Distributed Machine Learning on Mobile Devices: A Survey

RENJIE GU, Shanghai Jiao Tong University, China
SHUO YANG, Shanghai Jiao Tong University, China
FAN WU*, Shanghai Jiao Tong University, China

In recent years, mobile devices have gained increasingly development with stronger computation capability and larger storage. Some of the computation-intensive machine learning and deep learning tasks can now be run on mobile devices. To take advantage of the resources available on mobile devices and preserve users' privacy, the idea of mobile distributed machine learning is proposed. It uses local hardware resources and local data to solve machine learning sub-problems on mobile devices, and only uploads computation results instead of original data to contribute to the optimization of the global model. This architecture can not only relieve computation and storage burden on servers, but also protect the users' sensitive information. Another benefit is the bandwidth reduction, as various kinds of local data can now participate in the training process without being uploaded to the server. In this paper, we provide a comprehensive survey on recent studies of mobile distributed machine learning. We survey a number of widely-used mobile distributed machine learning methods. We also present an in-depth discussion on the challenges and future directions in this area. We believe that this survey can demonstrate a clear overview of mobile distributed machine learning and provide guidelines on applying mobile distributed machine learning to real applications.

CCS Concepts: • **Theory of computation → Multi-agent learning**; • **Computing methodologies → Machine learning**; **Distributed algorithms**.

Additional Key Words and Phrases: machine learning, distributed machine learning, mobile learning, federated learning

ACM Reference Format:
Renjie Gu, Shuo Yang, and Fan Wu. 2019. Distributed Machine Learning on Mobile Devices: A Survey. 1, 1 (September 2019), 28 pages. https://doi.org/00.0000/0000000.0000000

1 INTRODUCTION

Nowadays, machine learning, as well as deep learning, has become a hot topic and attracts great attention from both academia and industry. The core idea of machine learning is to use large amounts of data to fit a model that can generalize well to unseen inputs. As the increase of data size and model complexity, it becomes harder for a single server to accomplish a machine learning task. To address the problem, distributed machine learning is developed. A typical distributed machine learning task is done through the cooperation of multiple servers. They communicate with each other to transfer essential information during training via shared data buses or a fast

*Corresponding author, To whom correspondence should be addressed.

Authors' addresses: Renjie Gu, grj165@sjtu.edu.cn, Shanghai Jiao Tong University, 800 Dongchuan Road, Shanghai, China, 200240; Shuo Yang, yangshuo9999@gmail.com, Shanghai Jiao Tong University, 800 Dongchuan Road, Shanghai, China, 200240; Fan Wu, Shanghai Jiao Tong University, 800 Dongchuan Road, Shanghai, China, 200240, fwu@cs.sjtu.edu.cn.

XXXX-XXXX/2019/9-ART $15.00
https://doi.org/00.0000/0000000.0000000

local area network. Although this scheme is feasible and efficient, it costs much funds to build such high-performance server clusters in the cloud.

According to [36], by 2018, there are around 3 billion active smartphone users all over the world. So despite the use of traditional computers, the idea of doing machine learning on mobile devices has also emerged. For example, technologies such as face recognition, AI photographing, and voice assistant are all based on machine learning and popular among mobile phones. To support these applications, a well-performed machine learning model is first trained on servers using large amounts of data, and then sent to mobile devices to do inference and make predictions locally. This scheme brings all the burdens to the central servers, while wasting the computation and storage resources of mobile devices. Another fact is that the processors carried on mobile devices have already been powerful enough to support various kinds of tasks, including simple machine learning tools tasks which can actually be expressed as a series of matrix multiplication operations. Under this circumstance, making use of so much computing power and so much user data through machine learning seems to be an exciting and urgent task. What's more, frameworks for deep learning on mobile devices such as TensorFlow Lite has already been popular. The above evidences have shown that it is entirely possible to deploy a distributed machine learning task on mobile devices. Comparing with traditional centralized machine learning or server-based distributed machine learning, one main advantage of mobile distributed machine learning is that it can reduce the burden on servers by making use of idle resources on mobile devices. For those data whose amount is very large but only have low information density, they are not suitable for being uploaded to servers. Now mobile distributed machine learning enables them to contribute to the machine learning model as they can now be processed locally to extract valuable and high-density information.

Another important topic is data privacy. In machine learning area, the training data stored on servers are mainly obtained from users' uploads, which may include some of the users' sensitive data, such as the input words. The leak of sensitivity information can bring security risks to users. However, sensitive data may contain some useful information that can further improve the products. In machine learning area, most existing works that are based on centralized machine learning or traditional distributed machine learning are still using centralized data sets that are uploaded anonymously through the network and stored on a server or a data center. This causes the data being exposed to high risk of cyber attacks and insider attacks[51–53]. Some advanced attacks[48, 65] which have occurred recently are able to infer information from the final trained model and thus require us to perform a more complicated training process to get rid of them.

Researchers firstly proposed an idea about distributed deep learning without sharing data sets[64] in 2015. Later that year, researchers from Google designed federated optimization[32]. Its purpose is to improve communication efficiency during learning from decentralized data sets. They named their idea as *Federated Learning* and further improved the framework and algorithms in 2016 and 2017[31, 33, 34, 46]. Generally speaking, these works can be viewed as a kind of mobile distributed machine learning algorithms which mainly focus on how to make use of data without uploading data to the server so that the privacy of users can be preserved. If we can run some part of the machine learning process on users' devices and only upload results of the computation, sensitive data can thus be protected well since the attack against the result is much harder. However, besides privacy preservation, model performance and system scalability are also important topics in mobile distributed machine learning that are worthy to be studied but haven't been paid much attention to by federated learning. We hope that more works will exist in the future to build up a more complete and robust mobile distributed machine learning system.

The rest of this survey is organized as follows: In Section 2, we introduce the areas which are closely related to mobile distributed machine learning. In Section 3, we define the main task and give the general properties. In Section 4, the whole mobile distributed machine learning problem is

divided into three parts that are machine learning, distributed optimization, and data aggregation. Potential solutions are presented for each of these parts. In Section 5, we introduce works on federated learning, which can be viewed as attempts on mobile distributed machine learning with privacy preservation as the core property. In Section 6, we discuss the open problems and future directions of mobile distributed machine learning. Finally, we summarize in Section 7.

2 RELATED AREAS

2.1 Machine Learning

Generally speaking, the core idea of machine learning algorithms can be concluded as training the machine to learn the rules behind some happened things using a lot of data and then make some predictions on unknown things based on the learned knowledge. Many machine learning tasks were called PR(Pattern Recognition) before the concept of machine learning exists, including face recognition, voice recognition, character recognition and so on. Since we human beings cannot use the programming language to tell the machines all detailed rules that we are following when we are doing these tasks, machine learning is proposed to try to make the machines learn the hidden and even implied rules by themselves. Here we simply introduce several commonly used technical words in machine learning and show how the goal is achieved.

Suppose we are going to train a machine to classify whether a fruit is an apple or a banana. We first have to collect some samples that can be referenced and learned by the machine. So we randomly pick some apples and bananas and list the features of them, including shape, color, weight, size and so on. We also have to add correct labels to let the machine know what they really are. Now, a labeled fruit with a group of features together build up a sample. We hope that the machine can learn from these samples and becomes able to make good predictions given new groups of features. This learning process can be expressed as fitting a function that takes the features as inputs and outputs a value which is as close as possible to the true label. The function is also called the machine learning model. Now the whole procedure of machine learning can be divided into two parts:

(1) Design a good structure for the machine learning model according to the task.
(2) Use data to check the model's performance and optimize it by adjusting its parameters.

There are many well-designed models which can be referred to in the first step. They are known as different kinds of machine learning algorithms and are usually suitable for different kinds of tasks. For traditional machine learning algorithms such as the k-means algorithm[41], SVM[73] and the EM algorithm[17], you can refer to a machine learning survey [75] to know details about them.

However, which machine learning algorithm to be chosen is mainly determined by your task. The mobile distributed machine learning framework isn't sensitive about the choice. Since many machine learning lectures have already explained how to choose a suitable machine learning algorithm in detail, we will not further talk about it in this article.

For the second part, it is done through machine learning optimizers. A general process is shown in Figure 1.

As shown in Figure 1, in the beginning, we first preprocess the original data to extract the features in them. The results are divided into three sub-sets which will be used for different purposes. The training set is the biggest sub-set which is used to train the model. The validation set is used to validate the effectiveness of the model and check whether the model is being trained correctly. The validation results can also be used to tune the hyperparameters for a better and quicker training process. The test set is used to test the performance of the final model after the whole training process is finished. From the figure we can see that the optimizer is the one who actually takes

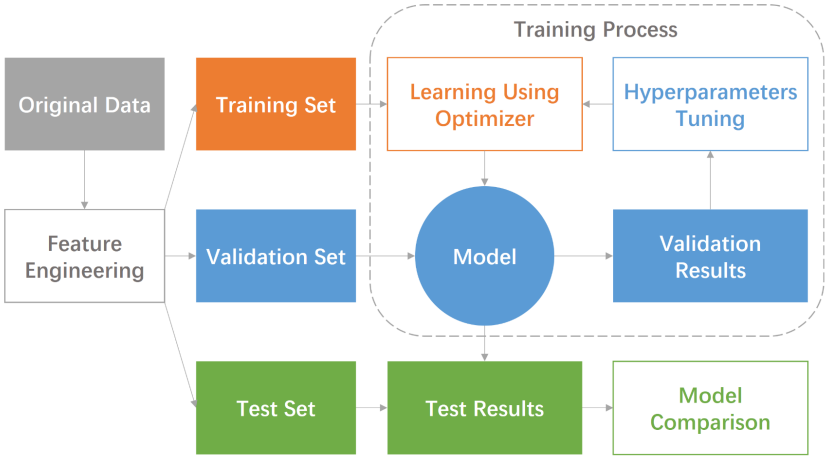


Fig. 1. Process of Machine Learning

charge of the training. Generally speaking, an optimizer focuses on how to optimize the model to reach its best performance based on the given data set. For example, GD(Gradient Descent) calculates the model's gradient based on the loss of every sample in the data set and uses this accurate gradient to optimize the model in each training iteration. However, the computation cost of GD is so high that it is not possible to be applied to those models with a large number of parameters. So many different schemes have been designed to find a better balance between the convergence speed and the computation cost. For those machine learning algorithms which are not suitable for using GD-based methods as their optimizers, they have their specific optimization algorithms, such as SMO(Sequential Minimal Optimization) for SVM and Canopy for k-means. Since the optimization process of mobile distributed machine learning is required to be run on mobile devices with limited resources, optimizers that need less computation in a training iteration is preferred in this scenario. We will further introduce machine learning optimizers and compare their advantages later in Section 4.

