

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—Federated learning, multi-objective evolutionary optimization, communication cost, deep neural networks, network connectivity

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 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, 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.* proposed two distributed training strategies [9] for the 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 [11] can further improve the convergence efficiency and robustness in 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 algorithm [2] introduced by McMahan *et al.* can improve communication efficiency by reducing local training mini-batch sizes or increasing local training passes to reduce communication rounds. Shokri *et al.* used the method of uploading the gradients located in the particular interval clipped by some threshold values [7], 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] 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 method occupy too many computation resources. More recently, Mocanu proposed a sparse evolutionary algorithm (SET) [17] 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 work proposes a framework for optimizing deep neural network models in

Hangyu Zhu and Yaochu Jin are with the Department of Computer Science, University of Surrey, Guildford, GU2 7XH, United Kingdom. Email: {hangyu.zhu;yaochu.jin}@surrey.ac.uk. (Corresponding author: Yaochu Jin)
Manuscript received December 6, 2018; revised xx, 2019.

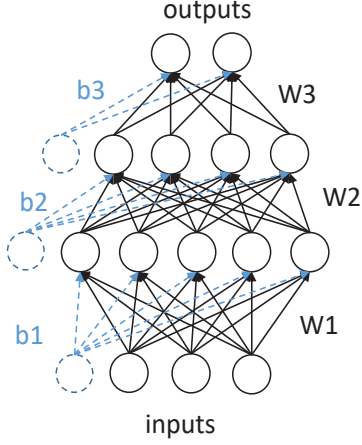


Fig. 1. A multilayer perceptron neural network containing an input layer, two hidden layers, and an output layer. Solid circles represent neurons while dashed ones represent biases. A deep neural network may have tens of hidden layers.

federated learning. The main contributions of the paper are as follows:

- 1) Federated learning is formulated as a bi-objective optimization problem, where the two objectives are the minimization of the communication cost and the maximization of the global learning accuracy. This bi-objective optimization is solved by a multi-objective evolutionary algorithm.
- 2) A modified SET algorithm is proposed to reduce the connections of deep neural networks, thereby indirectly reducing the number of model parameters to be communicated to the server.
- 3) The proposed framework is tested on two popular neural network models, a multilayer perceptron neural network (MLP) and a convolutional neural network (CNN) in the federated learning paradigm on various widely used IID or non-IID datasets.

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

II. PRELIMINARIES

In this section, we briefly review the basics of deep learning, federated learning and evolutionary optimization of neural networks.

A. Deep learning

Deep learning has achieved a big success in computer vision, speech recognition and many other domains [18]. Multilayer feedforward neural networks are the most common supervised deep learning architectures to build a projection from high-dimensional input data into pre-defined output labels.

Fig. 1 shows an illustrative example of a fully connected multilayer perceptron. Solid circles in the figure represent

'neurons' and circles in dashed line are called 'biases'. In the feed-forward propagation of a fully connected neural network, each node or neuron receives a weighted sum of all preceding neurons plus a bias value, which is the input of this neuron. Then the output of this neuron is computed by a nonlinear activation function σ as follows:

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

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 $\ell(\theta)$.

$$\min_{\theta} \ell(\theta) = \frac{1}{N} \sum_i \ell(\theta, x_i) \quad x_i \in \{x_1, x_2, \dots, x_N\} \quad (2)$$

where x_i is the i -th 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 descend (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 stochastic gradient descend (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, mini-batch stochastic gradient descent (mini-batch SGD) is proposed to select a randomly chosen mini-batch size of the training data for the gradient computation during every training iteration:

$$g_t = \frac{1}{n} \nabla_{\theta} \ell(\theta, x_{i:i+n}) \quad (3)$$

$$\theta_{t+1} = \theta_t - \eta g_t$$

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

B. Federated learning

Federated learning [19] is an emerging decentralized privacy-protection training technology which 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

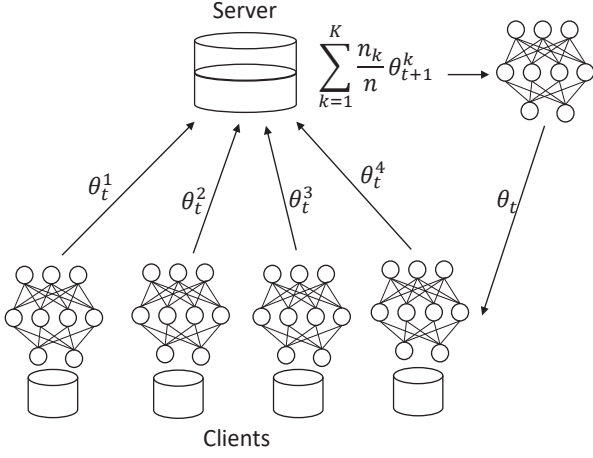


Fig. 2. Flowchart of federated learning.

clients are aggregated to obtain a new global model. Compared with the traditional centralized learning, federated learning has following unique features:

