electronics

MDPI

Article

Communication Cost Reduction with Partial Structure in Federated Learning

Dongseok Kang and Chang Wook Ahn *

AI Graduate School, Gwangju Institute of Science and Technology, 123 Cheomdangwagi-ro, Buk-gu, Gwangju 61005, Korea; dongseok176@gm.gist.ac.kr
* Correspondence: cwan@gist.ac.kr

Abstract: Federated learning is a distributed learning algorithm designed to train a single server model on a server using different clients and their local data. To improve the performance of the server model, continuous communication with clients is required, and since the number of clients is very large, the algorithm must be designed in consideration of the cost required for communication. In this paper, we propose a method for distributing a model with a structure different from that of the server model, distributing a model suitable for clients with different data sizes, and training a server model using the reconstructed model trained by the client. In this way, the server model deploys only a subset of the sequential model, collects gradient updates, and selectively applies updates to the server model. This method of delivering the server model at a lower cost to clients who only need smaller models can reduce the communication cost of training server models compared to standard methods. An image classification model was designed to verify the effectiveness of the proposed method via three data distribution situations and two datasets, and it was confirmed that training was accomplished only with a cost 0.229 times smaller than the standard method.

Keywords: federated learning; artificial intelligence; neural network

Citation: Kang, D.; Ahn, C.W. Communication Cost Reduction with Partial Structure in Federated Learning. *Electronics* **2021**, *10*, 2081. https://doi.org/10.3390/electronics10172081

Academic Editors: Juan M. Corchado, Stefanos Kollias and Javid Taheri

Received: 26 July 2021
Accepted: 25 August 2021
Published: 27 August 2021

1. Introduction

As we become better at using data to enable the use of more complex artificial neural network models, interest in data utilization has attracted the attention of all of society and is starting to be utilized on all devices [1,2]. As various devices begin to participate in training, research is being conducted on how to use a computing model in an environment where computing devices and data are distributed, as opposed to a centralized system with the typical deep learning models [3]. Recently, there has also been a series of active studies on federated learning, which considers privacy more than existing distributed computing models [4–6]. What is common between federated learning and the existing distributed computing is that there is a central server and that distributed computing devices are connected to it. The difference is whether or not the central server has the information and control authority of the distributed environment. Federated learning, designed with a widely deployed server model and uncontrollable distributed devices in mind, is in the spotlight because it respects increasing data privacy and allows data to be utilized by assuaging users' concerns with using their distributed data. A typical implementation of federated learning is shown in Figure 1.

Training a server model while respecting data privacy presents the following challenges: First, individual computing devices have different characteristics. They have a broad impact on learning, including when a device will be online, what computational power is available, whether the device will launch hostile attacks on the server, and whether communication with the server can be reliably delivered [7,8]. Therefore, it is necessary to design a robust algorithm to train the server model even when clients have different characteristics.

Second, the data are distributed on local devices, and any attempt by the server to move these data would violate the users' data privacy, while the data distribution of each client remains unknown. The ideal situation is when the distribution of data is independent across all devices and follows the same probability distribution, but the distribution of actual data varies greatly by each user. When training a server model that needs to be optimized for the entire dataset, the fact that the exact data of the client are not known reduces the performance of the server model. In federated learning, training is especially more difficult because it assumes a very large number of clients and different data distributions.

Third, communication with the server is not free. In general, as more devices and larger amounts of information are synchronized more frequently, the training difficulty of the server model decreases, but unrestricted communication with user devices can be dangerous [7,9,10]. For example, if it takes 100 rounds of communication to deploy and train a 100 MB server model, with an assumption that 100 communications are made from 1 million devices, the communication charges incurred from the communication traffic increase extraordinarily.

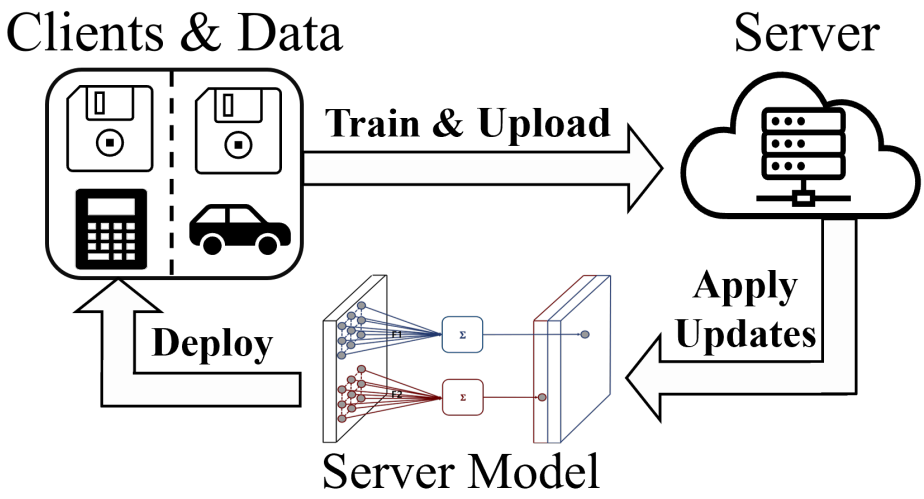

Figure 1. General procedure of federated learning.

