Toward Resource-Efficient Federated Learning in Mobile Edge Computing

Rong Yu and Peichun Li

Abstract

Federated learning is a newly emerged distributed deep learning paradigm, where the clients separately train their local neural network models with private data and then jointly aggregate a global model at the central server. Mobile edge computing is aimed at deploying mobile applications at the edge of wireless networks. Federated learning in mobile edge computing is a prospective distributed framework to deploy deep learning algorithms in many application scenarios. The bottleneck of federated learning in mobile edge computing is the intensive resources of mobile clients in computation, bandwidth, energy, and data. This article first illustrates the typical use cases of federated learning in mobile edge computing, and then investigates the state-of-the-art resource optimization approaches in federated learning. The resource-efficient techniques for federated learning are broadly divided into two classes: the black-box and white-box approaches. For black-box approaches, the techniques of training tricks, client selection, data compensation, and hierarchical aggregation are reviewed. For white-box approaches, the techniques of model compression, knowledge distillation, feature fusion, and asynchronous update are discussed. After that, a neural-structure-aware resource management approach with module-based federated learning is proposed, where mobile clients are assigned with different subnetworks of the global model according to the status of their local resources. Experiments demonstrate the superiority of our approach in elastic and efficient resource utilization.

Introduction

Recently, federated learning has attracted a lot of attention as a novel distributed deep learning paradigm [1]. In federated learning, the clients separately train their local deep neural network (DNN) models with their private data. The local model updates are then sent to the central server, while the private data remain on the clients. After collecting all local updates, the central server is responsible for aggregating a new global model, which will be delivered to the clients for the next round of model training. This distributed training iteration repeats until the global model converges to a satisfying test accuracy. By using federated learning, the individual privacy of clients could be effectively preserved as no private data is shared among the clients and the central server.

Mobile edge computing [2] is a rapidly developing technology that aims to deploy mobile applications at the edge of wireless networks by exploiting the underutilized computation and communication resources of edge devices, such as edge servers and mobile clients. As a complement to traditional centralized cloud computing, mobile edge computing has exhibited great potential in significantly reducing the traffic load of core networks, relieving the processing pressure of central servers, shortening the response latency of end-to-end operations, and therefore enhancing the overall system performance of wireless networks.

The fast-growing demands on artificial intelligence (AI) and deep learning applications call for a practically efficient computing pattern to support the system implementation. Federated learning in mobile edge computing, where mobile clients undertake the tasks of local model training, offers a prospective distributed framework to deploy deep learning applications. But the challenge is that federated learning is rather resource-consuming for mobile edge computing. On one hand, the training of a deep learning model usually requires massive computation resource, which is a heavy burden for mobile clients. On the other hand, as a distributed learning paradigm, federated learning also occupies considerable bandwidth resource, such that up to hundreds of communication rounds may be needed between the central server and the clients for model iterations. The deployment of federated learning in mobile edge computing has to overcome the difficulty of intensive resources in mobile clients. As a consequence, efficient resource management is crucial for federated learning in mobile edge computing.

The resource management for federated learning in mobile edge computing involves the joint optimization of multiple types of resources such as computation, bandwidth, energy, and data. In the literature, there are some efforts to tackle the problem but still far from enough. In this article, we investigate the state-of-the-art resource management approaches for federated learning in mobile edge computing. From the perspective of resource optimization, the applied techniques in existing work are broadly classified into two categories: the *black-box* and *white-box* approaches, depending on whether the optimization takes into consideration the internal structure of deep learning models. The details of these two classes of approaches are discussed later.

One of the shortcomings of most existing approaches is that they usually limit the clients to use an identical neural architecture as the central server. However, in many application areas, such

Digital Object Identifier:
10.1109/MNET.011.2000295

Rong Yu (corresponding author) is with Guangdong University of Technology; Peichun Li is with Guangdong Key Laboratory of IoT Information Technology.