- 1) The training data are spread on the local edges, which is not available to the global server cloud. However, the model we want to train is shared between the server and all clients.
- 2) Model training occurs on each local device instead of the global server where only aggregating uploaded models from clients is executed to obtain a shared global model.
- 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 multilayer perceptron neural network, federated learning aims to minimize the loss function $\ell(\theta)$ but in a distributed scheme:

$$\min_{\theta} \ell(\theta) = \sum_{k=1}^K \frac{n_k}{n} L_k(\theta) \quad \text{where} \quad L_k(\theta) = \frac{1}{n_k} \sum_{i \in P_k} \ell_i(\theta) \quad (4)$$

where k is the index of K total clients, $L_k(\theta)$ is the loss function of k -th local client, n_k equals 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 shared model parameter θ_t from the central server and then train their individual local models from their own data. After local training, each local device sends their trained local parameters (for instance θ_t^1) to 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 federated averaging (FedAvg) algorithm [2] can effectively reduce communication rounds by simultaneously increasing local training epochs and decreasing local mini-batch sizes in federated stochastic gradient descend algorithm

(FedSGD) [2]. The pseudo code of FedAvg is presented in **Algorithm 1**, where θ^k is the model parameter of the k -th client.

Algorithm 1 FederatedAveraging. 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

```

1: Server:
2: Initialize  $\theta_t$ 
3: for each communication round  $t = 1, 2, \dots$  do
4:   Select  $m = C \times K$  clients,  $C \in (0, 1)$  clients
5:   Download  $\theta_t$  to each client  $k$ 
6:   for each client  $k \in m$  do
7:     Wait Client  $k$  for synchronization
8:      $\theta_t = \sum_{k=1}^m \frac{n_k}{n} \theta^k$ 
9:   end for
10: end for
11: Client  $k$ :
12:  $\theta^k = \theta_t$ 
13: for each iteration from 1 to  $E$  do
14:   for batch  $b \in B$  do
15:      $\theta^k = \theta^k - \eta \nabla L_k(\theta^k, b)$ 
16:   end for
17: end for
18: return  $\theta^k$  to server

```

In the algorithm, n is the size of the whole data, and the global model parameter θ_t over t -th 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 to 1 determining the total fraction of clients allowed to update the shared global model. It was found in [2] that a large C is able to speed up the convergence, however, the findings in [9] suggested 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 is selected to communicate with the central server, can cover the whole data population, the client or replicas that has a larger data size converges to its optimum more quickly. Note that 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 negative effect on convergence performance.

The global weight convergence can also be affected by the probability differences between data distributed on client k and the whole data population, i.e., $\sum_{i=1}^L \|p^k(y=i) - p(y=i)\|$ [12], 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. This proposition indicates that training of the neural network is harder to converge on non-IID data than IID data in federated learning.

C. Evolutionary optimization of neural networks

In evolutionary optimization of the structure of the deep neural networks, the encoding scheme used by the evolutionary algorithm significantly affects the optimization efficiency.

Direct binary encoding such as the one introduced in [20] needs a large connection matrix to represent the structure of a neural network, which has poor scalability when the neural network contains multiple hidden layers with a large number of neurons. In order to enhance the scalability in evolving deep neural networks, we propose a modified sparse evolutionary training (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 [21]. Instead, SET starts from an initial Erdos Rnyi random graph [22] that determines the connectivity between every two neighboring layers of the neural network. The connection probability between two layers is described as follows in Eq. (5):

$$p(W_{ij}^k) = \frac{\varepsilon(n^k + n^{k-1})}{n^k n^{k-1}} \quad (5)$$

$$n^W = n^k n^{k-1} p(W_{ij}^k)$$

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 to remove a fraction ξ of the weights that have updated the smallest 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 mini-batch 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. Pseudo code of the modified SET is listed in **Algorithm 2**. By implementing the modified SET algorithm, a sparsely

Algorithm 2 Modified sparse evolutionary training algorithm

```

1: Set  $\varepsilon$  and  $\xi$ 
2: for each fully-connected layer of the neural network do
3:   Replace weight matrices by Erdos Rnyi random graphs given by  $\varepsilon$  in
   Eq. (5)
4: end for
5: Initialize weights
6: Start training
7: for each training iteration do
8:   Update corresponding weights
9:   for each weight matrix do
10:    Remove a fraction  $\xi$  of the smallest  $|weights|$ 
11:   end for
12: end for
```

connected neural network can be evolved, resulting in much fewer parameters to be downloaded or uploaded, thereby reducing the communication cost in federated learning.

However, the modified SET algorithm cannot evolve hyper parameters of the neural network model such as the number of hidden layers and may bring negative effect to the convergence of global learning on the server. To solve this

problem, we adopt a widely used multi-objective evolutionary algorithm, i.e., the elitist non-dominated sorting genetic algorithm (NSGA-II) [23] to optimize the connectivity and hyper parameters 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 considering the dominance relationship between the solutions in the population and a crowding distance calculated according to the distance between two neighbouring solutions. **Algorithm 3** outlines the main components of NSGA-II and more details can be found in [23].

