Multi-Objective Evolutionary Federated Learning

Hangyu Zhu and Yaochu Jin, *Fellow, IEEE*

***Abstract*— Federated learning is an emerging technique used to prevent the leakage of private information. Unlike centralized learning that needs to collect data from users and store them collectively on a cloud server, federated learning makes it possible to learn a global model while the data are distributed on the users' devices. However, compared with the traditional centralized approach, the federated setting consumes considerable communication resources of the clients, which is indispensable for updating global models and prevents this technique from being widely used. In this paper, we aim to optimize the structure of the neural network models in federated learning using a multi-objective evolutionary algorithm to simultaneously minimize the communication costs and the global model test errors. A scalable method for encoding network connectivity is adapted to federated learning to enhance the efficiency in evolving deep neural networks. Experimental results on both multilayer perceptrons and convolutional neural networks indicate that the proposed optimization method is able to find optimized neural network models that can not only significantly reduce communication costs but also improve the learning performance of federated learning compared with the standard fully connected neural networks.**

***Index Terms*— Communication cost, deep neural networks, federated learning, multi-objective evolutionary optimization, neural architecture search.**

I. Introduction

THE usage of smart phones has dramatically increased over the last decades [1]. Compared with classic PC devices, smart phones are more portable and user-friendly. Using smart phones has already become a significant part of modern people's daily life, while billions of data transferred between smart phones provide a great support for training machine learning models. However, traditional centralized machine learning requires local clients, e.g., smart phone users to upload their data directly to the central server for model training, which may cause severe private information leakages.

An emerging technology called federated learning [2] was proposed recently to allow the central server to train a good global model, while maintaining the training data to be distributed on the clients' devices. Instead of sending data directly to the central server, each local client downloads the current global model from the server, updates the shared model by training its local data, and then uploads the updated global model back to the server. By avoid sharing local private data, users' privacy can be effectively protected in federated learning.

Some research has been dedicated to further protect users' privacy and security in federated learning. Bonawitz *et al.* [3] gives an overview of cryptographic techniques like homomorphic encryption [4] to encrypt the uploaded information before averaging. Different from traditional encryption methods, differential privacy [5], which is used to decrease individuals' information influences when querying specific data repository, protects the privacy of deep learning by adding Gaussian noise [6]. This privacy protection technology is also suited for federated learning [7], [8].

Apart from privacy issues, the statistical challenge is a barrier for federated optimization. Improving the shared global model in federated learning is sometimes similar to training the distributed model by data parallelism. McDonald *et al.* [9] proposed two distributed training strategies for structured perceptron like iterative error-dependent mixing or uniform parameter mixing. Adjusted parameter mixing strategies like fish matrix implementation [10] and elastic averaging stochastic gradient descent (SGD) [11] can further improve the convergence efficiency and robustness in the distributed model mixture. However, the aforementioned algorithms are built under the assumption that data on each local edge is independent and identically distributed (IID), and non-IID local data distribution was not considered. To address this problem, Zhao *et al.* [12] did some experiments on highly skewed non-IID data and provided statistically divergence analysis.

Federated learning requires massive communication resources compared to the classic centralized learning. A federated averaging (FedAvg) algorithm [2] introduced by McMahan *et al.* can improve communication efficiency by reducing local training minibatch sizes or increasing local training passes to reduce communication rounds. Shokri and Shmatikov used the method of uploading the gradients located in the particular interval clipped by some threshold values, which is similar to the idea of structured updates introduced in [13].

Another method to reduce the communication cost is to scale down the uploaded parameters by reducing the complexity of the neural network models. The early ideas of evolving artificial neural network were introduced in [14], where systematic neural network encoding methods were presented. However, most of them are direct encoding methods that are not easily scalable to deep neural networks having a large number of layers and connections. In order to address this issue, neuroevolution of augmenting topologies (NEAT) [15]

Manuscript received December 6, 2018; revised February 19, 2019 and May 15, 2019; accepted May 24, 2019. This work was supported in part by the Royal Society Exchanges Program, under Grant EC\NSFC\170279. *(Corresponding author: Yaochu Jin.)*

The authors are with the Department of Computer Science, University of Surrey, Guildford GU2 7XH, U.K. (e-mail: hangyu.zhu@surrey.ac.uk; yaochu.jin@surrey.ac.uk).

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

Digital Object Identifier 10.1109/TNNLS.2019.2919699

and undirect graph encoding [16] were proposed to enhance the flexibility of neural network encoding. Although they are able to substantially improve the encoding efficiency, both NEAT and cellular graph methods occupy too many computation resources. More recently, Mocanu *et al.* [17] proposed a sparse evolutionary training (SET) to reduce the search space in optimizing deep neural networks containing a large number of connections.

To reduce the communication costs without seriously degrading the global learning accuracy, this paper proposes a framework for optimizing deep neural network models in federated learning. The main contributions of this paper are as follows.

1) Federated learning is formulated as a biobjective optimization problem, where the two objectives are the minimization of the communication cost and the maximization of the global learning accuracy. This biobjective optimization is solved by a multi-objective evolutionary algorithm.
2) A modified SET algorithm is proposed to reduce the connections of neural networks, thereby indirectly reducing the number of model parameters to be sent to the server.

Our experimental results indicate that the proposed algorithm can significantly reduce the complexity of the neural network models at the expense of minor performance degradation of the global model, thereby reducing the server–client communication.

The rest of this paper is organized as follows. Section II introduces the related background. A detailed description of the proposed algorithms is given in Section III. In Section IV, the experimental results are presented and discussed. Finally, this paper is concluded in Section V.

II. Preliminaries

In this section, we briefly review the basics of multilayer perceptron (MLP) and convolutional neural networks (CNNs), federated learning, and evolutionary optimization of neural networks.

A. Multilayer Perceptron Neural Networks

MLPs [18] are the most commonly used feedforward artificial neural networks containing at least three layers: the input layer, one hidden layer, and the output layer. Typically, nodes or neurons located between each layer of the MLP are fully connected without internal loops and use activation functions for the purpose of nonlinear projection and feature extraction upon outputs from the previous layer.

Fig. 1 shows an illustrative example of a fully connected MLP. Circles in solid lines in this figure represent "neurons," and circles in dashed line represent "biases." In the feed-forward propagation of a fully connected neural network, each node or neuron receives a weighted sum of the inputs of all preceding neurons plus a bias value as its input. Then, the output of this neuron is computed by a nonlinear activation function σ as follows:

$$y_{\text{neuron}} = \sigma\left(\sum_{i=1}^{N} x_i \cdot w_i + b\right) = \sigma(x^T w + b). \tag{1}$$

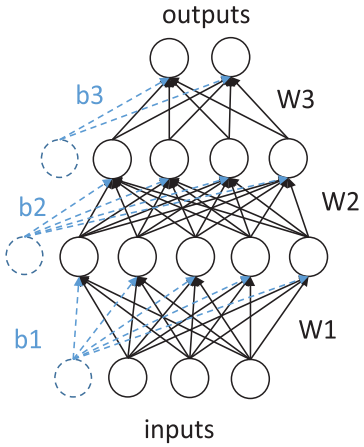

Fig. 1. This MLP neural network contains an input layer, two hidden layers, and an output layer. Solid circles: neurons. Dashed ones: biases.

When the feed-forward propagation passes through one or more hidden layers to the output layer, a predicted target $\hat{y}$ is achieved to compute the loss function $\ell(\hat{y}, y)$, which is typically the difference between the desired output y and predicted output $\hat{y}$. If we use θ to replace both weights and biases, the loss function can be reformulated as $\ell(\theta)$, and then the neural network tries to optimize the trainable parameter θ by minimizing the loss $\ell(\theta)$

$$\min_{\theta} \ \ell(\theta) = \frac{1}{N}\sum_{i} \ell(\theta, x_i) \quad x_i \in \{x_1, x_2 \ldots, x_N\} \tag{2}$$

where x_i is the ith training sample (can be a vector), and N is the size of training data. The objective is to find a specific parameter θ to minimize the *expected* loss through N data samples.

Gradient descent (GD) is commonly used to train neural networks in the back-propagation by computing the partial derivative of a loss function $\ell(\theta)$ over the whole N data samples with respect to each element in θ. However, this approach takes a very long time to compute the gradient in each iteration if the total number of input samples is very large. The SGD algorithm is at another extreme compared to GD—it only randomly chooses one training sample per iteration, which, however, may cause instability in training. To strike a balance between computation efficiency and training stability, minibatch SGD is proposed to select a randomly chosen minibatch size of the training data for gradient computation in each training iteration

$$\begin{aligned} g_t &= \frac{1}{n}\nabla_\theta \ell(\theta, x_{i:i+n}) \\ \theta_{t+1} &= \theta_t - \eta g_t \end{aligned} \tag{3}$$

where n is the size of minibatch, η is the learning rate, and g_t is the *average* gradient over data samples $x_{i:i+n}$ with respect to the elements in θ_t in the tth iteration. The training of the neural network is to update the parameter θ by iteratively subtracting ηg_t from the current model parameter θ_t.