0890-8044/21/$25.00 © 2021 IEEE

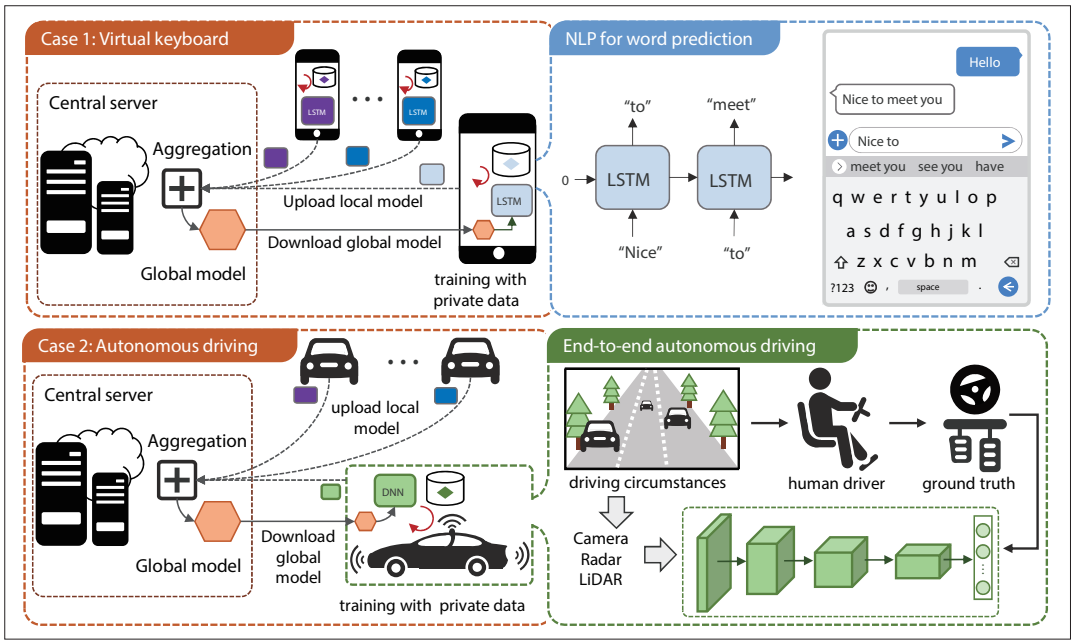

FIGURE 1. Application scenarios of federated learning in mobile edge computing.

as edge computing and the Internet of Things, there may be a variety of clients with diverse computation and communication capability. In this situation, the neural architectures should be adapted to match the clients' hardware configurations. Also, battery-powered devices tend to work in the low-power mode for energy saving. Employing a full-size DNN model in high-battery status and switching to a small submodel in low-battery status is an effective way to maintain the performance while prolonging the device lifetime [3].

In this article, we briefly overview the existing work and attempt to establish a new framework for federated learning to efficiently deal with the heterogeneous resources among massive edge devices. We propose a neural-structure-aware resource management approach, where mobile clients are assigned to different subnetworks of the global model according to the status of their local resources. Our approach offers an elastic learning framework for emerging applications such as edge computing and the Internet of Things, where clients have limited computation, bandwidth, power, and data resources.

The rest of the article is organized as follows. The following section takes two examples to illustrate the application scenarios of federated learning in mobile edge computing. Following that, we review the existing work on resource optimization in federated learning as the black-box and white-box approaches. Then an elastic resource management framework with module-based federated learning is discussed. The final section concludes the article and points out future work.

Federated Learning in Mobile Edge Computing

Federated learning has been leveraged to support a variety of deep learning applications in mobile edge computing. In the typical scenarios of federated learning in mobile edge computing, the mobile clients themselves participate in the local model training so as to protect data privacy. In this section, we take the virtual keyboard application on mobile phones and end-to-end autonomous driving as examples. We also discuss the constrained resources in communication, computation, power, and data for federated learning in mobile edge computing.

Typical Application Scenarios

Virtual Keyboard: The virtual keyboard is the very first application that the Google AI group used for testing when they proposed the concept of federated learning [4]. As shown in Fig. 1, the virtual keyboard application on mobile phones employs a natural language processing (NLP) deep learning model for word prediction. The actions of keyboard inputs and word selections are stored as local private data for training. The virtual keyboard with federated learning works as follows:

- The mobile phone downloads the global model from the central server and improves it by training on local data.
- Only the update of the model is sent to the central server by using a secure encrypted communication link.
- All the local model updates are immediately averaged by the central server to improve the global model.
- The iteration of model training and aggregation repeats until the global model converges. All the training data remains on the mobile phone. No private data is shared.

End-to-End Autonomous Driving: Following the principle of imitative learning, end-to-end autonomous driving is designed to emulate the manipulations of human drivers. A powerful onboard hardware platform and a deep learning model have been reported by Nvidia Inc. to support the implementation [5]. Figure 1 also illustrates end-to-end autonomous driving. The driving circumstances are perceived by onboard sensors such as cameras, radar, and lidar. The human driver's operations on steering wheel, gas, and brake pedals are captured as ground truth. By federated

reinforcement learning, the training procedure takes place as follows:

- The vehicle downloads the deep learning model from the central server for local model training.
- The captured images and point cloud data of onboard sensors are the input data of the model. The output predicted operations on steering wheel, gas, and brake pedals are compared to the ground truth of the human driver. The reward is computed and fed back to adjust the model by a backpropagation algorithm.
- The vehicle uploads the update of the local model to the central server, which will aggregate it with the other vehicles' updates to renew the global model.
- The iteration continues until the global model converges and the current round of training ends.

Learning with Constrained Resources

Compared to the traditional centralized model training, federated learning in mobile edge computing has to deliver training tasks to mobile clients, which generally have intensive resources in communication, computation, power, and data.

Communication Resource: In federated learning, up to hundreds of communication rounds may take place before the model converges. Regarding the variability and heterogeneity of wireless connections in mobile edge computing, the communication resource of mobile clients should be appropriately arranged. Traditional radio resource management such as bandwidth allocation and rate control, and the new technologies of non-orthogonal multiple access (NOMA) and network function virtualization (NFV) may be leveraged to improve the communication efficiency.

Computation Resource: In mobile edge computing, the mobile clients are computation-intensive due to practical constraints such as cost, size, and power consumption. Model training should be rethought and take into consideration the limited computation resource on mobile clients. Applying computation virtualization technologies like virtual machines (VMs) and containers (e.g., Docker and Kubernetes) for deep learning algorithm execution is a new trend to increase the computation elasticity and efficiency in recent years.

Power Resource: The clients in mobile edge computing are mostly battery-powered devices. Power management is essential to prolong the lifetime of mobile clients. The system should balance the learning performance and power consumption by scheduling mobile clients to switch between performance-optimal and power-optimal modes. The power consumption of federated learning in mobile edge computing mainly comes from two sources: the local computation to produce model updates and the wireless communication to upload model updates. In addition, training local models with lower operating frequency and sending model updates at lower transmission rate are beneficial for power saving.