If the server model converges quickly through the improvement of the learning algorithm, training can be completed with fewer communications. We propose a method to achieve fast convergence of the server model while communicating in a smaller size, which is a method to train a server model on the server by deploying a derived variant of the server model to the client in a federated learning environment. In the proposed method, when the server model is distributed, some layers of the server model are probabilistically deleted, and the derived model, which has a smaller size and a different structure, is distributed to the client, and the client trains it. Clients participating in every communication round train models with arbitrarily derived structures, and the results of this training are selectively applied to the common part between the derived model and the server model. This method of distributing the derived model in various sizes considering devices with various computing powers is advantageous in terms of communication cost compared to fully deploying the server model.

The advantages of the proposed federated learning algorithm are summarized as follows.

- By distributing the server model to the client as a smaller derivative model, the communication cost is reduced, simultaneously providing superior training efficiency over the standard method.
- Models distributed to clients do not require any additional computational work such as decompression and can be used for local data training as they are.

- Because the client does not know the complete structure of the server model, it provides protection against potential attacks such as malicious gradient uploads.

The remainder of the paper starts with a presentation of the background, including related works. Then, related works will be discussed. This is followed by a presentation of the proposed method. Based on this, we present twelve federated learning experiments. Finally, a conclusion is drawn, and an outlook on future work is described.

2. Background

2.1. Federated Learning

Federated learning aims to learn statistical models in a distributed environment. There are differences from existing distributed environment computing [11,12], edge computing [13,14], and fog computing [15,16]. Previous studies have focused on directly utilizing the computing resources of distributed devices such as Internet of Things devices. On the other hand, in federated learning, the focus is on the privacy of data resources, the availability of distributed devices, and the cost of communicating with a central server. These differences are as follows. 1. Data reside in distributed devices, and these data must not be interrogated or inferred. 2. Devices participating in training have different characteristics, including availability of devices, data bias, computing performance, and communication delay. In addition, these characteristics are neither arbitrarily adjusted nor predictable by the server. 3. For training the server model, devices communicate with the server regularly, and the required communication cost should be considered. These characteristics make it difficult to apply the previously researched distributed computing techniques, and when training an arbitrary model, the training efficiency is degraded compared to the general environment.

Assuming that N device owners $\mathcal{O}_1, ..., \mathcal{O}_N$ participating in federated learning each have their own data $d_1, ..., d_N$, it would be reasonable to see that each set of data has different distributions $\mathcal{D}_1, ..., \mathcal{D}_N$. In the traditional method, all the data sets $d = d_1 \cup ... \cup d_N$ that are collected as a single object are trained, but in federated learning, transferring raw data is prohibited. For example, let $\mathcal{Y} = \{0, 1\}$ be the label space in the data d. When all the data are collected in the server, each data point x in d has a probability such that $P(x_i \in d \text{ has label } y \in \mathcal{Y}) = P(x_j \in d \text{ has label } y \in \mathcal{Y})$, while $P(x_i \in d_k \text{ has the label } y \in \mathcal{Y}) \neq P(x_j \in d_k \text{ has label } y \in \mathcal{Y})$ in distributed devices. Each device trains its own model $\mathcal{M}_1, ..., \mathcal{M}_N$ only with the data it has. To train a neural network, we aim to minimize its loss function, and in doing so, a cross entropy function can be used as a loss function such as $loss(x, t) = -\sum_{i=1}^{n} t_i \log x_i$, for n classes, where t_i is the truth label, and x_i is the probability for the i^{th} class from the model output. When the server has the entire dataset d and has to update θ, the parameter of the model, we use the gradients for the entire training dataset: $\theta = \theta - \eta \cdot \nabla_\theta loss(\mathcal{M}(d_\mathcal{X}), d_\mathcal{Y})$, where η is the learning rate. However, to train the server model $\mathcal{M}$ in federated learning, it collects the gradients of the loss function ℓ from the clients, instead of the clients' data:

$$\min_{\theta} \ell(\theta) = \sum_{i=1}^{N} \frac{n_i}{n} L_i(\theta) \quad \text{where } L_i = \sum_{j \in d_i} \ell_j(\theta) \tag{1}$$

The number of data points in each device is n_i, and $\ell_j(\theta)$ represents loss functions of a model $\mathcal{M}_j$ with d_j. In a traditional environment, it is possible to optimize all the data at the same time by collecting each datum in the center and performing training, but in federated learning, only the results of learning are obtained without collecting distributed data. In this method of training the server model by collecting the results of several trainings, the difficulty of training the server model as above increases as the distribution of data varies as higher in general. In order to increase the training performance in consideration of the distribution of data, several studies have been conducted on how to collect and apply training results in each client [17].