B. Convolutional Neural Networks

CNNs [19] are well suited for dealing with very high-dimensional inputs and have shown consistently better

performances than MLPs for image classification. CNNs share a similar topological architecture with MLPs, but several variations are made to CNNs based on the structure of MLPs.

A CNN generally has three kinds of layers: convolutional layers, pooling layers, and fully connected layers. The convolutional layer consists of numerous kernel filters that can be recognized as an array of square block neurons, where the real number inside each square neuron is equivalent to the connection in the MLP. The convolutional layer does the "convolution" operations on the previous layer, where the kernel filters can be seen as training weights. The CNN can be mathematically described as follows:

$$y_{ij}^{l} = \sigma\left(\sum_{a=0}^{n-1}\sum_{b=0}^{n-1} f_{\mathrm{ab}} x_{(i+a)(j+b)}^{l-1}\right) \tag{4}$$

where f_{ab} is a $n \times n$ kernel filter, l is the layer number, x^{l-1} is the input of the convolutional layer, y_{ij}^{l} is the output of the convolutional layer, and σ is the activation function. Specifically, we use the rectified linear unit (ReLu) as our hidden neuron's activation function to relieve the effect of gradient vanishing [20] and softmax function in the output nodes for multi-class classification tasks. Formulas of ReLu and softmax function are given as

$$\begin{aligned} \sigma_{\mathrm{ReLu}}(z) &= \max(0, z) \\ \sigma_{\mathrm{soft\,max}}(z_i) &= \frac{\exp(z_i)}{\sum_{i=1}^{C}\exp(z_i)} \end{aligned} \tag{5}$$

where z is the output of the previous layer, and C is the total number of label classes we need to classify.

A pooling layer can be added in the CNN after several convolutional layers for the extraction of specific features from the hidden representations. For instance, a dimension of $m \times m$ max pooling window is generally created to extract the maximum luminance value of pixels within the corresponding pooling window for further enhancing the represented features of the filtered images from the previous convolutional layer. Besides max pooling, average pooling is also commonly used by averaging the feature values within the window.

The fully connected layer is applied at the back of the CNN. It is exactly the same as a traditional neuron layer in the MLP, with its input being the flattened image pixels from the output of its preceding layer. The main purpose of this layer is to classify the extracted features from the previous layers in the CNN into various classes.

The aforementioned minibatch GD is also applicable to the CNN. It should be noticed that we only calculate the partial derivative of weights in the convolutional and fully connected layers and do nothing with pooling layers when performing the back-propagation optimization on CNNs. This is because the pooling operation does not contain any trainable parameters with respect to the derivatives of the back-propagation.

C. Federated Learning

Federated learning [21] is an emerging decentralized privacy-protection training technology that enables client edges to learn a shared global model without uploading their private local data to a central server. In each training round, a local device downloads a shared model from the global server cloud, trains the downloaded model over the individuals' local data, and then sends the updated weights or gradients back to the server. On the server, the uploaded models from the clients are aggregated to obtain a new global model. Compared with the traditional centralized learning, federated learning has the following unique features.

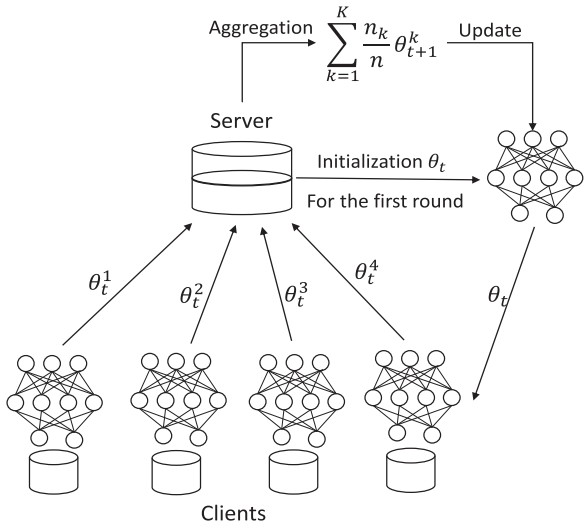

Fig. 2. Flowchart of federated learning. θ is the model parameters transferred between the server and the clients, n_k is the data size of client k, K is the total number of clients, and t is the communication round in federated learning. We just initialize global model parameters randomly at the beginning of the communication round and use updated model parameters afterward.

1) The training data are distributed on the local edges, which is not available to the global server cloud. However, the learned model is shared between the server and all clients.
2) Model training occurs on each local device instead of on the server. The server aggregates the local models uploaded from the clients to obtain a shared global model and send the global model back to the clients.
3) Federated learning has a much higher requirement on local computation powers and communication resources than the traditional centralized learning.

Similar to the learning algorithm of the MLP neural network, federated learning aims to minimize the loss function $\ell(\theta)$ but in a distributed scheme

$$\min_{\theta}\ \ell(\theta) = \sum_{k=1}^{K}\frac{n_k}{n}L_k(\theta) \quad \text{where } L_k(\theta) = \frac{1}{n_k}\sum_{i\in P_k}\ell_i(\theta) \tag{6}$$

where k is the index of K total clients, $L_k(\theta)$ is the loss function of kth local client, n_k is equal to the local data size, and P_k is the set of data indexes whose length is n_k, i.e., $n_k = |P_k|$. Optimizing the loss function $\ell(\theta)$ in federated learning is equivalent to minimizing the weighted average of local loss function $L_k(\theta)$.

The procedure of federated learning is shown in Fig. 2, where each client receives the parameters θ_t of the global model from the central server and then trains their individual local models using their own data. After local training, each local device sends their trained local parameters $\left(\text{i.e., } \theta_t^1\right)$ to

Algorithm 1 FedAvg. K Indicates the Total Numbers of Clients; B Is Size of Mini Batch; E Is Equal to Training Iterations; and η Is the Learning Rate

Server:
Initialize θ_t
for each communication round $t = 1, 2, \dots$ **do**
  Select $m = C \times K$ clients, $C \in (0, 1)$ clients
  Download θ_t to each client k
  for each client $k \in m$ **do**
    Wait **Client** k for synchronization
    $\theta_t = \sum_{k=1}^{m} \frac{n_k}{n} \theta^k$
  end for
end for
Client k**:**
$\theta^k = \theta_t$
for each iteration from 1 to E **do**
  for batch $b \in B$ **do**
    $\theta^k = \theta^k - \eta \nabla L_k(\theta^k, b)$
  end for
end for
return θ^k to server

the server to be aggregated to get an updated global model θ_{t+1} to be used for the next iteration's training. The subscript t denotes the time sequences or so-called communication rounds in federated learning.

The FedAvg algorithm [2] can effectively reduce communication rounds by simultaneously increasing local training epochs and decreasing local minibatch sizes in federated SGD (FedSGD) algorithm [2]. The pseudocode of FedAvg is presented in Algorithm 1, where θ^k are the model parameters of the kth client.

In Algorithm 1, n is the size of the whole data, and the global model parameter θ_t over tth communication round is calculated by a weighted average of θ^k from each client k. The client selection parameter C is a random number between 0 and 1 determining the total fraction of $C \times K$ clients allowed to update the shared global model.

The number of clients participating in federated learning may heavily affect the training performance, if the data on each clients do not cover the distribution of the overall data, which is very likely to happen in federated learning. As already found in [2], selecting more clients for training can speed up the convergence of the global model and enhance its performance, if the data on the participating clients in each communication round cannot cover the overall data distribution. More recently, it was theoretically shown [12] that the global weight convergence can be affected by the probability differences between the data distributed on client k and the whole data population, i.e., $\sum_{i=1}^{L} ||p^k(y = i) - p(y = i)||$, where L represents the total label classes, $p^k(y = i)$ is the probability of data occurrence corresponding to the label i for client k and $p(y = i)$ is that for the whole data population, respectively. Therefore, data distribution discrepancy on the client side is a root cause of the weight divergence, and clients with non-IID data are harder to train than those with IID data. Unfortunately, selecting the right number of participating clients is challenging in federated learning since the class balance, data distribution, and the amount of the data may vary a lot from client to client and also over time.

It should be mentioned that different observations have been made in other contexts of distributed learning where the data can be proactively divided over clients. For example, it was suggested in [9] that increasing the number of client shards may slow down the convergence of the weights in the IID environment. This happens because if the data distributed on the local devices, which are selected to communicate with the central server, can cover the whole data population, the client or replicas that have a larger data size converges to its optimum more quickly. Thus, the larger the number of client shards is, the smaller the expected amount of data that can be allocated to each client will be, if the whole data size is fixed. On the contrary, if the selected clients only hold a fraction of the whole training data, information deficiency of clients' data may cause a negative effect on convergence performance.