Algorithm 3 Elitist non-dominated sorting genetic algorithm NSGA-II

```

1: Randomly generation parent solutions  $P_t$ 
2: for each generations do
3:   Generate offsprings  $Q_t$  through crossover and mutation
4:   Combine  $P_t$  and  $Q_t$  to generation  $R_t$ 
5:   Calculate fitness values  $f$  of  $R_t$ 
6:   for each  $R_t$  do
7:     Do non-dominated and crowding distance sorting on  $f$ 
8:     Select high-ranking solutions from  $R_t$ 
9:     Let  $P_t = R_t$ 
10:   end for
11: end for
```

III. PROPOSED ALGORITHM

In this section, we firstly formulate federated learning as a bi-objective optimization problem. This is followed by a description of the encoding scheme adopted by the evolutionary algorithm. Finally, the overall framework is presented.

A. The objective functions and encoding of the neural networks

We reformulate federated learning as a two objective optimization problem [24]. One objective is the global model test error E_t and the other is the model complexity Ω_t over the t -th communication round. To minimize these two objectives, we evolve both the hyper parameters as well as the connectivity of the neural network models. The hyper parameters include the number of hidden layers, the number of neurons in each hidden layer, and the learning rate η of the mini-batch 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 Eq. (5), an integer, and the 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 integers are encoded using binary coding. Fig. 3 provides 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 convolutionary layers followed by a number of fully connected classification layers. Refer to Fig. 4 for an illustrative example.

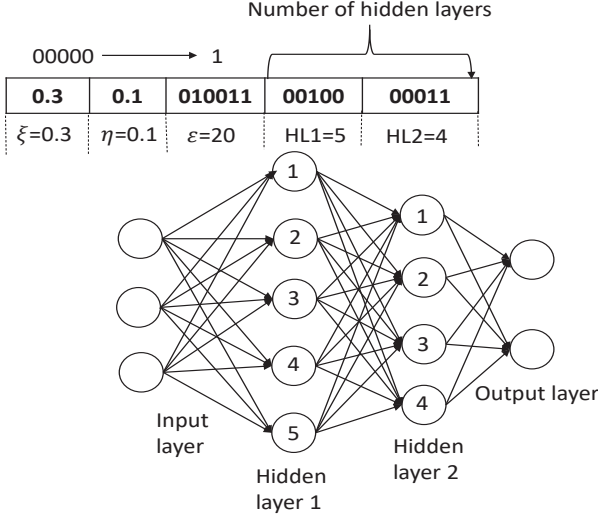


Fig. 3. A neural network and its chromosome. Note that when we decode the number of neurons, each variable will be increased by one to make sure that there is at least one neuron in a hidden layer. In the figure, HL1 and HL2 denote that the neural network has two hidden layers containing 5 and 4 neurons, respectively.

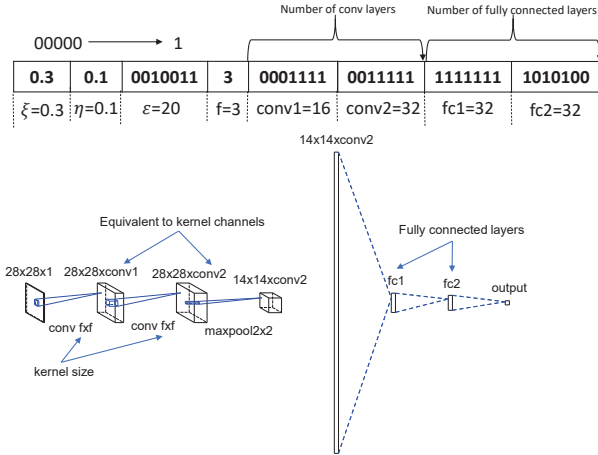


Fig. 4. An illustrative example of an individual encoding a convolutional neural network. Note that the minimum number of neurons in each hidden layer is 1. Two chromosomes conv1 and conv2 represent 16 and 32 filter channels, respectively, and fc1 and fc2 represent 128 and 84 neurons, respectively, in the fully connected layers. The padding type is set to be 'same', so the size of feathered images for each convolutional layer output remains the same before the maxpooling operation.

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 bi-objective 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 t -th communication round:

$$E_t = 1 - A_t$$

$$\Omega_t = \sum_{k=1}^K \Omega_k / K \quad (6)$$

where K is the total number of clients and Ω_i indicates the number of parameters of the k -th client model.

B. The modified 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. A detailed federated learning procedure is given in **Algorithm 4**.