Data Resource: One of the unique challenges of federated learning is that no data is shared among the clients and the central server. This implicitly causes the scarcity of training data and also the non-independent identical distribution (non-IID) among local datasets. For better learning accuracy under non-IID data, the distributed data should be used equally from local datasets. An interesting attempt to overcome the data deficit is by employing deep generative models like variational auto-encoder (VAE) and generative adversarial network (GAN) to generate new data samples. Besides that, another approach is to exploit the highly efficient label-less learning method [6], which has been demonstrated to significantly improve data utilization and facilitate data processing.

Black-Box vs. White-Box Approaches

In the literature, researchers have made efforts to propose approaches to resource optimization in federated learning. For ease of comparison, we broadly divide these studies into two categories: black-box and white box approaches. The black-box approaches, following the principle of traditional networking optimization, tend to arrange the network entities and system resources according to the observation on the external properties of the involved deep learning models. In contrast, the white-box approaches try to understand, at least partially, the internal structure as well as module functionality of the deep learning models. Network entities and system resources are configured along with the neural structure adaption for optimality. An overview of these two types of approaches is presented in the following.

Black-Box Approaches

Training Tricks: Using training tricks is a simple and effective way to improve the performance of federated learning. In [1], the hyperparameters of training, such as gradient descent strategies, batch size, epochs, learning rate, and decay rate, are empirically tuned to reduce the number of communication rounds. Additionally, a data augmentation method is also applied to enhance the inference accuracy. The experiment results show that these training tricks significantly reduce the communication rounds by one to two orders of magnitude on five different model architectures and four benchmark datasets.

Client Selection: The duration of one communication round is determined by the client with the longest computation and communication time, as synchronous model aggregation is adopted for federated learning in mobile edge computing. For this reason, it is crucial to select a set of appropriate clients with fine wireless channel conditions and sufficient computation capability. "Weak" clients will be less involved to avoid delaying the parameter aggregation [7]. For the sake of fairness and data distribution balance as well, the client scheduling should make up for the weak ones if their wireless channel conditions and computation resource recover to the average status.

Data Compensation: Previous theoretical analysis has revealed that the heterogeneity of data distribution among clients is a critical factor of restraining the convergence rate of federated learning. To balance the data heterogeneity, in [8], a small subset of local data from each client is globally shared to alleviate the degree of non-IID,

and hence potentially improve the model accuracy. In spite of having a performance enhancement, this approach has to take the risk of privacy leakage due to data sharing. In [9], a data compensation framework with the GAN model is proposed. Compressed data samples from clients are forwarded through multiple hops to the server for the GAN training. After that, the clients download the trained GAN to augment data quality for local model training. In this way, the non-IID data is compensated at the cost of additional computation workload of GAN model training and the underlying threat of data leakage among clients.

Hierarchical Aggregation: By incorporating the hierarchical network topology in federated learning, model aggregation could be conducted in a client-edge-cloud multi-stage procedure, that is, first through one or two stages of local aggregations at edge servers and then the global aggregation at the central server. In [10], multiple intermediate edge servers are introduced to perform multiple local aggregations before the global aggregation at the central server. The rationale behind this multi-stage aggregation is that the early model aggregations at the network edge are of lower communication cost and able to efficiently alleviate the uncertainty of model updates due to the randomness of local data. Once the central server has the refined model updates, the convergence of global aggregation will be fairly fast.

White-Box Approaches

Model Compression: Model compression consists of a set of optimization techniques to shrink the model size for easy deployment in a hardware environment, and hence, it is widely exploited to reduce the communication overhead in distributed learning. In [11], both quantization and low-rank approximation techniques are leveraged to minimize the communication cost in federated learning. The experiment there reports cost conservation of communication overhead by two orders of magnitude. Furthermore, sparsification techniques like gradient pruning have also been verified to be an efficient alternative for model compression in practice.

Knowledge Distillation: The core idea of knowledge distillation comes from the observation that a portion of the activations, for example, the soft probability distribution from the softmax output, of a well-trained deep learning model could be utilized as an additional regularizer to "teach" a new model. In fact, knowledge distillation is a highly efficient technique of knowledge transfer. For instance, the size of softmax output is tiny but carries rich information about the model knowledge. Inspired by this idea, knowledge-distillation-based federated learning in [12] chooses to share the class scores among the clients and aggregate to get a consensus for model updates. There is a gain of 20 percent in test accuracy for all participants, as reported. But unfortunately, in reality, this approach heavily relies on a large public dataset that is not commonly available.

Feature Fusion: By employing feature fusion technique in federated learning, the work in [13] proposes a two-stream feature extraction and fusion at the clients. More concretely, after receiving the global model update from the central server, each client copies the global feature extractor to replace the local one. The two-stream outputs from the global and local extractors are merged by the feature fusion module and then passed to the classifier for loss evaluation. For all training batches, the back-propagation will sequentially renew the classifier, the fusion module, and the local extractor, except for the global extractor. The global extractor acts as a regularizer to adjust the update of the local model, such that it is fit for both the local and global data distributions. The approach is effective in balancing the non-IID data and decreasing the communication rounds at the cost of extra computing overhead for the global feature extraction and fusion.