D. Elitist Nondominated Sorting Genetic Algorithm

We adopt the elitist nondominated sorting genetic algorithm (NSGA-II) [22], a widely used multi-objective evolutionary algorithm, to optimize the connectivity and hyperparameters of the neural network to simultaneously minimize the communication costs and maximize the global learning accuracy. NSGA-II is able to achieve a set of diverse Pareto optimal solutions by comparing the dominance relationships between the solutions in the population and a crowding distance calculated according to the distance between two neighboring solutions. The main procedure of NSGA-II can be summarized as follows.

1) *Step 1:* Randomly generate a parent population P_t of a size N for the first generation.
2) *Step 2:* Create an offspring population Q_t of the same size as the parent population P_t by using crossover and mutation operators on P_t. Merge P_t and Q_t into a combined population R_t, where $R_t = P_t \cup Q_t$ has a size of $2N$.
3) *Step 3:* Perform nondominated sorting to sort the combined population R_t into a number of nondominated fronts according to their dominance relationships. Thus, solutions in the same front are nondominated with each other. Once the combined population is sorted, calculate the crowding distance for each individual in the same nondominated front based on the distance to its neighboring solutions. Note that the solutions at the ends of each nondominated front are an infinite number so that they are always prioritized in selection.
4) *Step 4:* Generate the parent population for the next generation P_{t+1} by selecting N better solutions front by front from the sorted combined population R_t. If the number of solutions located in the selected last nondominated front is larger than that of solutions remains to be selected for P_{t+1}, the individuals with a larger crowding distance will be selected to promote the diversity of the population.
5) *Step 5:* Go to *step* 2 and repeat the whole procedure until a stop criterion is met.

NSGA-II is a powerful and robust multi-objective evolutionary algorithm for problems having two or three objectives. More recent evolutionary algorithms can be adopted if the number of objectives is larger than three, e.g., the evolutionary many-objective optimization algorithm using reference-point based nondominated sorting approach [23], the knee-driven evolutionary algorithm for many-objective optimization [24], and the reference vector-guided evolutionary for many-objective optimization [25]. Note also that computationally more efficient nondominated sorting algorithms can be used when the population size is large [26] or when the number of objectives is large [27].

The use of the NSGA-II to optimize the neural network model in federated learning will certainly increase the computational complexity of the algorithm. In NSGA-II, the fast nondominated sorting operation has a computational complexity of $O(mN^2)$ [28], where m is the number of objectives, and N is the number of populations. The computation complexity of crowding distance calculation is $O(mN \log N)$ in the worst case, when all the solutions are located in one nondominated front. Note, however, that in practice, the majority of the computational complexity mainly comes from a large number of time-consuming evaluations of the objective functions. For example, each evaluation of the objective functions in evolutionary optimization of the neural networks requires the training of the model, which can be computationally intensive if the amount of data is large. To address this issue, surrogate-assisted evolutionary optimization [29], [30] or Bayesian optimization [31] are helpful to reduce the computation cost.

III. Proposed Algorithm

In this section, we first introduce the modified SET algorithm. Then, we formulate federated learning as a biobjective optimization problem. This is followed by a description of the encoding scheme adopted by the evolutionary algorithm. Finally, the overall framework is presented.

A. Modified SET Algorithm

In evolutionary optimization of the structure of neural networks, the encoding scheme used by the evolutionary algorithm significantly affects the optimization efficiency. Direct binary encoding such as the one introduced in [32] needs a large connection matrix to represent the structure of a neural network, which is not scalable to neural networks containing multiple hidden layers and a large number of neurons. In order to enhance the scalability in evolving deep neural networks, we propose a modified SET [17] method to simultaneously improve the scalability and flexibility in evolving neural networks.

SET is different from typical methods for evolving the structure of neural networks. It does not directly encode the neural network and perform selection, crossover, and mutation as done in genetic algorithms [33]. Instead, SET starts from an initial Erdos Rényi random graph [34] that determines the connectivity between every two neighboring layers of the neural network. The connection probability between two layers is described as follows:

$$p\left(W_{ij}^{k}\right) = \frac{\varepsilon(n^{k} + n^{k-1})}{n^{k} n^{k-1}}$$
$$n^{W} = n^{k} n^{k-1} p\left(W_{ij}^{k}\right) \tag{7}$$

where n^k and n^{k-1} are the number of neurons in layer k and $k-1$, respectively, W_{ij}^{k} is the sparse weight matrix between the two layers, ε is a SET parameter that controls connection sparsity, and n^W is the total number of connections between the two layers. It is easy to find that the connection probability would become significantly lower, if $\varepsilon \ll n^k$ and $\varepsilon \ll n^{k-1}$.

Since the randomly initialized graph may not be suited for learning a particular data, Mocanu *et al.* suggest removing a fraction ξ of the weights with the smallest update during each training epoch, which can be seen as the selection operation of an evolutionary algorithm. However, removing the least important weights may cause fluctuation when minimizing the loss function using the minibatch SGD algorithm and this phenomenon turns out to be extremely severe in federated learning. To address this issue, we modify the operator by conducting the removal operation at the *last* training epoch only. Pseudocode of the modified SET is listed in Algorithm 2. By implementing the modified SET algorithm, a sparsely connected neural network can be evolved, resulting in much fewer parameters to be downloaded or uploaded, thereby reducing the communication cost in federated learning.

Algorithm 2 Modified SET Algorithm

```
Set ε and ξ
for each fully-connected layer of the neural network do
    Replace weight matrices by Erdos Rényi random graphs
        given by ε in (7)
end for
Initialize weights
Start training
for each training epoch do
    Training and updating corresponding weights
end for
for each weight matrix do
    Remove a fraction ξ of the smallest |weights|
end for
```

B. Objective Functions and Encoding of the Neural Networks

We reformulate federated learning as a two objective optimization problem [35]. One objective is the global model test error E_t, and the other is the model complexity Ω_t over the tth communication round. To minimize these two objectives, we evolve both the hyperparameters as well as the connectivity of the neural network models. The hyperparameters include the number of hidden layers, the number of neurons in each hidden layer, and the learning rate η of the minibatch SGD algorithm. The connectivity of the neural network is represented by the modified SET algorithm described in Algorithm 2, which consists of two parameters, namely, ε in (7), an integer, and the

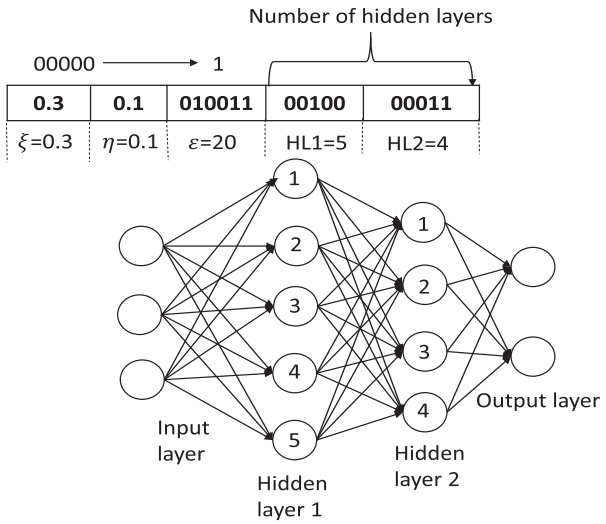

Fig. 3. Neural network and its chromosome. Note that when we decode the number of neurons, each variable will be increased by one to make sure that there is at least one neuron in a hidden layer. HL1 and HL2 denote that the neural network has two hidden layers containing five and four neurons, respectively.

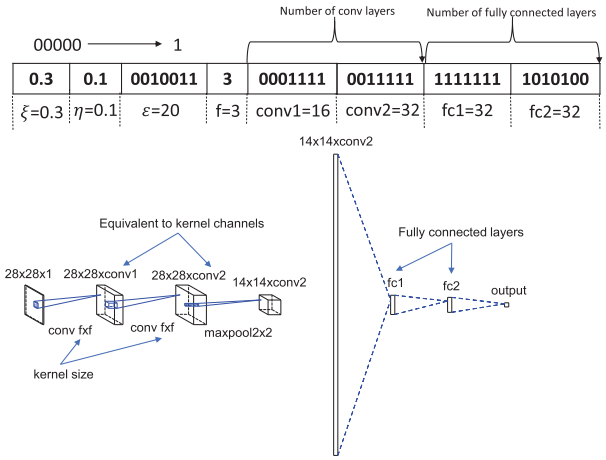

Fig. 4. Illustrative example of an individual encoding a CNN. Note that the minimum number of neurons in each hidden layer is 1. Two chromosomes conv1 and conv2 represent 16 and 32 filter channels, respectively, and fc1 and fc2 represent 32 and 32 neurons, respectively, in the fully connected layers. f is the size of the convolution kernel or filter. The padding type is set to be "same," so the size of featured images for each convolutional layer output remains the same before the max pooling operation.

fraction of weights to be removed, ξ, a real number between 0 and 1.