Algorithm 4 The modified SET FedAvg optimization. K indicates the total numbers of clients, k represents the k -th local client, B is the local mini-batch 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 2**

```

1: for each population  $i \in R$  do
2:   Globally initialize  $\theta_t^i$  with a Erdos Rnyi topology given by  $\varepsilon$  and equation (5)
3:   for each communication round  $t = 1, 2, \dots$  do
4:     Select  $m = C \times K$  clients,  $C \in (0, 1)$  clients
5:      $\Omega_t = 0$ 
6:     for each client  $k \in m$  do
7:       for each local epoch  $e$  from 1 to  $E$  do
8:         for batch  $b \in B$  do
9:            $\theta_e^k = \theta_t^i - \eta \nabla \ell(\theta_t^i; b)$ 
10:        end for
11:       remove a fraction of  $\xi$  smallest values in  $\theta^k$ 
12:     end for
13:      $\theta_{t+1}^i = \theta_t^i + \frac{\eta}{n} \theta^k$ 
14:      $\Omega^k = f(\theta^k)$  (calculate the number of weight parameters)
15:      $\Omega_t = \Omega_t + \frac{\eta}{n} \Omega^k$ 
16:   end for
17: end for
18: Evaluate test accuracy through  $\theta^i$  and test dataset
19: Calculate test error as objective one  $f_i^1$ 
20: Set  $\Omega_t$  as objective two  $f_i^2$ 
21: end for
22: return  $f^1$  and  $f^2$ 

```

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 hyper parameters and the connectivity of the neural network are determined by the evolutionary algorithm, the weights will be trained using the mini-batch 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.

C. Multi-objective evolutionary optimization

The bi-objective 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 pseudo code is summarized in **Algorithm 5**.

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

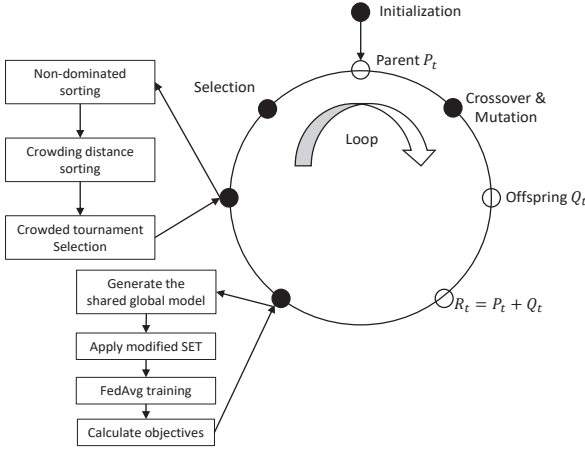


Fig. 5. A framework for multi-objective optimization of federated learning using NSGA-II.

Algorithm 5 Multi-objective evolutionary optimization

```

1: Randomly generate parent solutions  $P_t$  where  $|P_t| = M$ 
2: for each generation  $t = 1, 2, \dots$  do
3:   Generate offspring  $|Q_t| = M$  through crossover and mutation
4:    $R_t = P_t + Q_t$ 
5:   Evaluate  $f_t^1$  and  $f_t^2$  by Algorithm 4
6:    $f \leftarrow (f_t^1, f_t^2)$ 
7:   for each solution in  $R_t$  do
8:     Do non-dominated sorting and calculate crowding distance on  $f$ 
9:     Select high-ranking solutions from  $R_t$ 
10:    Let  $P_t = R_t$ 
11:   end for
12: end for
  
```

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 polynomial mutation [25] 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 non-dominance relationship and crowding distance. Finally, M high-ranking individuals from the combined population are selected as the parent of the next generation.

We repeat above procedure for several generations to generate a set of non-dominated 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.
- 4) SET parameters for sparse connection.

We select two popular neural network models: the multi-layer perceptron neural network (MLP) and the convolutional neural network (CNN), both trained and tested on a benchmark data set MNIST [26]. In optimizing both MLPs and CNNs, the mini-batch 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 mini-batch size B and training epochs E are 50 and 5, respectively. There are two ways of splitting MNIST dataset 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 dataset by the labelled 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 dataset and 50 generations on the non-IID dataset, 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 $n_m = 20$ [27] 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 less 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 work. In principle, these two parameters can also be binary coded and real coded, respectively, in genotypes for evolutionary optimization.

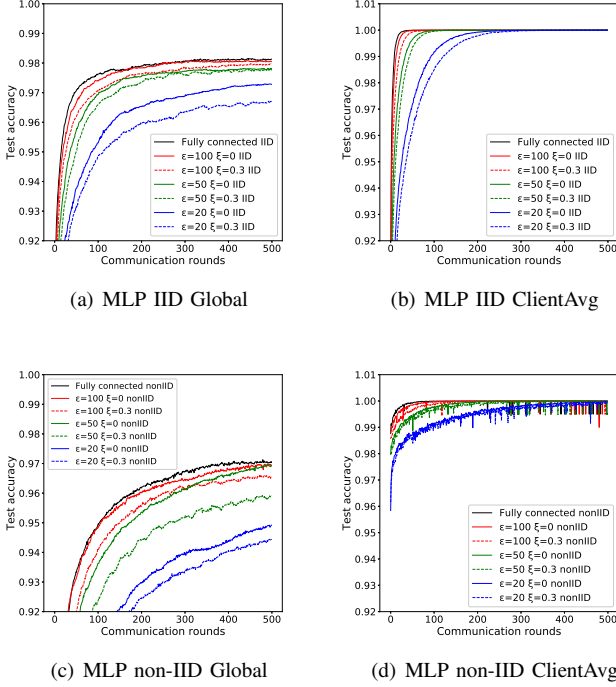


Fig. 6. The global model test accuracies and average client accuracies of MLPs on the IID and non-IID datasets. We select SET parameters ε and ξ to be (100,0), (100,0.3), (50,0), (50,0.3), (20,0), (20,0.3), and total communications rounds to be 500.