In recent years, deep learning[37] which is based on ANN(Artificial Neural Network) is proved to be very effective in many areas such as image recognition and natural language processing. Unlike the above-mentioned traditional machine learning algorithms, deep learning is an end-to-end method which can automatically extract features from raw data and learn without manual intervention. The word "deep" is used to describe the multi-layer structure of the neural network used by it. Deep learning needs much more training data and takes more computation costs during the training process due to the large number of parameters contained in its complex network. Considering that the scale of a deep learning model can be very large, it is usually trained and restored on high-performance servers. The concept of deep learning can be further divided into CNN(Convolutional Neural Networks), RNN(Recursive Neural Networks), GAN(Generative Adversarial Networks) and so on according to the structure of the neural network. Most of them choose to use GD-based methods and the backpropagation algorithm to optimize the model. Generally speaking, deep learning is a technique developed from traditional machine learning. It is based on a multi-layer neural network and benefited from the increases in the amount of training data. An example of a deep learning model is shown in Figure 2.

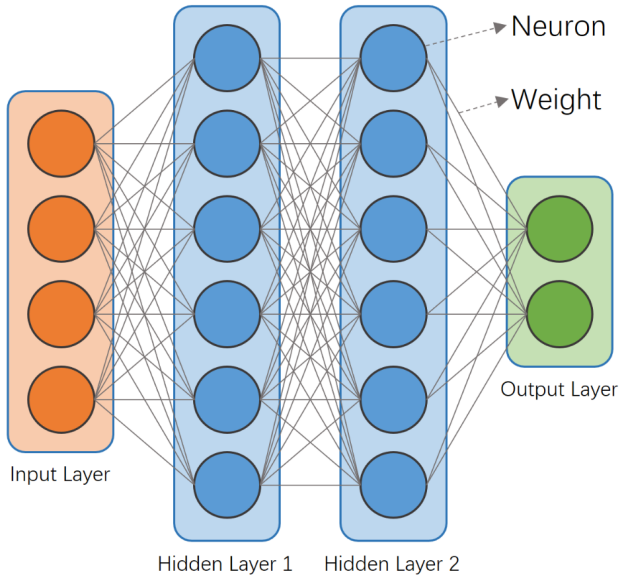


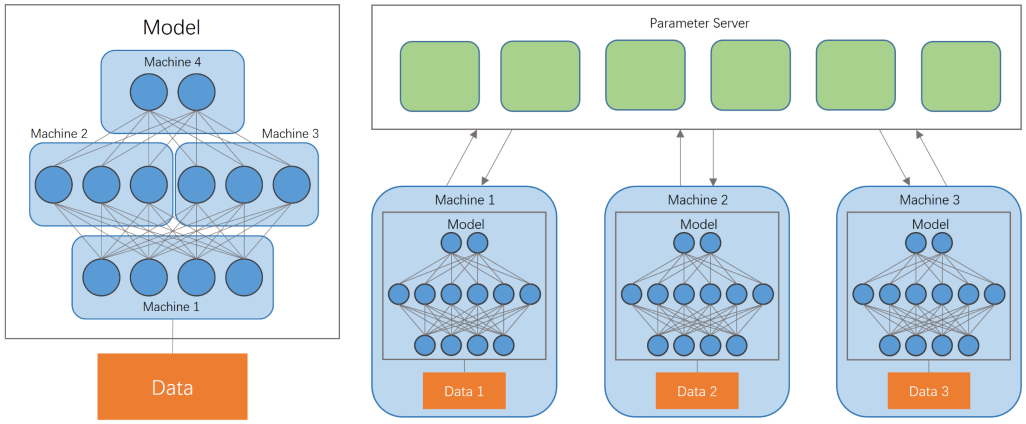
Fig. 2. Example of Deep Learning Model

Although centralized machine learning is very mature and has already solved many problems, it still has huge disadvantages. First of all, centralized schemes can only make use of the computation power of a single machine, which means it may cost a long time to get a well-trained model when dealing with complex tasks. Secondly, for those large-scale models that have billions and trillions of parameters and need huge amounts of data during the training process, the hardware on one machine can hardly support the learning task. So, to solve these challenges, distributed machine learning occurs.

2.2 Traditional Distributed Machine Learning

As we are living in the information era, the total amount of information is growing day and night. Sometimes the data set for a machine learning task is so big that it cannot be stored and processed by a single computer. According to [50], distributed machine learning with data parallelism has emerged to solve the above-mentioned problem. In this scheme, the whole data set is divided into sub-sets and stored distributedly on machines. Each machine also keeps a copy of the model and trains it based on the locally available part of data. After several iterations of training, the local models may become quite different from others. So the information is gathered to generate a final updated global model. This process is called data aggregation and can be done by a parameter server. Then, if the performance of the new global model is still not satisfying enough, another round of training can be started. With data parallelism, much more data is processed at the same time by different machines, which means it not only solves the problem on data storage, but also improves the efficiency of the training process.

In some special machine learning tasks, the model can be so large that it is too slow and even not able to be trained and run on a single machine. This problem is particularly serious in deep learning tasks. So large-scale distributed deep network is proposed in [15] trying to deal with it. Model parallelism methods are adopted and used to train large models with billions of parameters. In this scheme, each machine keeps a small part of the whole model. During training, the data



(a) Model Parallelism (b) Data Parallelism

Fig. 3. Architecture of Model Parallelism and Data Parallelism for Traditional Distributed Machine Learning

flows through machines in order to be processed by the local sub-models. On most occasions, every round of training needs the cooperation of all machines. So, to some degree, this process should be done serially since the inputs of some machines depend on the outputs of other machines. In this situation, using a scheduler to manage the training process may be helpful for solving the dependency between machines. Compared with data parallelism, model parallelism is more complex and also harder to implement due to the strong cooperation among machines. However, they are both important designs because they have solved two different limitations in the area of distributed machine learning. We show the architecture of model parallelism and data parallelism in Figure 3.

Besides the parallelism during the training process, the distributed communication scheme is another hot topic in distributed machine learning. Without a well-designed communication scheme, the limitation on the network bandwidth may become a troublesome bottleneck for the whole system. The schemes can be divided into the following three kinds:

- MapReduce/AllReduce: The first kind of distributed communication scheme is based on MapReduce/AllReduce. In MapReduce, the map operation distributes the data and tasks to machines and the reduce operation aggregate all results. [23, 47] are machine learning frameworks based on MapReduce. AllReduce is provided by MPI(Message Passing Interface) with a similar idea. It has been realized through many different topologies such as star topology, tree topology and ring topology. Each topology has its specific advantage. For example, compared to the simple star topology which is easy to be managed by the central master node, all nodes in ring topology[49] are machines with same jobs and they send messages in a ring to make full use of the network bandwidth. It is usually used to make multiple GPUs cooperate with each other efficiently. One disadvantage of MapReduce/AllReduce-based schemes is that it can be easily blocked by slow machines since all operations are synchronous and should be taken at the same time. To solve this problem, redundancy is usually added to the system.
- Parameter Server: The parameter server structure is also easy to be thought of. In this scheme, the parameter server can be either a single server or a server cluster that takes charge of the task arrangement but doesn't take part in the solving process. Each worker only has to communicate with the central server to pull or push data and has no need to be aware of other participating machines, which means the computation task of a worker is

independent of other participators so asynchronous working is possible. Compared with the above-mentioned MapReduce/AllReduce, it is more robust since it supports asynchronous communication and thus won't suffer from the slow machine problem. It has been applied to distributed deep learning tasks by Google in [15]. They further analyze its convergence in [38]. An example of this scheme is shown in Figure 3-b.

- Data Flow: For distributed machine learning with model parallelism, a specially designed scheme called data flow can be applied. Unlike the above-mentioned two schemes in which each worker has similar functions for the whole task, in this scheme, different parts of the model are distributed on different machines so their jobs vary from one to another. The whole computation process is organized using a directed acyclic graph. Nodes are units of the model and edges describe how data flows. If data flows between two units which are stored on different machines, communication will be taken place. An example of this scheme is shown in Figure 3-a. The disadvantage of this scheme is that the failure of any worker can cause the graph to be incomplete and the system can no longer run normally. If we use redundancy to solve this problem, backups for every worker is necessary. This may result in a very expensive cost if the number of machines is large. So the data flow scheme is more suitable for the cooperation among several powerful and stable machines.

From these techniques, we can reach the conclusion that traditional distributed machine learning usually focuses on the cooperation between powerful machines. Its core idea is using data parallelism or model parallelism to solve large-scale machine learning problems. The different communication schemes are designed for dealing with the low bandwidth of the local area network compared with the shared memory. However, in the scenario of mobile distributed machine learning, the challenges are mainly due to the limited resources and the unstable network, which is quite different from those of traditional distributed machine learning. So even if some techniques such as data parallelism and parameter server can be referred to for the design of mobile distributed machine leaning, we still have to pay attention to the special limitations in its scenario.

2.3 Mobile Machine Learning

In recent years, some works have already tried to run machine learning tasks on mobile devices. For example, [79] runs the trained and distilled model on mobile phones to recognize different kinds of pills using cameras. Compared with uploading photos of pills to the server to accomplish the recognition task, doing inference locally can protect users' privacy. Some other tasks such as face recognition and voice recognition are also necessary to be done locally for the similar purpose. In order to do these inference tasks more quickly and efficiently, NPU(Neural Processing Unit) has been designed for mobile devices. Some flagship products are already equipped with this technology.

According to our knowledge, there already exists some open-sourced project aiming to make the machine learning inference process easier to be done on mobile phones. However, few of them supports the training process. For example, TensorFlow Lite is one of the most commonly used mobile machine learning frameworks. It is widely used for doing inference based on trained deep neural networks. According to its roadmap for 2019, it will support training operations soon. From this we can see that although training models on mobile devices may not be possible for now, we will be able to achieve this goal in the near future as the calculation power of mobile devices keeps growing and the general mobile learning framework keeps being developed, which suggests that mobile distributed machine learning is also coming soon.

We summarize the difference between mobile inference and mobile training in Figure 4.