Consequently, we have two types of decision variables to be encoded in the chromosome of the evolutionary algorithm, i.e., real numbers and integers. Here, all the integers are encoded using binary coding and all real-valued parameters are real-encoded. For instance, the number of hidden layers and the number of nodes in each layer should be converted into binary numbers, while the real-valued parameters like learning rate and SET variables remain to be real values. Fig. 3 shows an example of an encoded individual and the corresponding MLP neural network, where $\xi = 0.3$, the learning rate $\eta = 0.1$, and $\varepsilon = 20$. In addition, the network has two hidden layers, each containing five and four neurons, respectively.

The encoding of the CNN is slightly different, mainly because a CNN contains a number of convolutional layers followed by a number of fully connected classification layers. Integers like the number of convolutional layers and the number of output channels for each convolutional layer are encoded using binary numbers. We just choose a value randomly between integer three and five for convenience for the kernel size. Refer to Fig. 4 for an illustrative example.

After generating a sparsely connected neural network model, we use the FedAvg algorithm to train the network and calculate the test accuracy A_t within a certain number of communication rounds t. This global test accuracy will be used to calculate the test error E_t of the global model, which is one of the objectives of the biobjective optimization problem. The model complexity Ω_t, the other objective, can be measured by averaging the number of weights uploaded from all clients in the tth communication round

$$\begin{aligned} E_t &= 1 - A_t \\ \Omega_t &= \sum_{k=1}^{K} \Omega_k / K \end{aligned} \tag{8}$$

where K is the total number of clients, and Ω_i indicates the number of parameters of the kth client model.

C. Modified SET Federated Averaging Algorithm

As mentioned above, the learning performance is evaluated by calculating the test error of the federated global model trained by the FedAvg algorithm (Algorithm 1). The modified SET algorithm is then integrated with the FedAvg algorithm to reduce the connectivity of the shared neural network model. The modified SET FedAvg optimization is described in Algorithm 3.

In the algorithm, i is one solution that represents a particular neural network model with a modified SET topology as a global model used in FedAvg, and R is the population size. Once the hyperparameters and the connectivity of the neural network are determined by the evolutionary algorithm, the weights will be trained using the minibatch SGD and the global model will be updated. This process repeats for a certain number of communication rounds before the two objectives can be calculated.

D. Multi-Objective Evolutionary Optimization

The biobjective optimization of federated learning can be solved using any multi-objective evolutionary algorithms. Here, we employ the popular NSGA-II for achieving a set of Pareto optimal solutions. A diagram of the overall algorithm is plotted in Fig. 5, and the pseudocode is summarized in Algorithm 4.

NSGA-II begins with the initialization of the population of size M where the binary and real-valued chromosomes are randomly initialized, which is the parent population at the first generation. Two parents are selected using the tournament selection to create two offspring by applying one-point crossover and flip mutation on the binary chromosome and the simulated binary crossover (SBX) and

Algorithm 3 Modified SET FedAvg Optimization. K Indicates the Total Numbers of Clients, k Represents the kth Local Client, B Is the Local Minibatch Size, E Is the Number of Local Training Iterations, η Is the Learning Rate, Ω Represents the Number of Connections, ε and ξ Are Both SET Parameters introduced in Algorithm 3

for each population $i \in R$ **do**
  Globally initialize θ_t^i with a Erdos Rényi topology given by ε andξ in (7)
  for each communication round $t = 1, 2, \ldots$ **do**
    Select $m = C \times K$ clients, $C \in (0, 1)$ clients
    $\Omega_t = 0$
    for each client $k \in m$ **do**
      for each local epoch e from 1 to E **do**
        for batch $b \in B$ **do**
          $\theta_e^k = \theta_t^i - \eta \nabla \ell(\theta_t^i; b)$
        end for
        remove a fraction of ξ smallest values in θ^k
      end for
      $\theta_{t+1}^i = \theta_t^i + \frac{n_k}{n}\theta^k$
      $\Omega^k = f(\theta^k)$ (calculate the number of weight parameters)
      $\Omega_t = \Omega_t + \frac{n_k}{n}\Omega^k$
    end for
  end for
  Evaluate test accuracy through θ^i and test data set
  Calculate test error as objective one f_i^1
  Set Ω_t as objective two f_i^2
end for
return f^1 and f^2

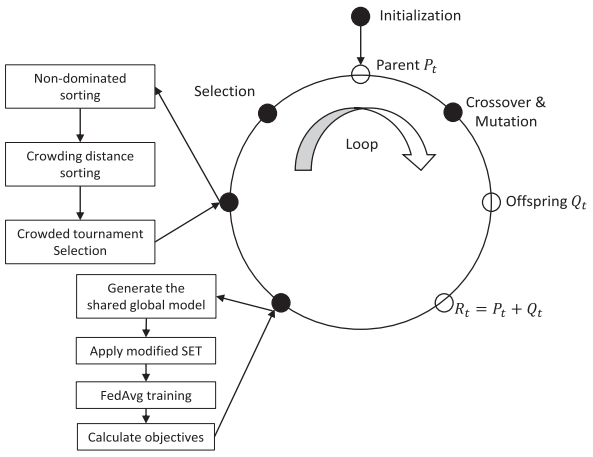

Fig. 5. Framework for multi-objective optimization of federated learning using NSGA-II.

polynomial mutation [36] on the real-valued chromosome. This process repeats until M offspring are generated.

We then calculate the two objectives of each individual in the offspring population. After that, the parent and offspring populations are combined and sorted according to the nondominance relationship and crowding distance. Finally, M high-ranking individuals from the combined population are selected as the parent of the next generation.

Algorithm 4 Multi-Objective Evolutionary Optimization

Randomly generate parent solutions P_t where $|P_t| = M$
for each generation $t = 1, 2, \ldots$ **do**
  Generate offspring $|Q_t| = M$ through crossover and mutation
  $R_t = P_t + Q_t$
  Evaluate f_t^1 and f_t^2 by Algorithm 5
  $f \leftarrow (f_t^1, f_t^2)$
  for each solution in R_t **do**
    Do nondominated sorting and calculate crowding distance on f
    Select high-ranking solutions from R_t
    Let $P_t = R_t$
  end for
end for

We repeat the above procedure for several generations to generate a set of nondominated solutions.

IV. Experimental Results

Two experiments are designed to examine the performance of the proposed multi-objective federated learning. The first experiment is conducted to compare the performance of federated learning using sparse neural network models with that using fully connected networks. The second experiment employs the widely used NSGA-II to achieve a set of Pareto optimal solutions which should be validated in both IID and non-IID environments.

A. Experimental Settings

In this section, we introduce some experimental settings in our case study. The settings include the following main parts: 1) neural network models we used in the experiment and their original settings; 2) parameters settings and data partition methods in federated learning; 3) parameters of NSGA-II; and 4) SET parameters for the sparse connection.

We select two popular neural network models: the MLP neural network and the CNN, both trained and tested on a benchmark data set MNIST [37]. In optimizing both MLPs and CNNs, the minibatch SGD algorithm has a learning rate of 0.1 and the batch size is 50. Our original MLP contains two hidden layers, each having 200 nodes (199 210 parameters in total) and uses the ReLu function as the activation function, as used in [2]. The CNN model has two 3×3 kernel filters (the first with 32 channels and the second with 64 channels) followed by a 2×2 max pooling layer, a 128 fully connected layer, and finally a 10 class softmax output layer (1 625 866 parameters in total). These can be seen as the *standard* neural network structures in our experiments.

The total number of clients K and a fraction of clients C are set to be 100 and 1 in federated learning, meaning that we use 100×1 clients on each communication rounds. For each local client training, the minibatch size B and training epochs E are 50 and 5, respectively. There are two ways of splitting the MNIST data set in our case study. One is IID, where the data is randomly shuffled into 100 clients with 600 samples

per client, and the other is non-IID, where we sort the whole MNIST data set by the labeled class, then divide it evenly into 200 fragments and randomly allocate two fragments to each client with only two classes.

The population size of NSGA-II is set to be 20 due to limited computational resources. The evolutionary optimization is run for 20 generations on the IID data set and 50 generations on the non-IID data set, because we are more interested in the learning performance on the non-IID data. The parameters of crossover and mutation operators are empirically set as follows. We apply one-point crossover with a probability of 0.9 and bit-flip mutation with a probability of 0.1 to the binary chromosome, and the SBX with a probability of 0.9 probability and $n_c = 2$, and the polynomial mutation with a probability of 0.1 and $n_m = 20$ [38] for the real-coded chromosome. In addition, the communication round required for fitness evaluations in NSGA-II is set to be 5 on the IID data and 10 on the non-IID data, because the global model trained on IID data needs fewer communication rounds to converge. Of course, evaluating fitness functions with a larger number of communication rounds can achieve more accurate fitness evaluations, but we are not allowed to do so, given very limited computation resources.

