Evolving Deep Neural Networks via Cooperative Coevolution With Backpropagation

Maoguo Gong, *Senior Member, IEEE*, Jia Liu, A. K. Qin, *Senior Member, IEEE*, Kun Zhao, and Kay Chen Tan, *Fellow, IEEE*

***Abstract*— Deep neural networks (DNNs), characterized by sophisticated architectures capable of learning a hierarchy of feature representations, have achieved remarkable successes in various applications. Learning DNN's parameters is a crucial but challenging task that is commonly resolved by using gradient-based backpropagation (BP) methods. However, BP-based methods suffer from severe initialization sensitivity and proneness to getting trapped into inferior local optima. To address these issues, we propose a DNN learning framework that hybridizes CC-based optimization with BP-based gradient descent, called BPCC, and implement it by devising a computationally efficient CC-based optimization technique dedicated to DNN parameter learning. In BPCC, BP will intermittently execute for multiple training epochs. Whenever the execution of BP in a training epoch cannot sufficiently decrease the training objective function value, CC will kick in to execute by using the parameter values derived by BP as the starting point. The best parameter values obtained by CC will act as the starting point of BP in its next training epoch. In CC-based optimization, the overall parameter learning task is decomposed into many subtasks of learning a small portion of parameters. These subtasks are individually addressed in a cooperative manner. In this article, we treat neurons as basic decomposition units. Furthermore, to reduce the computational cost, we devise a maturity-based subtask selection strategy to selectively solve some subtasks of higher priority. Experimental results demonstrate the superiority of the proposed method over common-practice DNN parameter learning techniques.**

***Index Terms*— Backpropagation (BP), cooperative coevolution (CC), deep neural networks (DNNs), evolutionary optimization.**

I. Introduction

NOWADAYS, deep neural networks (DNNs) [1] have become one of the most powerful machine learning techniques, featuring sophisticated architectures that empower them with remarkable learning capabilities to tackle various challenging tasks. In recent years, to accommodate the fast-increasing scale and complexity of applications, the size and complexity of DNNs keep growing. It significantly increases the difficulty of learning the parameters in DNNs, which corresponds to solve a large-scale, highly complex optimization problem, where an exponentially large number of parameters need to be learned by optimizing some objective function full of local optima.

Backpropagation (BP) [2], combined with some gradient-based optimization method, e.g., stochastic gradient descent (SGD) [3] and Adam [4], is a common practice for learning the parameters (i.e., connection weights and biases) in NNs. When applied to DNNs, BP suffers from the vanishing gradient problem [5], where the gradient (of the error signal) decreases exponentially as it propagates from the back to front layers, which significantly degrades the efficiency of learning the parameters in the front layers. Recent years have seen many approaches proposed to address this issue [6]–[9]. Furthermore, leveraging hardware to accelerate parameter learning also helps to mitigate this issue. In addition to the vanishing gradient issue, as a point-based local search technique, BP-based gradient descent inevitably suffers from severe initialization sensitivity and is prone to get stuck into inferior local optima (and leading to undesirable learning performance) when applied to DNNs. To address this issue, gradient descent algorithms with adaptive learning rates, e.g., Adadelta [10], and dedicated initialization methods [11], [12] have been proposed.

Besides BP, the parameters in DNNs may be also learned by using population-based search techniques, e.g., evolutionary algorithms (EAs) [13], [14], which are gradient-free and, thus, may avoid the abovementioned issues. EAs employ a population of candidate solutions, which is evolved by nature-inspired operations, to seek global optima. They have been intensively studied for learning the parameters in traditional NNs with simple and shallow architectures [15]. However, when applied to DNNs, the efficacy of EAs is greatly challenged by the giant searching space of parameters. Therefore, EAs are popular to evolve the hyperparameters, such as architectures [16]–[23] for DNNs that belong to automated machine learning (AutoML) [24]. In architecture learning, the hyperparameters, including the number of layers, the scale of convolutional kernels, feature maps, and specific connections that are with low-scale searching space, are optimized. The network parameters, including connecting weights and biases that are

Manuscript received June 20, 2019; revised December 3, 2019; accepted March 2, 2020. Date of publication March 25, 2020; date of current version January 5, 2021. This work was supported in part by the National Nature Science Foundation of China under Grant 61772393, in part by the Key Research and Development Program of Shaanxi Province under Grant 2018ZDXM-GY-045, in part by the Australian Research Council under Grant LP170100416, Grant LP180100114, and Grant DP200102611, and in part by the Research Grants Council of the Hong Kong Special Administrative Region, China, under Grant CityU11202418 and Grant CityU11209219. *(Corresponding author: Maoguo Gong.)*

Maoguo Gong and Kun Zhao are with the Key Laboratory of Intelligent Perception and Image Understanding of Ministry of Education, International Research Center for Intelligent Perception and Computation, Xidian University, Xi'an 710071, China (e-mail: gong@ieee.org).

Jia Liu is with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China.

A. K. Qin is with the Department of Computer Science and Software Engineering, Swinburne University of Technology, Melbourne, VIC 3122, Australia (e-mail: kqin@swin.edu.au).

Kay Chen Tan is with the Department of Computer Science, City University of Hong Kong, Hong Kong (e-mail: kaytan@cityu.edu.hk).

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

Digital Object Identifier 10.1109/TNNLS.2020.2978857

with high-scale searching space, are still optimized by BP. In fact, EA-based large-scale optimization per se remains an open problem. The most popular EA-based framework to deal with large-scale optimization is cooperative coevolution (CC) [25]. CC decomposes a large-scale problem into many small-scale (and preferably independent) subtasks and solves each subtask separately by using an EA, where each individual in EA's population with respect to a specific subtask is evaluated in a cooperative way by interacting with the other subtasks. Therefore, CC has been applied to many large-scale optimization problems [26], [27]. Some research attempts have been made to learn the parameters in traditional NNs via CC-based optimization methods [28]–[30], ending up with demanding computational cost. Therefore, the direct application of CC-based optimization to DNN parameter learning may hardly be effective in practice.

In this article, we propose a DNN parameter learning framework that hybridizes CC-based optimization with BP-based gradient descent, called BPCC, and implement it by devising a computationally efficient CC-based optimization method dedicated to DNN parameter learning, aiming to leverage on the advantages of these two different types of parameter learning techniques while mitigating their drawbacks to augment learning performance. In BPCC, BP will intermittently execute for multiple training epochs. After each BP's training epoch, if the loss value indicates that BP becomes less effective, then CC will kick in to execute to help BP escape from local optimum. After that, the best parameters obtained by CC will act as the starting point of BP in the next epoch. This learning process will terminate when some stopping criterion is met.

We implement the proposed BPCC framework via a newly devised CC-based optimization approach with reduced computational cost, which features neuron-based decomposition (ND), maturity-based selection (MS), and a self-adaptive differential evolution (SaDE)-based subtask solver [31]. Specifically, we propose to utilize neurons as basic decomposition units because a neuron acts as an independent feature extractor that extracts a certain feature from the output of its preceding layer. Accordingly, each subtask corresponds to optimize the parameters associated with a specific neuron. However, this decomposition strategy may result in too many subtasks to be solved, dramatically increasing the computational burden of CC. To address this issue, we propose to select to solve some subtasks based on the maturity of their corresponding neurons, where maturity reflects the degree of a neuron being activated or deactivated. When fully activated or deactivated, a neuron is regarded as having the largest maturity. If both the activation and deactivation degrees of a neuron are not enough strong, the maturity of the neuron is small, indicating obscurity in determining either the presence or absence of the feature captured by this neuron. In this case, it is desirable to let this neuron keep discovering a more meaningful feature by further learning its related parameters. Furthermore, we choose to employ a powerful EA, i.e., SaDE, as the subtask solver in CC. We name the proposed implementation as BPCC_ND_MS_SaDE and evaluate its performance on MNIST and CIFAR-10 data sets in comparison to the common-practice DNN parameter learning techniques to demonstrate its advantages.

The rest of this article is organized as follows. Section II describes the background and related works. Section III details the proposed BPCC framework and its implementation. In Section IV, experimental results are reported and discussed. Finally, Section V concludes this article and mentions future work.

II. Background

A. DNNs

DNNs [1] represent a special family of NNs characterized by multiple hidden layers between the input and output layers. They feature more complicated architectures than traditional NNs, endowing them with exceptional capacity to learn powerful feature representations from a massive amount of data.

Multilayer feedforward networks (MFNs) [32] are an intuitive DNN, where multiple hidden layers (more than two in the context of DNNs) between the input and output layers are fully connected in a feedforward way. In an MFN, each neuron except for those in the input layer acts as a feature extractor to extract a certain feature from the output of its preceding layer, where its incoming connection weights and bias define the parameters of the feature extractor. Accordingly, the whole network can be seen as the connection of numerous feature extractors corresponding to different neurons, where the synergy of the features extracted by these feature extractors determines the final output of the network. Therefore, neurons can be regarded as the fundamental units of an MFN.

Convolutional neural networks (CNNs) [33] are among the most popular DNNs, proficient in learning from the data with certain local structures, such as images. A typical CNN consists of an input layer, an output layer, and multiple hidden layers, including convolutional layers, pooling layers, and fully connected layers. The parameters in a CNN mostly come from convolutional layers and fully connected layers, where the parameters in a convolutional layer are from its involved convolution kernels and the parameters in a fully connected layer are from its involved neurons. Therefore, convolution kernels in convolutional layers and neurons in fully connected layers constitute the primitives of a CNN. In fact, a convolution kernel can be regarded as a special neuron that takes as input an image (or a feature map) and outputs a feature map.

In addition to MFNs and CNNs, there exist lots of other DNN's architectures suitable for different domains, e.g., recurrent neural networks [34] and deep belief networks [35]. As problem complexity and scale increase, the size and complexity of DNNs increase accordingly, posing a great challenge to effectively learn numerous parameters in DNNs. It is worth noting that DNN's parameters may also include structural parameters, e.g., the number of hidden layers and the number of neurons per layer, which, however, is out of the scope of our research. This article only considers connection weights and biases in DNN parameter learning.

B. Parameter Learning Techniques for DNNs

BP [36] is the most common choice of techniques for DNN parameter learning. It calculates the partial derivative of the error function (defined on training data) with respect to each parameter in the DNN via the chain rule [37]. The parameters in the DNN are then updated using the calculated derivatives in a way, such as a delta rule [37]. The parameter updating process is iterated until some stopping criteria are met.