2.2. Related Works

Deep learning technology is being used to utilize big data and is a field that is being actively researched [18,19]. Deep learning can extract high-level features from many low-level samples and learn hierarchical features of the data. It has been successfully used in computer vision, speech recognition, and natural language processing [20–23]. As this data-driven model has been successfully utilized, the importance of data collection has increased, and regulations have also increased [24]. In response to this trend, there was a need for a method to effectively train a deep learning model, and federated learning technology is being actively studied. In federated learning, both the amount of computation consumed by the client and the cost of communicating with the server must be considered. Unlike centralized systems in data centers, clients are unreliable with low bandwidth connections, so it is important to minimize communication costs with servers. Methods used in existing deep learning studies can be used to reduce communication costs. Quantization of the model, which converts a neural network with floating-point numbers into a neural network with lower bitwidth numbers, can effectively compress the model size with minimal performance degradation. This can be easily applied to federated learning [25,26]. There are many other methods, but not all of them are intuitively applicable to federated learning. Such a problem of finding a trade-off between communication cost and model performance is still an open question, and various attempts are being made to solve it [27,28].

There are three ways to reduce the communication cost: to compress the information sent from the client to the server; to let the server compress and distribute the model; or to achieve fewer communication rounds by improving the training efficiency. Several studies have been conducted on client-to-server communication compression, and its effectiveness has been shown [29–31]. How much communication cost can be reduced depends on the network the client is using, and the efficiency of these methods may vary. Although there is no study evaluating the effect of using the three methods at the same time, these methods alone can effectively reduce the communication cost, and the effect depends on the system configuration. For example, since home networks typically have larger download bandwidths than uploads, compressing client-to-server communications can be more effective than other methods. When sending updates to the model, aggressive lossy compression is sometimes used to approximate updates to actively reduce the communication size. Since federated learning assumes a very large number of clients, performance degradation can be minimized by averaging.

It is a general training method to distribute the models from the server to the client as is, but training them is possible even by compressing and distributing them. There is a study analyzing the effect of compressing and distributing a model in the server on the convergence of training [32]. Caldas proposes to distribute the compressed model on the server [33]. According to his study, clients unpack the distributed model, train it, and then compress it again and upload it to the server. Although the structure of the model trained on the client is the same as that of the server, it has been shown that the communication cost can be effectively reduced through weight subsampling and quantization of the model. Hamer, on the other hand, proposes a method to train an ensemble model in federated learning and presents a method to find the optimal mixture weight by distributing a pretrained partial model to a client [34]. In this method, the client and server exchange only a part of the model constituting the ensemble, thereby lowering the communication cost.

A computationally efficient training method can reduce the total number of training steps required for convergence, which in turn reduces the communication cost. Some training techniques used in deep learning can be intuitively applied to federated learning, and sometimes not. An example of an overall training method improvement designed for federated learning is FedAverage proposed by McMahan [35]. This study is one of the first papers that introduced the concept of federated learning, and it shows that as the local computation of the client increases, the training of the server model can also be accelerated.

3. Method

In general, when training a deep neural network model, the structure of the model is set to fixed because if the structure changes, performance convergence can be difficult. However, regarding the prerequisites of federated learning, it is still possible to consider model transformations as part of the overall learning process. (1) Clients have to download a new server model for every communication. (2) Each client has its own data distribution and computational performance, so a common server model is not always optimal. (3) Considering the fact that the total number of clients is very large, we propose a new federated learning method.

The overall structure follows the federated learning method proposed by McMahan [35], where the server creates a server model and distributes it to clients. Clients train the distributed model, using all the data they have. The model that has undergone a certain amount of training in the client is uploaded back to the server, and the server updates the server model by collecting the trained models. The parameter update of the server model is adjusted so that all clients contribute at the same rate in one communication round.

Before distributing the model to the client, the server creates a model with a smaller structure by probabilistically removing the intermediate layers so that the distributed model has a size smaller than or equal to the size of the server model. The method of generating the model with the new structure is shown in lines 9–14 of Algorithm 1 and in Figure 2. In the server model, two parts coexist with a structure that does not always change and a part whose structure changes probabilistically. In the part where the structure is stochastically changed, the intermediate layers are deleted with a probability of p, and the other part copies the server model as it is. We design the server model so that intermediate layers of the server model have the same input and output dimensions, so that there is no problem even if the intermediate structure is changed. As the number of skipped parts increases, the deformed model has a smaller size, and the communication cost is reduced thanks to its reduced structure.

The proposed algorithm considers that it can be an advantage for the client to have a smaller derivative model than the server model. A derived model can be created by omitting some structures from the entire structure of the server model. When the size of the model is reduced, it is possible to train local data with less communication and computational costs. Aside from the fact that the size of the model benefits communication costs, the size of the model also affects how well the client generalizes the local data. In a federated learning environment, the more the distribution of local data differs from the distribution of the entire dataset, the smaller the number of biased data may be. In this case, if the client has a smaller model, it gains an advantage in generalizing the local data. The results trained from the client can be selectively applied to the server model as follows.