There are two SET parameters ε and ξ controlling the sparsity level of our models in federated learning. A pair of empirical values $\varepsilon = 20$ and $\xi = 0.3$ are implemented in [17] for both MLPs and CNNs, which are also adopted in this paper. In principle, these two parameters can also be binary coded and real coded, respectively, in genotypes for evolutionary optimization.

B. Influence of the Neural Network Sparsity on the Performance

In the first part of our experiment, we propose different settings of the SET parameters for both MLPs and CNNs to examine the influence of different sparsity levels on global model test accuracy and discuss model convergence properties on both the server and the client in federated learning.

Three different ε values (100, 50, 20) and two different ξ values (0, 0.3) are selected for both MLPs and CNNs with the standard structures (the original fully connected structure introduced above), which derives the standard federated models with different sparseness. Note that the modified SET algorithm applied on the FedAvg algorithm removes a ξ fraction of the least important weights at the last iteration of each local training epoch before being uploaded to the server. The parameters of the global model on the server are aggregated by calculating the weighted average of the uploaded models as done in the standard federated learning.

In addition, both MLPs and CNNs are tested on the IID and non-IID data, and we run the modified SET FedAvg algorithm for 500 communication rounds for the MLPs and 200 communication rounds for CNNs. The reason for setting a smaller number of communication rounds for CNNs is that CNNs in federated learning are easier to converge but consume more time for a single communication round compared to that for MLPs. The results are shown in Figs. 6 and 7 for MLPs and CNNs, respectively.

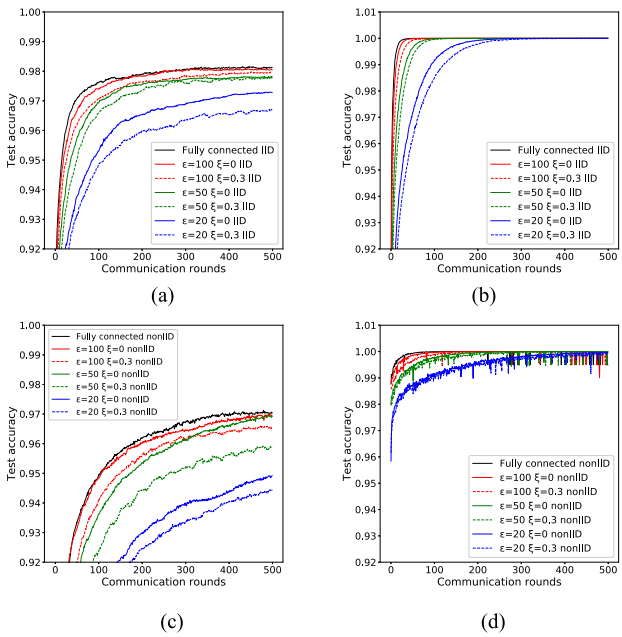

Fig. 6. Global model test accuracies and averaged client accuracies of MLPs on the IID and non-IID data sets. We select SET parameters ε and ξ to be (100,0), (100,0.3), (50,0), (50.0.3), (20,0), (20,0.3) and total communications rounds to be 500. (a) MLP IID global. (b) MLP IID clientavg. (c) MLP non-IID global. (d) MLP non-IID clientavg.

We discuss at first the convergence properties of the shared models on the server and the clients when learning the IID and non-IID data. The convergence performance on the clients is assessed through calculating the average training accuracy over all clients. The average training accuracy reaches nearly 100% within only a few rounds on the non-IID data and also becomes higher than 95% within the first 25 to 50 rounds of communication on the IID data, refer to Figs. 6(b) and (d) and 7(b) and (d). By contrast, learning converges much slower on the server, in particular, on the non-IID data, as shown in Figs. 6(a) and (c) and 7(a) and (c). This indicates that learning on the server becomes more challenging, in particular, on non-IID data.

To take a closer look at the learning behavior on the server, we compare the global test accuracies on the server as the sparsity level of the neural network models varies. An observation that can be made from the results shown in Figs. 6(a) and (c) and 7(a) and (c) is that reducing the network connectivity may lead to a degradation of the global test accuracies on both IID and non-IID data set. However, the test accuracy enhances as ξ decreases, i.e., when less "least important" weights are removed from neural network models on each client before uploading them to the server. For instance, a global test accuracy of 96.93% has been achieved when $\varepsilon = 50, \xi = 0$ in the SET algorithm that result in 72 051 connections on average, as shown in Fig. 6(c). This accuracy is higher than 96.54% when the SET parameters $\varepsilon = 100, \xi = 0.3$ that result in 87 300 connections on average at the 500th round. This implies that removing a larger fraction of weights is detrimental to the learning performance.

Nevertheless, it is clearly seen that there is a trade-off between the global test accuracy and average model

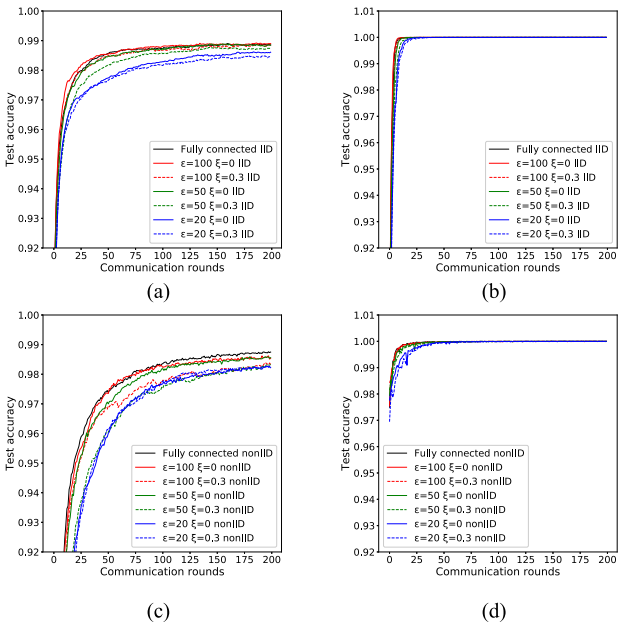

Fig. 7. Global model test accuracies and averaged client accuracies of CNNs on the IID and non-IID data sets. We select SET parameters ε and ξ to be (100,0), (100,0.3), (50,0), (50.0.3), (20,0), (20,0.3) and total communications rounds to be 200. (a) CNN IID global. (b) CNN IID clientavg. (c) CNN non-IID global. (d) CNN non-IID clientavg.

TABLE I

GLOBAL TEST ACCURACIES AND THE NUMBER OF AVERAGE CONNECTIONS

Local data distributions		IID		non-IID	
		Accuracy	Connections	Accuracy	Connections
Fully connected	MLP	98.13%	199,210	97.04%	199,210
	CNN	98.85%	1,625,866	98.75%	1,625,866
Sparsely connected	MLP	96.69%	19,360	94.45%	18,785
	CNN	98.44%	185,407	98.32%	184,543

complexity of the local models. The experimental results of the fully connected neural network model and mostly sparsely connected neural network model found by the proposed algorithm (whose SET parameters are $\varepsilon = 20, \xi = 0.3$) are listed in Table I. We can see that the global test accuracies of the sparsely connected MLPs (having about only 10% of the total number connections in the fully connected models) is only about 2% lower than that of the fully connected one on both IID and non-IID data sets. The global test accuracy of the sparse CNN, which has only about 12% of the total number of connections of the fully connected CNN, is only 0.45% worse than the fully connected CNN. Moreover, it should be pointed out that test accuracies of both MLPs and CNNs deteriorate more quickly on the non-IID data than on the IID data as the sparsity level of the network increases.

Overall, the global model test accuracy on the server tends to decline when we tune the SET parameters to rise the sparseness of the shared neural network model in our experiment. In other words, using the modified SET FedAvg algorithm only cannot maximize the global learning accuracy and minimize the communication costs at the same time.

C. Evolved Federated Learning Models

In the second part of our empirical studies, we employ NSGA-II to achieve a set of Pareto optimal neural network models that balance a tradeoff between global learning performance and communication costs in federated learning. Both IID and non-IID data sets will be used in multi-objective evolutionary optimization of federated learning. It is also interesting to investigate if the structure of the neural network models optimized on IID data sets still work on non-IID data sets and vice versa.

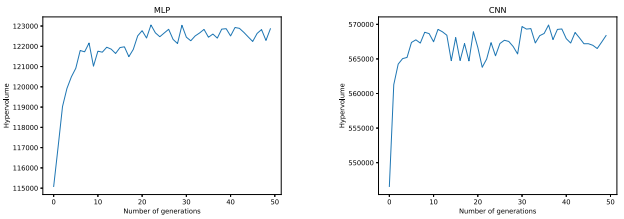

Fig. 8. Change of hypervolume over the generations in training MLP and CNN on non-IID data sets. (a) Hypervolume for MLP. (b) Hypervolume for CNN.