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 Fig. 6 and Fig. 7 for MLPs and CNNs, respectively.

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

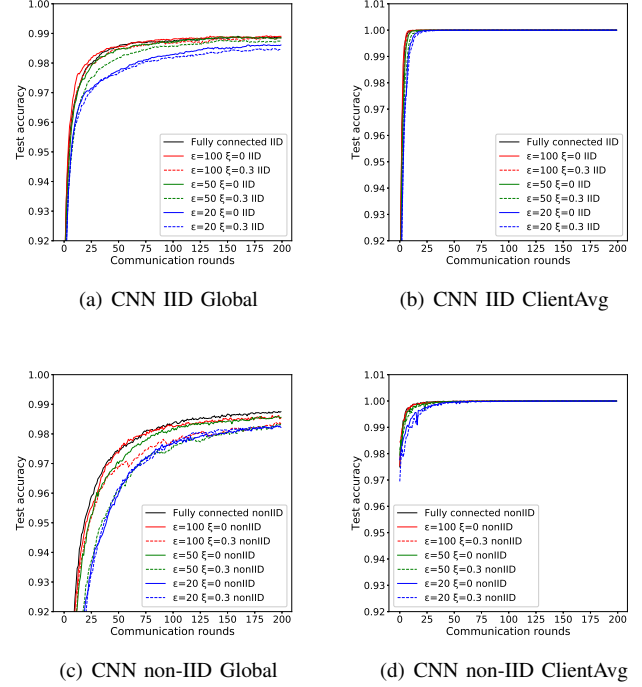


Fig. 7. The global model test accuracies and averaged client accuracies of CNNs on the IID and non-IID data sets. We select SET parameters ε and ξ to be (100,0), (100,0.3), (50,0), (50,0.3), (20,0), (20,0.3), and total communications rounds to be 200.

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. 5 (b)(d) and Figs. 6 (b)(d). By contrast, learning converges much slower on the server, in particular on the non-IID data, as shown in Figs. 6(a)(c) and Figs. 7(a)(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 in Figs. 6(a)(c) and Figs. 7(a)(c) is that reducing the network connectivity may lead to a degradation of the global test accuracies on both IID and non-IID dataset. 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 72051 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 87300 connections on average at the 500-th 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 complexity of the local models. The experimental results of the fully con-

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

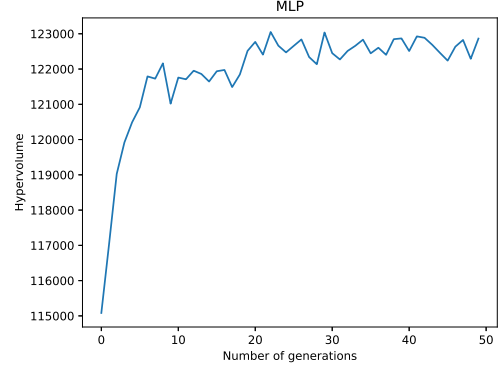
nected neural network model and mostly sparsely connected neural network model found by the proposed algorithm (whose SET parameters are $\varepsilon = 20, \xi = 0.3$) are listed in Table I. We can see that the global test accuracies of the sparsely connected MLPs (having about only 10% of the total 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 datasets. 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.

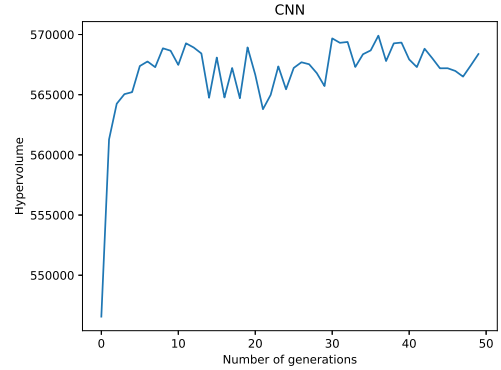
C. Evolved federated learning models

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 trade-off between global learning performance and communication costs in federated learning. Both IID and non-IID datasets 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 datasets still work on non-IID datasets, and vice versa.

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 one 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 non-dominated solution set in the population over the generations [28] in evolving MLP and CNN on non-IID datasets are plotted in Fig. 8. From the figure, we can see that the hypervolumes of both runs increase at the beginning and start fluctuating from around the 20-th generation onward. These results imply that approximately 20 generations are needed for federated learning to converge on non-IID datasets used in this work.



(a) Hypervolume for MLP



(b) Hypervolume for CNN

Fig. 8. Change of hypervolume over the generations in training MLP and CNN on non-IID datasets.

The total communication rounds for each population is set to be 5 for IID datasets and 10 for non-IID datasets, respectively, before the objective values are calculated. Of course, setting a 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, and 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.