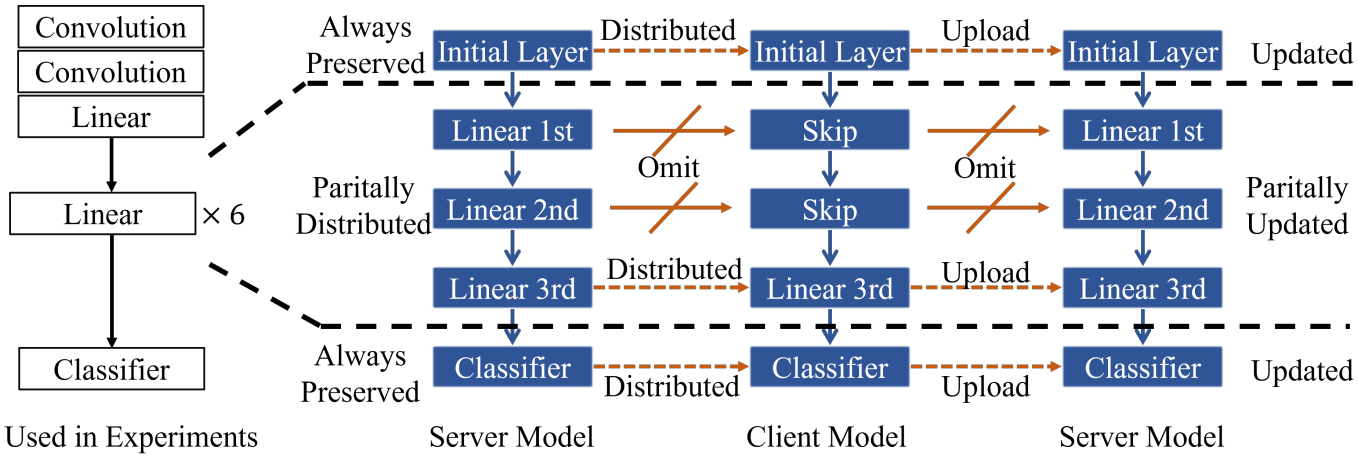

Figure 2. This figure shows the configuration of the server model used in the experiment and the process of generating an arbitrary derivative model to be distributed to the client.

Algorithm 1: FederatedPartial. T is the number of communication rounds, B is the local minibatch size, E is the numberof local epochs, η is the learning rate, i and j are indices of layers to be replaced, and p is the probability of being distributed.

Initialize w_1
for *round* $t \leftarrow 1$ *to* T **do**
  $S_t \leftarrow$ random n clients
  foreach *client* $k \in S_t$ **in parallel do**
    $w_{new} \leftarrow$ `BuildPartialModel` (w_t, p, i, j)
    $k.trained \leftarrow$ `ClientUpdate` (k, w_{new}, B, E, η)
  $w_{t+1} \leftarrow$ `ServerUpdate` (S_t, w_t, n)

Function `BuildPartialModel`(w, p, i, j)**:**
  $w_{new} \leftarrow$ (copy w)
  for $k \leftarrow i$ *to* j **do**
    if $p < U[0, 1]$ **then**
      delete kth layer in w_{new}
  return w_{new}

Function `ClientUpdate`(k, w, B, E, η)**:**
  `// Run on client`
  B $\leftarrow$ (split data of k into batches of size B)
  for *epoch* $i \leftarrow 1$ *to* E **do**
    foreach *batch* $b \in$ B **do**
      $w \leftarrow w - \eta \nabla_w \mathcal{L}(w; b)$
  return w

Function `ServerUpdate`(S, w, n)**:**
  foreach *client* $k \in S$ **do**
    foreach *layer* $w_i \in k.trained$ **do**
      if w_i is in w **then**
        $w \leftarrow w + \frac{1}{n} w_i$
  return w

The client trains the distributed model with local data and sends the entire model to the server. The server needs to selectively apply updates because the models it receives have a different structure from the server model. For this, the server selects only the common parts applicable to the server model among the whole structure of the received model and applies the update. When applying updates, the updates are adjusted and applied so that each client contributes the same proportion.

4. Experiment

In order to deploy and train a model with a structure different from the server model and to find out whether there is a benefit in the cost of convergence when actual convergence is achieved, an image classification model has been designed upon which federated learning is performed and analyzed. In the experiment, it is assumed that 100 clients participate, and only 20% of clients participate in training for every communication round. The experimental parameters used in the experiment are shown in Table 1.

Table 1. This table shows the federated learning results of the image classification model using the standard method and the proposed algorithm, FederatedPartial. The ratios represent the cost of FederatedPartial compared to the standard method.