We, hereby, describe and explain some key notations to help in the understanding of the proposed method in Section III. Let us consider an MFN with L layers, including one input layer (the first layer), one output layer (the last layer), and $L-2$ hidden layers. The parameter set of this MFN, represented by $P=\{W,B\}$, consists of a set of connection weight matrices $W=\left\{\boldsymbol{W}^{ll+1}\in\Re^{n_l\times n_{l+1}}\mid l=1,\ldots,L-1\right\}$ and a set of bias vectors $B=\left\{\boldsymbol{B}^{l}\in\Re^{n_l}\mid l=2,\ldots,L\right\}$. Here, $\boldsymbol{W}^{ll+1}$ denotes the connection weight matrix between the lth layer with n_l neurons and the $(l+1)$th layer with n_{l+1} neurons. Each element in $\boldsymbol{W}^{ll+1}$ is represented by w_{ij}^{ll+1} that denotes the connection weight between the ith neuron in the lth layer and the jth neuron in the $(l+1)$th layer. $\boldsymbol{B}^l$ denotes the bias vector for the lth layer with each element denoted by b_i^l representing the bias of the ith neuron in this layer. Given a training set composed of N training cases (i.e., input–output pairs) denoted by $D=\left\{(\boldsymbol{x}_k,\ \boldsymbol{y}_k)\mid \boldsymbol{x}_k\in\Re^d,\boldsymbol{y}_k\in\Re^C,k=1,\ldots,N\right\}$, where $\boldsymbol{x}_k$ and $\boldsymbol{y}_k$ denote an input vector in $\Re^d$ and its target output represented in $\Re^C$, with $\Re$ being the set of real numbers. The parameter learning task aims at finding the optimal parameter values to minimize an error function $E(P)$ that can be typically formulated as

$$E(P)=\frac{1}{2N}\sum_{k=1}^{N}\left\|\boldsymbol{o}^L(\boldsymbol{x}_k)-\boldsymbol{y}_k\right\|_2^2 \tag{1}$$

where $\boldsymbol{o}^L(\boldsymbol{x}_k)$ stands for the final output of the network with respect to input $\boldsymbol{x}_k$ and $\|\cdot\|_2^2$ denotes the squared l_2-norm used to measure the squared Euclidean distance between $\boldsymbol{o}^L(\boldsymbol{x}_k)$ and $\boldsymbol{y}_k$. Suppose that $\boldsymbol{o}^l(\boldsymbol{x}_k)\in\Re^{n_l}$ denotes the outputs of n_l neurons in the lth layer with respect to input $\boldsymbol{x}_k$. The input and output of the first layer are the same, i.e., $\boldsymbol{o}^1(\boldsymbol{x}_k)=\boldsymbol{x}_k$ and $n_1=d$. The output of the last layer, i.e., $\boldsymbol{o}^L(\boldsymbol{x}_k)$, is the final output of the network so that $n_L=C$. Mathematically, the output of the jth neuron in the lth layer $(l>1)$ with respect to $\boldsymbol{x}_k$ can be formulated as

$$o_j^l(\boldsymbol{x}_k)=A\left(\sum_{i=1}^{n_{l-1}}w_{ij}^{(l-1)l}\cdot o_i^{l-1}(\boldsymbol{x}_k)+b_j^l\right) \tag{2}$$

where A denotes an activation function, e.g., a sigmoid function of the form $f(x)=1/(1+\exp(x))$.

The partial derivative of $E(P)$ with respect to each connection weight w_{ij}^{ll+1} $(i=1,\ldots,n_l, j=1,\ldots,n_{l+1},\ l=1,\ldots,L-1)$ and each bias b_i^l $(i=1,\ldots,n_l,\ l=2,\ldots,L)$ can be calculated via BP. Then, the parameters can be updated using the calculated derivatives by

$$w_{ij}^{ll+1}=w_{ij}^{ll+1}-\eta\frac{\partial E(P)}{\partial w_{ij}^{ll+1}},\quad i=1,\ldots,n_l,\quad j=1,\ldots,n_{l+1},\quad l=1,\ldots,L-1 \tag{3}$$

$$b_i^l=b_i^l-\eta\frac{\partial E(P)}{\partial b_i^l},\quad i=1,\ldots,n_l,\ l=2,\ldots,L \tag{4}$$

where η denotes the learning rate that controls the step size of learning and needs to be set properly to avoid learning too fast or too slow. The parameter updating process will be iterated until a certain stopping criterion is met.

BP-based parameter learning often operates in a minibatch mode. In this mode, a small subset of training cases randomly selected from the whole training set, so-called a minibatch, is used for an iteration of BP, and a certain number of such iterations constitute a training epoch. Typically, BP's learning process consists of multiple training epochs, where each epoch covers the whole training set that is randomly partitioned into multiple minibatches with each minibatch used by one iteration of the epoch. When applied to DNNs, BP suffers from the notorious vanishing gradient problem. Furthermore, as a point-based local search approach, it also suffers from severe sensitivity to parameter initialization and is prone to get trapped into inferior local optima.

Population-based search techniques, e.g., EAs [13], provide another way to learn DNN's parameters, which may avoid the issues encountered by BP due to their gradient-free nature. EAs have been intensively studied for learning the parameters in traditional NNs with simple and shallow architectures [14]. However, a DNN typically involves far more parameters than a traditional NN, leading to a large-scale optimization problem that greatly challenges the efficacy of existing EA-based parameter learning methods. There are attempts to evolve parameters in unsupervised models that are single-layer networks pretraining a deep network [38]–[41] and to evolve architectures or other hyperparameters of DNNs [16]–[20], as introduced earlier. However, few works can be found to directly evolve the network parameters due to the large-scale searching space. CC-based optimization, as proposed by Potter and De Jong [25], provides an effective EA-based framework to deal with large-scale optimization problems.

C. Cooperative Coevolution

CC is a type of EA framework for large-scale optimization. It decomposes a n-dimensional decision vector into L subcomponents. Then, the L subcomponents are evolved separately with EAs. The framework of CC is shown in Fig. 1, where each subcomponent is evolved separately. This greatly reduces the searching space in each evolving process. Parameter learning via CC-based optimization has been used for training traditional NNs [28]–[30], achieving promising performance at the expense of demanding computational cost. Accordingly, direct application of such methods to DNNs may likely result in prohibitive computational cost and, thus, becomes practically infeasible.

III. BPCC and Its Implementation

BP-based gradient descent and CC-based optimization provide two different ways to learn the parameter in DNNs, each with pros and cons. BP stands for the mainstream DNN parameter learning techniques with good computational efficiency but suffers from initialization sensitivity and proneness to getting trapped into inferior local optima. CC is gradient-free

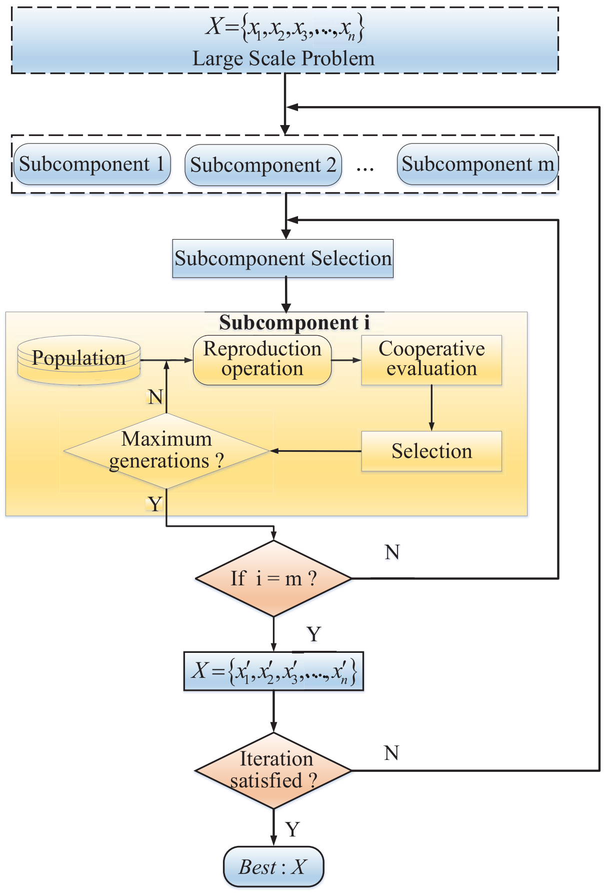


Fig. 1. Framework of CC.

and, thus, may avoid the issues suffered by BP although it is typically computationally expensive. In this article, we propose to hybridize CC with BP so that these two learning techniques may complement each other to augment learning performance. This section will describe the proposed hybridization framework and its implementation via a newly devised CC-based optimization method dedicated to DNN parameter learning, which features ND, MS, and a SaDE-based subtask solver.

A. Framework

DNN parameter learning corresponds to optimize an objective function of DNN's parameters, typically formulated via an error term measuring the difference between the output of the network with respect to the input and the target output of the input plus a regularization term intending to avoid overfitting in training and, thus, improve generalization performance. Without loss of generality, we employ the following objective function $F(P)$ for the DNNs studied in this article, which will be minimized with respect to DNN's parameter set P to derive the optimal parameter values:

$$F(P) = \frac{1}{2|D|}\sum_{(\boldsymbol{x},\boldsymbol{y})\in D}\|\boldsymbol{o}^L(\boldsymbol{x}) - \boldsymbol{y}\|_2^2 + \beta\|\boldsymbol{W}\|_2. \quad (5)$$

Here, the first term at the right-hand side of the equation is the accumulated training error over a certain training set D composed of $|D|$ training cases. The second term is the

Algorithm 1 Description of the BPCC Parameter Learning Framework

Step 1:
Initializing the parameters in the DNN and using the initialized parameter values as the starting point for BP.
Step 2:
Randomly selecting a training set D from the whole training set on which BP is applied for one training epoch in a minibatch mode.
Step 3:
If the difference between the values of the training objective function obtained after the current and previous BP's training epochs is below a certain threshold, applying CC-based optimization on the training set D that was used in Step 2 while using the parameter values derived by BP in Step 2 as the starting point for CC. CC-based optimization will terminate when stopping criterion STOP_{CC} is met.
Step 4:
If BPCC's stopping criterion $\text{STOP}_{\text{BPCC}}$ is not met, using the best parameter values obtained by CC in Step 3 as the new starting point for BP and going to step 2. Otherwise, terminating BPCC and returning the best parameter values found so far as BPCC's learning result.

weighted l_2-norm of $\boldsymbol{W}$ (all connection weights in a DNN are represented as a vector), which is a very commonly used regularization term in DNN's objective functions, where weighting parameter β controls the contribution of the regularization term to $F(P)$.

The proposed BPCC parameter learning framework hybridizes CC-based optimization with BP-based gradient descent to minimize (5). In BPCC, BP acts as the backbone and executes intermittently for multiple training epochs. To reduce the computational cost, each training epoch of BP randomly selects a training set D from the whole training set and partitions D into multiple minibatches used by the multiple iterations of the epoch. After each BP's training epoch finishes, the difference between the training objective function values obtained after the current and previous training epochs will be calculated. If the difference is below a certain threshold, indicating that BP becomes less effective, CC will kick in to search better parameter values, using the parameter values derived by BP as the starting point, and execute until stopping criterion STOP_{CC} is met. The best parameter values obtained by CC will then become the starting point of BP in its next training epoch. The learning process of BPCC will keep iterating until stopping criterion $\text{STOP}_{\text{BPCC}}$ is met. Algorithm 1 elaborates on the proposed BPCC framework.