Asynchronous Update: Borrowing the method of asynchronous learning, the work in [14] proposes an asynchronous model update strategy for federated learning. Through a layerwise analysis, the approach splits the DNN into two parts: shallow layers and deep layers. The former will be updated more frequently than the latter, because it learns more general features, and therefore is more critical to the global update than the latter. Meanwhile, using the framework of asynchronous learning, the central server is allowed to aggregate models without collecting all the local ones. Temporal coefficients corresponding to the time duration are multiplied to the model weights for better aggregation. This asynchronous update approach is flexible and efficient in the time dimension for model aggregation. But the convergence curves in the experiment show that the learning procedure might have to experience unusual fluctuation in test accuracy.

For comparison, a list of the aforementioned black-box and white-box approaches for resource management in federated learning is shown in Table 1. Empirically, there are some suggestions for applying these approaches in practice:

- Using synchronous update is simple, efficient, and practically useful in most scenarios.
- Model compression should always be applied with top priority to reduce communication overheads.
- By leveraging edge computing, hierarchical aggregation could reduce traffic among client-edge-cloud entities.
- There is no conflict in applying both black-box and white-box approaches at the same time. The incorporation of these approaches may "double" the performance gains.

Neural-Structure-Aware Resource Management

In this section, we propose a module-based white-box approach where the global model is partitioned into multiple submodels for training. With model partition, different clients may be assigned to jointly train an identical submodel or separately train different submodels using local resources. From the standpoint of distributed training, this novel federated learning framework has concurrently integrated the two parallel computing patterns of data parallelism and model parallelism. Accordingly, the resources of mobile clients could be efficiently orchestrated to fulfill the submodel training.

Type	Technique	Statement	Pros	Cons
Black-box	Training tricks [1]	Hyperparameter setting, data augmentation, etc.	Simple and effective	Empirical tuning needed in every single case
	Client selection [7]	Selecting clients with sufficient communication and computation resources	Leveraging the diversity of clients	Implicitly causing unbalanced usage of clients' data
	Data compensation [8, 9]	Balancing non-IID data by partial data sharing [8] or GAN model [9]	Improving data efficiency	High risk of privacy leakage [8] or heavy computing overhead [9]
	Hierarchical aggregation [10]	Exploiting edge servers for local aggregations before global aggregation	Taking advantage of edge computing	Additional system cost for edges and clients management
White-box	Model compression [11]	Reducing parameters by pruning, quantization, sparsification, etc.	Useful in reducing the communication payload	Inducing unpredictable loss of model accuracy
	Knowledge distillation [12]	Extracting activations (e.g., softmax output) as regularizer for training	Using knowledge transfer to enhance efficiency	Relying on a public dataset that is not commonly available
	Feature fusion [13]	Merging features of local and global models in local iteration	Effective in balancing non-IID data	Extra computing cost for global feature extraction and fusion
	Asynchronous update [14]	Local (sub)models updated and aggregated asynchronously	Temporally flexible in model aggregation	Suffering from unusual fluctuation in convergence

TABLE 1. Black-box and white-box approaches.

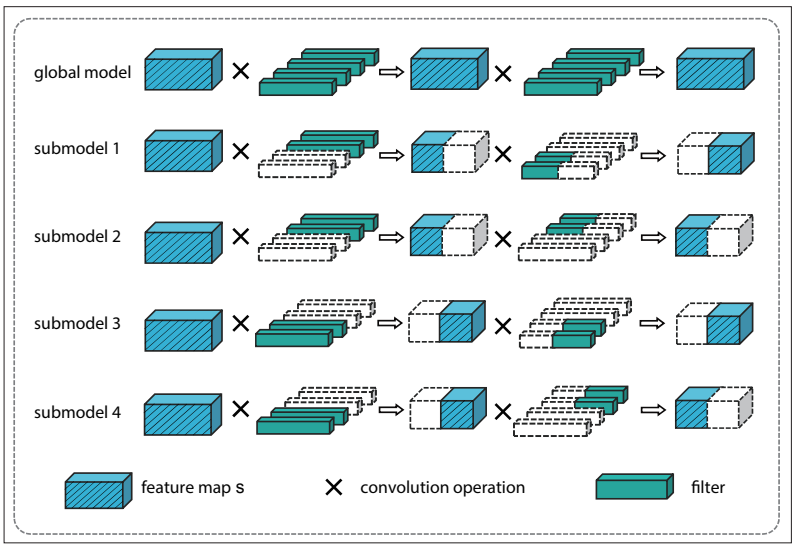

FIGURE 2. Filter grouping.

Model Partition and Aggregation

Without loss of generality, we take the convolutional neural network (CNN) as an example, and consider model partition by width (i.e., the number of filters in convolutional layers). As shown in Fig. 2, the global model is divided into multiple submodels by grouping the filters into blocks. We define *partitioning factor* of a layer as the number of output channel groups of the layer. Partitioning factors might be different for different layers and usually given values by the powers of two. A submodel is composed of any number of filter blocks from all convolutional layers, along with the input, output, and fully connected layers. In Fig. 2, the two-layer CNN with partitioning factor {2, 2} means that the numbers of output channels of filter blocks in the first and second layers are shrunk to 1/2 of the global model. For filter blocks in the first layer, the input stays the same as the global model, but the output is reduced to 1/2, and thus the size of each block in the first layer is $1 \times 1/2 = 1/2$ of that of the global model. For the second layer, both input and output are reduced to 1/2, and thus, the size of the filter block is reduced by $1/2 \times 1/2 = 1/4$. The four optional submodels 1 to 4 with different filters for the structure of filter block sizes (1/2, 1/4) are shown in Fig. 2. Also, there are structures of filter block sizes (1/2, 1/2) and (1, 1/2), but those are not shown here. The submodel search algorithm is discussed later.