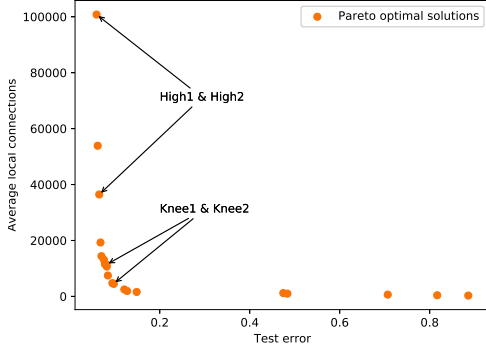
The range of the learning rate is between 0.01 and 0.3 for both MLPs and CNNs, because too large values may harm the global convergence in federated learning.

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.

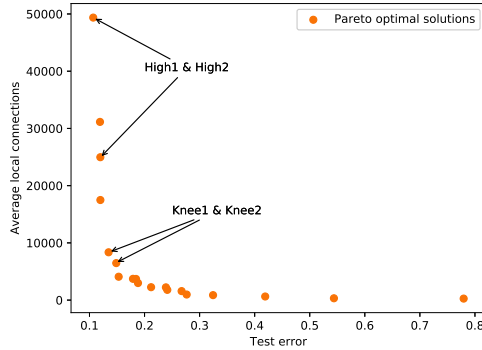
The final non-dominated MLP and CNN solutions optimized on the IID and non-IID datasets are presented in Fig. 9 and Fig. 10, respectively, where each point represents one solution corresponding to a particular structure of the neural network model in federated learning. However, not all non-

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



(a) Evolved Pareto frontier of MLPs trained on IID datasets

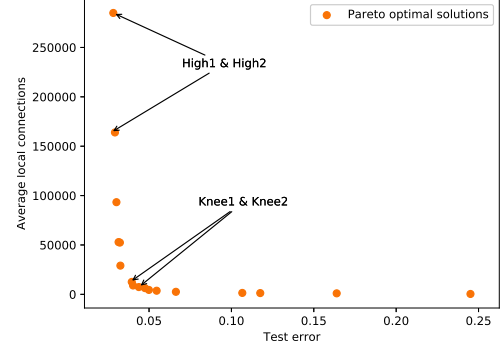


(b) Evolved Pareto frontier of MLPs trained on non-IID datasets

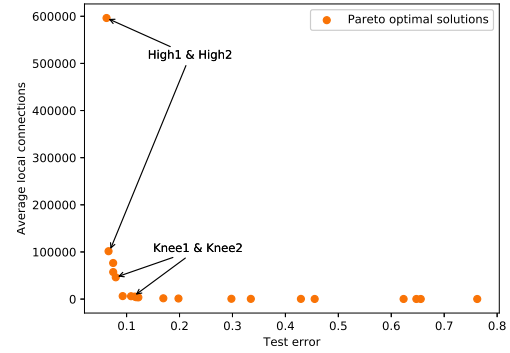
Fig. 9. Pareto frontier of MLPs, of which four solutions, High1, High2, Knee1, Knee2 are selected for validation.

dominated 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 work, we select two types of non-dominated solutions, namely those solutions with a very low global test error, and those solutions near the knee point of the frontier, as suggested in [20], [29].

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 Fig. 9 and Fig. 10) for further performance verification and compare their



(a) Evolved Pareto frontier of CNNs trained on IID datasets



(b) Evolved Pareto frontier of CNNs trained on non-IID datasets

Fig. 10. Pareto frontier of CNNs, of which High1, High2, Knee1, and Knee2 are selected for validation.

TABLE III
HYPER-PARAMETERS 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

performance with the fully connected MLPs and CNNs. Recall that only 5 and 10 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 communication rounds is increased to 500 for MLPs and 200 for CNNs, as set in the original federated learning. All validation results are listed in Tables III, IV, V, and VI, and the global test accuracies of the selected solutions are also presented in Fig. 11 and Fig. 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.

- Solution High1 of MLP has global test accuracies of

TABLE IV
HYPER-PARAMETERS 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
HYPER-PARAMETERS OF HIGH1, HIGH2, KNEE1, AD 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

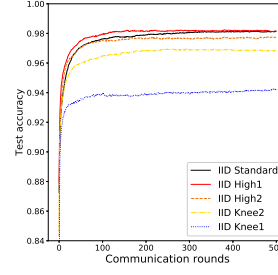
98.16% and 97.42% on IID and non-IID datasets, both of which are better than that of the fully connected MLP. In addition, this evolved model has only on average 91,933 and 91,527 connections on IID data and non-IID data, respectively, which is approximately 46% of 199,210 connections the fully connected network has.

- 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.
- Knee1 and Knee2 have test accuracies of 96.84% and 94.24%, respectively, on IID datasets. 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.