In this article, we define a dynamic threshold (in Step 3) as $\tau/\sqrt{t}$, where τ is a problem-dependent parameter. Accordingly, the threshold value decreases as BP's training epoch t increases to accommodate the fact that the change of the training objective function value tends to become smaller and smaller as BP executes. Furthermore, STOP_{CC} and $\text{STOP}_{\text{BPCC}}$ are defined as reaching the maximum number of CC's iterations ($\#maxCCIter$) and the maximum number of BP's training epochs ($\#maxBPEpoch$), respectively.

The BPCC framework can be implemented via any specific BP-based gradient descent algorithm and CC-based optimization method. In Section III-B, we will detail our proposed implementation of the BPCC framework.

B. Implementation

To implement the proposed BPCC framework, we need to implement its BP and CC components. There exist many BP-based gradient descent algorithms for DNN parameter learning although CC-based optimization methods for learning the parameters in DNNs are yet common. In this work, we employ the well-known SGD and Adam algorithms to implement the BP component, which has been widely applied to DNNs, and devise a dedicated CC-based optimization method with reduced computational cost to implement the CC component.

The decomposition strategy plays a crucial role in a CC-based optimization method. Overdecomposition can produce simple subtasks but may end up with a considerable number of them, which would degrade computational efficiency. Under-decomposition can reduce the number of subtasks but may increase the complexity and dependence of them, which would degrade optimization accuracy. In practice, a desirable decomposition strategy is expected to generate a limited number of simple and less dependent subtasks. In this article, we propose an ND strategy that uses neurons as basic decomposition units. Accordingly, each of the resulting subtasks corresponds to optimize the parameters associated with a specific neuron. Due to the massive number of neurons in a DNN, this strategy will typically lead to overdecomposition, ending up with numerous subtasks to be resolved. To address this issue, we further propose an MS strategy to selectively resolve some subtasks corresponding to the least mature neurons. In addition to decomposition, the EA employed to solve individual subtasks is another key factor that may significantly influence the ultimate performance of CC-based optimization. In this work, we utilize a powerful EA, i.e., the SaDE algorithm [31], to solve each subtask. Thus, the proposed CC-based optimization method is named CC_ND_MS_SaDE, and its corresponding implementation of the BPCC framework is called BPCC_ND_MS_SaDE. In the following, we will elaborate on the decomposition strategy, the selection strategy, and the optimization process in CC_ND_MS_SaDE, respectively.

1) Decomposition Strategy: Decomposition is the very first step in CC-based optimization. Specifically, to learn the parameters in a DNN, the ultimate task of learning the parameters in the whole network will be first decomposed into many subtasks with each subtask targeting at a small portion of parameters. A DNN is typically composed of tons of neurons (a convolution kernel in a CNN can be regarded as a special neuron as discussed in Section II-A). Each neuron, except for those in the input layer, acts as an independent feature extractor to extract a certain feature from the output of its preceding layer. It is the synergy of the features extracted by all the neurons in a DNN that determines the final output of the DNN. Therefore, neurons can be regarded as the most fundamental elements of a DNN, and thus, we use them as decomposition units. Fig. 2 shows the proposed ND strategy for MFNs and CNNs, respectively.

To better explain the proposed ND using mathematical notations, we take as an example the MFN illustrated in Fig. 2(a). This MFN has four layers (i.e., $L = 4$), including one input layer, two hidden layers, and one output layer. Its parameter set is represented by $P = \{W, B\}$, where $W = \{\boldsymbol{W}^{(ll+1)} \in \mathfrak{R}^{(n_l+1)\times n_{l+1}} \mid l = 1, 2, 3\}$ with $w_{ij}^{ll+1}(i = 1, \ldots, n_l, j = 1, \ldots, n_{l+1})$ denoting an element in $\boldsymbol{W}^{(ll+1)}$ and $B = \{\boldsymbol{B}^l \in \mathfrak{R}^{n_l} \mid l = 2, 3, 4\}$ with $b_i^l(i = 1, \ldots, n_l)$ denoting an element in $\boldsymbol{B}^l$. The ND strategy will lead to a set of $n_2 + n_3 + n_4$ subtasks, i.e., $\text{sub}_r(r = 1, \ldots, n_2 + n_3 + n_4)$, as shown in the following equation:

$$\underbrace{w_{11}^{12}, \ldots, w_{n_1 1}^{12}, b_1^2}_{P_{sub_1}}, \underbrace{w_{12}^{12}, \ldots, w_{n_1 2}^{12}, b_2^2}_{P_{sub_2}}, \ldots, \underbrace{w_{1n_2}^{12}, \ldots, w_{n_1 n_2}^{12}, b_{n_2}^2}_{P_{sub_{n_2}}}$$
$$\text{(The 1}^{\text{st}}\text{ hidden layer with } \boldsymbol{W}^{12} \text{ and } \boldsymbol{B}^2)$$
$$\underbrace{w_{11}^{23}, \ldots, w_{n_2 1}^{23}, b_1^3}_{P_{sub_{n_2+1}}}, \underbrace{w_{12}^{23}, \ldots, w_{n_2 2}^{23}, b_2^3}_{P_{sub_{n_2+2}}}, \ldots, \underbrace{w_{1n_3}^{23}, \ldots, w_{n_2 n_3}^{23}, b_{n_3}^3}_{P_{sub_{n_2+n_3}}}$$
$$\text{(The 2}^{\text{nd}}\text{ hidden layer with } \boldsymbol{W}^{23} \text{ and } \boldsymbol{B}^3)$$
$$\underbrace{w_{11}^{34}, \ldots, w_{n_3 1}^{34}, b_1^4}_{P_{sub_{n_2+n_3+1}}}, \underbrace{w_{12}^{34}, \ldots, w_{n_3 2}^{34}, b_2^4}_{P_{sub_{n_2+n_3+2}}}, \ldots, \underbrace{w_{1n_4}^{34}, \ldots, w_{n_3 n_4}^{34}, b_{n_4}^4}_{P_{sub_{n_2+n_3+n_4}}}$$
$$(\mathit{The\ output\ layer\ with}\ \boldsymbol{W}^{34} \text{ and } \boldsymbol{B}^4). \quad (6)$$

Here, a subtask, e.g., sub_1, corresponds to a neuron, targeting at a parameter set represented by $P_{\text{sub}_1} = \{W_{\text{sub}_1}, B_{\text{sub}_1}\}$, where W_{sub_1} and B_{sub_1} denote the set of connection weights and the set of biases in subtask sub_1, respectively. Such a decomposition strategy can result in intuitive subtasks that are simple and less dependent upon each other but inevitably suffers from overdecomposition because there usually exist lots of neurons in a DNN. As a result, it will be computationally prohibitive if all of the generated subtasks need to be resolved in each iteration of CC-based optimization. To address this issue, we propose to selectively resolve a portion of subtasks with top priority, which will be detailed in Section III-B2.

For a CNN, as illustrated in Fig. 2(b), each convolution kernel can be regarded as a special neuron that takes as input an image (or a feature map) and outputs a feature map. Since such a kernel typically consists of a small number of parameters and the number of kernels in one convolutional layer is not large in this work, we propose to combine all convolutional kernels in one convolutional layer into one subtask. Accordingly, the connection weights and biases of these kernels are combined into W_{sub_r} and B_{sub_r}, respectively. As to fully connected layers, each neuron in a layer corresponds to one subtask as defined in the MFN. It is noted that there exist no parameters in pooling layers.

2) Selection Strategy: It has been commonly observed that not all neurons in a DNN make equal contributions when resolving a specific task. Some neurons will be greatly activated (or deactivated) leading to large (or small) output values, which shows strong evidence on the presence (or absence) of certain features captured by these neurons. Others will

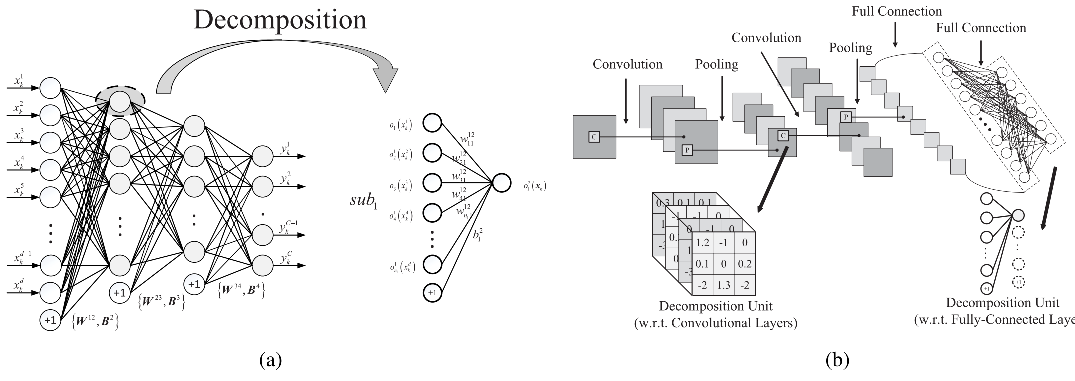

Decomposition
sub_1
(a)
Convolution
Pooling
Convolution
Pooling
Full Connection
Full Connection
Decomposition Unit (w.r.t. Convolutional Layers)
Decomposition Unit (w.r.t. Fully-Connected Layers)
(b)

Fig. 2. Illustration of ND for (a) MFN and (b) CNN.

output medium-scale values, reflecting ambiguity on the presence or absence of the corresponding features. The former neurons can be regarded as being more mature than the latter ones. In fact, mature neurons are highly preferred in a well-trained DNN because they may lead to network sparsity desired for good generalization. Neuroscience research has verified that only a small portion of biological neurons in the animal's brain would be stimulated when the brain processes the information in any specific task [42]. Therefore, it is desirable to let less mature neurons to keep learning their parameters, aiming to reach augmented maturity corresponding to a higher degree of activation or deactivation. Accordingly, we propose to select some subtasks corresponding to the least mature neurons in each iteration of CC-based optimization and only solve these selected subtasks. Here, the maturity of a neuron is defined as the accumulated output value of the neuron over all inputs, i.e., $\{\boldsymbol{x}_k | \ \boldsymbol{x}_k \in \Re^d, k = 1, \ldots, N\}$.