During model training, it is critical to balance the use of local data in clients. The central server should periodically exchange the filter blocks among submodels to ensure that all filters of the global model will be equally trained by all clients. This submodel exchange should be elaborately conducted to fairly utilize the local data, meanwhile maintaining the inter-layer connections of the submodels with best effort. A simple submodel exchange strategy is illustrated in Fig. 3, where each client is arranged to sequentially train the four submodels in Fig. 2 one by one for every round of iteration.

For model aggregation, the central server should reassemble all the partitioned submodels and reconstruct the global model. Notice that multiple copies of the global model may be partitioned and trained at the same time. These copies of the global model may be partitioned in different ways and trained by different clients. Define multiplexing factor as the number of global model copies. When aggregating the global model, the central server will first reconstruct all the copies of the global model, and then perform gradient aggregation among the copies. The aggregation coefficients of model gradients are proportional to the sizes of the local datasets, as the same rule as in traditional federated learning applies.

Aggregate Gradient Information

Next, we investigate how much the different submodels contribute to the learning performance. We are interested in the amount of training information that is carried by a submodel when it is uploaded to the central server for aggregation. For a CNN model with L layers, at the ith layer,

there are m_i neural computing units, that is, the filters of a convolutional layer or the weights of a fully connected layer. Previous studies have disclosed that the representation capability of a neural network grows exponentially with the depth and polynomially with the width of the model. Following this scaling law and borrowing the form of capacity in [15], the capacity of such a CNN could be approximately estimated by $C = \log_2(\Pi_i \beta_i m_i) = \Sigma_i \log_2(\beta_i m_i)$, where β_i is a constant that explicitly reflects the number of representable functions of a computing unit of the ith layer (e.g., the kernel size of a convolutional layer). In this form, the capacity of a neural network is viewed as an information-theoretic measure of its representation capability.

In federated learning, the clients should report gradient information to the central server in each round of iteration. The more accurate the reported gradients, the better the learning efficiency. To express the learning efficiency of a specific neural network model, we propose the concept of aggregate gradient information (AGI). Let α_i denote the number of tunable parameters of a computing unit of the ith layer. The AGI is defined by

$$I = \sum_{i=1,\ldots,L} \alpha_i m_i \log_2(\beta_i m_i). \quad (1)$$

Here, AGI is interpreted as the effective information that is learned during training and recorded in the neural model.

From an information-theoretic perspective, we treat the training procedure of federated learning as the process of gradually reducing the *uncertainty* about the optimal model parameters. We aim to find out the optimal weight W^* with the minimum value of loss function. In each round of iteration, the model gradient could be viewed as a measure of the uncertainty of the optimal W^*. The larger the gradient's absolute value, the more the uncertainty. Generally, the gradient's absolute value decreases round by round, and eventually at W^*, the uncertainty reaches zero. Apparently, the larger the neural network capacity, the better it presents the gradient, and thus the more information it carries. In this sense, AGI is also an indicator that reflects the accuracy of the gradient for the parameter update during model training.

Resource-Efficient Learning Problem

To allocate resources for model training, we evaluate the energy and time costs for different submodel structures. Let j and k index the clients and the rounds of iterations in federated learning; $M_{j,k}$, $f_{j,k}$, and $r_{j,k}$ denote the neural structure, computing frequency and transmitting rate of the jth client in the kth round of iteration, respectively. Given the structure $M_{j,k}$, the model size $W_{j,k}$ is easy to calculate, while the training workload per datapoint $Q_{j,k}$ could be approximately computed or empirically measured. We consider the condition of symmetric wireless channels, that is, the uplink and downlink have same channel status. Meanwhile, let $E^c_{j,k}$, $E^t_{j,k}$, $T^c_{j,k}$, and $T^t_{j,k}$ denote the computing energy cost, transmitting energy cost, computing time cost, and transmitting time cost of the jth client in the kth round of iteration. We have

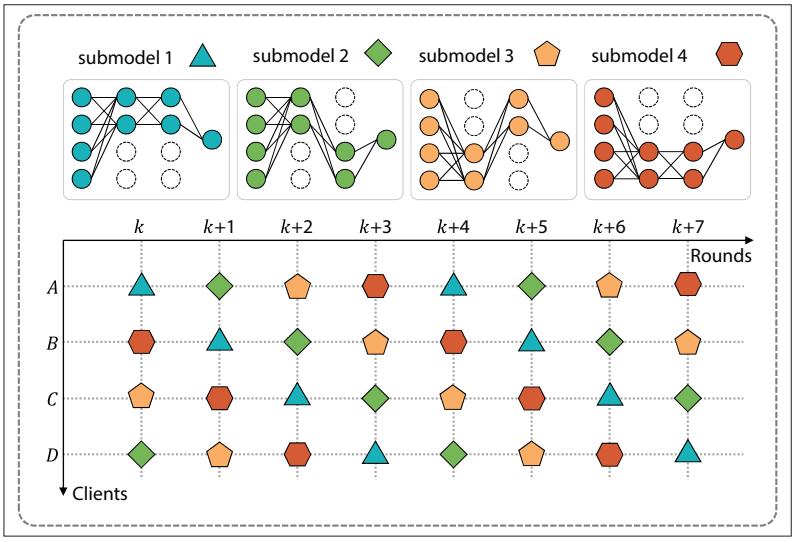

FIGURE 3. Submodel exchanging.