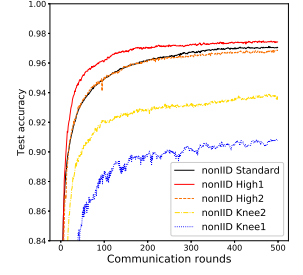
Similar observations can be made on the two high-accuracy

TABLE VI
HYPER-PARAMETERS OF HIGH1, HIGH2, KNEE1, AND KNEE2 FOR *CNNs*
EVOLVED ON *non-IID* DATA AND THEIR VALIDATION RESULTS

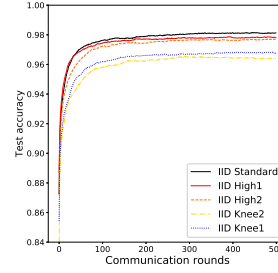
Parameters	Knee1	Knee2	High1	High2	Standard
Conv layer1	17	5	53	33	32
Conv layer2	/	/	/	/	64
Fully connected layer1	29	21	208	31	128
Fully connected layer2	/	/	/	/	/
Kernel size	5	5	5	5	3
ε	18	8	66	20	/
ξ	0.1451	0.1892	0.0786	0.1354	/
Learning rate η	0.2519	0.2388	0.2776	0.2503	0.1
Test accuracy IID	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



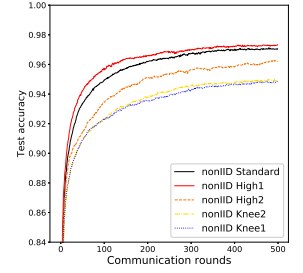
(a) Solutions evolved on IID data and validated on IID data



(b) Solutions evolved on IID data and validated on non-IID data

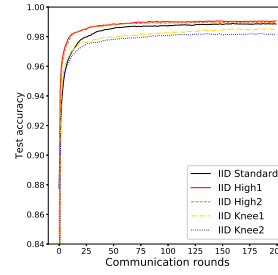


(c) Solutions evolved on non-IID data and validated on IID data

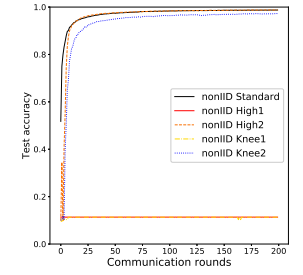


(d) Solutions evolved on non-IID data and validated on non-IID data

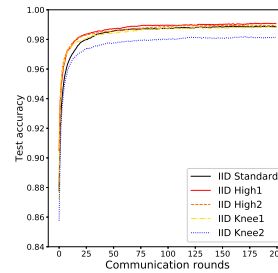
Fig. 11. The global test accuracies of the selected Pareto optimal MLPs validated on both IID and non-IID data. The test accuracies of the fully connected MLP are also plotted in the figure for comparison.



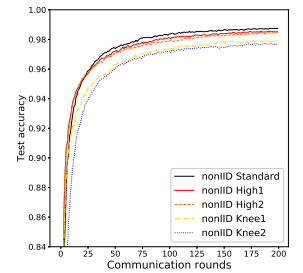
(a) Solutions evolved on IID data and validated on IID data



(b) Solutions evolved on IID data and validated 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

Fig. 12. The 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.

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 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.

The following conclusions can be drawn on the four selected solutions evolved on non-IID data.

- Of the four Pareto optimal MLPs, the test accuracies of High1 is 0.28% higher than the fully connected MLP on non-IID data and 0.31% lower when validated on IID data, while High1 has only about 45,000 connections, which is 22% of the connections of the fully connected MLP. In contrast to the models evolved on IID data, Pareto optimal solutions evolved on non-IID data still work slightly better when they are validated on IID datasets.
- The two high-accuracy CNN solutions, High1 and High2, exhibits a test accuracy of 99.06% and 98.93%, respectively, when they are validated on IID data, which are 0.21% and 0.08% higher than the fully connected CNN. They have, however, only around 34% and 5.5% of the connections of the fully connected model. Knee1 also has a test accuracy of 98.84% (the fully connected one has an accuracy of 98.85%), but has only 2.4% of connections compared to the fully connected CNN model. All solutions work well when they are validated on non-IID, although their test accuracies are slightly lower than the fully connected model. Specifically, the accuracies of the two high-accuracy solutions are about 0.4% lower, and the accuracy of the two knee solutions are about 0.8% lower than the fully connected model. Of course, all solutions have much less connections on average than the fully connected one.

Below are a few additional interesting findings from our experiment. First, the model structures evolved on IID datasets are always deeper than that evolved on non-IID datasets. Second, the optimized learning rates are near 0.3 for MLPs. Third, the learning process on the server diverges when the numbers of connections becomes dramatically small. Fourth, the local models may have different complexities when the data on different clients are different, even if the global model has the same structure.

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 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 and 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 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.

V. CONCLUSIONS AND FUTURE WORK

This work proposes a multi-objective federated learning to simultaneously maximize the learning performance and minimize the communication cost. To improve the scalability in evolving deep neural networks, a modified SET method is adopted to indirectly encode the connectivity of the neural network. Our experimental results show that the modified SET algorithm can effectively reduce the connections of deep neural networks by encoding only two hyper parameters. Selected solutions from the non-dominated frontier obtained by the multi-objective algorithm have higher global test accuracies and much less connections in the network, thereby dramatically reduce the communication cost without deteriorating the learning performance on both IID and non-IID datasets.