(a) Inference on Mobile Devices (b) Training on Mobile Devices

Fig. 4. Process of Inference and Training for Mobile Machine Learning

From the figure we can see that the mobile inference process just returns a prediction result which may be used to produce a service for the user locally. It has nothing to do for the server. However, if some part of the training process can be done locally on mobile devices, the generated trained model can not only become personalized but also provide valuable information for the global model stored on the server if the update of the local model is uploaded. This is exactly the idea of mobile distributed machine learning. What's more, the additional training process is not conflict with the original inference process and can even improve the accuracy of it because the model is further trained.

2.4 Edge Computing

[63] introduces that edge computing refers to the enabling technologies allowing computation to be performed at the edge of the network, on downstream data on behalf of cloud services and upstream data on behalf of IoT services. Its core idea can be concluded as using resources on edge nodes to do computation and thus reducing the burden of the server. However, in this scenario, if the number of nodes is large, the server still has to spare a lot of resources to communicate with them and may cause network congestion. So MEC(Mobile Edge Computing) was proposed. According to [44, 45], it wants further transfer the workload of the centralized server to the edge of the network by building MEC servers at the edge. With the help of 5G mobile network, this scheme can also reduce the latency and the usage of the global bandwidth. Now, this technology is renamed as Multi-access Edge Computing.

The above-mentioned techniques are all designed for helping the heavily loaded server. Although the original purpose of mobile distributed machine learning is to make use of local data, which is a little bit different from that of edge computing, it has a similar effect on reducing the burden of the centralized server. The development of MEC reminds us that we can also use it to deal with some of the limitations in mobile distributed machine learning. The MEC servers locate at the edge of the network and are designed to provide low-latency services. We have already mentioned that one of the challenges of deploying mobile distributed machine learning is that complex and heavy learning tasks may not be able to be done on mobile devices with limited hardware resources. So if we can let mobile devices communicate with the MEC servers to get additional help, this will be a possible solution to the problem. We will introduce the detail of it later in Section 6.

3 OVERVIEW

Mobile distributed machine learning is motivated by the strong need of making the best use of user data generated on mobile devices and protecting users' privacy at the same time. In contrast to traditional machine learning algorithms that require centralized data sets, distributed machine learning on mobile devices separate the task into sub-problems. The mobile devices solve sub-problems according to local user data. Finally, the server aggregates all intermediate results and gets the trained model. In this section, we formally define the task in Section 3.1. Then we discuss the general properties in Section 3.2.

3.1 Task Definition

Here we first introduce some notations and definitions which will be used:

- A real matrix $\mathbf{W} \in \mathbb{R}^{d_1 \times d_2}$ is the model learned from decentralized data.
- A list $\{C_1, C_2, \ldots, C_m\}$ contains all m clients that are run on different mobile devices.They are also known as the workers of mobile distributed machine learning.
- A list $\{D_1, D_2, \ldots, D_m\}$ contains all local datasets used by the clients.
- A server S that communicates with clients and arranges their works. S can either be a single computer or a server cluster whose structure is transparent to the clients.

Then a general process of mobile distributed machine learning can be expressed as follow:

(1) The workers $\{C_1, C_2, \ldots, C_m\}$ run on mobile devices download hyperparameters from the server S and get themselves initialized.
(2) The server S arranges the tasks that should be done in this round and sends them to the workers $\{C_1, C_2, \ldots, C_m\}$.
(3) The workers $\{C_1, C_2, \ldots, C_m\}$ train the local models based on the local datasets $\{D_1, D_2, \ldots, D_m\}$. Updates will be generated after a certain number of local training iterations.
(4) The workers $\{C_1, C_2, \ldots, C_m\}$ upload the updates to the server S.
(5) The server S aggregates all updates to generate a final update and applies it to the global model $\mathbf{W}$. A single round of learning ends here. Go back to step 2 to start a new round of learning.

Now we can divide the whole process of mobile distributed machine learning into three stages according to the general process: local training, communication and aggregation. We will show how they cooperate with each other and what kinds of techniques can be used to improve them.

Local training is done on workers in step 3. The server has nothing to do in this stage so it can be accomplished totally offline. However, since we have mentioned in Section 2 that the main constraint of mobile machine learning is that the resources are very limited, the local machine learning optimizer should be chosen and designed carefully so that it can be run on mobile devices without significant bad effect on users' daily usage experience. In this process, the input is the initialized model and the output is the update. The update can be generated based on either the new model which has been trained on the local dataset or the difference between the two models.

Communication takes place in step 1, step 2 and step 4. It is related to both the server and the workers, which means they have to cooperate with each other in this stage. The communication process mainly contains two key problems: task arrangement and data transfer. From the server's perspective, it should concern about the task arrangement part, whose main purpose is to divide the original complicated problem into easier sub-problems. This can be done through distributed optimization algorithms. The input is the original problem and the output is a set of sub-problems. Some simple information about the local datasets and the available resources can be added as additional inputs to make the sub-problems become more suitable for being solved by the corresponding workers. From the workers' perspective, the data transfer part is the main challenge

because usually mobile network resources are limited and expensive. Compression methods can be used to reduce the cost of communication. So the input of this part is the original data and the output is the compressed data.

The aggregation is done on the server in step 5. Since the number of updates received from workers can be very large, it is inefficient and even impossible to apply all updates to the global model one by one. So the main purpose of the data aggregation process is to extract and make the best use of the information contained in the updates. This process takes all received updates as inputs. Its output can be either a final update which can be directly applied to the global model or a new model which replaces the old one.

Before we introduce the possible solutions for each stage in Section 4, we first claim the general properties of mobile distributed machine learning to show our scenario in detail. We also analyze and conclude the challenges we may face to demonstrate the key points needed to be cared about if we would like to implement mobile distributed machine learning in real life.

3.2 General Property

We could say that the only difference between mobile distributed machine learning and traditional distributed machine learning is that the place where the training process is done has changed. However, this change introduces additional constraints for the whole system due to the limitations on mobile devices:

- The hardware resources available on mobile devices are poor and limited. (compared with PCs and servers)
- The network condition for mobile devices is unstable. The communication cost is very high.

Here we will continue to show how the above two limitations lead to the general properties of mobile distributed machine learning.

Let us first discuss the first limitation. Although the development of mobile devices has kept accelerating in recent years, their computation power is still far less than that of personal computers, not to mention the server clusters. The key reason for this phenomenon is that mobile devices need to be portable. To achieve this goal, the measurement adopted by mobile devices is to use small batteries with very limited energy as their power source. However, the energy density of batteries hasn't been improved much while the CPUs used on mobile devices have become more and more powerful. Compared with the old mobile phones which can standby for more than a week, smartphones can only support one day's daily usage with the same battery capacity due to the energy-consuming chips. In this situation, chips are specially designed for mobile devices to balance between the need for greater power and the need for lower energy consumption. So unlike the CPUs used on personal computers which can easily reach 100W in power with stable power supply, the peak power of CPUs used on mobile devices is usually under 10W. Considering that mobile devices have to spare some resources and energy to support its basic functions, the part which can be used for supporting mobile machine learning becomes even less. Moreover, the mobile devices usually have no additional cooler and just use passive cooling methods to ensure their small size. This also limits the maximum power of them.

Now we can get the first general property:

- The machine learning sub-problems solved on mobile devices should be easy.

Firstly, mobile devices cannot handle with complex machine learning tasks because the power is limited. Secondly, since only passive cooling methods are used, a hard machine learning task is very likely to cause the mobile device becomes hot. This mustn't happen as it will take huge negative effect on the user experience because no one would like to hold a hot phone in their hands

or in their pockets. So the machine learning sub-problems solved on mobile devices are better to be easy enough so that the users are even unaware of its running process at all.

Another property which can be concluded from the limitation of resources is that:

- The time spent on a round of local training process on a mobile device should be short.

This is not very obvious so we will explain it in detail. We have already mentioned that the hardware resources available on mobile devices are poor and limited compared with personal computers. So the systems run on mobile devices are specially designed. In a word, they may pause the apps running in the background to save resources for the app that is currently being used. Considering that most users may just use an app for a little while at one time, it is better to finish the machine learning task during the time that the app is running in the foreground. Otherwise, if the training process hasn't been finished when the app is paused or quit, the generated update may not be sent to the server in time and become useless at the next start because it has delayed too much.

Then let us consider the second limitation which is about the mobile network and the communication process. For now, although the 4G network has already been deployed widely and the 5G network is coming soon, the cost of data on mobile devices is still expensive. What's more, local data on mobile devices may contain users' sensitive information that is not suitable for being uploaded to the server even if the communication process is anonymous. So here we can get the third property:

- Local data shouldn't be uploaded to the server in principle.

However, considering that uploading a small subset of local data is feasible and may be helpful for the aggregation process on the server, it is permitted under special situations with guaranteed privacy. It is worthy to be mentioned that this "no uploading" property is one of the causes of the significant difference between traditional distributed machine learning and mobile distributed machine learning. Without uploading data, the local datasets used for training are specific to their owners and can never become IID(Independent and Identically Distributed) which is a necessary condition for many machine learning algorithms. Moreover, this will also cause the sizes of local datasets to be unbalanced, which results in the amount of useful information contained in different workers' updates being unbalanced. So how to determine which updates are more valuable becomes another problem needed to be solved.

The fourth property can also be concluded due to the high cost of communication:

- The data transferred in a round of learning shouldn't be too much.

This property also aims at protecting the experience of users' daily usage. If too much data has to be transferred, it may be a heavy burden for the mobile devices even if they are connected to WLANs and can cause other apps that need to use the network to be blocked.

The fifth property is about the frequency of the communication process:

- The communication frequency should be low and it is better to let the workers decide when to communicate.

As the network condition for mobile devices is unstable, a scheme with frequent communication is unsuitable for the current scenario. What's more, a low communication frequency with a long training time is beneficial for reducing the communication cost. The reason why we suggest to let the workers decide when to make the communication is that usually workers' computation power varies from one to another so the server has no idea when the sub-problem will be solved by a worker. This design gives freedom to the workers so that they can choose to do the communication at the time when they are connected to WLANs, which further reduces the communication cost.

From the above five general properties, we can conclude the main challenges of mobile distributed machine learning:

- Transform the original machine learning task to easier sub-problems which can be solved on mobile devices easily and quickly.
- Deal with the non-IID dataset and cover its negative effects.
- Compress the transferred data between the server and the workers.
- Lower the frequency of the communication process and improve its efficiency.

Most of them haven't been solved yet in the scenario of mobile distributed machine learning. So in Section 4, we introduce several works which can be referred to as basic algorithms for the possible solutions.