Evolving deep neural network structures based on the modified SET FedAvg algorithm is computationally highly intensive. For example, one run of evolutionary optimization of CNNs with a population size of 20 for 50 generations took us more than 1 week on a computer with GTX 1080Ti GPU and i7-8th 8700 CPU, preventing us from running the evolutionary optimization for a large number of generations. In order to monitor the convergence of the multi-objective optimization, the hypervolumes calculated based on the nondominated solution set in the population over the generations [39] in evolving MLP and CNN on non-IID data sets are plotted in Fig. 8. From this figure, we can see that the hypervolumes of both runs increase at the beginning and start fluctuating from around the 20th generation onward. These results imply that approximately 20 generations are needed for federated learning to converge on non-IID data sets used in this paper.

The total communication rounds for each population is set to be 5 for IID data sets and 10 for non-IID data sets, respectively, before the objective values are calculated. Of course, setting large communication rounds may achieve more accurate evaluations of the objectives, which is unfortunately prohibitive given limited computation resources.

We set the maximum number of hidden layers of MLPs to be 4, and the maximum number of neurons per layer is 256. For CNNs, we set the maximum number of convolutional layers to be 3, the maximum number of kernel channels to be 64, the maximum number of fully connected layers to be 3, and the maximum neurons in the convolutional layers to be 256. The kernel size is either 3 or 5, which is also evolved.

The range of the learning rate is between 0.01 and 0.3 for both MLPs and CNNs, because too large values may harm the global convergence in federated learning.

Recall that the SET parameters ε and ξ are binary coded and real coded, respectively. The maximum value of ε is set to 128, and ξ ranges from 0.01 to 0.55. A summary of the experimental settings is given in Table II.

The final nondominated MLP and CNN solutions optimized on the IID and non-IID data sets are presented in Figs. 9 and 10, respectively, where each point represents

TABLE II

EXPERIMENTAL SETTINGS FOR MULTI-OBJECTIVE OPTIMIZATION OF FEDERATED LEARNING

Genotypes	MLP IID	MLP nonIID	CNN IID	CNN nonIID
Populations	20	20	20	20
Generations	20	50	20	50
Learning rate	0.01-0.3	0.01-0.3	0.01-0.3	0.01-0.3
Hidden layers	1-4	1-4	/	/
Hidden neurons	1-256	1-256	/	/
Conv layers	/	/	1-3	1-3
Kernel channels	/	/	1-64	1-64
Fully connected layers	/	/	1-3	1-3
Fully connected neurons	/	/	1-256	1-256
Kernel sizes	/	/	3 or 5	3 or 5
ε sizes	1-128	1-128	1-128	1-128
ξ sizes	0.01-0.55	0.01-0.55	0.01-0.55	0.01-0.55

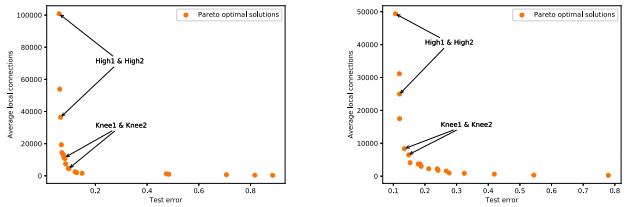

Fig. 9. Pareto frontier of MLPs, of which four solutions, High1, High2, Knee1, and Knee2 are selected for validation. (a) Evolved Pareto frontier of MLPs trained on IID data sets. (b) Evolved Pareto frontier of MLPs trained on non-IID data sets.

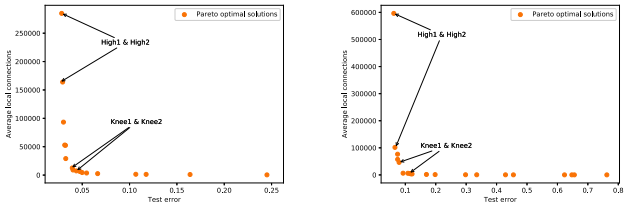

Fig. 10. Pareto frontier of CNNs, of which High1, High2, Knee1, and Knee2 are selected for validation. (a) Evolved Pareto frontier of CNNs trained on IID data sets. (b) Evolved Pareto frontier of CNNs trained on non-IID data sets.

one solution corresponding to a particular structure of the neural network model in federated learning. However, not all nondominated solutions are of interest, since some of them have very large test errors, even if they have very simple model structures with very limited average local model connections. In this paper, we select two types of nondominated solutions, namely, those solutions with a very low global test error and those solutions near the knee point of the frontier, as suggested in [32].

We choose two high-accuracy Pareto solutions (High1 and High2) and two solutions around the knee point (Knee1 and Knee2) of both MLPs and CNNs (refer to Figs. 9 and 10) for further performance verification and compare their performance with the fully connected MLPs and CNNs. Recall that only five and ten communication rounds are used over IID data and non-IID data, respectively, for fitness evaluations in the evolutionary optimization. For a fair comparison, however, the number of communication rounds are increased to 500 for MLPs and 200 for CNNs, as set in the original federated learning. All validation results are listed in Tables III–VI, and the global test accuracies of the selected solutions are also presented in Figs. 11 and 12.

From the results presented in Figs. 11 and 12, we can make the following observations on the four selected Pareto optimal MLP models evolved on the IID data.

1) Solution High1 of MLP has global test accuracies of 98.16% and 97.42% on IID and non-IID data sets, both of which are better than that of the fully connected MLP. In addition, this evolved model has only on average 91 933 and 91 527 connections on IID data and non-IID data, respectively, which is approximately 46% of 199 210 connections the fully connected network has.
2) Solution High2 has a lower test accuracy of 0.39% on IID data and 0.22% on non-IID data but it has only 16.5% of connections compared to the fully connected MLP.

TABLE III

HYPERPARAMETERS OF HIGH1, HIGH2, KNEE1, AND KNEE2 FOR *MLPs* EVOLVED ON *IID* DATA AND THEIR VALIDATION RESULTS

Parameters	Knee1	Knee2	High1	High2	Standard
Hidden layer1	10	15	152	73	200
Hidden layer2	27	123	49	22	200
ε	28	60	121	48	/
ξ	0.3969	0.2021	0.1314	0.1214	/
Learning rate η	0.2591	0.3	0.2951	0.283	0.1
Test accuracy IID	94.24%	96.84%	98.16%	97.74%	98.13%
Connections IID	4,374	10,815	91,933	32,929	199,210
Test accuracy nonIID	90.77%	93.77%	97.42%	96.82%	97.04%
Connections nonIID	4,026	10,206	91,527	33,594	199,210

TABLE IV

HYPERPARAMETERS OF HIGH1, HIGH2, KNEE1, AND KNEE2 FOR *MLPs* EVOLVED ON *non-IID* DATA AND THEIR VALIDATION RESULTS

Parameters	Knee1	Knee2	High1	High2	Standard
Hidden layer1	49	53	86	109	200
Hidden layer2	/	/	/	/	200
ε	10	8	66	34	/
ξ	0.1106	0.0764	0.1106	0.1566	/
Learning rate η	0.3	0.2961	0.3	0.3	0.1
Test accuracy IID	96.78%	96.41%	97.82%	97.68%	98.13%
Connections IID	7,749	5,621	45,329	22,210	199,210
Test accuracy nonIID	94.85%	94.88%	97.32%	96.21%	97.04%
Connections nonIID	8,086	6,143	45,530	24,055	199,210

TABLE V

HYPERPARAMETERS OF HIGH1, HIGH2, KNEE1, AND KNEE2 FOR *CNNs* EVOLVED ON *IID* DATA AND THEIR VALIDATION RESULTS

Parameters	Knee1	Knee2	High1	High2	Standard
Conv layer1	34	6	25	18	32
Conv layer2	6	6	38	20	64
Fully connected layer1	11	9	38	102	128
Fully connected layer2	/	/	/	/	/
Kernel size	5	5	5	5	3
ε	24	39	121	41	/
ξ	0.4702	0.3901	0.0685	0.0625	/
Learning rate η	0.2094	0.1576	0.2279	0.1888	0.1
Test accuracy IID	98.51%	98.19%	99.07%	98.96%	98.85%
Connections IID	12,360	7,127	268,150	158,340	1,625,866
Test accuracy nonIID	11.35%	97.21%	11.35%	98.79%	98.75%
Connections nonIID	6,071	6,804	24,853	157,511	1,625,866

TABLE VI

HYPERPARAMETERS OF HIGH1, HIGH2, KNEE1, AND KNEE2 FOR *CNNs* EVOLVED ON *non-IID* DATA AND THEIR VALIDATION RESULTS

Parameters	Knee1	Knee2	High1	High2	Standard
Conv layer1	17	5	53	33	32
Conv layer2	/	/	/	/	64
Fully connected layer1	29	21	208	31	128
Fully connected layer2	/	/	/	/	/
Kernel size	5	5	5	5	3
ε	18	8	66	20	/
ξ	0.1451	0.1892	0.0786	0.1354	/
Learning rate η	0.2519	0.2388	0.2776	0.2503	0.1
Test accuracy IID	98.84%	98.15%	99.06%	98.93%	98.85%
Connections IID	48949	6262	622090	107224	1,625,866
Test accuracy nonIID	97.92%	97.7%	98.52%	98.46%	98.75%
Connections nonIID	39457	6804	553402	90081	1,625,866