$$E^c_{j,k} = \varepsilon_c Q_{j,k} f^2_{j,k} D_j,$$

$$E^t_{j,k} = \frac{2N_0 W_{j,k}}{|h_{j,k}| r_{j,k}} 2^{\frac{r_{j,k}}{B}-1},$$

$$T^c_{j,k} = \frac{Q_{j,k}}{f_{j,k}}, \quad T^t_{j,k} = \frac{2W_{j,k}}{r_{j,k}},$$

where ε_c is the power consumption coefficient related to the hardware architecture of the processor, D_j is the number of training datapoints of the jth client, and N_0, B, and $|h_{j,k}|$ are the Gaussian noise power at the radio receivers, the wireless bandwidth, and the path loss of wireless channel of the jth client at the kth round of iteration, respectively. As a result, the total energy cost E_k could be calculated by adding up the corresponding energy costs of communication and computation of all clients. The total time cost T_k is the maximum of that of all clients, and for each client, its time cost is the sum of that for communication and computation.

Let $I_{j,k}$ denote the AGI of the jth client in the kth round of iteration. For a given power budget P_{bud}, the resource-efficient federated learning problem is formulated by

$$\max_{\{M_{j,k}, f_{j,k}, r_{j,k}\}} U_k = \frac{\sum_j I_{j,k}}{E_k T_k}, \quad (2)$$

subject to $0 < E_k \le P_{bud} T_k$. Note that if the clients have different dataset sizes, each client's AGI will have a weight that is proportional to the logarithm of its dataset size. The objective of Eq. 2 is to maximize the total AGI at the costs of energy and time consumption (i.e., to improve model learning efficiency in each iteration). Figure 4 describes the neural-structure-aware resource management approach.

Heuristic Search Algorithm

Given the clients, the power budget, and the specific CNN model with partitioning and multiplexing factors, to solve Eq. 2, the heuristic algorithm with three-level recursive search is described as follows.

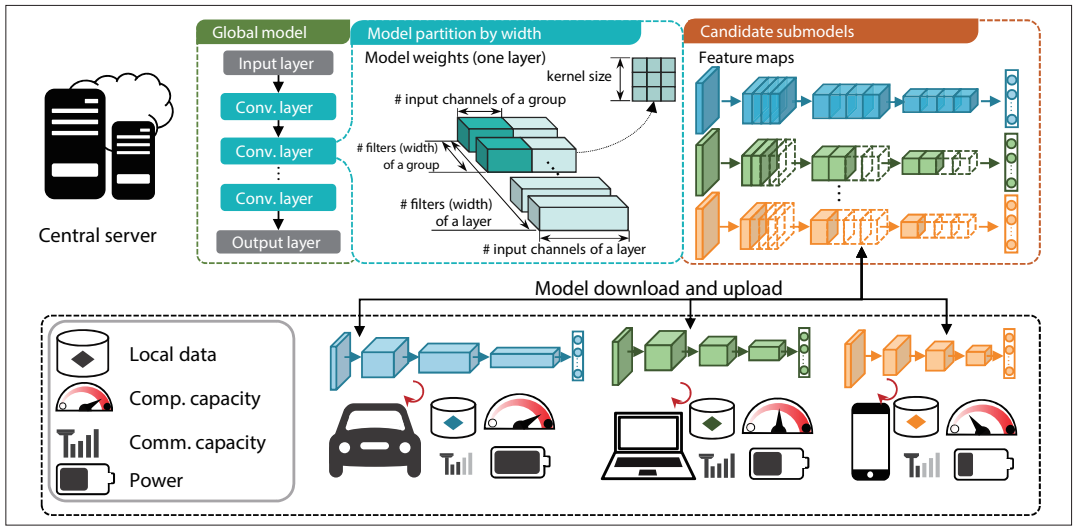

FIGURE 4. Neural-structure-aware resource management.

(S1) Submodel Structure Search: For each model partition, sort its submodels by their sizes from large to small, and the clients by their channel conditions from good to bad. Match the clients with the submodels in order. Call program S_2 to get the suboptimal energy and time costs, and then the suboptimal utility. Compare all suboptimal utilities to get the optimal one.

(S2) Time Duration Search: Given the searching range of time cost, take the binary search technique. In each round of search, call program *S*3 to get the suboptimal computing frequency and transmitting rate. Compute energy cost and compare it to the budget. The binary search goes until the energy cost is close enough to the budget. Return the suboptimal energy and time costs.

(S3) Resource Configuration Search: With the input time cost and model partition, for all clients, compute the suboptimal computing frequency and transmitting rate with the minimal energy cost. Return the suboptimal computing frequency, transmitting rate and energy cost.

Consider J clients with N submodels and T searching time slots in *S*2. For *S*1, the computational complexity is induced by sorting and estimated by $\mathcal{O}(J\log_2 J + N\log_2 N)$. For *S*2, the computational complexity of binary searching within T slots is estimated by $\mathcal{O}(\log_2 T)$. For S3, we can directly calculate the suboptimal computing frequency and transmitting rate with given time cost. Thus, the overall computational complexity is $\mathcal{O}(J\log_2 J + N\log_2 N + J\log_2 T)$.