To define neuron maturity in a mathematically way, we take as an example subtask sub_1 in Fig. 2(a), which corresponds to the first neuron in the first hidden layer. The parameters involved in this subtask can be represented by $P_{\text{sub}_1} = \{W_{\text{sub}_1}, B_{\text{sub}_1}\}$ where $W_{\text{sub}_1} = \{w_{11}^{12}, \ldots, w_{n_1 1}^{12}\}$ and $B_{sub_1} = \{b_1^2\}$. Suppose that the output of the jth neuron in the lth layer with respect to input $\boldsymbol{x}_k$ is denoted by $o_j^l(\boldsymbol{x}_k)$. The output of the neuron corresponding to sub_1 with respect to $\boldsymbol{x}_k$ can be calculated by

$$o_1^2(\boldsymbol{x}_k) = A\left(\sum_{i=1}^{n_1} w_{i1}^{12} \cdot o_i^1(\boldsymbol{x}_k) + b_1^2\right) \tag{7}$$

where $o_i^1(\boldsymbol{x}_k)$ is the ith element of $\boldsymbol{x}_k$ because the first layer is the input layer. Accordingly, we define the maturity of the neuron corresponding to sub_1 as

$$M_1 = \sum_{k=1}^{N} G\big(o_1^2(\boldsymbol{x}_k)\big), \quad \text{with}$$
$$G(s) = \begin{cases} 0, & \alpha \ll s \ll 1-\alpha \\ 1, & \text{otherwise} \end{cases} \tag{8}$$

where $G(s)$ is a threshold function measuring neuron maturity in a crisp manner via parameter α. In this article, we fix α at 0.3. Following this definition, we can calculate neuron maturity M_r for each subtask $\text{sub}_r(r = 1, \ldots, n_2 + n_3 + n_4)$. In the beginning of each iteration of CC-based optimization, we will (re)calculate the maturity values for all subtasks, rank all subtasks in the ascending order of their maturity values, and select a certain number of top-ranked subtasks to resolve. For a CNN, only the subtasks corresponding to the neurons in fully connected layers will participate in this MS process. The subtasks corresponding to conventional layers will always be addressed in each iteration of CC-based optimization.

This MS strategy makes CC-based optimization to focus on a portion of neurons that are more needed for further learning and, thus, reduces the considerable computational burden due to overdecomposition caused by the ND strategy.

3) Optimization Process: CC-based optimization employs an iterative optimization process to resolve the set of subtasks generated from decomposition. In each iteration, some subtasks will be selected from the entire list of subtasks and resolved individually via a specific EA. In this work, suppose that R subtasks $\text{sub}_r(r = 1, \ldots, R)$ are generated by the ND strategy and their corresponding maturity values $M_r(r = 1, \ldots, R)$ are calculated. Let $P^* = \{P_{sub_1}^*, P_{\text{sub}_2}^*, \ldots, P_{sub_R}^*\}$ denote the set of optimal parameter values found so far with respect to the objective function in (5), where $P_{\text{sub}_r}^*$ is the subset of P^* with respect to sub_r. In each iteration of the optimization process, all R subtasks will be first ranked in the ascending order of their maturity values. After ranking, top Q subtasks will be selected, i.e., $sub_{r_i}(r_i = 1, \ldots, Q)$, and resolved individually via the EA. The iteration of the optimization process will continue until stopping criterion STOP_{CC} is met.

The objective function used by the EA can be different from that defined in (5), which may consider the characteristics of the subtask. In this article, we assume that a desirable parameter setting for a subtask should lead to both superior model performance and high neuron maturity and, accordingly, propose the following objective function for subtask sub_r (w.l.o.g), which will be maximized with respect to $P_{\text{sub}_r} = \{W_{\text{sub}_r}, B_{\text{sub}_r}\}$ to seek the optimal values of P_{sub_r}

$$\begin{aligned} F_s(P_{\text{sub}_r}) &= \frac{1}{\frac{1}{2|D_s|}\sum_{(\boldsymbol{x},\boldsymbol{y})\in D_s} \|\boldsymbol{o}^L(\boldsymbol{x}) - \boldsymbol{y}\|_2^2 + \beta_s \|W_{\text{sub}_r}\|_2 + 10^{-8}} \\ &\quad + \sigma_s M_r. \end{aligned} \tag{9}$$

This objective function is composed of two parts. The first part considers model performance in terms of the accumulated training error and the regularization of the connection weights related to sub_r, which are balanced via weighting parameter β_s, where a small constant value 10^{-8} is used to avoid zero in the denominator. The second part considers the maturity of the neuron corresponding to sub_r, where the weighting parameter σ_s is used to control the contribution from this part to the entire objective function. It is worth noting that the second part does not exist for subtasks corresponding to convolutional layers on a CNN. Since the evaluation of $F_s\left(P_{\text{sub}_r}\right)$ relies on all the parameters in the DNN to calculate $\boldsymbol{o}^L(\boldsymbol{x}_k)$ as formulated in (2), for the parameters not serving as functional arguments, i.e., those in P but not in P_{sub_r}, their values will be taken from P^* and remain unchanged during the execution of the EA for solving sub_r. This objective function evaluation process is commonly known as cooperative evaluation. Furthermore, to reduce the demanding computational cost expended by the EA to evaluate the objective function, $F\left(P_{\text{sub}_r}\right)$ is formulated on a small subset D_s randomly selected from the training set D used in the preceding BP's training epoch. D_s will be regenerated whenever a new subtask is to be resolved. In this work, the size of D_s (i.e., $|D_s|$) is set to 20% of $|D|$.

The ultimate performance of CC-based optimization is also influenced by the efficacy of the EA employed as the subtask solver. In this article, we use SaDE [31], a well-known variant of the differential evolution (DE) algorithm [43]. DE is one of the most powerful EAs for solving continuous optimization problems. The success of DE is mainly attributed to its unique differential mutation scheme that distinguishes it from the other existing EAs. DE has demonstrated remarkable superiority in various challenging applications although its practical performance is sensitive to its employed search strategy and parameter setting. Finding the most suitable search strategy and its associated parameter setting via trial and error among many candidates is time consuming and may require human expertise that is not always available. Furthermore, a single best-calibrated algorithmic configuration in terms of the employed search strategy and its associated parameter setting cannot guarantee consistent effectiveness at different searching stages since subregions of the search space explored at varying searching stages may prefer different configurations.

To relieve the abovementioned practical difficulty faced by DE, SaDE was proposed in [31] to adapt the employed search strategy and its associated parameter setting to varying searching stages. Specifically, SaDE employs a pool of powerful but complementary search strategies. In each generation, for every individual in the current population, one search strategy will be chosen from the pool based on the selection probabilities of all strategies in the pool, which are calculated as per the success and failure rates of each strategy for generating promising offspring (i.e., those entering the population in the next generation) in the preceding LP generations. To determine the parameter setting for the selected search strategy, two control parameters CR and F are considered. For each search strategy in the pool, its associated CR value will be archived whenever it generates promising offspring in the preceding LP generations, so-called learning period. The median of the CR values recorded in the archive with respect to each strategy is obtained at the end of the current generation and used as the mean value in a normal distribution with the fixed standard deviation to generate the CR value used by the corresponding strategy in the next generation. SaDE randomly generates the F value for the selected strategy according to a normal distribution with the fixed mean and standard deviation so that both exploration (via large F values) and exploitation (via small F values) can be maintained during the search. The population size, denoted by NP, typically depends on the scale and complexity of the problem to be solved as well as the available computational budget. Therefore, SaDE leaves it as a user-defined parameter. Accordingly, there exist two manually specified control parameters in SaDE, i.e., NP and LP. The details of SaDE and its pseudocode can be referred to [31] and [44].

To solve a specific subtask, e.g., sub_r, by using SaDE, $P^*_{\text{sub}_r}$ taken from P^* will be first utilized to initialize a population that contains NP individuals $P^{i_p}_{\text{sub}_r}(i_p = 1, \ldots, NP)$ as

$$P^{i_p}_{\text{sub}_r}(j) = \begin{cases} P^*_{\text{sub}_r}(j) \cdot \text{rand}(0,2), & i_p = 1, \ldots, \text{NP}-1 \\ P^*_{\text{sub}_r}(j), & i_p = \text{NP} \end{cases} \tag{10}$$

where $P^*_{\text{sub}_r}(j)$ and $P^{i_p}_{\text{sub}_r}(j)$ represent the jth element of $P^*_{\text{sub}_r}$ and the jth element of the i_pth individual in the population used for addressing sub_r, respectively. After that, SaDE will evolve the population via cooperative evaluation of the objective function defined in (9) to seek the optimal values of P_{sub_r}. When the number of generations reaches the prespecified maximum number of generations ($\#maxEAGen$) used to terminate the execution of SaDE, the best parameter values found so far by SaDE, denoted by $P^{\text{gbest}}_{\text{sub}_r}$, will replace $P^*_{\text{sub}_r}$ in P^* if this replacement results in the better objective function value calculated on D by (5). If the replacement happens, the updated P^* will then be used for resolving the next subtask.

The pseudocode of the proposed BPCC_ND_MS_SaDE algorithm is described in Algorithm 2, and the code in MATLAB is available at GitHub.[1]

IV. Experimental Study

In this section, we examine the proposed parameter learning method, i.e., BPCC_ND_MS_SaDE, in comparison to common-practice BP-based methods on two popular image data sets to test the following hypotheses.

1) Neuron-based decomposition (ND) and MS strategies play a significant role in the proposed parameter learning method.
2) The proposed parameter learning method outperforms both standalone BP-based learning and standalone CC-based learning for optimizing the training objective function.
3) The proposed parameter learning method outperforms common-practice methods in terms of both training (optimization) and testing (generalization) performances.

[1]https://github.com/liusiqinqinqin/BPCC

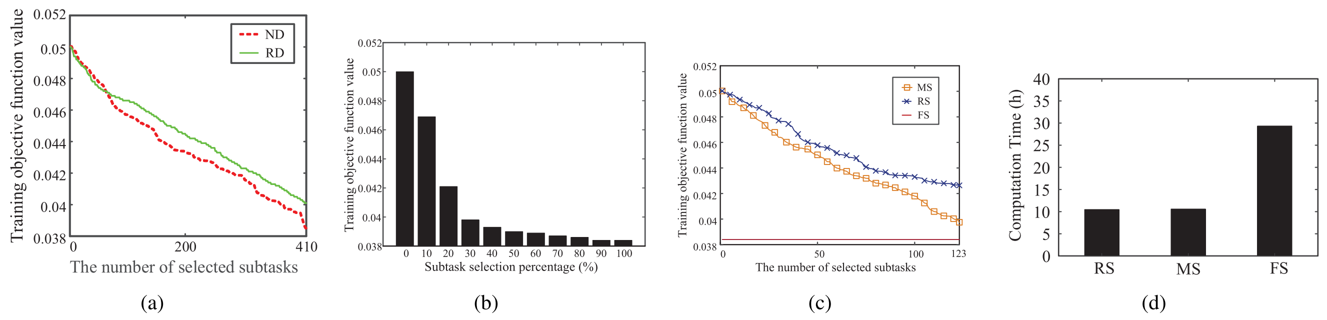


Fig. 3. (a) Comparison of the changing curves of the training objective function value as the number of subtasks being addressed increases with respect to ND and RD in the first iteration of CC, while CC is triggered for the first time. (b) Relation between the obtained training objective function value and the selection percentage of subtasks in the first iteration of CC, while CC is triggered for the first time. (c) Comparison of the changing curves of the training objective function value as the number of selected subtasks increases up to 123 (i.e., 30% of the total number of subtasks) with respect to MS and RS in the first iteration of CC, while CC is triggered for the first time. (d) Comparison of the computation time of the first iteration of CC, while CC is triggered for the first time with respect to MS, RS, and FS being used as the selection strategy, where the selection percentages for MS and RS are both set to 30%.