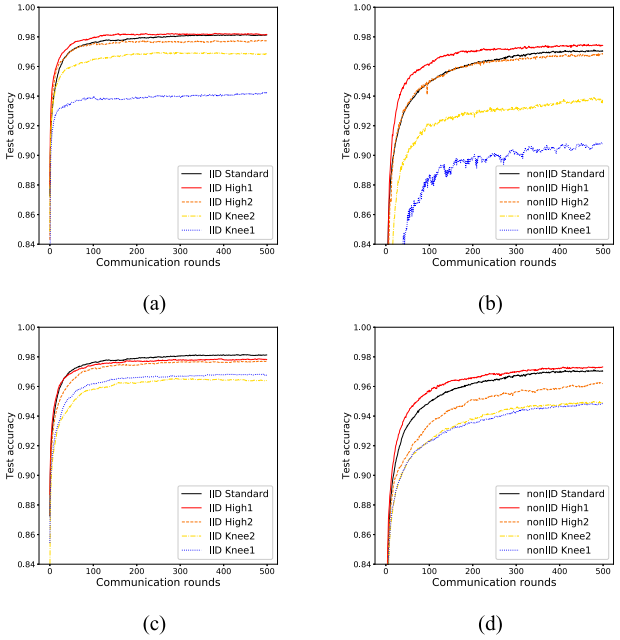

Fig. 11. Global test accuracies of the selected Pareto optimal MLPs validated on both IID and non-IID data. The test accuracies of the fully connected MLP are also plotted in the figure for comparison. (a) Solutions evolved on IID data and validated on IID data. (b) Solutions evolved on IID data and validated on non-IID data. (c) Solutions evolved on non-IID data and validated on IIDdata. (d) Solutions evolved on non-IID data and validated on non-IID data.

3) Knee1 and Knee2 have test accuracies of 96.84% and 94.24%, respectively, on IID data sets. Note, however, that but their performance becomes much worse on non-IID data and the test accuracies decrease to only 93.77% and 90.77%. This means that knee solutions evolved on IID data may not be suited for non-IID data.

Similar observations can be made on the two high-accuracy Pareto optimal CNNs, High1 and High2, on IID data and their test accuracies are 99.07% and 98.96%, respectively, both of which are higher than that of the fully connected CNN. The two knee solutions also have acceptable global test accuracies on IID data with a much smaller number of connections. However, it is surprising to see that both Knee1 and High1 fail to converge on the non-IID data, even if they both converge very well on the IID data, meaning that the

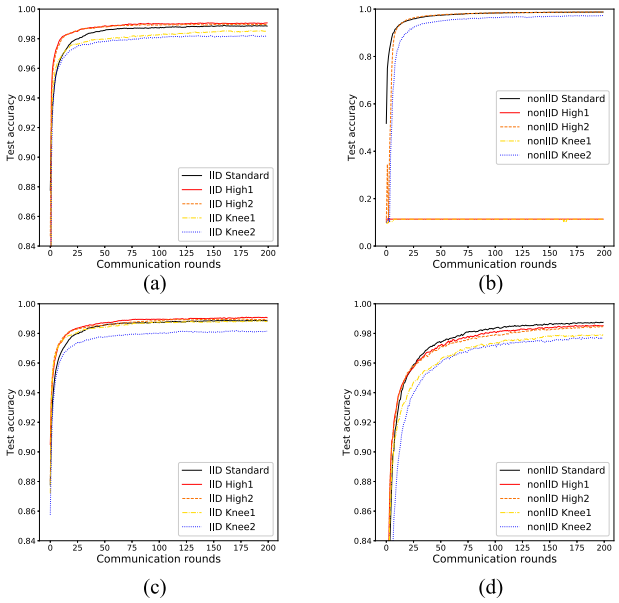

Fig. 12. Global test accuracies of the selected Pareto optimal CNNs validated on both IID and non-IID data. The test accuracies of the fully connected CNN are also plotted for comparison. (a) Solutions evolved on IID data and validated on IID data. (b) Solutions evolved on IID data and validated non-IID data. (c) Solutions evolved on non-IID data and validated on IID data. (d) Solutions evolved on non-IID data and validated on non-IID data.

Pareto optimal CNNs generated on IID data may completely break down on non-IID data. Note that the high-accuracy solution High2 also converge well on non-IID data and has a test accuracy of 98.79%, which is 0.04% higher than the fully connected model, but has only around 10% of connections of the fully connected CNN.

Based on the above validation results, we recommend to select the following solutions from the Pareto frontier for final implementation. The high-accuracy MLP solution, High1 with two hidden layers should be selected. This model has 152 and 49 neurons in the first and second hidden layers, respectively. SET parameters of the network are $\varepsilon = 121$ and $\xi = 0.1314$, and the learning rate is 0.2951. It has better global test accuracies than the fully connected model on both IID and non-IID data, while it has around 46% of the connections of the standard fully connected network. By contrast, the high-accuracy CNN solution High2 with two 5×5 convolutional layers should be selected. The first and second layers of this network have 18 and 20 filters, respectively. The fully connected layer has 102 nodes, the SET parameters are $\varepsilon = 41$ and $\xi = 0.0625$, and the learning rate is 0.1888. The network has better global test accuracies than the standard fully connected model on both IID and non-IID data, but has only about 9.7% of the connections of the standard fully connected networks. In addition, one knee point CNN solution Knee1 has one 5×5 convolutional layer, 17 kernel filters, 29 nodes in the fully connected layer, whose SET parameters are $\varepsilon = 18$ and $\xi = 0.1415$, and the learning rate is 0.2591, has a similar global test accuracy on IID data, and 0.8% worse than the fully connected one on non-IID data. This network has only about 3% of the connections of the standard fully

connected model. Thus, Knee1 of the CNN model is also recommendable.

The following observations can be made from our experimental results.

1) The proposed algorithm can achieve a set of Pareto optimal solutions, from which we can select multiple solutions based on different preferences for different learning tasks.
2) The solutions we selected for comparison, either the knee points or solutions with high accuracy, can significantly reduce the number of parameters in the global model to be transferred between the server and the clients without seriously deteriorating the model performance in federated learning. Actually, the proposed algorithm has also found two solutions that have higher test accuracies than the original settings on both IID and non-IID data sets, one being an MLP that has 46% of the connections in the standard MLP and the other being a CNN that has only 9.7% the connections in the fully connected one. By significantly reducing the global model size transferred between devices, the proposed multi-objective evolutionary algorithm can effectively enhance the communication efficiency at a rate of at least 50% in training a neural network model in federated learning.
3) The global model structures in federated learning evolved from non-IID data are more robust than those evolved from the IID data. Specifically, our solutions in the optimal Pareto frontier evolved from the IID data set may not fit very well when the data becomes non-IID. As seen from our previous experimental results, some of our solutions cannot converge at all. On the contrary, solutions evolved from the non-IID data set also perform well on the IID data set. This implies that it is harder for federated learning on non-IID data to converge than the traditional distributed learning only on IID data.
4) The model structures evolved on IID data sets are usually deeper than that evolved on non-IID data sets. In addition, the proposed algorithm allows different clients to have different model sizes, which is computationally more efficient and enables more reduction in communication cost.

V. Conclusion

This paper proposes a multi-objective federated learning to simultaneously maximize the learning performance and minimize the communication cost using a multi-objective evolutionary algorithm. To improve the scalability in evolving large neural networks, a modified SET method is suggested to indirectly encode the connectivity of the neural network. Our experimental results demonstrate that the modified SET algorithm can effectively reduce the number of connections of neural networks by encoding only two hyperparameters. Selected solutions from the nondominated frontier obtained by the multi-objective algorithm confirm that the proposed algorithm is able to generate neural network models for federated learning that exhibit better global learning accuracy and have much fewer connections, thereby dramatically reducing the communication cost without deteriorating the learning performance on both IID and non-IID data sets.

A lot of research works remain to be done in federated learning. For instance, both the modified SET FedAvg algorithm and the FedAvg algorithm do not work very well on complicated data sets like non-IID CIFAR-10. Although the proposed algorithm is applicable to deep networks in principle, the network models studied in this paper are fairly simple. Thus, it is of great interest to investigate the performance of the proposed algorithm in optimizing deep neural networks having dozens of hidden layers. In addition, it is still unclear if missing data caused by package loss in communications between clients and the server will significantly affect the performance of federated learning. Finally, adversarial attacks [40] on the parameters uploaded to the central server may directly damage the global model. Thus, preserving privacy while maintaining robustness in federated learning will be a very important research challenge.