Experiment Evaluation

In the experiment, we consider the scenario of federated learning with 64 mobile clients for image classification tasks on the MNIST dataset. For the 70,000 samples in the MNIST dataset, 60,000 of them are used for training and the others for testing. The training data are assigned to the clients in two ways. For the IID setting, we shuffle the data and uniformly divide it to all clients, each with 937 samples. For the non-IID setting, we sort the data by labels, split them into 320 shards, with 5 shards for each client. In the experiment, the global CNN model has the structure {16C3-32C3-MP2-32C3-MP2-10C3-GAP-10SM}, with partitioning factors {2, 2, 2, 1} for the convolutional layers, respectively.[1] The hyperparameters are set to batch size 10, local epoch number 1, learning rate 0.05, and decay rate per round 0.996.

We compare our module-based federated learning (MFL) with the traditional federated learning (TFL) in test accuracy, energy consumption, and convergence time. Two power conditions, low power (LP) mode with power budget P_{bud} = 10 W and high power (HP) mode with power budget P_{bud} = 100 W, are considered, and accordingly, the optimal model partitions {16*L*, 16*M*, 16*S*, 16*T*}, {32*L*, 16*M*, 8*S*, 8*T*} are searched for them, respectively. Here, #*L*, *M*, *S*, *T*, denote the number of submodels in "Large," "Medium," "Small," and "Tiny" sizes that have convolutional layers {16C3, 32C3, 32C3, 10C3}, {16C3, 16C3, 32C3, 10C3}, {16C3, 16C3, 16C3, 10C3}, {8C3, 16C3, 16C3, 10C3}, respectively. As we know, using different power modes allows mobile clients to adapt to the available resources and the application features.

Figure 5a shows the experiment results under normal communication condition. It is observed that our MFL outperforms the TFL in both low and high power modes. Particularly in low power mode, compared to TFL, MFL has an energy saving about 43 and 30 percent for IID and non-IID data, respectively, to reach the test accuracy of 96 percent. In high power mode, the advantages of MFL in energy conservation are about 37 and 22 percent for IID and non-IID data, respectively. Meanwhile, as the record on the convergence time, MFL takes much less time than TFL to reach the test accuracy of 96 percent. The convergence time of MFL is less than that of TFL by about 33 and 18 percent for IID and non-IID data in low power mode, and about 33 and 17 percent for IID and non-IID data in high power mode, respectively. In Fig. 5b, the experiment results under fading channel condition are plotted. In this case, it is supposed that the clients may fail to return the gradients with a 20 percent probability in each round. We can observe that MFL still maintains satisfying performance. This indicates that MFL is reliable in dynamic wireless communication environments.

As the record of our experiments, MFL will eventually reach a sufficiently close converged accuracy when compared to TFL. Although this is not plotted in Fig. 5, the numeric results report

[1] Here, we use 16C3 to represent a convolution layer with 16 filters with 3 × 3 kernel size, and similarly, MP2 for 2 × 2 max pooling, GAP for global average pooling, and 10SM for 10 output units activated by softmax. Our experiment results suggest setting the partitioning factor from 1 to 4.

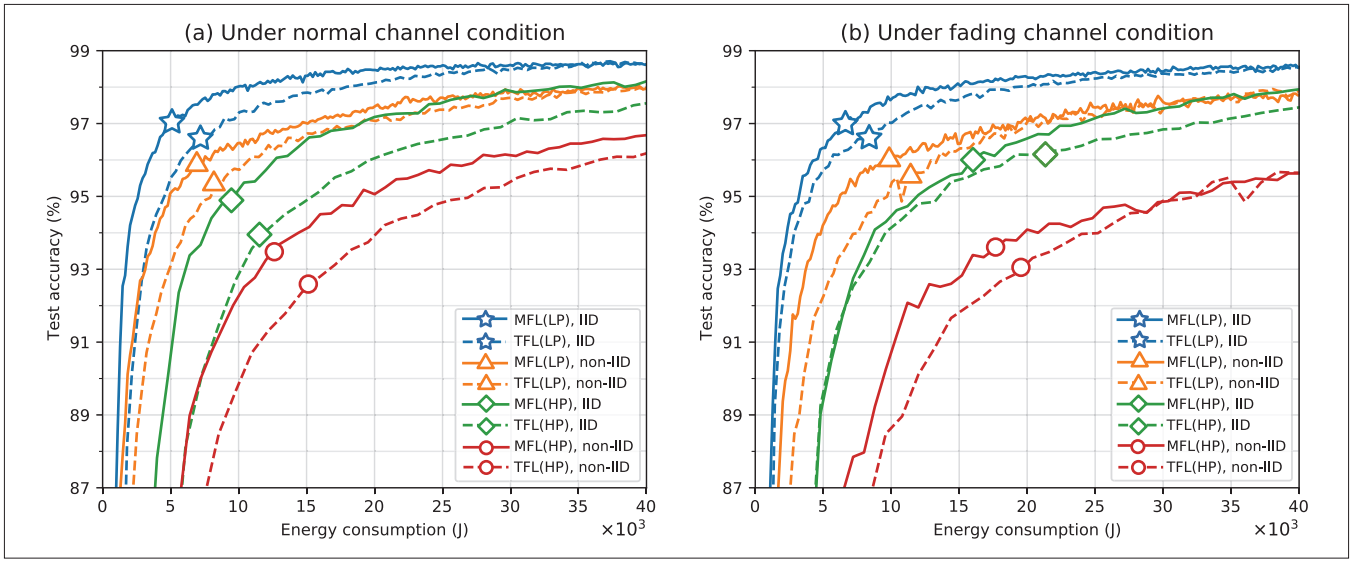

FIGURE 5. Module-based federated learning vs. traditional federated learning.