Our experimental results also confirm that there is a trade-off between learning performance and model complexity and models with a smaller number of connections often have lower accuracies. Meanwhile, the Pareto optimal solutions trained on non-IID datasets can always perform well on IID data, but not vice versa. The MLP solutions near the knee point trained on IID data usually cannot perform very well on non-IID data, while those high-accuracy Pareto optimal MLPs do not suffer from this problem. For CNNs, it is surprising to observe that solutions optimized on IID datasets completely fail to work on non-IID datasets, indicating that CNNs evolved using IID data may not be robust to data distribution changes.

A lot of work 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 datasets like non-IID CIFAR-10. In addition, it is still unclear if missing data caused by package loss from communications between clients and the server will significantly affect the performance of federated learning. Adversarial attacks [30] 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.

VI. ACKNOWLEDGEMENT

We are grateful to Y. Zhao for sharing his code.

REFERENCES

- [1] J. Poushter *et al.*, “Smartphone ownership and internet usage continues to climb in emerging economies,” *Pew Research Center*, vol. 22, pp. 1–44, 2016.
- [2] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, *et al.*, “Communication-efficient learning of deep networks from decentralized data,” *arXiv preprint arXiv:1602.05629*, 2016.
- [3] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, “Practical secure aggregation for privacy-preserving machine learning,” in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1175–1191, ACM, 2017.
- [4] A. A. Atayero and O. Feyisetan, “Security issues in cloud computing: The potentials of homomorphic encryption,” *Journal of Emerging Trends in Computing and Information Sciences*, vol. 2, no. 10, pp. 546–552, 2011.
- [5] C. Dwork, “Differential privacy: A survey of results,” in *International Conference on Theory and Applications of Models of Computation*, pp. 1–19, Springer, 2008.
- [6] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, “Deep learning with differential privacy,” in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 308–318, ACM, 2016.
- [7] R. Shokri and V. Shmatikov, “Privacy-preserving deep learning,” in *Proceedings of the 22nd ACM SIGSAC conference on Computer and Communications security*, pp. 1310–1321, ACM, 2015.
- [8] R. C. Geyer, T. Klein, and M. Nabi, “Differentially private federated learning: A client level perspective,” *arXiv preprint arXiv:1712.07557*, 2017.
- [9] R. McDonald, K. Hall, and G. Mann, “Distributed training strategies for the structured perceptron,” in *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 456–464, Association for Computational Linguistics, 2010.
- [10] D. Povey, X. Zhang, and S. Khudanpur, “Parallel training of dnns with natural gradient and parameter averaging,” *arXiv preprint arXiv:1410.7455*, 2014.
- [11] S. Zhang, A. E. Choromanska, and Y. LeCun, “Deep learning with elastic averaging sgd,” in *Advances in Neural Information Processing Systems*, pp. 685–693, 2015.
- [12] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, “Federated learning with non-iid data,” *arXiv preprint arXiv:1806.00582*, 2018.
- [13] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, “Federated learning: Strategies for improving communication efficiency,” *arXiv preprint arXiv:1610.05492*, 2016.
- [14] X. Yao, “Evolving artificial neural networks,” *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1423–1447, 1999.
- [15] K. O. Stanley and R. Miikkulainen, “Evolving neural networks through augmenting topologies,” *Evolutionary Computation*, 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 *Proceedings of 25th European Conference on Modeling and Simulation ECMS*, pp. 410–416, 2011.
- [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 Communications*, vol. 9, no. 1, p. 2383, 2018.
- [18] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, p. 436, 2015.
- [19] B. McMahan and D. Ramage, “Federated learning: Collaborative machine learning without centralized training data,” *Google Research Blog*, 2017.
- [20] Y. Jin and B. Sendhoff, “Pareto-based multi-objective machine learning: An overview and case studies,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 38, no. 3, pp. 397–415, 2008.
- [21] D. Whitley, “A genetic algorithm tutorial,” *Statistics and Computing*, vol. 4, no. 2, pp. 65–85, 1994.
- [22] P. Erdos and A. Rényi, “On the evolution of random graphs,” *Publ. Math. Inst. Hung. Acad. Sci.*, vol. 5, no. 1, pp. 17–60, 1960.
- [23] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: Nsga-ii,” *IEEE transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.
- [24] K. Deb, “Multi-objective optimization,” in *Search methodologies*, pp. 403–449, Springer, 2014.
- [25] R. B. Agrawal, K. Deb, and R. Agrawal, “Simulated binary crossover for continuous search space,” *Complex Systems*, vol. 9, no. 2, pp. 115–148, 1995.
- [26] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [27] K. Kumar, “Real-coded genetic algorithms with simulated binary crossover: studies on multimodal and multiobjective problems,” *Complex Syst.*, vol. 9, pp. 431–454, 1995.
- [28] N. Beume, B. Naujoks, and M. Emmerich, “SMS-EMOA: Multiobjective selection based on dominated hypervolume,” *European Journal of Operational Research*, vol. 181, no. 3, pp. 1653–1669, 2007.
- [29] X. Zhang, Y. Tian, and Y. Jin, “A knee point-driven evolutionary algorithm for many-objective optimization,” *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 6, pp. 761–776, 2015.
- [30] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, “How to backdoor federated learning,” *arXiv preprint arXiv:1807.00459*, 2018.