Acknowledgment

The authors would like to thank Y. Zhao for sharing his code.

References

[1] J. Poushter, "Smartphone ownership and Internet usage continues to climb in emerging economies," *Pew Res. Center*, vol. 22, pp. 1–44, 2016.
[2] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, "Communication-efficient learning of deep networks from decentralized data," 2016, *arXiv:1602.05629*. [Online]. Available: https://arxiv.org/abs/1602.05629
[3] K. Bonawitz *et al.*, "Practical secure aggregation for privacy-preserving machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 1175–1191.
[4] A. A. Atayero and O. Feyisetan, "Security issues in cloud computing: The potentials of homomorphic encryption," *J. Emerg. Trends Comput. Inf. Sci.*, vol. 2, no. 10, pp. 546–552, 2011.
[5] C. Dwork, "Differential privacy: A survey of results," in *Proc. Int. Conf. Theory Appl. Models Comput.* Berlin, Germany: Springer, 2008, pp. 1–19.
[6] M. Abadi *et al.*, "Deep learning with differential privacy," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 308–318.
[7] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, 2015, pp. 1310–1321.
[8] R. C. Geyer, T. Klein, and M. Nabi, "Differentially private federated learning: A client level perspective," 2017, *arXiv:1712.07557*. [Online]. Available: https://arxiv.org/abs/1712.07557
[9] R. McDonald, K. Hall, and G. Mann, "Distributed training strategies for the structured perceptron," in *Proc. Hum. Lang. Technol., Annu. Conf. North Amer. Chapter Assoc. Comput. Linguistics*, 2010, pp. 456–464.
[10] D. Povey, X. Zhang, and S. Khudanpur, "Parallel training of DNNs with natural gradient and parameter averaging," 2014, *arXiv:1410.7455*. [Online]. Available: https://arxiv.org/abs/1410.7455
[11] S. Zhang, A. E. Choromanska, and Y. LeCun, "Deep learning with elastic averaging SGD," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 685–693.
[12] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-IID data," 2018, *arXiv:1806.00582*. [Online]. Available: https://arxiv.org/abs/1806.00582
[13] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," 2016, *arXiv:1610.05492*. [Online]. Available: https://arxiv.org/abs/1610.05492
[14] X. Yao, "Evolving artificial neural networks," *Proc. IEEE*, vol. 87, no. 9, pp. 1423–1447, Sep. 1999.
[15] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evol. Comput.*, vol. 10, no. 2, pp. 99–127, 2002.

[16] J. Fekiač, I. Zelinka, and J. C. Burguillo, "A review of methods for encoding neural network topologies in evolutionary computation," in *Proc. 25th Eur. Conf. Modeling Simulation (ECMS)*, 2011, pp. 410–416.
[17] D. C. Mocanu, E. Mocanu, P. Stone, P. H. Nguyen, M. Gibescu, and A. Liotta, "Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science," *Nature Commun.*, vol. 9, no. 1, 2018, Art. no. 2383.
[18] D. Pham, "Neural networks in engineering," in *WIT Transactions on Information and Communication Technologies*, vol. 6. Univ. of Wales College of Cardiff, U.K.: Transactions on Information and Communications Technologies, 1994.
[19] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time series," *Handbook Brain Theory Neural Netw.*, vol. 3361, no. 10, p. 1995, 1995.
[20] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *Uncertain. Fuzziness Knowl.-Based Syst.*, vol. 6, no. 2, pp. 107–116, 1998.
[21] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 2, p. 12, 2019.
[22] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.
[23] K. Deb and H. Jain, "An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: Solving problems with box constraints," *IEEE Trans. Evol. Comput.*, vol. 18, no. 4, pp. 577–601, Aug. 2013.
[24] X. Zhang, Y. Tian, and Y. Jin, "A knee point-driven evolutionary algorithm for many-objective optimization," *IEEE Trans. Evol. Comput.*, vol. 19, no. 6, pp. 761–776, Dec. 2015.
[25] R. Cheng, Y. Jin, M. Olhofer, and B. Sendhoff, "A reference vector guided evolutionary algorithm for many-objective optimization," *IEEE Trans. Evol. Comput.*, vol. 20, no. 5, pp. 773–791, Oct. 2016.
[26] X. Zhang, Y. Tian, R. Cheng, and Y. Jin, "An Efficient approach to nondominated sorting for evolutionary multiobjective optimization," *IEEE Trans. Evol. Comput.*, vol. 19, no. 2, pp. 201–213, Apr. 2015.
[27] X. Zhang, Y. Tian, R. Cheng, and Y. Jin, "A decision variable clustering-based evolutionary algorithm for large-scale many-objective optimization," *IEEE Trans. Evol. Comput.*, vol. 22, no. 1, pp. 97–112, Feb. 2018.
[28] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II," in *International Conference on Parallel Problem Solving from Nature* (Lecture Notes in Computer Science). Springer, 2000, pp. 849–858.
[29] Y. Jin, "Surrogate-assisted evolutionary computation: Recent advances and future challenges," *Swarm Evol. Comput.*, vol. 1, no. 2, pp. 61–70, Jun. 2011.
[30] Y. Jin, H. Wang, T. Chugh, D. Guo, and K. Miettinen, "Data-driven evolutionary optimization: An overview and case studies," *IEEE Trans. Evol. Comput.*, to be published.
[31] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas, "Taking the human out of the loop: A review of Bayesian optimization," *Proc. IEEE*, vol. 104, no. 1, pp. 148–175, Jan. 2016.
[32] Y. Jin and B. Sendhoff, "Pareto-based multiobjective machine learning: An overview and case studies," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 38, no. 3, pp. 397–415, May 2008.
[33] D. Whitley, "A genetic algorithm tutorial," *Statist. Comput.*, vol. 4, no. 2, pp. 65–85, Jun. 1994.
[34] P. Erdos and A. Rényi, "On the evolution of random graphs," *Publications Math. Inst. Hung. Acad. Sci.*, vol. 5, no. 1, pp. 17–61, 1960.
[35] K. Deb, "Multi-objective optimization," in *Search Methodologies*. Springer, 2014, pp. 403–449.
[36] K. Deb and R. B. Agrawal, "Simulated binary crossover for continuous search space," *Complex Syst.*, vol. 9, no. 2, pp. 115–148, 1995.
[37] 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.
[38] K. Kumar, "Real-coded genetic algorithms with simulated binary crossover: Studieson multimodaland multiobjective problems," *Complex Syst.*, vol. 9, no. 9, pp. 431–454, 1995.
[39] N. Beume, B. Naujoks, and M. Emmerich, "SMS-EMOA: Multiobjective selection based on dominated hypervolume," *Eur. J. Oper. Res.*, vol. 181, no. 3, pp. 1653–1669, 2007.
[40] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," 2018, *arXiv:1807.00459*. [Online]. Available: https://arxiv.org/abs/1807.00459

Hangyu Zhu received the B.Sc. degree from Yangzhou University, Yangzhou, China, in 2015, and the M.Sc. degree from RMIT University, Melbourne, VIC, Australia, in 2017. He is currently pursuing the Ph.D. degree, with a focus on evolutionary neural architecture search for federated learning, with the Department of Computer Science, University of Surrey, Guildford, U.K.

Yaochu Jin (M'98–SM'02–F'16) received the B.Sc., M.Sc., and Ph.D. degrees from Zhejiang University, Hangzhou, China, in 1988, 1991, and 1996, respectively, and the Dr.-Ing. degree from Ruhr University Bochum, Bochum, Germany, in 2001.

He was a Finland Distinguished Professor at the University of Jyvaskyla, Jyväskylä, Finland, and a Changjiang Distinguished Visiting Professor at Northeastern University, Shenyang, China. He is currently a Distinguished Chair Professor of Computational Intelligence with the Department of Computer Science, University of Surrey, Guildford, U.K., where he is the Head of the Nature Inspired Computing and Engineering Group. He has coauthored more than 300 peer-reviewed journal and conference papers. He has been granted eight patents on evolutionary optimization. His current research interests include the cross-disciplinary areas of computational intelligence, computational neuroscience, and computational systems biology, especially in the application of nature-inspired algorithms to solving real-world optimization, learning, and self-organization problems.

Dr. Jin was a recipient of the 2015 and 2017 IEEE Computational Intelligence Magazine Outstanding Paper Award and the 2018 IEEE Transactions on Evolutionary Computation Outstanding Paper Award. He is the Editor-in-Chief of the IEEE TRANSACTIONS ON COGNITIVE AND DEVELOPMENTAL SYSTEMS and *Complex & Intelligent Systems*. He was the Vice President for Technical Activities of the IEEE Computational Intelligence Society from 2014 to 2015. He is an IEEE Distinguished Lecturer from 2017 to 2019.