that in the IID case, MFL achieves almost the same converged accuracy as TFL. The tiny gap is merely 0.1 percent. In the non-IID case, MFL suffers a rather small performance drop of 0.89 percent on the converged accuracy. In addition, MFL could flexibly use the submodel structures under different conditions. It tends to use smaller submodels in a low power condition and larger submodels in a high power condition. It is an elastic and efficient resource management approach.

Conclusion and Future Work

This article investigates the existing works on resource optimization in federated learning and broadly classifies them into black-box and white-box approaches. The module-based federated learning framework and the neural-structure-aware resource management approach are proposed. Experiments show their advantages in improving the resource efficiency.

In principle, our learning framework supports model partition in terms of width, depth, and kernel size. In our future work, it will be interesting to further explore the resource management approach with submodel structures that are all flexible in width, depth, and kernel size. Deep reinforcement learning may be employed to adaptively find out the optimal resource configuration. In addition, the incorporation of neural-structure-aware resource management with efficient black-box approaches may also bring surprising results.

Acknowledgment

The work is supported in part by the program of NSFC under Grant no. 61971148, the Natural Science Foundation of Guangxi Province under Grant no. 2018GXNSFDA281013, and the Foundation for Science and Technology Project of Guilin City under Grant no. 20190214-3.

References

[1] H. B. McMahan *et al.*, "Communication-Efficient Learning of Deep Networks from Decentralized Data," *Proc. AISTATS 2017*, 20–22 Apr. 2017, pp. 1273–82.
[2] M. Chen and Y. Hao, "Task Offloading for Mobile Edge Computing in Software Defined Ultra-Dense Network," *IEEE JSAC*, vol. 36, no. 3, Mar. 2018, pp. 587–97.
[3] H. Cai *et al.*, "Once-for-All: Train One Network and Specialize It for Efficient Deployment," *ICLR*, 2020.
[4] A. Hard *et al.*, "Federated Learning for Mobile Keyboard Prediction," arXiv preprint arXiv:1811.03604, 2018.
[5] M. Bojarski *et al.*, "End to End Learning for Self-Driving Cars," arXiv preprint arXiv:1604.07316, 2016.
[6] M. Chen and Y. Hao, "Label-Less Learning for Emotion Cognition," *IEEE Trans. Neural Networks and Learning Systems*, vol. 99, 13 Aug. 2019, pp. 1–11.
[7] T. Nishio and R. Yonetani, "Client Selection for Federated Learning with Heterogeneous Resources in Mobile Edge," *Proc. IEEE ICC 2019*, 20–24 May 2019, pp. 1–7.
[8] Y. Zhao *et al.*, "Federated Learning with Non-IID Data," arXiv preprint arXiv:1806.00582, 2018.
[9] E. Jeong *et al.*, "Multi-Hop Federated Private Data Augmentation with Sample Compression," *Proc. IJCAI Wksp. FML '19*, 12 Aug. 2019, pp. 1–8.
[10] L. Liu *et al.*, "Client-Edge-Cloud Hierarchical Federated Learning," *Proc. IEEE ICC 2020*, June 2020.
[11] J. Konečnỳ *et al.*, "Federated Learning: Strategies for Improving Communication Efficiency," arXiv preprint arXiv:1610.05492, 2016.
[12] D. Li and J. Wang, "FedMD: Heterogenous Federated Learning via Model Distillation," *Proc. Wksp. FL-NeurIPS '19*, 13 Dec. 2019, pp.1–8.
[13] X. Yao *et al.*, "Towards Faster and Better Federated Learning: A Feature Fusion Approach," *Proc. ICIP 2019*, 22–25 Sept. 2019, pp. 175–79.
[14] Y. Chen *et al.*, "Communication-Efficient Federated Deep Learning with Layerwise Asynchronous Model Update and Temporally Weighted Aggregation," *IEEE Trans. Neural Networks and Learning Systems*, vol. 99, Dec. 2019, pp. 1–10.
[15] P. Baldi and R. Vershynin, "On Neuronal Capacity," *NeurIPS 2018*, 3–8 Dec. 2018, pp. 7740–49.

Biographies

Rong Yu (yurong@ieee.org) [M' 08] received his Ph.D. degree in electronic engineering from Tsinghua University, China, in 2007. After that, he worked as an assistant professor in the School of Electronic and Information Engineering at South China University of Technology. In 2010, he joined the School of Automation at Guangdong University of Technology, where he is currently a full professor. His research interests include wireless networking and mobile computing such as mobile cloud, edge computing, deep learning, connected vehicles, smart grid, and the Internet of Things.

Peichun Li (peichun@mail2.gdut.edu.cn) [S' 19] is an M.S. student in the School of Automation, Guangdong University of Technology. His research interests include edge computing and deep learning, particularly in efficient algorithms for artificial intelligence applications.