A. Test Problems and Protocols

Two data sets are employed in our experiments, i.e., MNIST and CIFRA-10, which are the two widely employed benchmarks in DNN-related studies. MNIST is an image data set of handwritten digits 0 $\sim$ 9. It contains 60 000 training cases (with 6000 images per digit) and 10 000 test cases (with 1000 images per digit). Each image is represented by 28×28 pixels. CIFAR-10 is an image data set containing 60 000 color images of 32×32 pixels from ten classes (with 6000 images per class). The ten classes include airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck that are mutually exclusive. The 60 000 images in this data set are divided into 50 000 for training and 10 000 for testing, where each class in the training and test sets has 5000 and 1000 images, respectively.

In our experiments, DNN parameter learning is based upon the training set, and the performances of different parameter learning techniques are compared in terms of the eventually achieved values of the objective function formulated on the training set, as defined in (5). The practical performances of the learned DNNs (reflecting their generalization capabilities) are evaluated and compared on the test set. For the MNIST data set, two kinds of DNN architectures are considered, i.e., MFN and CNN, which have been commonly used for this data set. For the CIFAR-10 data set, only CNN is considered because of its ubiquitous utilization for this data set. Since this article is not focused on how to determine the best DNN architecture, we only choose to use some example configurations of the DNN architectures under examination to investigate the effectiveness of the proposed parameter learning method.

B. Analysis of Decomposition and Selection Strategies

Decomposition plays a key role in CC. In this experiment, we will compare ND with random decomposition (RD) to demonstrate the effectiveness of the proposed ND strategy. We employ the training set of the MNIST data set to learn the parameters of a two-hidden-layer MFN with its architecture denoted by 784-300-100-10, indicating 784, 300, 100, and 10 neurons in the input, first hidden, second hidden, and output layers, respectively. The ND strategy, as described in Section III-B1, produces a total of 410 (300+100+10) subtasks for such an MFN. To make a fair comparison between ND and RD, the RD strategy is specified to also generate 410 subtasks in total. Specifically, it randomly partitions a total of 266 610 (785*300 + 301*100 + 101*10) parameters in this MFN into 410 groups with each group corresponding to one subtask that contains around 650 parameters.

To compare the performances of two decomposition strategies, we utilize BPCC_ND_MS_SaDE as the base method and remove its selection component, i.e., all 410 subtasks will be addressed in each iteration of CC, to make the comparison to be only focused on the decomposition component. ND and RD are then employed, respectively, as the decomposition strategy for comparison. The parameter setting in the base method is specified as follows: $\#maxBPEpoch = 50$, $\#maxCCIter = 1$, $\#maxEAGen = 50$, $\eta = 0.1$, $\tau = 0.005$, $\alpha = 0.3$, $\beta = 0.00001$, $\beta_s = 0.001$, $\sigma_s = 0.0005$, $Q = 30\% \times R$, $\mathrm{SZ}_D = 10\,000$, $\mathrm{SZ}_{D_s} = 20\% \times \mathrm{SZ}_D$, $\mathrm{NP} = 100$, and $\mathrm{LP} = 20$. We choose to use the following method to initialize connection weights $w_{ij}^{ll+1} (i = 1, \ldots, n_l, j = 1, \ldots, n_{l+1})$, $l = 1, \ldots, L-1$ and biases $b_i^l (i = 1, \ldots, n_l, l = 2, \ldots, L)$:

$$w_{ij}^{ll+1} = \frac{2\sqrt{6}(\mathrm{rand}(0,1) - 0.5)}{\sqrt{n_l + n_{l+1}}}$$
$$b_i^l = 2(\mathrm{rand}(0,1) - 0.5) \qquad (11)$$

where rand(0, 1) denotes a real number randomly generated in the value range between 0 and 1.

Note that in any iteration of CC, the execution order of all subtasks matches subtask indices. For ND, subtask indices are determined in the way illustrated in Section III-B1. For RD, indices are randomly allocated during random partition.

Fig. 3(a) compares the changing curves of training objective function values as the number of subtasks being addressed increases with respect to ND and RD in the first iteration of CC, while CC is triggered for the first time. For both ND and RD, the illustrated CC iteration starts from the same BP result, and thus, the two curves share the same starting point. It is observed in Fig. 3(a) that after a small number (i.e., 62) of subtasks are addressed, ND-based CC consistently

Algorithm 2 BPCC_ND_MS_SaDE

```
Input:
  $D_{all}$: Training data; $\#maxBPEpoch$: The maximum number of BP's epochs; $\eta$: Learning rate in BP; $\beta$: Weighting parameter in Eq. (5); $\tau$: Threshold parameter; $SZ_D$: The size of the selected training subset for each BP's epoch; $\beta_s$, $\sigma_s$: Weighting parameters in Eq. (9); $\alpha$: Maturity parameter in Eq. (8); $Q$: The number of selected subtasks for MS; $\#maxCCIter$: The maximum number of CC's iterations; $SZ_{D_s}$: The size of the selected training subset for solving each subtask in a CC's iteration; $\#maxEAGen$: The maximum number of EA's generations; $NP$: Population size; $LP$: Learning period
Output:
  DNN's parameters: $P^*_{\#maxBPEpoch} = \{W^*, B^*\}$
Algorithm:
  Initialize DNN's parameters: $P^*_0 = \{W^0, B^0\}$.
  for $t = 1, \cdots, \#maxBPEpoch$ do
    1. Randomly select $D$ of size $SZ_D$ from $D_{all}$;
    2. Apply BP by staring from $P^*_{t-1}$ to learn DNN's parameters on $D$ for one epoch, with the result assigned to $P^*_t$;
    3. Calculate $F(P^*_t)$ and $F(P^*_{t-1})$ on $D$ by using Eq. (5).
    4. If $F(P^*_t) - F(P^*_{t-1}) < \tau / \sqrt{t}$:
      1. Initialize CC's starting point $P^{CC}_0$ as $P^*_t$;
      2. Decompose $P^{CC}_0$ into $P^{CC}_0 = \{P^{CC}_{0,sub_1}, .., P^{CC}_{0,sub_R}\}$;
      3. Perform the MS-based ranking of all $R$ subtasks and select to solve the top $Q$ subtasks $\{sub_1, \cdots, sub_Q\}$.
      4. for $c = 1, \cdots, \#maxCCIter$ do
           for $q = 1, \cdots, Q$ do
             1. Randomly select $D_s$ of size $SZ_{D_s}$ from $D$;
             2. Initialize a population of size $NP$ for SaDE by using Eq. (10);
             3. Execute $SaDE$ with the objective function calculated by Eq. (9) and the eventually achieved best result denoted by $P^{gbest}_{c,sub_q}$;
             4. Create $P_{temp}$ by initializing it as $P^*_t$ and set its $sub_q$ component to $P^{gbest}_{c,sub_q}$;
             5. Calculate $F(P^*_t)$ and $F(P_{temp})$ on $D$ by using Eq. (5).
             6. if $F(P_{temp}) < F(P^*_t)$: $P^*_t = P_{temp}$. end if
           end for
         end for
    end if
  end for
```

outperforms RD-based CC in terms of the obtained training objective function value as the number of subtasks being addressed keeps increasing. After addressing all 410 subtasks, the training objective function value achieved by ND-based CC is significantly smaller than that obtained by RD-based CC. The results verify the effectiveness of the proposed ND strategy.

Although the abovementioned experiment demonstrates the effectiveness of the ND strategy, addressing all the subtasks generated by ND in each iteration of CC will typically lead to prohibitive computation time and disable practical feasibility. To address this issue, we have proposed to merely solve some subtasks selected in a certain way from all the subtasks generated by ND. In this experiment, we will compare the proposed MS, as described in Section III-B2, with random selection (RS) while using full selection (FS) that addresses all subtasks as did in the previous experiment as the reference.

We use the same base method and experimental setup as those employed in the previous experiment and make the comparison to be only focused on the selected component. Fig. 3(b) shows the relationship between the value of objective function and the selection percentage of subtasks when the MS strategy is used. It is observed that the training objective function value quickly decreases to a small value as the selection percentage increases to 30%. After that, the decrease slows down as the selection percentage continues to increase. Considering the fact (also observed in the previous experiment) that solving all subtasks in an iteration of CC is usually computationally expensive and the experimental observation that selectively addressing merely 30% of all subtasks via MS may lead to the similar result as that obtained by addressing all subtasks, we compare MS with RS subjected to selection percentages not exceeding 30% to control computation time at a reasonable level. Different from MS that selects the top 30% of the subtasks in the ranked list generated as per the ascending order of maturity scores, RS randomly selects 30% of all subtasks.

Fig. 3(c) compares the changing curves of training objective function values as the number of selected subtasks increases up to 123 (i.e., 30% of the total number of subtasks) with respect to MS and RS in the first iteration of CC, while CC is triggered for the first time. The training objective function value obtained by FS, i.e., addressing all subtasks, is depicted as a straight line in Fig. 3(c) to denote the best achievable result in this CC iteration. It is observed that using MS makes the training objective function value to reach the best achievable result after 30% of all subtasks are addressed, which, however, cannot be achieved by using RS. Furthermore, MS has a much quicker decrease in the training objective function value than RS, indicating that more important subtasks can be selected to address earlier than less important ones.

Fig. 3(d) compares the computation time of the first iteration of CC, while CC is triggered for the first time with respect to MS, RS, and FS being used as the selection strategy, where the selection percentages for MS and RS are both set to 30%, and FS is performed in the same way as in the previous experiment. It is observed that FS is much more time consuming than both MS and RS, while MS is nearly as time efficient as RS. However, as demonstrated in Fig. 3(c), MS can result in a much better training objective function value than RS, approaching that achieved by FS. The results verify the effectiveness of the proposed MS strategy.

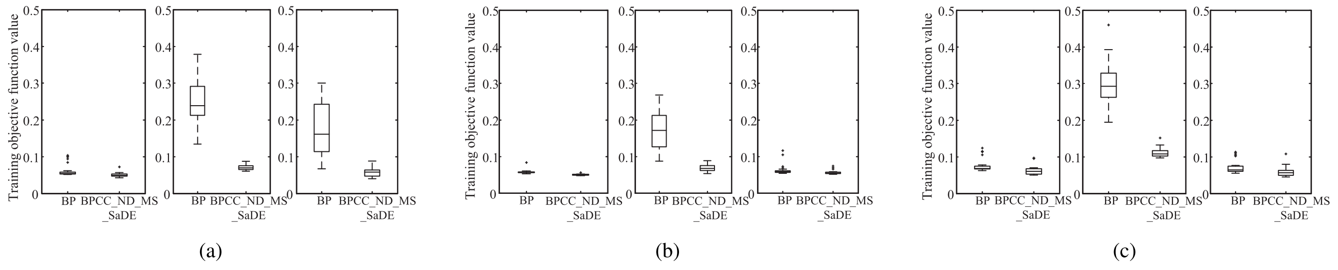


Fig. 4. Comparison of experimental results via box plots. (a)–(c) Network architectures A_1, A_2, and A_3, respectively, and the left, middle, and right parts of each subfigure compare BPCC_ND_MS_SaDE and BP via the box plots of their eventually obtained training objective function values over 30 runs for initialization methods I_1, I_2, and I_3, respectively.