4 ALGORITHMS FOR MOBILE DISTRIBUTED MACHINE LEARNING

The whole mobile distributed machine learning can be roughly divided into three parts:

- Machine Learning Optimizers (Section 4.1) mainly focus on how to train a model more efficiently. There already exists many powerful optimization algorithms which are developed from GD(Gradient Descent). However, as mentioned in General Property (Section 3.2), since the scenario of mobile distributed machine learning is quite different from that of traditional machine learning, some state of the art algorithms may not be suitable for this task. So in this part, we introduce several GD-based machine learning optimizers which are efficient and may cooperate well with distributed communication algorithms. For those machine learning models which are not suitable for being applied with GD, as mentioned in Section 2.1, their specially designed optimization algorithms are recommended to be chosen and we will not discuss them here.
- Distributed Optimization Algorithms (Section 4.2) pay attention to the arrangement of tasks. Since it is hardly possible to train a whole model on a single mobile device, we have to divide the original task into sub-problems. Each worker just deals with its sub-problem and uses an appropriate machine learning algorithm as its local solver. In a word, the distributed optimization algorithm plays the role of a manager and the mobile devices serve as workers. A well-arranged distributed system can not only train a better model but also save resources for mobile devices.
- Data Aggregation Methods(Section 4.3) aim at making full use of the updates generated by workers. The weights for different users' contribution to the global model should be carefully balanced according to their attributes. Moreover, it can also be considered as an additional firewall for privacy. Although mobile distributed machine learning doesn't require the local sensitive data being uploaded to the server, we have already mentioned in Section 1 that sensitive information can be inferred from the model. So it is necessary to apply methods such as k-anonymity and differential privacy during data aggregation to further protect the information contained in the uploaded data.

4.1 Machine Learning Optimizers

In this part, we mainly focus on optimization algorithms for machine learning. They will work as local optimizers. Since we have already mentioned in Section 3.2 that computation done on mobile devices shouldn't be too complicated to have negative effect on experience of user's daily usage, we don't have to take GD into consideration due to its huge computation cost in each training step. However, SGD and its different versions are very suitable for this work as their computation cost in a round is much lower than traditional GD. Here we will introduce several optimizers which are developed from SGD and analyze their advantages.

- SGD[59]: SGD(Stochastic Gradient Descent) is an ancient algorithm firstly proposed in 1951. The main advantage of SGD is that it greatly reduces the computation cost in each iteration

compared with GD(Gradient Descent). Its core equations are very similar to those of GD and are shown as follows:

$$g_t = \nabla f(\theta_{t-1}) \tag{1}$$

$$\theta_t = \theta_{t-1} - \eta \cdot g_t \tag{2}$$

where t is the timestamp for the current training iteration, θ_{t-1} is the model at time $t-1$, g_t is the corresponding gradient for the loss function f at point $t-1$, η is the learning rate. The feature that it only uses one sample to compute gradients in each iteration can benefit distributed machine learning a lot. Many improved versions of SGD like mini-batch gradient descent also have similar advantages. Although the solution for distributed machine learning on mobile devices is unlikely to choose the original SGD as its local optimizer since a much more efficient algorithm is expected under the environment of mobile devices, we have to say that SGD still works well on many other occasions.

- SVRG[29]: SVRG(Stochastic Variance Reduced Gradient) aims at accelerating the convergence speed of SGD by applying noise reduction methods. It is commonly known that compared with GD(Gradient Descent), SGD does much less computation in each iteration but has a lower convergence speed. [6] shows that one main reason for the decrease in convergence speed of SGD is the noise exists in the estimate of gradient, which can be also considered as the variance of computed gradients. So SVRG uses gradient aggregation methods to make corrections based on historical information and thus reduces the noise. The core equations are shown as follow:

$$g'_t = \nabla f(\theta_{t-1}) - \nabla f(\bar{\theta}) + \bar{g} \tag{3}$$

$$\theta_t = \theta_{t-1} - \eta \cdot g'_t \tag{4}$$

where $\bar{\theta}$ is an averaged model which is updated every k iterations and $\bar{g}$ is a gradient averaged among all data samples at point $\bar{\theta}$. The first term $\nabla f(\theta_{t-1})$ on the right side of Equation 3 is exactly the g_t used in SGD. The core idea of SVRG is to make the upper bound of gradients' variance keep being reduced during training by adding additional computation for correction which is $(-\nabla f(\bar{\theta}) + \bar{g})$. [46] shows that SVRG can cooperate well with some distributed optimization algorithms like DANE[62] by working as the local optimizer.
- ADAM[30]: Adam is an optimization algorithm based on SGD aiming to accelerate the convergence speed by adaptively tuning the learning rate. It was proposed in 2014. Before Adam, there already exists some algorithms trying to improve SGD through making use of the momentum of gradients such as SGDM[54] (SGD with Momentum), AdaGrad[20] and RMSProp[72]. Adam combines the advantage of AdaGrad and RMSProp and makes use of both the first moment estimate and the second moment estimate. Its equations are shown as follow:

$$m_t = \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t \tag{5}$$

$$v_t = \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2 \tag{6}$$

$$\hat{m}_t = \frac{1-\beta_1^t}{m_t} \tag{7}$$

$$\hat{v}_t = \frac{1-\beta_2^t}{v_t} \tag{8}$$

$$\theta_t = \theta_{t-1} - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{9}$$

Here m_t is the first moment estimate and v_t is the second moment estimate. β_1 and β_2 are control parameters. $\hat{m}_t$ and $\hat{v}_t$ are the bias-corrected version of the moment estimates. ϵ is a small constant used to prevent division by zero. Experiments show that Adam can make the

training process become faster and more stable. Since in the scenario of distributed learning on mobile devices, communication rounds may be limited, an efficient optimizer which can accelerate the training process may help a lot on the performance of the trained model. However, [74] shows that on some special occasions, ADAM may fail to reach convergence.

- Hogwild![55]: Hogwild! is a work aiming to prove that parallel SGD can be implemented without any locking. It shows that when the optimization problem is convex and sparse, most updates only modify small subsets of all parameters, which means the whole update process can be run asynchronously without locking. However, it has only been tested on traditional machine learning problems such as sparse SVM and matrix completion. It may not be suitable for complex tasks such as deep learning.
- ASGD[16]: ASGD(Asynchronous SGD) is a simple attempt for making use of more machines to train a huge deep network through asynchronous methods with SGD. Compared with common synchronous SGD, ASGD won't suffer from the slow machine problem which is a huge obstacle if we want to deploy distributed machine learning on mobile devices. Since no waiting is needed, all machines can make best use of their resources and together accelerate the convergence speed of the training process. However, the problem is that in the asynchronous method, the delayed gradients may be unsuitable for being applied to the current updated model. This can cause fluctuation in weights and have a negative effect on the performance of the final model according to [3, 39].
- EASGD[80]: EASGD's(Elastic ASGD) purpose is to reduce the communication cost between workers and the parameter server during parallel training. Each worker's local model won't be updated equally according to the global model after one communication round. The communication and coordination of work among all workers is controlled by an elastic force that links the local parameters with a center variable stored by the parameter server. The update rules are shown as follows:

$$\theta_t^i = \theta_{t-1}^i - \eta \cdot (g_t^i + \rho \cdot (\theta_{t-1}^i - \bar{\theta}_{t-1})) \tag{10}$$

$$\bar{\theta}_t = \bar{\theta}_{t-1} + \eta \cdot \sum_{i=1}^{p} \rho \cdot (\theta_{t-1}^i - \bar{\theta}_{t-1}) \tag{11}$$

  where i is a random index of a worker, p is the number of workers, ρ is the control parameter for the elasticity and $\bar{\theta}_t$ is the center variable. In this scheme, historical information is needed because the center variable is updated as a moving average which is taken in both time and space over all local parameters. According to the paper, this elastic force design allows workers to do more exploration in its nearby parameter space which can do good to the performance of the model. But it is worthy to worry about what if the elastic hyperparameter isn't set properly and causes the workers to explore too far away from the center variable, the model may become even worse. The effectiveness of EASGD is only analyzed in the quadratic and strongly convex case.
- AdaDelay[68]: AdaDelay(Adaptive Delay) is an asynchronous SGD-based method that tolerates stale gradients. In the first stage, the delay model does updates as soon as a gradient is received from any worker. In the second stage, the delay model takes larger update steps when it receives a gradient from a worker that sends updates infrequently. A smaller update step is taken when the gradient is received from a worker that sends updates frequently. The second stage is designed to reduce the bias caused by the aggressive first stage. Convergence analysis for the algorithm dealing with smooth stochastic convex optimization problems is given in the paper.

Table 1. Comparison of machine learning optimizers suitable for distributed mobile learning

Name	Computation Cost	Convergence Speed	Robustness	Non-Convex	Historical Information	Async	Delay Solution
SGD	Normal	Normal	Normal	Supported	Not Required	No	/
SVRG	Very High	Fast	Normal	Not Supported	Required	No	/
ADAM	High	Very Fast	A Little Weaker	Supported	Required	No	/
Hogwild!	Normal	Normal	Weak	Not Supported	Not Required	Yes	No
ASGD	Normal	Slow	Weak	Supported	Not Required	Yes	No
EASGD	High	Slow	Weak	Not Supported	Required	Yes	Not Needed
AdaDelay	Normal	Normal	Normal	Not Supported	Not Required	Yes	Delay-Tolerant
DC-ASGD	A Little Higher	Normal	Normal	Supported	Required	Yes	Compensation

- DC-ASGD[83]: The first version of DC-ASGD(Delay Compensated ASGD) was finished in 2016. It was published in ICML'17. It focuses on reducing the error caused by delayed gradients in ASGD. The main idea of DC-ASGD is to use the first-order term in Taylor series to compensate for the delayed gradient and use an approximation of the Hessian Matrix to reduce computation costs. At time $t + \tau$, to update model $\theta_{t+\tau}$, the original delayed gradient $g(\theta_t)$ for old model θ_t will be replaced by the delay-compensated gradient which can be expressed as follow:

  $$g(\theta_t) + \lambda g(\theta_t) \odot g(\theta_t) \odot (\theta_{t+\tau} - \theta_t)$$

  λ is a variance control parameter controlled by the server. So at time $t + \tau$, if the server needs to compensate for the delayed gradients, the only additional knowledge needed is the historical model θ_t, which means that this method is easy to be realized. The convergence rate of DC-ASGD when dealing with smooth and non-convex loss functions is analyzed and the proof is given in the paper. Experiments are done on CIFAR-10 and ImageNet datasets and results show that DC-ASGD outperforms both SSGD and ASGD. However, since none of the experiments covers the situation that more than 16 workers working together, DC-ASGD's performance under a large number of workers still needs to be studied.