	Cost Hit 0.8 in Rounds	Cost Hit 0.8 in Traffic	Method	Rounds Ratio	Traffic Ratio	Data Distribution	Dataset	Variables	Optimizer
Exp1	960	2.86×10^9	Standard	0.892	0.569	(10, 90)	Fashion MNIST	T: 2000 B: 50 E: 5 η: 0.001 i: 3 j: 9 p: $\frac{2}{3}$	SGD with 0.9 Momentum
Exp2	857	1.63×10^9	Partial						
Exp3	1032	3.08×10^9	Standard	0.757	0.500	(90, 10)			
Exp4	781	1.54×10^9	Partial						
Exp5	1007	3.00×10^9	Standard	0.781	0.517	(10, 10, 80)			
Exp6	787	1.55×10^9	Partial						
Exp7	323	9.57×10^8	Standard	0.346	0.229	(10, 90)	MNIST		
Exp8	111	2.19×10^8	Partial						
Exp9	316	9.42×10^8	Standard	0.413	0.273	(90, 10)			
Exp10	131	2.57×10^8	Partial						
Exp11	279	8.31×10^8	Standard	0.431	0.284	(10, 10, 80)			
Exp12	120	2.36×10^8	Partial						

There are two datasets to be used for the image classification model, MNIST and Fashion-MNIST, and the distribution of the dataset is artificially adjusted to assume a federated learning environment [36,37]. The MNIST dataset has numbers from 0 to 9 as labels and contains grayscale images of handwritten digits. Fashion-MNIST has 10 types of fashion items as labels and consists of grayscale pictures of them. Both datasets consist of 60,000 training data points and 10,000 test data points with an image size of 28×28. Examples of images in the datasets are shown in Figure 3. In order to ensure that the distribution of the local data of each client is clearly different from the entire dataset, the data are distributed so that each client has a maximum of one to three labels. There are three types of data distributions to 100 clients: (1) 10 clients have 1 label, and the remaining 90 clients have 2 labels; (2) 90 clients have 1 label, and the remaining 10 clients have 2 labels; (3) 10 clients have 1 label, another 10 clients have 2 labels, and the remaining 80 clients have 3 labels. The three situations are depicted in Figure 4.

Figure 3. An example of grayscale handwritten digits and 10 fashion items from the dataset MNIST and Fashion-MNIST.

The model used in the experiment is an artificial neural network model composed of a combination of a convolutional layer and a linear layer. Each layer goes through the ELU activation function, and some layers are replaced with Skip layers at the deployment time. This model has 14,911 parameters, with downloads and uploads made by 100 clients per one communication round, resulting in a total communication cost of 2,982,200. When a new model is created to distribute the model to the client, the middle part of the server model is newly designed and deployed each time. An example of the deployment process for this model is shown in Figure 2.

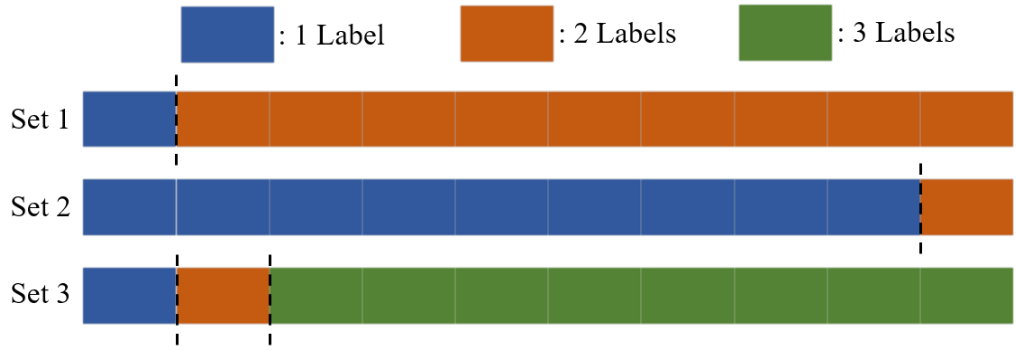

Figure 4. A graphical set of bars to show how to partition the entire dataset so that each client has an extreme data distribution.

Using the federated averaging algorithm proposed by McMahan [35] as a standard, twelve experiments were designed by combining the standard method, the proposed method, and three distributions of two datasets. Unlike the proposed method, the standard method does not undergo transformation in the deployment and update of the server model. Both methods were tested with the same parameters such as maximum communication round, running rate, and batch size.

In order to evaluate the effectiveness of the proposed method in federated learning, it is necessary to measure how much communication cost is consumed until the server model has a certain accuracy. At this time, it is necessary to determine how to measure the cost by considering the characteristics of federated learning: that the learning curve is not smooth and fluctuates greatly. There is a gap between when the target performance is first achieved and when it stably converges beyond the target performance. In our experiment, we recorded the communication cost when the server model satisfied the target performance more than three times within five communication times. The target performance measure of the server model is set as the inference accuracy of the server model for the entire dataset. The target performance was set to an accuracy of 0.8, each experiment was repeated four times, and the cost of the four trials was averaged.