C. In-Depth Evaluation of the Proposed Method

The experiments described in Section IV-B have demonstrated the effectiveness of the ND and MS strategies in the proposed BPCC_ND_MS_SaDE. In this section, we will make an in-depth evaluation of BPCC_ND_MS_SaDE to investigate its sensitivity to parameter initialization and proneness to getting stuck into inferior local optima in comparison to BP. We will also analyze how BPCC_ND_MS_SaDE works in comparison to standalone CC-based learning to demonstrate its advantages.

DNN parameter learning needs to solve a large-scale and highly multimodal optimization problem, where the complexity of the problem highly depends on network architectures. Therefore, it is more reasonable to investigate the effectiveness of BPCC_ND_MS_SaDE under varying problem complexity, i.e., using different network architectures. In this experiment, we employ the training set of the MNIST data set to learn the parameters of three MFNs with different architectures denoted by A_1: 784-500-150-10, A_2: 784-300-100-10, and A_3: 784-196-10, respectively, which are commonly used for the MNIST data set [33], [45]. Note that A_3 is a shallow NN, used here to serve as a baseline for comparison. Moreover, since BPCC_ND_MS_SaDE starts with BP, we also need to investigate its performance under different parameter initialization methods due to the initialization sensitivity issue suffered by BP. Accordingly, we choose to use the following three methods to initialize connection weights $w_{ij}^{ll+1}(i = 1, \ldots, n_l, j = 1, \ldots, n_{l+1}), l = 1, \ldots, L-1$ and biases $b_i^l(i = 1, \ldots, n_l, l = 2, \ldots, L)$:

$$\begin{aligned} I_1\text{: } w_{ij}^{ll+1} &= 2(\text{rand}(0,1) - 0.5) \\ b_i^l &= 2(\text{rand}(0,1) - 0.5) \end{aligned} \tag{12}$$

$$\begin{aligned} I_2\text{: } w_{ij}^{ll+1} &= 4(\text{rand}(0,1) - 0.5) \\ b_i^l &= 4(\text{rand}(0,1) - 0.5) \end{aligned} \tag{13}$$

$$\begin{aligned} I_3\text{: } w_{ij}^{ll+1} &= \frac{2\sqrt{6}(\text{rand}(0,1) - 0.5)}{\sqrt{n_l + n_{l+1}}} \\ b_i^l &= 2(\text{rand}(0,1) - 0.5) \end{aligned} \tag{14}$$

where rand(0, 1) denotes a real number randomly generated in the value range between 0 and 1. Here, I_1 and I_2 are the two commonly used initialization methods, and I_3 is a special method dedicated to the MNIST data set [46].

We evaluate the performance of BPCC_ND_MS_SaDE in terms of its finally achieved training objective function value and compare it with that obtained by BP under each of the three initialization methods (I_1, I_2, and I_3) for three network architectures, A_1 A_2, and A_3, respectively. In each test scenario (i.e., a specific initialization method and a specific network architecture), both BPCC_ND_MS_SaDE and BP are executed for 30 runs, where, in each run, both methods start with the same initialization. The parameter setting of BPCC_ND_MS_SaDE is the same as that in the previous experiment. BP's parameters are set the same as their counterparts in BPCC_ND_MS_SaDE.

Fig. 4 illustrates the box plot comparison of experimental results, and Fig. 4(a)–(c) refers to network architectures A_1, A_2, and A_3, respectively, and the left, middle, and right parts of each subfigure compare BP and BPCC_ND_MS_SaDE via the box plots of their eventually obtained training objective function values over 30 runs for initialization methods I_1, I_2, and I_3, respectively. It is observed that in many test scenarios, such as (A_1, I_2), (A_1, I_3), (A_2, I_2), and (A_3, I_2), BP demonstrates severe initialization sensitivity via large performance variation over 30 runs. Furthermore, the significant performance difference is observed when BP employs different initialization methods under any specific network architecture (especially A_1). In contrast, BPCC_ND_MS_SaDE shows much smaller performance variation across all test scenarios and much more consistent performance with respect to different initialization methods. Meanwhile, the median value of 30 experimental results achieved by BPCC_ND_MS_SaDE consistently outperforms that obtained by BP across all test scenarios. These comparisons demonstrate that BPCC_ND_MS_SaDE is more robust (i.e., less sensitivity to initialization) and effective (i.e., less prone to getting stuck into inferior local optima) than BP.

BPCC_ND_MS_SaDE hybridizes CC_ND_MS_SaDE with BP, where BP acts as the backbone and CC is only triggered to execute once some conditions are met. To gain a better insight into how BPCC_ND_MS_SaDE works, Fig. 5 compares its convergence curve in one execution run with that of BP for each of the three network architectures A_1, A_2, and A_3, where I_3 is used for initialization and both methods in comparison start with the same initialization. It is observed that in the beginning, BPCC_ND_MS_SaDE and BP share the same convergence curve until the first time CC is triggered. The execution of CC helps further reduce the training objective function value from that obtained by BP in the corresponding training epoch, depicted by a vertical drop in the convergence curve (denoted by the dashed circle).

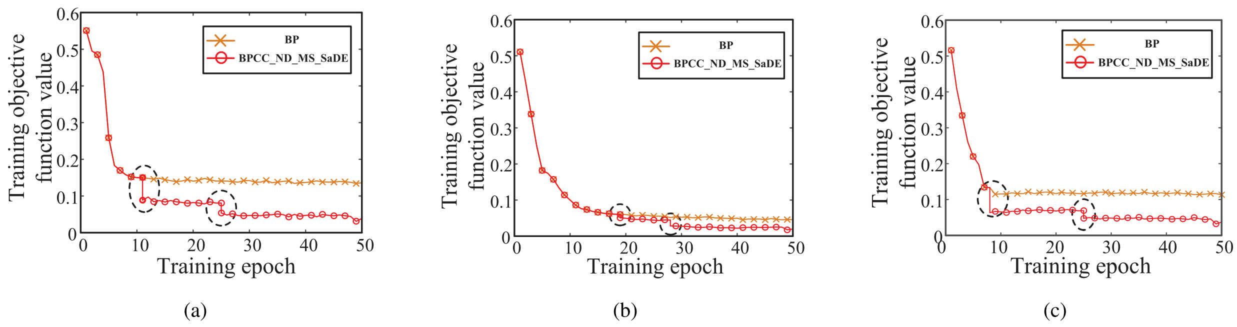


Fig. 5. Comparison of the convergence curve of BPCC_ND_MS_SaDE in one execution run with that of BP for network architectures A_1, A_2, and A_3 shown in (a)–(c), respectively, where I_3 is used for initialization, and both methods start with the same initialization. Dashed circle depicts CC being triggered for execution.

TABLE I

Relative Performance of CC_ND_MS_SaDE (i.e., Standalone CC) With Its Total Number of Iterations Set to 3 in Contrast to BPCC_ND_MS_SaDE

Network Architecture	Training Objective Function Value	Computation Time
A_1	5.7	1.48
A_2	5.0	1.49
A_3	6.1	1.50

When using CC, the objective value decreases lower than that of BP in the final epoch. Since the local search capability of BP is better than CC, this phenomenon means that CC can help in global search. For different network architectures, CC may be triggered in different training epochs, as shown in Fig. 5. It may also be triggered for different numbers of times although, in the current experiment, it was only triggered twice with respect to all three architectures.

Compared with one BP training epoch, one CC iteration is often much more time consuming. Therefore, it is computationally prohibitive to execute a pure CC-based learning method, such as CC_ND_MS_SaDE, for a sufficient number of iterations. Given this fact, we restrict the total number of iterations to 3 for CC_ND_MS_SaDE and compare it with BPCC_ND_MS_SaDE under the same experimental setup as that used to produce Fig. 5, where CC_ND_MS_SaDE employs the same parameter setting as the related setting in BPCC_ND_MS_SaDE. The reason for restricting the total number CC iterations to 3 is to allow CC_ND_MS_SaDE to have one more CC iteration than BPCC_ND_MS_SaDE, which involves two CC iterations. In this way, we intend to make a fair comparison of the performances of BPCC and standalone CC under a reasonable computational budget. Table I reports the relative performance of CC in contrast to BPCC under each of the three network architectures A_1, A_2, and A_3, which is calculated by $\mathrm{Perf}_{\mathrm{CC}}/\mathrm{Perf}_{\mathrm{BPCC}}$, where $\mathrm{Perf}_{\mathrm{CC}}$ and $\mathrm{Perf}_{\mathrm{BPCC}}$ refer to certain performance metrics (i.e., the training objective function value and computation time) for CC and BPCC, respectively. It is observed that the training objective function value achieved by CC is about $5 \sim 6$ times larger than that of BPCC although the computation time of CC is around 50% longer than that of BPCC. These results demonstrate the superiority of BPCC over standalone CC.

D. Comparison of the Proposed Method With Common-Practice BP-Based Methods

The experiments described in Section IV-C have demonstrated that the proposed BPCC_ND_MS_SaDE outperforms the standalone BP and standalone CC in terms of the finally achieved training objective function value. In this section, we will further evaluate the proposed method for image classification applications in terms of both training and testing performances on image data sets MNIST and CIFAR-10 and compare the proposed method with BP by incorporating two widely used gradient descent algorithms: SGD and ADAM. In addition to MFNs used in previous experiments, this experiment will also consider another popular category of DNNs (i.e., CNNs) due to its prevalence in image classification applications.

We first compare the training and test errors achieved by BPCC_ND_MS_SaDE and BP on MNIST by, respectively, using SGD and ADAM as the gradient descent algorithm in both methods for different initialization methods and different network architectures. Specifically, three initialization methods I_1, I_2, and I_3 are considered in this experiment. In addition to the three MFNs, i.e., A_1, A_2, and A_3, employed in previous experiments, we further consider three CNNs with different architectures, i.e., A_4: I-C1(5×5, 6)-S1(2)-C2(5×5, 12)-S2(2)-10, A_5: I-C1(5 × 5, 6)-S1(2)-C2(5 × 5, 12)-S2(2)-C3(4 × 4, 120)-10, and A_6: I-C1(5 × 5, 6)-S1(2)-C2(5 × 5, 12)-100-10, which have been used in existing research for MNIST. Here, $\mathrm{C}1(a \times a, b)$ represents a convolution operation, where a convolutional kernel of size $a \times a$ is applied to the input feature maps (i.e., the output of the previous layer or the original image) to yield b feature maps as output. Note that the convolutional kernel is inherently 3-D where the size of the third dimension is always equal to the total number of input feature maps and, thus, omitted. $\mathrm{S}(a)$ denotes a pooling operation with a filter of size $a \times a$. I stands for an identity mapping that takes as input the original image and outputs itself. The last one or two numbers (e.g., 10 and 100) represent the number of neurons in the fully connected layer.