We regard SGD as a standard and compare these deep learning optimizers in Table 1. The features taken into consideration are explained as follows:

- Computation Cost: Usually a higher computation cost means a more complex training process. For the above-mentioned algorithms which are developed from SGD, this feature is mainly related to the amount of additional information other than the data point itself used in each training step. For distributed mobile learning, optimizers with lower computation cost are preferred since they consume fewer resources and thus cause fewer negative effects on user experience.
- Convergence Speed: It is widely known that compared with traditional GD, SGD requires much less computation in each step while its convergence speed is slower. So algorithms like SVRG and ADAM use additional information to accelerate the training. However, in distributed mobile learning, the trade-off between computation cost and convergence speed needs to be carefully considered. In order to guarantee the experience of users' normal usage, a little sacrifice on convergence speed may be permitted.
- Robustness: This feature represents the ability of an algorithm to adapt for different tasks. According to analyses made in [3, 39, 56], ASGD and ADAM may not always reach convergence successfully. So before applying the algorithms which are not so robust, we have to ensure that the task doesn't match the special occasion where the optimizer may not train the model to convergence.

- Non-convex: Some algorithms do not guarantee convergence when dealing with non-convex loss functions. Considering that most machine learning tasks appear in recent years are complicated and usually based on deep learning, which means non-convex problems may be very popular, we add this feature into the table.
- Historical Information: It has already been mentioned that SVRG, DC-ASGD and ADAM use historical information to make corrections on gradients for different purposes. This requires additional computation and space. Luckily, in DC-ASGD and ADAM, all additional operations can be done by the server and are transparent to clients. While making use of additional information usually improves the performance of the model, it will also make the algorithm become more complex and increase the burden on servers. So it is very necessary to take a good balance on this considering the task and the limitations.
- Async: As the number of workers grows, a synchronous system may become inefficient due to problems like the slow machine problem. So we add this feature to show whether asynchronous training is supported or not. Here, Hogwild!, ASGD, EASGD, AdaDelay and DC-ASGD are all designed for asynchronous schemes, which can be a huge advantage of them.
- Delay Solution: This feature is added for those optimizers which support asynchronous training. Hogwild! and ASGD have no solution for delayed gradients. EASGD doesn't need a delay solution because its global model is a moving average which is not updated through sub-models' gradients and there are no gradients transferred. For AdaDelay, it is designed to be delay-tolerant. For DC-ASGD, it compensates delayed gradients by using Taylor series expansion.

4.2 Distributed Optimization Algorithms

When we are running distributed machine learning algorithms on mobile devices, the availability of network resources becomes a serious problem. It is a little bit different from that of distributed systems which are designed for cloud computing or parallel training, for example, *Project Adam* [14] from *Microsoft Research*. This kind of algorithms mainly focus on how to better managing large scale of machines and making full use of their hardware to solve a huge complex task. In order to do that, a common solution is to transform the original hard problem into many much easier sub-problems. In contrast to those machines provided with a stable network condition and sufficient electric power, since no signal and low power are problems that can easily appear on mobile devices, it is quite obvious that reducing communication rounds and improving efficiency become the most important targets in this scenario.

- ADMM[7]: ADMM is short for alternating direction method of multipliers. In ADMM, machines distributedly use augmented Lagrangian methods to solve local sub-problems based on local data and alternately compute some shared variables in order to solve the large global problem. The idea of this algorithm can be traced back to the mid-1970s and has been used in distributed SVM training[21] in 2010. The authors improved it and applied it to machine learning applications.
- DANE[62]: DANE(Distributed Approximate NEwton) is an approximate Newton-like method. In every iteration, each machine separately takes an approximate Newton step with implicitly using its local Hessian and make two rounds of communication. In contrast to ADMM, DANE can be benefited by the fact that sub-problems are often similar in applications of machine learning. DANE performs well on smooth and strongly convex problems.
- CoCoA[67]: CoCoA's first version is proposed in 2014[28]. It can be viewed as a general framework for distributed optimization in machine learning tasks with smooth loss functions.

Table 2. Comparison of data aggregation algorithms suitable for distributed mobile learning

Name	Computation Cost	Convergence Speed	Non-Convex	Additional Space	Async
Simple Averaging	Normal	Normal	Not Supported	Not Required	Yes
Federated Averaging	Normal	Fast	Not Supported	Not Required	Yes
Ensemble-Compression	High	Fast	Supported	Large	No
Codistillation	High	Fast	Supported	Huge	Yes

By making use of convex duality, after dividing the task into approximate local sub-problems, it can be chosen whether to solve the primal sub-problem or its dual problem. The main advantage of CoCoA is its flexibility. It allows machines to choose the best local solver for sub-problems and training the model to arbitrary accuracy. What's more, the balance between local computation and communication can also be adjusted easily by setting a specific parameter. Experiments show that compared with other distributed optimization algorithms like ADMM, mini-batch SGD, mini-bath CD (Contrastive Divergence), L-BFGS (Limited-memory quasi-Newton method) and OWL-QN[78], CoCoA can achieve up to a 50× speedup when dealing with problems like SVM, logistic regression and lasso.

- CoCoA+[42]: CoCoA+ also makes use of the primal-dual problem to get optimization. Compared with CoCoA, CoCoA+ additionally guarantees the convergence rate on non-smooth loss functions, which means it can deal with more general local sub-problems. Furthermore, in order to get rid of the slow down problem caused by averaging updates, CoCoA+ choose to add all updates. Some other changes are also made to the algorithm to support the adding operation.
- DiSCO[81]: DiSCO(Distributed Self-Concordant Optimization) is also a Newton-type method. Compared with DANE, DISCO uses a distributed preconditioned conjugate gradient method to compute inexact Newton steps in each iteration and gets a superior communication efficiency. However, the number of steps in each iteration may change since it depends on many different factors. One significant advantage of DiSCO is that compared with other above-mentioned algorithms, it has fewer parameters which need to be paid attention and adjusted.
- DiSCO-S & DiSCO-F[43]: DiSCO-S is an improved version of DiSCO. It uses a matrix which can be viewed as an approximated or stochastic Hessian as its preconditioning matrix and uses Woodbury Formula to solve the linear system more efficiently. While dataset is partitioned by samples in DiSCO's default setting, DiSCO-F partitions dataset by features and thus only needs fewer MPI_Allreduce communication rounds. However, since in our scenario the dataset is already partitioned and distributed on devices by samples, DiSCO-F may not be suitable for this job.
- Hydra[57]: Hydra(HYbriD cooRdinAte) is a randomized coordinate descent method. This kinds of methods are becoming more and more popular in many learning tasks such as boosting and large-scale regression. In Hydra's design, the original data are partitioned and assigned to one node from the cluster of machines. Each node independently updates a random subset of its data based on a designed closed-form formula in each iteration. The updates are all parallelized. It has been tested on a LASSO instance described by a 3TB matrix.

4.3 Data Aggregation Methods

After the clients successfully finish their training process and generate the updates, how to make use of these data will be the next problem. If we just apply updates to the global model one by one,

it may cause the global model changes so quickly and frequently that its performance becomes very unstable. This operation may also be time-consuming if the number of updates is large. What's more, once the global model is updated for even one time, the updates that haven't been used thus become relatively delayed and less accurate for being applied to the new model. So it is necessary to design a data aggregation process whose job is to generate a final update which can be directly applied to the global model. Moreover, since the aggregation process is considered to be irreversible, an additional benefit of this operation is that it can prevent attackers from inferring sensitive information from the updated global model. Privacy preserving methods such as differential privacy can also be applied during aggregation to further ensure security. Here we will introduce several data aggregation methods.

- Simple Averaging: It is easy to be thought of and commonly used. Simple averaging just average all updates to get the final update. A constraint of simple averaging is that all sub-models should be initialized equally to the global model because it makes no sense to apply a completely different model's gradient to another one. However, according to our knowledge, there exists no proof that guarantees the convergence speed of this method, which means it may perform badly in some special situations.
- Federated Averaging[46]: This is improved from simply averaging. Its core ideas can be concluded as doing common initialization instead of independent initialization on all sub-models and doing weighted averaging among all received updates. The common initialization is the base of federated averaging. Simple experiments on MNIST dataset show that the final model averaged from parent models which are initialized using different random seeds suffers from severe performance loss. However, with shared initialization, the final averaged model gets an additional reduction in the loss on the whole training set. That is why common initialization is necessary for federated learning. The weighted averaging mainly aims at making better use of those updates which contain more knowledge. Since it is natural to consider the updates generated from larger datasets as more valuable ones, the weights for federated averaging are thus closely related to the sizes of datasets.
- Ensemble-Compression[70]: This method is quite different from the above-mentioned two averaging-based algorithms. For methods like Model Averaging which do aggregation by averaging model parameters, the effectiveness under convex optimization problems is proved. However, this work shows that for non-convex optimization problems such as DNN, averaging-based methods are bad in performance. Ensemble-Compression method is thus proposed. Suppose there are K clients, the first ensemble step is to produce a new global model which is made up of K sub-models and one additional layer used to average those K sub-models' outputs and give the final result. Since the size of the updated global model is at least K times larger than that of the local model after the ensemble step, the compression step uses distillation-based methods such as [8, 27, 60] to get a compressed new global model with the same size as the local model. Distillation-based methods can be replaced by other suitable compression methods[12, 18, 19, 24, 25, 58]. A problem of this method is that many compression methods may require the global dataset which is unavailable in mobile distributed machine learning. Finding a powerful compression method can be run without the original training dataset thus becomes a key challenge. Another problem is that it is not sure whether this ensemble-compression scheme is suitable for tasks done by thousands even millions of workers together because no more than 8 workers are deployed in the experiment of this work. What's more, since this Ensemble-Compression scheme average all K sub-models' outputs, it will be blocked at the ensemble process if less than K sub-models are received,

which means it doesn't support totally asynchronous training scheme and may easily suffer from the slow machine problem.