5. Discussion

Twelve experiments were conducted to compare the standard method and the proposed method, and the results are recorded in Table 1. In the three data distributions of Fashion-MNIST, FederatedPartial reached the target performance with a cost 0.6–0.7 times smaller compared to the standard method. Figure 5 shows the learning curves of Exp3 and Exp4.

In the most extreme case of data distribution difference (90, 10), FederatedPartial required 0.612 times less cost than the standard method, but it required 0.672 times less cost in the less localized data (10, 90). This is because FederatedPartial has a greater effect when the data distribution difference between clients is large, which can be interpreted as the relationship between the data and model size of each client. When one client has a large difference from the distribution of the entire dataset, it means that the data have very few variants, with which it is easier to train via a smaller model. FederatedPartial distributes server models of various sizes to clients and shows that the training efficiency of the central server model can also be increased by distributing a server model suitable for clients that need a smaller model to increase the client's learning efficiency.

In federated learning, it is generally known that the greater the difference in the data distribution of each client, the more difficult it is to train. In Exp5, compared to Exp1, a little more cost was required to reach the target performance. Exp5 is designed so that each client has three labels of data, which is closer to the original dataset distribution than the data distribution in Exp1, but Exp5 recorded a higher cost than Experiment 1. This shows that the difference between the data distribution of each client and the distribution of the original dataset affects the training, and how the client group is divided also affects the

training of the server model. This difference also appears in the comparison between Exp7 and Exp11 using the MNIST dataset. It can be seen that the server model suffers slightly from training when there is a diverse group of clients.

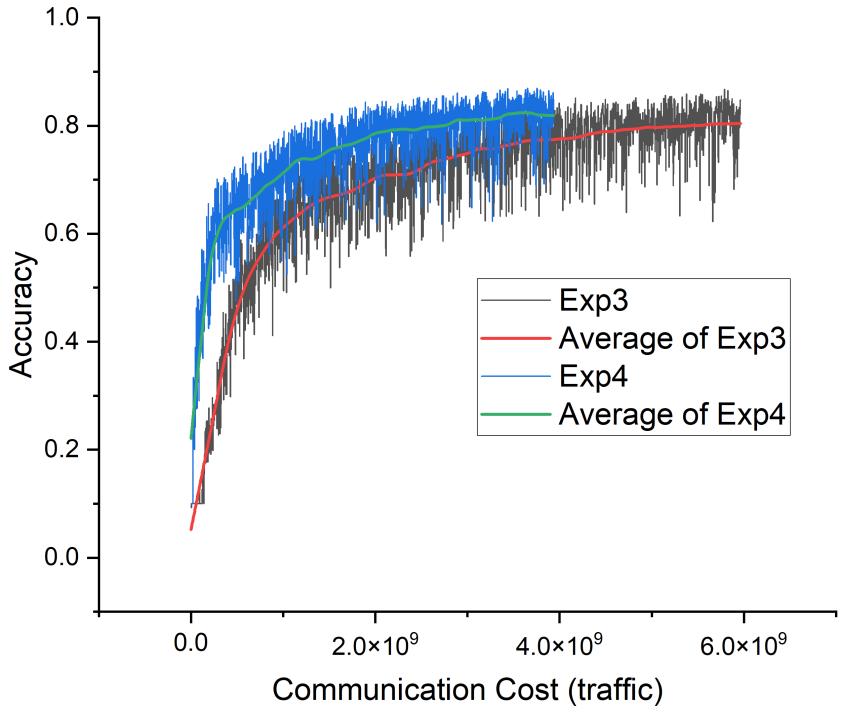

Figure 5. Learning curves of Exp3 and Exp4.

In an experiment using the MNIST dataset, FederatedPartial was able to train the server model using a traffic cost 0.22 to 0.28 times smaller than the standard method to reach the target performance. All six experiments showed the same tendency as the experiment conducted on Fashion-MNIST. The structure of the FedParitial algorithm and the server model used to train MNIST and Fashion-MNIST is the same, but FederatedPartial showed a higher effect in MNIST. This phenomenon seems to have been possible because FederatedPartial appropriately provides a model suitable for the size of the client data when the server model has a large and complex model, but the data the client has only require a simple model. Compared to Fashion-MNIST, MNIST can be sufficiently trained with a simpler model, and this difference can be seen as the reason for the difference in the efficiency of the algorithms proposed in the two datasets.

Both the standard method and the proposed method showed a reduction in communication cost but showed different ratios of efficiency in both the number of communications and communication traffic. In Exp1 and 2, for the server model to reach the target performance, FederatedPartial required a cost of traffic 0.569 times smaller and a cost of communication rounds 0.892 times smaller compared to the standard method. In all comparative experiments, the reduction in traffic cost was greater, which means that the total cost of traffic consumed by the standard method is not essential. In federated learning, what is more important in learning the server model is not the size of the communication but the importance of the information contained in the communication and frequent synchronization between the client and the server model. When applying the federated learning technology, whether the traffic cost or the number of communications should take precedence with regards to the system configuration and the characteristics of the client still remains questionable, so further research on this difference in efficiency is needed.

