data. This means that with a distribution that is even a little different from distribution of training data, the output will be much different. Thus with different background in the test set, the performance decays a lot. However, in NNN, the background are restrained by the architecture, thus the model will specially focus on the foreground. Therefore, NNN greatly improves the performance in this problem. The digits with bg-img are more difficult to recognize because the background greatly disturb the digits as shown in Fig. 4. Although NNN cannot achieve equivalent performance compared with traditional classifiers trained by images with the same kind of background as shown in [23] (77.39% and 85.42% respectively by SVM, and 83.32% and 89.70% respectively by denoising auto-encoder), it outperforms traditional classifiers greatly when the backgrounds of training and test sets are different. The robust learning and inference capability of NNN is larger than that of traditional network architectures in this dataset.

TABLE I
AUC, AP, AND CA VALUES OF THE TEST RESULT ON DIGITS IN DIFFERENT TEST SET. ARC. DENOTES ARCHITECTURE AND PAR. DENOTES PARAMETER LEARNING METHOD.

Test Set	Arc.	DBN		CNN		NNN	
	Par.	BP	EDPM	BP	EDPM	BP	EDPM
MNIST	AUC	.9898	.9864	**.9968**	.9888	.9846	.9921
	AP	.9941	.9896	**.9953**	.9921	.8734	.9814
	CA(%)	98.97	98.74	**99.14**	98.73	98.69	98.86
bg-img	AUC	.6395	.4005	.6002	.4139	.7205	**.7434**
	AP	.2860	.0389	.1514	.0379	.3300	**.3776**
	CA(%)	29.71	10.28	24.24	13.14	37.18	**57.33**
bg-rand	AUC	.6230	.3671	.5981	.4183	.7082	**.8926**
	AP	.3020	.0340	.1478	.0323	.3658	**.5685**
	CA(%)	30.66	9.55	22.18	15.62	32.84	**67.00**

V. CONCLUSIONS AND FUTURE WORK

This paper attempts a new neural network architecture which is called nucleus neural network (NNN). NNN mimics nuclei in brain where there is no regular layers so as to relax the limitation of layers. To evaluate the architectures efficiently, we directly model the architecture without involvement of connecting weights and biases. The objective function is defined to well represent the relationship between input and output neurons with the architecture and is optimized by the widely used particle swarm optimization algorithm. With the optimized architecture, the connecting weights and biases are learned by establishing a probability model in order to well represent the input data. After optimization of architecture and parameters respectively, we find that this novel architecture is robust to irrelevant components in data. This means that NNN works when the background of test data is different from that of training data. Therefore, we reconstruct a new dataset based on the MNIST dataset. From the experiments, NNN is superior over traditional deep learners on this dataset.

However, since NNN is a brand new architecture, there are still many defects including the low architecture and parameter learning efficiency due to difficult parallelization and unsatisfactory classification accuracy. In the future work, we will further improve the efficiency of learning methods via parallel computation and attempt feedback connections.

Moreover, we will stack NNN to mimic the hierarchical architecture in brain.

REFERENCES

[1] I. Arel, D. C. Rose, T. P. Karnowski *et al.*, "Deep machine learning-a new frontier in artificial intelligence research," *IEEE Computational Intelligence Magazine*, vol. 5, no. 4, pp. 13–18, 2010.

[2] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.

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

[5] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin, "Large-scale evolution of image classifiers," in *ICML*, 2017, pp. 2902–2911.

[6] H. Liu, K. Simonyan, and Y. Yang, "DARTS: differentiable architecture search," in *ICLR*, 2019.

[7] Y. Sun, X. Wang, and X. Tang, "Sparsifying neural network connections for face recognition," in *CVPR*, 2016, pp. 4856–4864.

[8] J. Liu, M. Gong, Q. Miao, X. Wang, and H. Li, "Structure learning for deep neural networks based on multiobjective optimization," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 6, pp. 2450–2463, 2018.

[9] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *CVPR*, 2018, pp. 8697–8710.

[10] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, F.-F. Li, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," in *ECCV*, 2018, pp. 19–34.

[11] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *AAAI*, 2019.

[12] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, "Efficient architecture search by network transformation," in *AAAI*, 2018.

[13] K. Kandasamy, W. Neiswanger, J. Schneider, B. Poczos, and E. P. Xing, "Neural architecture search with bayesian optimisation and optimal transport," in *NeurIPS*, 2018, pp. 2016–2025.

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

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

[16] G. Bouma, "Normalized (pointwise) mutual information in collocation extraction," *Proceedings of GSCL*, pp. 31–40, 2009.

[17] K. W. Church and P. Hanks, "Word association norms, mutual information, and lexicography," *Computational Linguistics*, vol. 16, no. 1, pp. 22–29, 1990.

[18] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. Le, and A. Kurakin, "Large-scale evolution of image classifiers," in *ICML*, 2017.

[19] Y. Jeong, J. Park, S. Jang, and K. Y. Lee, "A new quantum-inspired binary pso: Application to unit commitment problems for power systems," *IEEE Transactions on Power Systems*, vol. 25, no. 3, pp. 1486–1495, 2010.

[20] A. Modiri and K. Kiasaleh, "Modification of real-number and binary pso algorithms for accelerated convergence," *IEEE Transactions on Antennas and Propagation*, vol. 59, no. 1, pp. 214–224, 2011.

[21] Y. L. Cun, B. Boser, J. S. Denker, D. Henderson, and L. D. Jackel, "Handwritten digit recognition with a back-propagation network," in *NIPS*, 1997.

[22] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[23] 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," *Journal of Machine Learning Research*, vol. 11, no. Dec, pp. 3371–3408, 2010.

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

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