- Codistillation[2]: This work tries to use distillation-based methods to train a model more efficiently than distributed SGD on large-scale datasets. The basic idea of it is to train models on different locally available datasets and transfer information between them. After some burn-in steps, a worker will keep copies of others' models and make predictions for local samples with them. Then it will tune its parameter according to the outputs given by others. The reason why we put this algorithm in this data aggregation section instead of the machine learning optimizer section is that although it is designed to replace distributed SGD when dealing with large scale machine learning tasks, it can be combined with standard distributed SGD, resulting in a procedure that workers exchange gradients to do distributed SGD in a group and exchange model parameters between groups to do codistillation for data aggregation. Another advantage of codistillation is that it can give good support to asynchronous training more easily because making use of predictions given by stale models might be less of a problem than applying delayed gradients. Details of this technology are given in the paper. However, a severe problem of codistillation in the scenario of mobile distributed learning is that the number of workers is so large that it is impossible to save copies of all other models on a single mobile device. Some changes in its architecture should be done to solve this problem. Combining codistillation with edge computing may be another topic worth studying.
- Other Methods: Some distributed optimization algorithms such as CoCoA+[42] have already proposed suitable data aggregation methods for themselves. So if you choose to use one of them, no additional data aggregation method is needed.

We regard simple averaging as a standard and compare these data aggregation methods in Table 2. Most features we take into consideration has already been explained in Table 1. Here we just add one new feature:

- Additional Space: For averaging-based methods, usually the additional space they use is so small that it can be ignored because it is not necessary for them to save all sub-models independently. For ensemble-compression, in order to do the ensemble operation, it requires the server to save each whole sub-model, which means if the number of workers is big, the additional space used for this purpose will be very large. For codistillation, since a worker needs to keep copies of others' model, this requires huge additional space on every worker.

5 DEVELOPMENT OF FEDERATED LEARNING

Federated Learning was proposed by researchers from Google in 2015 to implement machine learning that can fully protect private data. Its core idea is to solve machine learning problems locally on users' devices and aggregate updates on the server without uploading the original user data, which is very similar to the scheme of mobile distributed machine learning. From 2016 to 2018, Google published several related articles to complement federated learning's framework. In 2019, applications of federated learning appear. In this section, the whole process of federated learning's development as well as its contribution is summarized. This part of works can be viewed as some attempts on mobile distributed machine learning with privacy preservation as the primary goal. They mainly focus on how to design a good communication scheme which requires fewer communication rounds to save resources for mobile devices and how to aggregate so large amounts of data on the server. Less attention is paid to the improvement of the final model's performance in federated learning.

5.1 Origin

The idea of federated learning can be traced back to DSSGD (Distributed Selective Stochastic Gradient Descent) [64]. It is published in October 2015 and is about distributed deep learning without sharing data sets. In the general procedure, each client firstly select some parameters, download them from the global model which is saved on server and replace the corresponding local parameters with them. Then clients independently train their local models based on the local data. After the training process, clients will select some updated parameters and upload their changes to the server. Although there are several available strategies for this selection, all clients should agree on one strategy and use it consistently during the whole training. Normalization can be done and noise can be added to the data message before uploading to ensure privacy and security. The server aggregate all uploaded changes and update the parameters of the global model. Some useful information like which parameter has changed most will be given by the server as information that can be referenced by clients to design their selection of parameters in the next download step. According to the paper, one advantage of DSSGD is that a participant can get rid of falling into local optima by constantly gaining additional information from others. DSSGD can also adapt to the situation that some clients become offline. However, it is easy to find out that DSSGD just uses simple SGD as core learning algorithm and the server works like some shared memory, which suggests that efforts can be paid to further improve the learning process and the communication scheme. So researchers from Google follow its idea of distributed learning and privacy preservation and try to apply the system on mobile devices, which is known as federated learning.

5.2 Basic Model

5.2.1 First Attempt. In November 2015, researchers from Google submitted their first attempt on federated learning [32]. The idea of federated learning can be viewed as an improved version of DSSGD's idea which is optimized for mobile devices. In this work, three basic properties of federated learning's scenario are proposed, which are known as *Non-IID*, *Unbalanced* and *Massively Distributed*. The fourth one *Limited Communication* is introduced in next paper [46]. Their meanings are as follows:

- Non-IID: Since data on a client is usually generated by a particular user, no local dataset can be the representative of the overall distribution.
- Unbalanced: Some users may be more active than others, which causes the amount of local data is unbalanced among different clients.
- Massively Distributed: The number of clients participating in one training process is much bigger than the average number of data points stored on a single client.
- Limited Communication: Mobile devices are usually available to limited and expensive network resources and are frequently offline. (This is proposed in their next paper [46].)

What's more, this work is concerned with dealing with the sparse data and makes use of the sparsity structure to develop an efficient federated optimization algorithm called DSVRG (Distributed Stochastic Variance Reduced Gradient) based on SVRG [29, 35] and DANE[62]. DSVRG uses SVRG as a local solver to produce an approximate solution instead of exactly solving the DANE sub-problem so that the algorithm will become more efficient and is suitable for being run on mobile devices. DSVRG also makes further changes on some equations to gain better performance. During the experiment, a binary classification task is designed. It is about predicting whether a post will receive comments. The result shows that DSVRG outperforms existing algorithms such as DANE, DiSCO and CoCoA in this specific scenario.

5.2.2 Complement. Later in 2016, [33] complements and improves the above work. DSVRG is renamed as FSVRG(Federated SVRG) here. Details and motivation of its design are explained carefully. Equations and mathematical proofs for it are also given. Even though few innovations can be found in this paper, it still contributes a lot to the development of federated learning since it guarantees the effectiveness and reliability of FSVRG, which is the basic and important algorithm in a federated learning system.

5.3 Communication Efficient Techniques

5.3.1 Reducing Communication Rounds. After the proposal of DSVRG which can be seen as the basic strategy for federated learning, since the result of experiment looks satisfying, researchers from Google shift their focus to developing a more efficient communication scheme, which is one of the biggest challenges for distributed machine learning on mobile devices. In [46], *Limited Communication*, as the fourth basic characteristics of federated learning's scenario, is proposed. So in order to solve the above-mentioned problem, especially to reduce communication rounds, FedAvg (FederatedAveraging) is designed. During the authors' exploration, they find out that if clients' local models are trained from independent random initialization, the merged global model suffers from great reduction in performance. However, if local models share a same random initialization and then be trained independently, the final global model merged from local models through naive parameter averaging is able to perform well. So shared initialization at the beginning of each communication round can ensure the effectiveness of FedAvg. What's more, compared with the old strategy in DSVRG that communication happens after every single training iteration, FedAvg only requires communication every E training iterations, which is the main method to reduce communication rounds.

The experiment plays an important role in this work. The authors have tested FedAvg on many public datasets and applied different parameter settings during the tests. They spend almost half of the paper to describe details of the experiments and analyze the results which are quite complicated. As shown by results, in some scenarios, FedAvg can make a 95× speedup for training the model to reach a certain accuracy. However, since the performance of FedAvg varies greatly among different datasets, FedAvg may not be suitable for all kinds of tasks. Further improvement is still needed when we try applying it to solve more common tasks.

5.3.2 Communication Compression. Different from reducing communication rounds, here communication compression means reducing the amount of data traffic in a single round. According to [34], as on most occasions the bandwidth of the uplink is much poorer than that of the downlink, reducing the uplink communication cost is a more urgent task. So the approaches are mainly about lower the size of the updates. They can be generally classified as *structured updates* and *sketched updates*:

- Structured updates: These kinds of methods apply restrictions to updates during the whole training process so that they have a pre-specified structure. In other words, with these methods, clients will directly give compressed updates after training is finished in each round.
- Sketched updates: These kinds of methods do compression on the generated update after training before communication.

For structured updates, two methods are provided in this paper:

- Low rank: Every update $\mathbf{H} \in \mathbb{R}^{d_1 \times d_2}$ to local model is enforced to be a low rank matrix whose rank is at most k by expressing $\mathbf{H}$ as a product of two special matrices:

$$\mathbf{H} = \mathbf{A} \cdot \mathbf{B},\ \mathbf{A} \in \mathbb{R}^{d_1 \times k} \text{ and } \mathbf{B} \in \mathbb{R}^{k \times d_2}$$

A is generated randomly independently in each round and clients only need to optimize **B**. What's more, in implementation, **A** can be generated through a random seed shared with server and thus only **B** needs to be sent to the server.

- Random mask: During training, updates are enforced to be a sparse matrix by abandoning some information using a random mask. Random masks are generated independently by the clients in each round. Only non-zeros entries of the update matrix are required to be uploaded if the random masks are known by the server. To further reduce the communication cost, the clients' random seeds can be shared with the server so that the random masks can be generated locally by both the server and the clients and no longer need to be transferred.

For sketched updates, three tools are introduced:

- Subsampling: This is quite similar to random mask except that it is done after the training step. Each client sends a random subset of the original computed update to the server. According to the paper, since subsampled updates will be averaged by the server, we can consider that average as an unbiased estimator of original updates' average. The explanation makes this method sounds more reliable.
- Probabilistic quantization: Its main idea is to transform weights in the update matrix into the n-bit form, where n a fixed number. For example, transforming a 4-byte weight into a 4-bit data provides 8× compression. The algorithm first gets the maximum weight and the minimum weight in the update matrix. Since all weights are in this interval, it divides this interval into 2^n smaller intervals. For each weight, the algorithm finds out which interval it belongs to and problematically changes it to one of the interval endpoints. According to our research, this method is popular in data compression.
- Improving the quantization by structured random rotations: Probabilistic quantization may lose a lot of information when the data has a special distribution. For example, when the interval is $[-1, 1]$ and most of the weights are 0, then 1-bit quantization will lead to a large error since all 0 is changed to either -1 or 1. According to [71], this problem can be solved by applying a random rotation on the matrix before the quantization. An inverse rotation is also needed when decoding the data.

In this paper, the above methods are combined and many experiments are done. While the accuracy of their original model on the CIFAR-10 dataset is around 0.9, they can reduce the uploaded data during communication by a half as well as obtain a modest accuracy, say 0.85. The techniques proposed are also efficient in LSTM next-word prediction. More detailed results can be found in the paper.

[9] is another related work. It is done by researchers from Carnegie Mellon University and Google. In general, it gives some lossy compression methods and Federated Dropout as solutions. Federated Dropout is an idea developed from Dropout[69]. With Federated Dropout, a worker trains an update to a smaller incomplete sub-model instead of training an update to the whole global model. Experiments show that Federated Dropout can reduce both the local computation cost and the communication cost.

5.4 Benchmark

In [10], researchers from Carnegie Mellon University cooperate with Google to purpose LEAF, which is an open-source benchmarking framework for federated settings. Three datasets suitable for federated learning are specially generated and already provided in LEAF. More datasets from different domains are planned to be supported in the future. Scripts for pre-processing, prototyping and testing are also available.