6. Conclusions

In this study, we showed that the training of a server model is possible even when deploying it in various structures and sizes. With the proposed FederatedPartial method, it was possible to achieve the target performance of the server model at a lower cost compared to the standard method in terms of the number of communications and communications traffic. In our federated learning experiments, different efficiencies were shown depending on the difference between the distribution of data from the client and the distribution of the entire dataset, but when the server model was large enough to learn the entire dataset, it showed a noticeable communication cost reduction efficiency. It is worth further researching the relationship between the reduction of communication cost and the data distribution of clients, as well as the multi-objective optimization of communication traffic and communication frequency.

Author Contributions: Conceptualization, methodology, software, validation, investigation, data curation, writing—original draft preparation, visualization, D.K.; writing—review and editing, formal analysis, supervision, resources, project administration, funding acquisition, C.W.A. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by GIST Research Institute(GRI) grant funded by the GIST in 2021 and the National Research Foundation of Korea(NRF) grant funded by the Korea government. (MSIT) (2021R1A2C3013687).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Ghosh, A.; Chakraborty, D.; Law, A. Artificial intelligence in Internet of things. *CAAI Trans. Intell. Technol.* **2018**, *3*, 208–218. [CrossRef]
2. Misra, N.; Dixit, Y.; Al-Mallahi, A.; Bhullar, M.S.; Upadhyay, R.; Martynenko, A. IoT, big data and artificial intelligence in agriculture and food industry. *IEEE Internet Things J.* **2020**. [CrossRef]
3. Hard, A.; Rao, K.; Mathews, R.; Ramaswamy, S.; Beaufays, F.; Augenstein, S.; Eichner, H.; Kiddon, C.; Ramage, D. Federated learning for mobile keyboard prediction. *arXiv* **2018**, arXiv:1811.03604.
4. Li, L.; Fan, Y.; Tse, M.; Lin, K.Y. A review of applications in federated learning. *Comput. Ind. Eng.* **2020**, *149*, 106854. [CrossRef]
5. Yang, Q.; Liu, Y.; Cheng, Y.; Kang, Y.; Chen, T.; Yu, H. Federated learning. *Synth. Lect. Artif. Intell. Mach. Learn.* **2019**, *13*, 1–207. [CrossRef]
6. Li, T.; Sahu, A.K.; Talwalkar, A.; Smith, V. Federated learning: Challenges, methods, and future directions. *IEEE Signal Process. Mag.* **2020**, *37*, 50–60. [CrossRef]
7. Bagdasaryan, E.; Veit, A.; Hua, Y.; Estrin, D.; Shmatikov, V. How to backdoor federated learning. In Proceedings of the International Conference on Artificial Intelligence and Statistics, Palermo, Italy, 3–5 June 2020; pp. 2938–2948.
8. Nishio, T.; Yonetani, R. Client selection for federated learning with heterogeneous resources in mobile edge. In Proceedings of the ICC 2019—2019 IEEE International Conference on Communications (ICC), Shanghai, China, 20–24 May 2019; pp. 1–7.
9. Wang, Z.; Song, M.; Zhang, Z.; Song, Y.; Wang, Q.; Qi, H. Beyond inferring class representatives: User-level privacy leakage from federated learning. In Proceedings of the IEEE INFOCOM 2019—IEEE Conference on Computer Communications, Paris, France, 29 April–2 May 2019; pp. 2512–2520.
10. Sofia, D.; Lotrecchiano, N.; Trucillo, P.; Giuliano, A.; Terrone, L. Novel Air Pollution Measurement System Based on Ethereum Blockchain. *J. Sens. Actuator Netw.* **2020**, *9*, 49. [CrossRef]
11. Mcdonald, R.; Mohri, M.; Silberman, N.; Walker, D.; Mann, G. Efficient Large-Scale Distributed Training of Conditional Maximum Entropy Models. Available online: https://storage.googleapis.com/pub-tools-public-publication-data/pdf/35648.pdf (accessed on 27 August 2021).
12. Gupta, O.; Raskar, R. Distributed learning of deep neural network over multiple agents. *J. Netw. Comput. Appl.* **2018**, *116*, 1–8. [CrossRef]
13. Shi, W.; Cao, J.; Zhang, Q.; Li, Y.; Xu, L. Edge computing: Vision and challenges. *IEEE Internet Things J.* **2016**, *3*, 637–646. [CrossRef]
14. Satyanarayanan, M. The emergence of edge computing. *Computer* **2017**, *50*, 30–39. [CrossRef]
15. Byers, C.C. Architectural imperatives for fog computing: Use cases, requirements, and architectural techniques for fog-enabled iot networks. *IEEE Commun. Mag.* **2017**, *55*, 14–20. [CrossRef]
16. OpenFog Consortium Architecture Working Group. OpenFog Architecture Overview. Available online: https://www.iiconsortium.org/pdf/OpenFog_Reference_Architecture_2_09_17.pdf (accessed on 27 August 2021).
17. Kairouz, P.; McMahan, H.B.; Avent, B.; Bellet, A.; Bennis, M.; Bhagoji, A.N.; Bonawitz, K.; Charles, Z.; Cormode, G.; Cummings, R.; et al. Advances and open problems in federated learning. *arXiv* **2019**, arXiv:1912.04977.

18. Chen, X.W.; Lin, X. Big data deep learning: Challenges and perspectives. *IEEE Access* **2014**, *2*, 514–525. [CrossRef]
19. Schmidhuber, J. Deep learning in neural networks: An overview. *Neural Netw.* **2015**, *61*, 85–117. [CrossRef]
20. Islam, S.S.; Rahman, S.; Rahman, M.M.; Dey, E.K.; Shoyaib, M. Application of deep learning to computer vision: A comprehensive study. In Proceedings of the 2016 5th International Conference on Informatics, Electronics and Vision (ICIEV), Dhaka, Bangladesh, 13–14 May 2016; pp. 592–597.
21. Chen, D.; Mak, B.K.W. Multitask learning of deep neural networks for low-resource speech recognition. *IEEE/ACM Trans. Audio Speech Lang. Process.* **2015**, *23*, 1172–1183.
22. Majumder, N.; Poria, S.; Gelbukh, A.; Cambria, E. Deep learning-based document modeling for personality detection from text. *IEEE Intell. Syst.* **2017**, *32*, 74–79. [CrossRef]
23. Jiang, Z.; Li, L.; Huang, D.; Jin, L. Training word embeddings for deep learning in biomedical text mining tasks. In Proceedings of the 2015 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), Washington, DC, USA, 9–12 November 2015; pp. 625–628.
24. Voigt, P.; Von dem Bussche, A. The EU General Data Protection Regulation (GDPR). In *A Practical Guide*, 1st ed.; Springer International Publishing: Cham, Switzerland, 2017; Volume 10, p. 3152676.
25. Basu, D.; Data, D.; Karakus, C.; Diggavi, S. Qsparse-local-SGD: Distributed SGD with quantization, sparsification, and local computations. *arXiv* **2019**, arXiv:1906.02367.
26. Konečnỳ, J.; McMahan, H.B.; Yu, F.X.; Richtárik, P.; Suresh, A.T.; Bacon, D. Federated learning: Strategies for improving communication efficiency. *arXiv* **2016**, arXiv:1610.05492.
27. Barnes, L.P.; Inan, H.A.; Isik, B.; Özgür, A. rTop-k: A statistical estimation approach to distributed SGD. *IEEE J. Sel. Areas Inf. Theory* **2020**, *1*, 897–907. [CrossRef]
28. Zhang, Y.; Duchi, J.C.; Jordan, M.I.; Wainwright, M.J. Information-Theoretic Lower Bounds for Distributed Statistical Estimation with Communication Constraints. Available online: https://people.eecs.berkeley.edu/~jordan/papers/zhang-etal-nips14.pdf (accessed on 27 August 2021).
29. Suresh, A.T.; Felix, X.Y.; Kumar, S.; McMahan, H.B. Distributed mean estimation with limited communication. In Proceedings of the International Conference on Machine Learning, Sydney, Australia, 6–11 August 2017; pp. 3329–3337.
30. Konečnỳ, J.; Richtárik, P. Randomized distributed mean estimation: Accuracy vs. communication. *Front. Appl. Math. Stat.* **2018**, *4*, 62. [CrossRef]
31. Alistarh, D.; Grubic, D.; Li, J.; Tomioka, R.; Vojnovic, M. QSGD: Communication-efficient SGD via gradient quantization and encoding. *Adv. Neural Inf. Process. Syst.* **2017**, *30*, 1709–1720.
32. Chraibi, S.; Khaled, A.; Kovalev, D.; Richtárik, P.; Salim, A.; Takáč, M. Distributed fixed point methods with compressed iterates. *arXiv* **2019**, arXiv:1912.09925.
33. Caldas, S.; Konečny, J.; McMahan, H.B.; Talwalkar, A. Expanding the reach of federated learning by reducing client resource requirements. *arXiv* **2018**, arXiv:1812.07210.
34. Hamer, J.; Mohri, M.; Suresh, A.T. FedBoost: A Communication-Efficient Algorithm for Federated Learning. In Proceedings of the 37th International Conference on Machine Learning, Virtual Event, 13–18 July 2020; pp. 3973–3983.
35. McMahan, B.; Moore, E.; Ramage, D.; Hampson, S.; y Arcas, B.A. Communication-efficient learning of deep networks from decentralized data. In Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, Fort Lauderdale, FL, USA, 20–22 April 2017; pp. 1273–1282.
36. LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]
37. Xiao, H.; Rasul, K.; Vollgraf, R. Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms. *arXiv* **2017**, arXiv:1708.07747.