In each test scenario that corresponds to a specific network architecture, a specific gradient descent algorithm, and a specific initialization method, the two methods in comparison are executed for five runs, where, in each run, both methods

TABLE II

COMPARISON OF BPCC_ND_MS_SADE AND BP IN TERMS OF THE AVERAGE CLASSIFICATION ERROR RATE (%) OVER FIVE RUNS ON THE TRAINING AND TEST SETS OF THE MNIST DATA SET BY, RESPECTIVELY, USING SGD AND ADAM AS THE GRADIENT DESCENT ALGORITHM IN BOTH METHODS WITH RESPECT TO THREE INITIALIZATION METHODS (I_1, I_2, AND I_3) AND SIX NETWORK ARCHITECTURES (A_1, A_2, A_3, A_4, A_5, AND A_6)

Methods			Network Architectures											
			A_1		A_2		A_3		A_4		A_5		A_6	
			$Train$	$Test$	$Train$	$Test$	$Train$	$Test$	$Train$	$Test$	$Train$	$Test$	$Train$	$Test$
I_1	SGD	BP	3.17	4.06	5.09	5.25	5.82	6.39	2.65	2.78	4.01	4.75	2.34	2.50
		BPCC_ND_MS_SaDE	**2.37**	**2.98**	**2.83**	**3.65**	**4.54**	**5.37**	**1.02**	**1.97**	**2.48**	**3.39**	**1.21**	**1.63**
	ADAM	BP	2.73	3.46	5.47	5.65	5.22	6.09	1.64	2.38	3.51	3.61	1.28	1.90
		BPCC_ND_MS_SaDE	**1.93**	**2.59**	**2.61**	**3.48**	**3.29**	**5.19**	**1.06**	**1.61**	**2.13**	**3.04**	**0.86**	**1.35**
I_2	SGD	BP	11.25	14.74	8.18	9.47	10.18	12.07	9.29	11.47	13.18	14.16	16.19	18.34
		BPCC_ND_MS_SaDE	**5.92**	**8.40**	**3.17**	**5.83**	**5.71**	**8.14**	**4.52**	**6.23**	**7.29**	**8.16**	**6.36**	**10.86**
	ADAM	BP	9.89	11.76	7.04	8.13	7.36	10.86	7.63	8.43	8.92	10.94	12.15	13.65
		BPCC_ND_MS_SaDE	**7.27**	**9.05**	**5.05**	**6.91**	**6.43**	**8.84**	**4.82**	**6.26**	**6.85**	**8.15**	**8.16**	**10.31**
I_3	SGD	BP	2.78	2.96	3.02	3.18	4.28	4.15	0.92	1.06	0.97	0.93	1.09	1.21
		BPCC_ND_MS_SaDE	**1.07**	**2.22**	**1.72**	**2.42**	**2.62**	**3.07**	**0.71**	**0.91**	**0.63**	**0.84**	**0.72**	**0.98**
	ADAM	BP	1.93	2.71	2.61	2.94	3.51	3.52	0.88	1.02	0.74	0.83	0.81	1.04
		BPCC_ND_MS_SaDE	**1.87**	**2.34**	**2.08**	**2.67**	**2.16**	**3.11**	**0.73**	**0.92**	**0.59**	**0.75**	**0.74**	**0.95**

TABLE III

COMPARISON OF BPCC_ND_MS_SADE AND BP IN TERMS OF THE AVERAGE CLASSIFICATION ERROR RATE (%) OVER FIVE RUNS ON THE TRAINING AND TEST SETS OF THE CIFAR-10 DATA SET BY, RESPECTIVELY, USING SGD AND ADAM AS THE GRADIENT DESCENT ALGORITHM IN BOTH METHODS WITH RESPECT TO THREE INITIALIZATION METHODS (I_1, I_2, AND I_3) AND THREE NETWORK ARCHITECTURES (A_7, A_8, AND A_9)

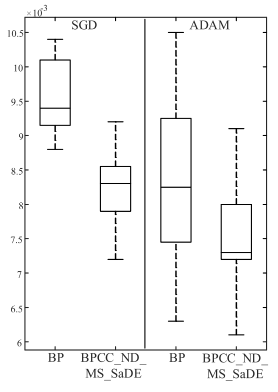
Methods			Network Architectures					
			A_7		A_8		A_9	
			$Train$	$Test$	$Train$	$Test$	$Train$	$Test$
I_1	SGD	BP	45.32	47.59	42.19	45.28	39.04	41.46
		CCBP_ND_MS_SaDE	**33.26**	**39.76**	**32.71**	**38.89**	**30.27**	**35.80**
	ADAM	BP	36.48	41.48	34.88	39.76	31.25	35.04
		CCBP_ND_MS_SaDE	**26.57**	**34.11**	**27.13**	**34.06**	**22.62**	**30.91**
I_2	SGD	BP	65.22	70.78	61.58	65.32	53.07	61.37
		BPCC_ND_MS_SaDE	**49.81**	**55.82**	**43.51**	**49.07**	**40.93**	**47.75**
	ADAM	BP	56.31	61.23	50.18	56.19	48.04	53.08
		BPCC_ND_MS_SaDE	**44.36**	**52.46**	**42.33**	**48.15**	**37.27**	**45.90**
I_3	SGD	BP	40.39	42.94	43.17	45.68	36.57	38.76
		BPCC_ND_MS_SaDE	**35.20**	**38.19**	**35.12**	**38.52**	**32.71**	**35.09**
	ADAM	BP	33.15	36.18	33.84	38.47	30.56	34.04
		BPCC_ND_MS_SaDE	**28.30**	**33.66**	**29.03**	**34.21**	**28.23**	**31.58**

have the same initialization. We allow certain parameters of BPCC_ND_MS_SaDE to be fine-tuned for different network architectures and different gradient descent algorithms, i.e., the learning rate (η) of the gradient descent algorithm and the threshold parameter (τ) for triggering CC, aiming to reveal the best achievable performance. The remaining parameters of BPCC_ND_MS_SaDE are set same in all test scenarios as follows: $\#maxBPEpoch = 50$, $\#maxCCIter = 1$, $\#maxEAGen = 50$, $\alpha = 0.3$, $\beta = 0.00005$, $\beta_s = 0.005$, $\sigma_s = 0.0005$, $Q = 30\% * R$, $\mathrm{SZ}_D = 10\,000$, $\mathrm{SZ}_{D_s} = 20\% * \mathrm{SZ}_D$, NP $= 100$, and LP $= 20$.

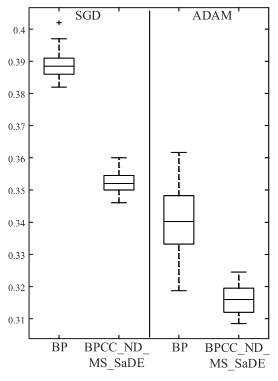
Across all test scenarios, the parameters of BP are set the same as their counterparts in BPCC_ND_MS_SaDE to ensure a fair comparison. Table II reports the average classification errors over five runs obtained on the training and test sets of the MNIST data set. It is observed that on both training and test sets, BPCC_ND_MS_SaDE consistently outperforms the compared common-practice learning techniques, i.e., BP with SGD and BP with ADAM, across all test scenarios.

We then compare the training and test errors of BP and BPCC_ND_MS_SaDE on CIFAR-10, following the same experimental setup as that used for MNIST. In this experiment, we merely consider CNNs because of its dominant utilization of CIFAR-10. We use three CNNs with different architectures, i.e., A_7: I-C1(5 × 5, 32)-S1(2)-C2(5 × 5, 64)-S2(2)-C3(5 × 5, 240)-10, A_8: I-C1(5 × 5, 32)-S1(2)-C2(5 × 5, 64)-S2(2)-C3(5 × 5, 120)-100-10, and A_9: I-C1(5 × 5, 32)-S1(2)-C2(5 × 5, 64)-S2(2)-100-10, which have been used in existing research for CIFAR-10. The way to set the parameters of BPCC_ND_MS_SaDE and BP is the same as that used in the previous experiment. Table III reports the average classification errors over five runs obtained on the training and test sets of the CIFAR-10 data set. It is observed again that on both training and test sets, BPCC_ND_MS_SaDE consistently outperforms the compared common-practice learning techniques, i.e., BP with SGD and BP with ADAM, across all test scenarios.

The abovementioned experiments indicate that BPCC_ND_MS_SaDE is superior over common-practice



(a)



(b)

Fig. 6. Comparison of the box plots of the test errors obtained by BP and BPCC_ND_MS_SaDE over 30 runs by incorporating SGD and ADAM, respectively, with respect to (a) A_5 on MNIST and (b) A_9 on CIFAR-10.

TABLE IV

STATISTICAL COMPARISON OF BPCC_ND_MS_SADE AND BP VIA t-TEST AT THE SIGNIFICANT LEVEL OF 0.05 AND COHEN'S d EFFECT SIZE, WHERE $H = 1$ INDICATES THAT BPCC_ND_MS_SADE PERFORMS STATICALLY SIGNIFICANTLY BETTER THAN BP

		H	p-value	d
A_5 (MNIST)	SGD	1	$1.60 \times 10^{-8}\%$	2.78
	ADAM	1	1.22%	8.24
A_9 (CIFAR-10)	SGD	1	$1.32 \times 10^{-21}\%$	0.87
	ADAM	1	$1.72 \times 10^{-9}\%$	2.97

BP learning techniques in image classification applications. To further consolidate such a finding, we select, according to Tables II and III, network architectures A_5 and A_9 with initialization method I_3, which can lead to the superior performance on MINIST and CIFAR-10, respectively, execute both BPCC_ND_MS_SaDE and BP under each of these two network architectures for 30 runs and make the box plot comparison as well as conduct the t-test [47] to reveal performance distinction in a more statistically rigid way. A t-test is a type of inferential statistic used to determine if there is a significant difference between the means of two groups, which may be related to certain features. Fig. 6 compares the box plots of the test errors obtained by BP and BPCC_ND_MS_SaDE over 30 runs by incorporating SGD and ADAM, respectively. It is observed that in all comparison scenarios, BPCC_ND_MS_SaDE outperforms BP in terms of both the average and stability of performance, where ADAM performs better than SGD in terms of median performance but is less stable. Table IV reports the t-test results at the significant level of 0.05 and Cohen's d to measure the effect size [48]. It is observed that BPCC_ND_MS_SaDE outperforms BP by a significant amount in all comparison scenarios, indicated by $H = 1$ and Cohen's d of above 0.8 across all scenarios.

V. CONCLUSION

We proposed a DNN learning framework that hybridizes CC-based optimization with BP-based gradient descent, named BPCC, and devising a computationally efficient CC-based optimization method dedicated to DNN parameter learning to implement the proposed framework, aiming to leverage on the advantages of two different types of parameter learning techniques while mitigating their drawbacks to augment learning performance. In BPCC, BP will intermittently execute for multiple training epochs. In CC-based optimization, the overall parameter learning task is decomposed into many subtasks of learning a small portion of parameters. These subtasks will be selectively addressed in a cooperative mode as per some definition of task priority. We implemented the proposed framework into a learning method called BPCC_ND_MS_SaDE, where ND, MS, and SaDE represent ND, MS, and CC based upon SaDE, respectively. The experiments on two widely used image data sets MNIST and CIFAR-10 demonstrated that the proposed method consistently outperformed common-practice BP-based parameter learning methods. Our future work will be focused on further improving the proposed learning framework and its implementations, e.g., design of decomposition strategies for more categories of DNNs, design of more effective selection strategies, an improvement on CC-based optimization, and development of parallel implementations on high-performance computing facilities [49] to speed up computation.