5.5 System Design

In [5], Google gives their system design towards federated learning at scale. Unlike the above-mentioned papers which mainly focus on introducing the theoretical techniques used by federated learning, this work actually gives many valuable suggestions on how to deploy the federated learning system in real life. Architectures of the server and the clients are shown in detail. Federated Averaging[46] still performs as the core algorithm. Synchronous training is used in this work.

5.6 Applications

Federated learning has already been used in some applications[1, 11, 26, 77] by Google. Most of them are about tasks for Gboard such as keyboard query suggestions and next word prediction. [1] uses federated learning to provide more accurate, useful searching results so that people can find what they are looking for faster. [77] applies federated learning to Gboard to improve its query suggestions. [26] trains a recurrent neural network language model using federated learning for the purpose of next-word prediction in a virtual keyboard for smartphones. [11] demonstrates that a character-level recurrent neural network is able to learn OOV(Out-Of-Vocabulary) words under federated learning settings.

5.7 Other Related Works

Besides Google, many other researchers have also done some work related to federated learning. In 2017, [66] combine federated learning with multi-task learning. [22] tries to add differential privacy methods to federated learning. In 2018, [82] analyzes the effect of the Non-IID setting on federated learning's performance. [4] tries to backdoor federated learning and gives the attack model. Its main purpose is to make the model generated by federated learning gives wrong answers or even attacker-chosen answers. [40] introduces secure FTL(Federated Transfer Learning) to improve statistical models under a data federation. It shows that federated learning is suitable for being applied to machine learning tasks in the scenario of a bank where sensitive information mustn't be shared. In 2019, [13] proposes a novel lossless privacy-preserving tree-boosting system known as SecureBoost in the setting of federated learning. [76] is a survey about federated learning's concept and applications. It is given by the team who has proposed FTL[40] in 2018. [61] proposes STC(Sparse Ternary Compression) which is robust to Non-IID data. It works as a substitute for Federated Averaging which may not always perform well under the Non-IID setting.

6 FUTURE DIRECTIONS

Here we will introduce several future research directions of mobile distributed machine learning. Some of them are motivated by the problems that remain unsolved for now. Others are ideas which may improve the efficiency or the robustness of the system.

- General Mobile Training Library: By June 2019, although the computation power of mobile devices is sufficient for training small machine learning models, existed open-source mobile machine learning libraries such as TensorFlow Lite only supports inference operations which focus on using a trained model to predict values. Without a general mobile training library, deploying machine learning tasks on mobile applications can be exhausting because developers have to realize the operations that will be used in the training process by themselves for each application, which is time-consuming and inefficient. However, we believe that this problem will be solved very soon because according to the 2019 roadmap of TensorFlow Lite, its support for on-device training will be added. Many other popular machine learning frameworks such as Caffe and PyTorch may also have been working on this.

- Non-IID Training Set: This problem was first introduced in federated learning. It also exists in the scenario of mobile distributed machine learning and is caused by the no-uploading property. [82] has analyzed the negative effect of non-IID datasets and proposed a simple data-sharing strategy to deal with it. However, it may be hard for mobile devices to share a small part of the local dataset with others because the process is hard to be managed and the communication cost can be high. We also have to ensure that the privacy is still preserved all the time. Another kind of possible solutions are to develop machine learning optimizers which are not sensitive to the distribution of training set. This may also be difficult to realize because it is a new task and does not have many works that can be referenced since all existing algorithms for traditional machine learning do not have to worry about non-IID training set at all as it can be easily solved by shuffling the dataset.
- Aggregation Methods: Although we have introduced several kinds of data aggregation methods in 4.3, we don't think any of them can perform so well that little improvement is able to be done. Since in traditional distributed machine learning, the number of workers is usually under one hundred, no one has the experience to aggregate tens of thousands of updates in a round in the scenario of mobile distributed machine learning. Moreover, the above-introduced methods all have disadvantages. The averaging-based methods may cause a decrease in the convergence speed because it can reduce the scale of updates on weights. They may also not be suitable for non-convex problems. The distillation-based methods may not be able to merge tens of thousands of models because it will need very much additional computation to handle this complex task. So how to develop an outstanding and efficient data aggregation method which can make the best use of the information contained in the updates is an urgent problem need to be solved.
- Mobile Edge Computing: We have already introduced mobile edge computing in Section 2, here we will show how to combine distributed machine learning with it. The local training process on mobile devices can be divided into two steps:

(1) Use samples and their corresponding labels to get the loss function which only contains the weights of the model as its variables.
(2) Calculate the partial derivatives of the loss function and use gradient descent to update the model.

We can see that the local data are only used in the first step. The second step is where complex computation actually takes place. If the scale of the loss function is too large, resources available on a single mobile device may not be sufficient for the computation process. With the help of MEC, mobile devices can upload the loss function to the MEC servers to get the problem solved there. Since we can use many samples to generate a complex sum loss function, it will be impossible to recover the original data from the uploaded loss function itself, which means this scheme guarantees the preservation for the user privacy at the same time. The disadvantage of this assistant training scheme is that while the training process totally done on mobile devices can be repeated multiple times using many different data samples to generate one update, one uploaded loss function only contains several selected samples. In other words, compared with the fact that a lot of data samples can contribute to the update in a single round using the original local training scheme, only limited information is available in one uploaded loss function if the assistant training scheme is used. However, since this assistant training process doesn't conflict with the original local one and they can be run parallel, it can at least be used to accelerate the learning.

7 CONCLUSION

In this survey paper, we have introduced the development of machine learning in recent years, from traditional machine learning to deep learning and followed by distributed machine learning and mobile learning. We have discussed their purposes and shown the necessity of mobile distributed machine learning in this information era. After that, the general properties of mobile distributed machine learning have been claimed, and a clear task definition for it has been given. To explain the architecture of mobile distributed machine learning, we have divided the whole learning process into three steps and listed possible solutions for them. Features of the solutions have been clearly compared and summarized in Table 1 and Table 2. When trying to apply mobile distributed machine learning to a particular task, users and developers can refer to this comparison as guidelines. Federated learning, as an example of mobile distributed machine learning whose main purpose is privacy preservation, has also been mentioned and its developing progress has been introduced. We have also discussed the future directions of mobile distributed machine learning research. In general, there still exists many challenges if we want to implement mobile distributed machine learning on popular real-life applications, which suggests that more efforts should be paid on the further improvement of this system, including aspects such as efficiency, privacy and robustness.

REFERENCES

[1] ai.google. 2018. Under the hood of the pixel 2: How ai is supercharging hardware. (2018). https://ai.google/stories/ai-in-hardware/

[2] Rohan Anil, Gabriel Pereyra, Alexandre Passos, Robert Ormandi, George E Dahl, and Geoffrey E Hinton. 2018. Large scale distributed neural network training through online distillation. *arXiv preprint arXiv:1804.03235* (2018).

[3] Haim Avron, Alex Druinsky, and Anshul Gupta. 2015. Revisiting asynchronous linear solvers: Provable convergence rate through randomization. *Journal of the ACM (JACM)* 62, 6 (2015), 51.

[4] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. 2018. How To Backdoor Federated Learning. *CoRR* abs/1807.00459 (2018). arXiv:1807.00459 http://arxiv.org/abs/1807.00459

[5] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konecny, Stefano Mazzocchi, H Brendan McMahan, et al. 2019. Towards Federated Learning at Scale: System Design. *arXiv preprint arXiv:1902.01046* (2019).

[6] Léon Bottou, Frank E Curtis, and Jorge Nocedal. 2018. Optimization methods for large-scale machine learning. *Siam Review* 60, 2 (2018), 223–311.

[7] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein, et al. 2011. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning* 3, 1 (2011), 1–122.

[8] Cristian BuciluǍ, Rich Caruana, and Alexandru Niculescu-Mizil. 2006. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 535–541.

[9] Sebastian Caldas, Jakub Konecný, H. Brendan McMahan, and Ameet Talwalkar. 2018. Expanding the Reach of Federated Learning by Reducing Client Resource Requirements. *CoRR* abs/1812.07210 (2018). arXiv:1812.07210 http://arxiv.org/abs/1812.07210

[10] Sebastian Caldas, Peter Wu, Tian Li, Jakub Konecný, H. Brendan McMahan, Virginia Smith, and Ameet Talwalkar. 2018. LEAF: A Benchmark for Federated Settings. *CoRR* abs/1812.01097 (2018). arXiv:1812.01097 http://arxiv.org/abs/1812.01097

[11] Mingqing Chen, Rajiv Mathews, Tom Ouyang, and Françoise Beaufays. 2019. Federated Learning Of Out-Of-Vocabulary Words. *CoRR* abs/1903.10635 (2019). arXiv:1903.10635 http://arxiv.org/abs/1903.10635

[12] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. 2015. Compressing neural networks with the hashing trick. In *International Conference on Machine Learning*. 2285–2294.

[13] Kewei Cheng, Tao Fan, Yilun Jin, Yang Liu, Tianjian Chen, and Qiang Yang. 2019. SecureBoost: A Lossless Federated Learning Framework. *CoRR* abs/1901.08755 (2019). arXiv:1901.08755 http://arxiv.org/abs/1901.08755

[14] Trishul M Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. 2014. Project Adam: Building an Efficient and Scalable Deep Learning Training System.. In *OSDI*, Vol. 14. 571–582.

[15] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Quoc V. Le, and Andrew Y. Ng. 2012. Large Scale Distributed Deep Networks. In *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Eds.). Curran

Associates, Inc., 1223–1231. http://papers.nips.cc/paper/4687-large-scale-distributed-deep-networks.pdf
[16] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. 2012. Large scale distributed deep networks. In *Advances in neural information processing systems*. 1223–1231.
[17] Arthur P Dempster, Nan M Laird, and Donald B Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)* 39, 1 (1977), 1–22.
[18] Misha Denil, Babak Shakibi, Laurent Dinh, Nando De Freitas, et al. 2013. Predicting parameters in deep learning. In *Advances in neural information processing systems*. 2148–2156.
[19] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. 2014. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in neural information processing systems*. 1269–1277.
[20] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research* 12, Jul (2011), 2121–2159.
[21] Pedro A Forero, Alfonso Cano, and Georgios B Giannakis. 2010. Consensus-based distributed support vector machines. *Journal of Machine Learning Research* 11, May (2010), 1663–1707.
[22] Robin C. Geyer, Tassilo Klein, and Moin Nabi. 2017. Differentially Private Federated Learning: A Client Level Perspective. *CoRR* abs/1712.07557 (2017). arXiv:1712.07557 http://arxiv.org/abs/1712.07557
[23] A. Ghoting, R. Krishnamurthy, E. Pednault, B. Reinwald, V. Sindhwani, S. Tatikonda, Y. Tian, and S. Vaithyanathan. 2011. SystemML: Declarative machine learning on MapReduce. In *2011 IEEE 27th International Conference on Data Engineering*. 231–242. https://doi.org/10.1109/ICDE.2011.5767930
[24] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. 2014. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115* (2014).
[25] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*. 1135–1143.
[26] Andrew Hard, Kanishka Rao, Rajiv Mathews, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. 2018. Federated Learning for Mobile Keyboard Prediction. *CoRR* abs/1811.03604 (2018). arXiv:1811.03604 http://arxiv.org/abs/1811.03604
[27] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).
[28] Martin Jaggi, Virginia Smith, Martin Takác, Jonathan Terhorst, Sanjay Krishnan, Thomas Hofmann, and Michael I Jordan. 2014. Communication-efficient distributed dual coordinate ascent. In *Advances in neural information processing systems*. 3068–3076.
[29] Rie Johnson and Tong Zhang. 2013. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in neural information processing systems*. 315–323.
[30] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
[31] Jakub Konecnỳ. 2017. Stochastic, Distributed and Federated Optimization for Machine Learning. *arXiv preprint arXiv:1707.01155* (2017).
[32] Jakub Konečnỳ, Brendan McMahan, and Daniel Ramage. 2015. Federated optimization: Distributed optimization beyond the datacenter. *arXiv preprint arXiv:1511.03575* (2015).
[33] Jakub Konecnỳ, H Brendan McMahan, Daniel Ramage, and Peter Richtárik. 2016. Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527* (2016).
[34] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. 2016. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492* (2016).
[35] Jakub Konecnỳ and Peter Richtárik. 2013. Semi-stochastic gradient descent methods. *arXiv preprint arXiv:1312.1666* (2013).
[36] Jelle Kooistra. 2018. *Global Mobile Market Report.* Technical Report. Newzoo.
[37] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436.
[38] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. 2014. Scaling Distributed Machine Learning with the Parameter Server. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. USENIX Association, Broomfield, CO, 583–598. https://www.usenix.org/conference/osdi14/technical-sessions/presentation/li_mu
[39] Xiangru Lian, Yijun Huang, Yuncheng Li, and Ji Liu. 2015. Asynchronous parallel stochastic gradient for nonconvex optimization. In *Advances in Neural Information Processing Systems*. 2737–2745.
[40] Yang Liu, Tianjian Chen, and Qiang Yang. 2018. Secure Federated Transfer Learning. *CoRR* abs/1812.03337 (2018). arXiv:1812.03337 http://arxiv.org/abs/1812.03337
[41] Stuart Lloyd. 1982. Least squares quantization in PCM. *IEEE transactions on information theory* 28, 2 (1982), 129–137.

[42] Chenxin Ma, Virginia Smith, Martin Jaggi, Michael I Jordan, Peter Richtárik, and Martin Takáč. 2015. Adding vs. averaging in distributed primal-dual optimization. *arXiv preprint arXiv:1502.03508* (2015).

[43] Chenxin Ma and Martin Takáč. 2016. Distributed inexact damped newton method: Data partitioning and load-balancing. *arXiv preprint arXiv:1603.05191* (2016).

[44] P. Mach and Z. Becvar. 2017. Mobile Edge Computing: A Survey on Architecture and Computation Offloading. *IEEE Communications Surveys Tutorials* 19, 3 (thirdquarter 2017), 1628–1656. https://doi.org/10.1109/COMST.2017.2682318

[45] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief. 2017. A Survey on Mobile Edge Computing: The Communication Perspective. *IEEE Communications Surveys Tutorials* 19, 4 (Fourthquarter 2017), 2322–2358. https://doi.org/10.1109/COMST.2017.2745201

[46] H Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, et al. 2016. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629* (2016).

[47] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, et al. 2016. Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research* 17, 1 (2016), 1235–1241.

[48] Milad Nasr, Reza Shokri, and Amir Houmansadr. 2018. Machine learning with membership privacy using adversarial regularization. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 634–646.

[49] Pitch Patarasuk and Xin Yuan. 2009. Bandwidth optimal all-reduce algorithms for clusters of workstations. *J. Parallel and Distrib. Comput.* 69, 2 (2009), 117–124.

[50] Diego Peteiro-Barral and Bertha Guijarro-Berdiñas. 2013. A survey of methods for distributed machine learning. *Progress in Artificial Intelligence* 2, 1 (2013), 1–11.

[51] Raluca Ada Popa. 2014. *Building practical systems that compute on encrypted data.* Ph.D. Dissertation. Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science.

[52] Raluca Ada Popa, Catherine Redfield, Nickolai Zeldovich, and Hari Balakrishnan. 2011. CryptDB: protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. ACM, 85–100.

[53] Raluca Ada Popa, Catherine Redfield, Nickolai Zeldovich, and Hari Balakrishnan. 2012. CryptDB: processing queries on an encrypted database. *Commun. ACM* 55, 9 (2012), 103–111.

[54] Ning Qian. 1999. On the momentum term in gradient descent learning algorithms. *Neural networks* 12, 1 (1999), 145–151.

[55] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. 2011. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in neural information processing systems*. 693–701.

[56] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. 2018. On the convergence of adam and beyond. (2018).

[57] Peter Richtárik and Martin Takáč. 2016. Distributed coordinate descent method for learning with big data. *The Journal of Machine Learning Research* 17, 1 (2016), 2657–2681.

[58] Roberto Rigamonti, Amos Sironi, Vincent Lepetit, and Pascal Fua. 2013. Learning separable filters. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2754–2761.

[59] Herbert Robbins and Sutton Monro. 1951. A stochastic approximation method. *The annals of mathematical statistics* (1951), 400–407.

[60] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. 2014. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550* (2014).

[61] Felix Sattler, Simon Wiedemann, Klaus-Robert Müller, and Wojciech Samek. 2019. Robust and Communication-Efficient Federated Learning from Non-IID Data. *CoRR* abs/1903.02891 (2019). arXiv:1903.02891 http://arxiv.org/abs/1903.02891

[62] Ohad Shamir, Nati Srebro, and Tong Zhang. 2014. Communication-efficient distributed optimization using an approximate newton-type method. In *International conference on machine learning*. 1000–1008.

[63] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. 2016. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal* 3, 5 (Oct 2016), 637–646. https://doi.org/10.1109/JIOT.2016.2579198

[64] Reza Shokri and Vitaly Shmatikov. 2015. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*. ACM, 1310–1321.

[65] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership inference attacks against machine learning models. In *Security and Privacy (SP), 2017 IEEE Symposium on*. IEEE, 3–18.

[66] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet S Talwalkar. 2017. Federated Multi-Task Learning. In *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, Inc., 4424–4434. http://papers.nips.cc/paper/7029-federated-multi-task-learning.pdf

[67] Virginia Smith, Simone Forte, Ma Chenxin, Martin Takáč, Michael I Jordan, and Martin Jaggi. 2018. Cocoa: A general framework for communication-efficient distributed optimization. *Journal of Machine Learning Research* 18 (2018), 230.

[68] Suvrit Sra, Adams Wei Yu, Mu Li, and Alexander J Smola. 2015. Adadelay: Delay adaptive distributed stochastic convex optimization. *arXiv preprint arXiv:1508.05003* (2015).

[69] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15, 1 (2014), 1929–1958.

[70] Shizhao Sun, Wei Chen, Jiang Bian, Xiaoguang Liu, and Tie-Yan Liu. 2017. Ensemble-compression: A new method for parallel training of deep neural networks. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 187–202.

[71] Ananda Theertha Suresh, Felix X Yu, Sanjiv Kumar, and H Brendan McMahan. 2017. Distributed mean estimation with limited communication. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 3329–3337.

[72] T Tieleman and Geoffrey Hinton. 2012. COURSERA: Neural networks for machine learning. *University of Toronto* (2012).

[73] Vladimir Vapnik. 2013. *The nature of statistical learning theory*. Springer science & business media.

[74] Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. 2017. The Marginal Value of Adaptive Gradient Methods in Machine Learning. In *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, Inc., 4148–4158. http://papers.nips.cc/paper/7003-the-marginal-value-of-adaptive-gradient-methods-in-machine-learning.pdf

[75] Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus Ng, Bing Liu, Philip S. Yu, Zhi-Hua Zhou, Michael Steinbach, David J. Hand, and Dan Steinberg. 2007. Top 10 Algorithms in Data Mining. *Knowl. Inf. Syst.* 14, 1 (Dec. 2007), 1–37. https://doi.org/10.1007/s10115-007-0114-2

[76] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. 2019. Federated Machine Learning: Concept and Applications. *CoRR* abs/1902.04885 (2019). arXiv:1902.04885 http://arxiv.org/abs/1902.04885

[77] Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. 2018. Applied Federated Learning: Improving Google Keyboard Query Suggestions. *CoRR* abs/1812.02903 (2018). arXiv:1812.02903 http://arxiv.org/abs/1812.02903

[78] Jin Yu, SVN Vishwanathan, Simon Günter, and Nicol N Schraudolph. 2010. A quasi-Newton approach to nonsmooth convex optimization problems in machine learning. *Journal of Machine Learning Research* 11, Mar (2010), 1145–1200.

[79] Xiao Zeng, Kai Cao, and Mi Zhang. 2017. MobileDeepPill: A Small-Footprint Mobile Deep Learning System for Recognizing Unconstrained Pill Images. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '17)*. ACM, New York, NY, USA, 56–67. https://doi.org/10.1145/3081333.3081336

[80] Sixin Zhang, Anna E Choromanska, and Yann LeCun. 2015. Deep learning with elastic averaging SGD. In *Advances in Neural Information Processing Systems*. 685–693.

[81] Yuchen Zhang and Xiao Lin. 2015. DiSCO: Distributed optimization for self-concordant empirical loss. In *International conference on machine learning*. 362–370.

[82] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. 2018. Federated Learning with Non-IID Data. *CoRR* abs/1806.00582 (2018). arXiv:1806.00582 http://arxiv.org/abs/1806.00582

[83] Shuxin Zheng, Qi Meng, Taifeng Wang, Wei Chen, Nenghai Yu, Zhi-Ming Ma, and Tie-Yan Liu. 2017. Asynchronous stochastic gradient descent with delay compensation. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 4120–4129.