REFERENCES

[1] G. Hinton *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, Nov. 2012.

[2] D. E. Rumelhart, G. E. Hinton, R. J. Williams, "Learning internal representations by error propagation" in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1. Cambridge, MA, USA: Bradford Books, 1986, pp. 318–362.

[3] H. Robbins and S. Monro, "A stochastic approximation method," *Ann. Math. Statist.*, vol. 22, no. 3, pp. 400–407, 1985.

[4] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*. [Online]. Available: http://arxiv.org/abs/1412.6980

[5] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *Int. J. Uncertainty, Fuzziness Knowl.-Based Syst.*, vol. 06, no. 2, pp. 107–116, Apr. 1998.

[6] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 20, no. 1, pp. 30–42, Jan. 2012.

[7] 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.

[8] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015, *arXiv:1502.03167*. [Online]. Available: http://arxiv.org/abs/1502.03167

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[10] M. D. Zeiler, "ADADELTA: An adaptive learning rate method," 2012, *arXiv:1212.5701*. [Online]. Available: http://arxiv.org/abs/1212.5701

[11] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural Networks: Tricks of the Trade*. Berlin, Germany: Springer, 1998, pp. 9–50.

[12] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," *J. Mach. Learn. Res.*, vol. 9, pp. 249–256, May 2010.

[13] T. Bäck and H.-P. Schwefel, "An overview of evolutionary algorithms for parameter optimization," *Evol. Comput.*, vol. 1, no. 1, pp. 1–23, Mar. 1993.

[14] J. J. Liang, S. Baskar, P. N. Suganthan, and A. K. Qin, "Performance evaluation of multiagent genetic algorithm," *Natural Comput.*, vol. 5, no. 1, pp. 83–96, Mar. 2006.

[15] X. Yao, "Evolving artificial neural networks," *Proc. IEEE*, vol. 87, no. 9, pp. 1423–1447, Sep. 2002.

[16] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, pp. 281–305, Feb. 2012.

[17] F. Assunção, N. Lourenço, P. Machado, and B. Ribeiro, "Evolving the topology of large scale deep neural networks," in *Proc. Eur. Conf. Genetic Program.* Cham, Switzerland: Springer, 2018, pp. 19–34.

[18] M. Arjovsky, A. Shah, and Y. Bengio, "Unitary evolution recurrent neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1120–1128.

[19] T. Desell, "Large scale evolution of convolutional neural networks using volunteer computing," in *Proc. Genetic Evol. Comput. Conf. Companion (GECCO)*, 2017, pp. 127–128.

[20] J. Liu, M. Gong, Q. Miao, X. Wang, and H. Li, "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.

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

[22] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, 2019, pp. 4780–4789.

[23] Y. Zhou, G. G. Yen, and Z. Yi, "Evolutionary compression of deep neural networks for biomedical image segmentation," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Sep. 13, 2019, doi: 10.1109/TNNLS.2019.2933879.

[24] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 2962–2970.

[25] M. A. Potter and K. A. De Jong, "A cooperative coevolutionary approach to function optimization," in *Proc. Int. Conf. Parallel Problem Solving Nature*, 1994, pp. 249–257.

[26] Z. Yang, Y. Ding, Y. Jin, and K. Hao, "Immune-endocrine system inspired hierarchical coevolutionary multiobjective optimization algorithm for IoT service," *IEEE Trans. Cybern.*, vol. 50, no. 1, pp. 164–177, Jan. 2020.

[27] Z. Yang, Y. Jin, and K. Hao, "A bio-inspired self-learning coevolutionary dynamic multiobjective optimization algorithm for Internet of Things services," *IEEE Trans. Evol. Comput.*, vol. 23, no. 4, pp. 675–688, Aug. 2019.

[28] N. García-Pedrajas, C. Hervás-Martínez, and J. Muñoz-Pérez, "COVNET: A cooperative coevolutionary model for evolving artificial neural networks," *IEEE Trans. Neural Netw.*, vol. 14, no. 3, pp. 575–596, May 2003.

[29] S. Chand and R. Chandra, "Cooperative coevolution of feed forward neural networks for financial time series problem," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2014, pp. 202–209.

[30] A. P. Topchy and O. A. Lebedko, "Neural network training by means of cooperative evolutionary search," *Nucl. Instrum. Methods Phys. Res. A, Accel. Spectrom. Detect. Assoc. Equip.*, vol. 389, nos. 1–2, pp. 240–241, Apr. 1997.

[31] A. K. Qin, V. L. Huang, and P. N. Suganthan, "Differential evolution algorithm with strategy adaptation for global numerical optimization," *IEEE Trans. Evol. Comput.*, vol. 13, no. 2, pp. 398–417, Apr. 2009.

[32] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Netw.*, vol. 2, no. 5, pp. 359–366, Jan. 1989.

[33] 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.

[34] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[35] 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, Jul. 2006.

[36] H. Leung and S. Haykin, "The complex backpropagation algorithm," *IEEE Trans. Signal Process.*, vol. 39, no. 9, pp. 2101–2104, Sep. 1991.

[37] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning Internal Representations by Error Propagation*. Cambridge, MA, USA: MIT Press, 1988.

[38] Y. Sun, G. G. Yen, and Z. Yi, "Evolving unsupervised deep neural networks for learning meaningful representations," *IEEE Trans. Evol. Comput.*, vol. 23, no. 1, pp. 89–103, Feb. 2019.

[39] L. Feng, Y.-S. Ong, S. Jiang, and A. Gupta, "Autoencoding evolutionary search with learning across heterogeneous problems," *IEEE Trans. Evol. Comput.*, vol. 21, no. 5, pp. 760–772, Oct. 2017.

[40] 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.

[41] C. Zhang, P. Lim, A. K. Qin, and K. C. Tan, "Multiobjective deep belief networks ensemble for remaining useful life estimation in prognostics," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 10, pp. 2306–2318, Oct. 2017.

[42] B. A. Olshausen and D. J. Field, "Emergence of simple-cell receptive field properties by learning a sparse code for natural images," *Nature*, vol. 381, no. 6583, pp. 607–609, Jun. 1996.

[43] R. Storn and K. Price, "Differential evolution: A simple and efficient heuristic for global optimization over continuous spaces," *J. Global Optim.*, vol. 13, no. 4, pp. 341–359, 1997.

[44] A. K. Qin, X. Li, H. Pan, and S. Xia, "Investigation of self-adaptive differential evolution on the CEC-2013 real-parameter single-objective optimization testbed," in *Proc. IEEE Congr. Evol. Comput.*, Jun. 2013, pp. 1107–1114.

[45] M. Ranzato, C. Poultney, S. Chopra, and Y. LeCun, "Efficient learning of sparse representations with an energy-based model," in *Proc. Adv. Neural Inf. Process. Syst.*, Cambridge, MA, USA, 2006, pp. 1137–1144.

[46] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2010, pp. 249–256.

[47] T. G. Dietterich, "Approximate statistical tests for comparing supervised classification learning algorithms," *Neural Comput.*, vol. 10, no. 7, pp. 1895–1923, Oct. 1998.

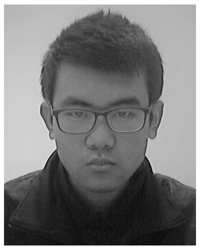
[48] K. Kelley and K. J. Preacher, "On effect size," *Psychol. Methods*, vol. 17, no. 2, p. 137, 2012.

[49] T. H. Wong, A. K. Qin, S. Wang, and Y. Shi, "cuSaDE: A CUDA-based parallel self-adaptive differential evolution algorithm," in *Proc. IES*, 2014, pp. 375–388.

Maoguo Gong (Senior Member, IEEE) received the B.S. degree (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, where he was promoted as an Associate Professor and as a Full Professor in 2008 and 2010, respectively, with exceptive 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 also an Associate Editor of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION and the IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS.



Jia Liu received the B.S. degree in intelligence science and technology and the Ph.D. degree in pattern recognition and intelligent systems 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.

A. K. Qin (Senior Member, IEEE) received the Ph.D. degree from the Nanyang Technological University, Singapore, in 2007.

He was with the University of Waterloo, Waterloo, ON, Canada, and the French Institute for Research in Computer Science and Automation, Rocquencourt, France, from 2007 to 2012. From 2012 to 2017, he was a Lecturer with the School of Science, RMIT University, Melbourne VIC, Australia. He is currently an Associate Professor with the Department of Computer Science and Software Engineering and the Data Science Research Institute, Swinburne University of Technology, Melbourne. He has authored over 60 publications. Two of his coauthored journal articles are the most cited ones (Web of Science) among articles published in the 2006 and 2009 IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION (TEVC), respectively. His current research interests include evolutionary computation, machine learning, image processing, GPU computing, and service computing.

Dr. Qin's 2009 article won the 2012 IEEE TEVC Outstanding Paper Award. One of his conference papers had been nominated for the Best Paper Award at the 2012 Genetic and Evolutionary Computation Conference. He is also chairing the IEEE Computational Intelligence Society (CIS) Task Force on collaborative learning and optimization.

Kun Zhao received the B.S. degree from the School of Science, Yanshan University, Qinhuangdao, China, in 2015. He is currently pursuing the M.E. degree in circuit and system with the School of Electronic Engineering, Xidian University, Xi'an, China.

His current research interests include computational intelligence and image understanding.

Kay Chen Tan (Fellow, IEEE) received the B.Eng. (Hons.) and Ph.D. degrees from the University of Glasgow, Glasgow, U.K., in 1994 and 1997, respectively.

He is currently a Professor with the Department of Computer Science, City University of Hong Kong, Hong Kong. He is actively pursuing research in artificial/computational intelligence and machine learning, with applications to evolutionary multi-objective optimization, data analytics, prognostics, BCI, operational research, and so on.

Dr. Tan is an elected AdCom Member of the IEEE Computational Intelligence Society (2014–2019) and an IEEE Distinguished Lecturer (2011–2013 and 2015–2017). He received the 2016 IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS Outstanding Paper Award. He was the awardee of the 2012 IEEE Computational Intelligence Society Outstanding Early Career Award for his contributions to evolutionary computation in multiobjective optimization. He also received the Recognition Award in 2008 from the International Network for Engineering Education and Research (iNEER). He also serves on the editorial board of over ten international journals, such as the IEEE TRANSACTIONS ON CYBERNETICS and *Evolutionary Computation*. He was the General Co-Chair of the IEEE World Congress on Computational Intelligence (WCCI) 2016 in Vancouver, Canada, and the IEEE Congress on Evolutionary Computation (CEC) 2019 in Wellington, New Zealand. He was the Editor-in-Chief (EiC) of the *IEEE Computational Intelligence Magazine*. He is also the EiC of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION.