Data Analytics for Fog Computing

Qian, Jia

Publication date:
2022

Document Version
Publisher's PDF, also known as Version of record

Ph.D. Thesis
Doctor of Philosophy

DTU Compute
Department of Applied Mathematics and Computer Science

Data Analytics for Fog Computing

Jia Qian

Kongens Lyngby 2021

DTU Compute
Department of Applied Mathematics and Computer Science
Technical University of Denmark

Richard Petersen Plads
Building 321
2800 Kongens Lyngby, Denmark
compute@compute.dtu.dk
www.compute.dtu.dk

English Summary

Fog Computing (FC) is a rising field that addresses the primary drawbacks of Cloud Computing (CC), such as: privacy issues, latent response and bandwidth expense. Fog Nodes (FNs) encapsulate the computation, storage, and networking services within one platform. FNs reside in the Fog Layer, a middleware that mediates between Cloud and Edge devices, close to the edge side such that the computational tasks are pushed towards edge devices.

In this thesis, we propose to use Federated Learning (FL) as the Machine Learning (ML) framework for the distributed data analytics scenario such that there is no requirement of data transmission. We demonstrate the feasibility of applying Active Learning (AL) under the federated framework to reduce the expense of manual labeling.

We investigate the data heterogeneity problem in FL, which arises due to the unbalanced data distribution across distributed devices or the biased data sampling schemes, e.g., active sampler. This has significant theoretical and practical meaning, as in real life the data is often Non Independent and Identically Distributed (non-i.i.d.). This challenges the model training process, requiring careful consideration of the learning rate and the aggregation strategies to be applied.

FL leaves the majority of the computational workload to the edge devices leaving only simple arithmetic operations on the server. We suggest a new framework that is more flexible to the variable allocation of computational resources, and applicable to the scenarios where the computational resource of devices is insufficient to implement the task.

Lastly, we analyze the minimal requirements to perform an input reconstruction attack. Recent work shows the possibility of reconstructing the inputs with the knowledge of gradients and model parameters. We quantify the lower bound condition to accomplish a full reconstruction using Multilayer Perceptron (MLP) and Convolutional Neural Network (CNN). This supports FL practitioners in securely designing network architecture, depth, and batch size.

Danish Summary

Fog computing (FC) er et voksende forskningsfelt, der adresserer de primære ulemper ved Cloud computing (CC), såsom: privatlivs-ufordringer og udgifter til kommunikation. En såkaldt Fog Node (FN) indkapsler beregnings-, lagrings- og netværkstjenester inden for én platform. Og det lag, hvor FN residerer, kaldes Fog Layer, og det forbinder Cloud- og Edge-enheder, men så tæt på kantsiden, at beregningsopgaverne skubbes mod kantenhederne.

I denne afhandling har vi foreslået at bruge FL som machine learning ramme for et distribueret dataanalyse-scenario, således at der ikke er noget krav om datatransmission. Dernæst har vi undersøgt muligheden for at anvende såkaldt Active learning (AL) i en federated learning-ramme, og derved potentielt begrænse udgifterne til manuel labeling. Vi undersøgte også data-heterogenitetsproblemet i FL, som kan opstå på grund af en ubalanceret datafordeling på tværs af de distribuerede enheder. Dette har betydelig teoretisk og praktisk betydning, da dataene i praktiske anvendelser ofte ikke er identisk og uafhængit fordels. Dette udfordrer ML-processen og kræver grundig overvejelse af læringshastigheden og de aggregeringsstrategier, der skal anvendes.

FL overlader størstedelen af den beregningsmæssige arbejdsbyrde til kantenhederne og udfører kun begrænsede mængder af aritmetiske operationer på serveren. Vi foreslår en ny ramme, der er mere fleksibel i forhold til beregningsbudget, og den er relevant for applikationer og scenarier, kant-enhederne har få beregningsressourcer.

Endelig bidrager vi med analysen af minimale krav til input-rekonstruktionen ud fra gradienter. De tidligere arbejder viser muligheden for at rekonstruere input med kendskab til gradienter og modelparametre. Vi tager et skridt videre for at udlede en nedre grænse for hele rekonstruktionerne ved i både MLP og CNN neurale netværk. Det kan hjælpe FL-praktikere, når man skal designe netværksarkitekturen, f.eks. bestemme dybden af netværk og batchstørrelser.

Preface

This Ph.D. thesis was prepared at the Cognitive Systems Section at the Technical University of Denmark to partially fulfill the requirements for acquiring a Ph.D. degree in computer science.

This thesis consists of a summary report and a collection of four papers: one conference paper, one journal paper, and two papers under review. The Ph.D. project was carried out at DTU during the period of July 2018 - September 2021 except for a four-month external stay at TU Kaiserslautern under Marius Kloft.

The work presented in this thesis was funded by a European Training Network (ETN)-Fog Computing for Robotics and Industrial Automation (FORA). It was supervised by Professor Lars Kai Hansen and co-supervised by Associate Professor Xenofon Fafoutis.

Kongens Lyngby, September 30, 2021

Jia Qian

Acknowledgments

First and foremost I would like to express my sincere gratitude to my supervisor Lars Kai Hansen for the continuous support and encouragement throughout my phd study. His guidance helped me in all the time of research and writing of this thesis. I also would like to thank my co-supervisor Xenofon Fafoutis for the inspiring research discussion. A special thanks to Jan Larsen who opened up the research door to me.

I feel grateful to work in a stimulating environment where I had the enlightening conversations with my colleagues Max, Mathias, Sayantan and Cong. I would like to extend my sincere thanks to them for the insightful comments and suggestions.

I would like to thank the European Unions Horizon 2020 research and innovation program under the Marie Sklodowska-Curie grant agreement No. 764785, FORA-Fog Computingfor Robotics and Industrial Automation for funding this research.

Finally, I would like to thank my families (mum and dad) for the unconditional love and support. I know you are always there for me. Also, thanks to Simon for your understanding, being so supportive as well as happy distractions to rest my mind outside of my research.

List of Publications

All publications are peer-reviewed except the works with * are under review.

Included in Thesis

A Jia Qian, Sarada Prasad Gochhayat, and Lars Kai Hansen. Distributed active learning strategies on edge computing. In *2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*, pages 221–226. IEEE, 2019

B Jia Qian, Lars Kai Hansen, Xenofon Fafoutis, Prayag Tiwari, and Hari Mohan Pandey. Robustness analytics to data heterogeneity in edge computing. *Computer Communications*, 164:229–239, 2020

C *Jia Qian and Mohammadreza Barzegaran. A decomposed deep training solution for fog computing platforms (submitted to tec2021 workshop). 2021

D *Jia Qian, Hiba Nassar, and Lars Kai Hansen. Minimal conditions analysis of gradient-based reconstruction in federated learning (submitted to neurips workshop). *arXiv preprint arXiv:2010.15718*, 2020

Not Included in Thesis

E Jia Qian, Prayag Tiwari, Sarada Prasad Gochhayat, and Hari Mohan Pandey. A noble double-dictionary-based ecg compression technique for ioth. *IEEE Internet of Things Journal*, 7(10):10160–10170, 2020

F Mohammadreza Barzegaran, Nitin Desai, Jia Qian, Koen Tange, Bahram Zarrin, Paul Pop, and Juha Kuusela. Fogification of electric drives: An industrial use case. In *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 77–84. IEEE, 2020

G Mohammadreza Barzegaran, Nitin Desai, Jia Qian, and Paul Pop. Electric drives as fog nodes in a fog computing-based industrial use case. *The Journal of Engineering*, 2021

H Prayag Tiwari, Jia Qian, Qiuchi Li, Benyou Wang, Deepak Gupta, Ashish Khanna, Joel JPC Rodrigues, and Victor Hugo C de Albuquerque. Detection of subtype blood cells using deep learning. *Cognitive Systems Research*, 52: 1036–1044, 2018

Contents

List of Abbreviations

ADL Active Deep Learning

AL Active Learning

BALD Bayesian Active Learning by Disagreement

CC Cloud Computing

CNN Convolutional Neural Network

DL Deep Learning

DP Differential Privacy

EC Edge Computing

ELBO Evidence Lower Bound

FC Fog Computing

FCDDT Fog Computing-based Decomposed Deep Training

FCP Fog Computing Platform

FGSM Fast Gradient Sign Method

FL Federated Learning

FN Fog Node

GAN Generative Adversarial Network

GDPR General Data Protection Regulation

i.i.d. Independent and Identically Distributed

KL divergence Kullback–Leibler divergence

L2GD Loopless Local GD

LSTM Long Short Term Memory

MCMC Markov Chain Monte Carlo

ME Maximal Entropy

ML Machine Learning

MLP Multilayer Perceptron

MSE Mean Square Error

non-i.i.d. Non Independent and Identically Distributed

PCA Principle Component Analysis

PGD Projected Gradient Descent

RNN Recurrent Neural Network

SGD Stochastic Gradient Descent

SSL Semi-supervised Learning

VI Variational Inference

CHAPTER 1

Introduction

This chapter will briefly present the potential impacts and challenges of Big Data and then move to the generic motivation of the advent of Fog Computing. In the end, we will propose Federated Learning as the ML framework to finally enable the intelligence on edges.

1.1 Fog Computing

Nowadays, Big Data is largely changing and influencing the way we live and make decisions in our daily lives. It also has significant impact on business, ranging from economics, pharmaceutical, and law to the areas where data analytics enable the auto-diagnose [8, 47, 75]. In most cases, the realization occurs through Cloud Computing, where a centralized server is equipped with rich computational resources and different communication solutions. In the past decade, companies like Google (GCP), Microsoft (Azure), Amazon (AWS) have built giant data processing platforms to analyze data and respond to queries from customers. They offered different services for customers from various domains to develop their data processing center, for instance, storage, data analytics, networking, etc.

However, the growing awareness of privacy concerns brought more and more challenges to the conventional CC. For instance, the release of General Data Protection Regulation (GDPR) [95] and the new function of iOS [1], users deciding if "allowing App to track activity across other companies' apps and websites", which largely tighten up the full potentials of Big Data. In addition, the exponential growth of (IoT) devices introduces further hassle since it is almost impossible to connect all of them to the centralized server due to the tremendous workload. Moreover, some applications are critically sensitive to latency caused by distant transmission from server to edge side. To this end, we urgently need a new diagram to address the issues mentioned above.

An architectural means to solve such issues is FC, defined as a "system-level architecture that distributes resources and services of computing, storage, control and networking anywhere along the continuum from Cloud to Things" [21]. A closely related term is Edge Computing (EC) [84] that emphasizes the computation more than FC, which is more focused on infrastructure. We will interchangeably use them in the following context. The vicinity property of EC outperforms CC with lower

latency and bandwidth [100]. As shown in Figure 1.1, FNs are placed at the edge of the network, in close proximity to data sources. FC is literally pushing computation, data storage, data analytics, and service to the edge side, away from Cloud. For further insights, FC may stimulate and lead the business revolution. For instance, the achievement of healthcare occurs through wearable devices, which offer remote diagnosis and patients monitor [74]. The realization of Industrial 4.0 relies on the convergence of operational and informational technologies through the implementation of FC [68]. In the financial sector, they [99] demonstrate a prototype of blockchain mining using a mobile edge computing network. Vehicular edge computing is designed to mainly address the safety issue for the autonomous system [57]. Researchers [12, 69] and industry [64, 94] have developed several types of FNs for applications in a range from powerful high-end FNs to low-end FNs with limited resources.

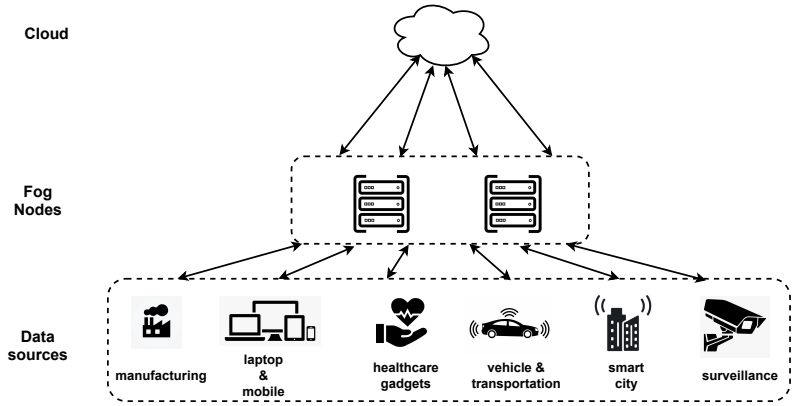


Figure 1.1: Edge/Fog computing diagram. The figure is from appendix C.

EC enables Edge intelligence, moving the AI frontier to the edge of the network to completely unleash the power of Big Data [105]. Edge intelligence includes the process of collecting data, performing analysis, and optionally sharing the metadata with the server with the evolution of GPUs. To this end, a favorable ML framework FL meets the requirements. It [45] was firstly proposed by Google where they presented a solution for mobile applications, jointly training a global model that may capture information of the whole environment by exchanging gradients between distributed devices and the server. Another similar idea is called parameter server [52], but with the focus on the architecture and system design. In our case, the distributed workers (devices) could be FNs or edge devices (data producers), and the server could be a server on the Cloud or a FN, depending on the specific application.

1.2 Data Analytics on Fog Computing

FL [45] belongs to the distributed ML category, and it is a framework where we have a server and a set of distributed devices collaboratively training a global model with the knowledge of the combined environment. The set of devices possess their personalized data and one copy of model parameters, and based on which they update the gradients and share it with the server. The server has no direct access to the training data rather than the gradients and model parameters information, which vastly decreases the chance of sensitive information disclosure, but it is not sufficient, and we will discuss it later. Moreover, the transmission of gradients or model parameters could save upload bandwidth with the compression technique and appropriate model complexity comparing with the transmission of training data in CC [33].

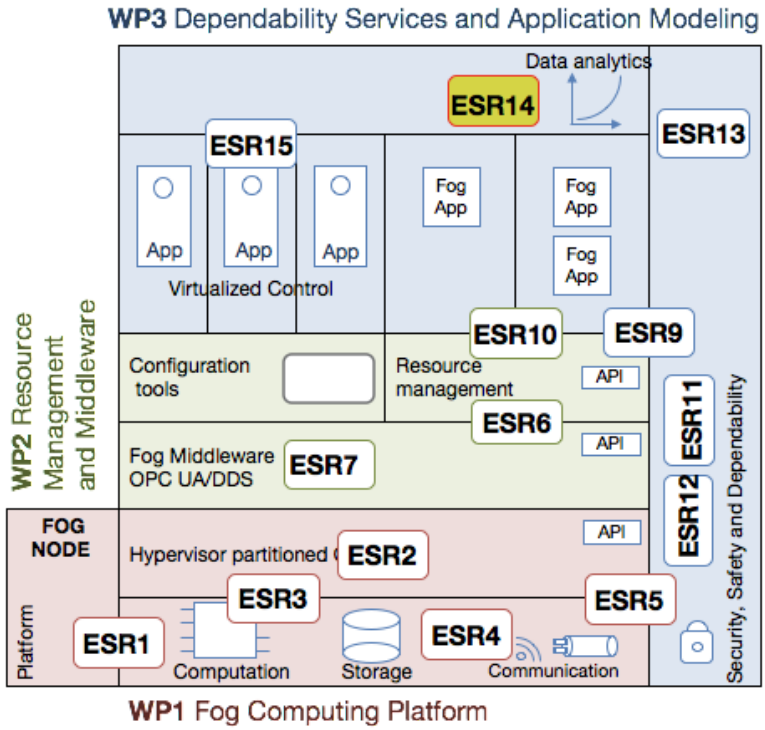


Figure 1.2: Overview of FORA project (sourced from FORA website).

1.3 Scope of Thesis

The project FORA—Fog Computing for Robotics and Industrial Automation is a European Training Network (ETN), which funds and trains 15 Ph.D. candidates in the area of Fog Computing as shown in Figure 1.2. My research mainly focuses on the data analytics part (ESR14), applying ML on FC. We suggest FL as the possible solution and study a few potential questions under the federated diagram. This research is mainly focused on ML side considering Fog Computing Platform (FCP) as a black box assuming FN design including the hypervisor, communication, computation and storage, task scheduling, and resource management have been carried out and implemented inside the box. The relevant research works correspond to the other ESRs shown in Figure 1.2.

1.4 Thesis Structure

In Chapter 2, we will introduce the related concepts that will appear in the research and in Chapter 3 we will show our contributions and wrap the work in Chapter 4.

CHAPTER 2

Background

This chapter will lay out the background knowledge for the rest of the thesis, covering some concepts and definitions appearing in the contributions or ideas relevant to the contributions.

2.1 Deep Learning

Deep Learning (DL) [48] has gained significant success and beat the state-of-the-art in many fields such as object detection, speech recognition, medical diagnosis [92], and many other domains, e.g. genomics [60]. The advent of neural networks avoids the non-trivial feature extraction work, which was preliminarily required in conventional ML methods. Instead, the model takes raw data as the input. The model automatically carries out the feature extraction work in the hierarchical way, which significantly declines the requirement of high-quality features extraction, particularly for high-dimensional input as images and signal data. There are several fundamental neural network architectures such as Multilayer Perceptron (MLP), Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), and Long Short Term Memory (LSTM).

2.1.1 Network Architecture

Feed-forward neural network, also known as MLP, is the stack of linear and non-linear functions, often combining in the interleaving fashion. Generally, it consists of one input layer, one output layer, and all the rest layers between input and output layers are called hidden layers. A neural network with L hidden layer is expressed as

$$f_L = W_1 \circ \sigma \circ W_2 \cdots \circ \sigma \circ W_L \tag{2.1}$$

where σ is the non-linear function and W_i is the weights of layer i (linear function).

Sigmoid, Hyperbolic Tangent Function, ReLU and their variations are commonly used non-linear function (a.k.a activation function). A three-layer perceptron (one hidden layer) can approximate any continuous function if the infinite number of units is allowed in the hidden layer [39][22]. The richness of representation brings more

and more interest to researchers, particularly when hardware advances rapidly. One illustration is shown in Figure 2.1.

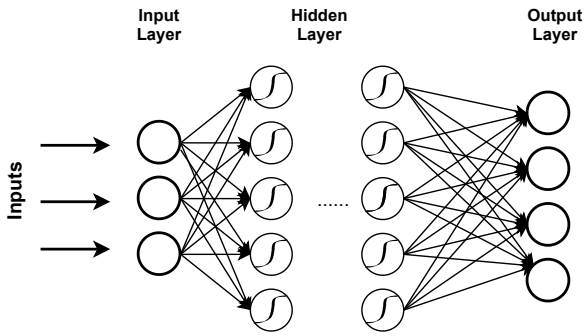


Figure 2.1: Feed-forward Neural Network.

CNN largely reduces the number of model parameters compared with MLP. Instead, one unit only connects to the receptive field where a convolution kernel is applied rather than connecting with all units (nodes) in the previous layer. The convolution kernel is shared in the convolutional layer, which made CNN shift-invariant, i.e. certain features can be detected regardless of their location in the input image. Illustratively it is shown in Figure 2.2. Say, we have the square input image $x \in \mathbb{R}^{C \times d \times d}$, with C channels, width (length) equal to d. We define the convolution operator as:

$$z_{mij} = (\sum_{c=1}^{C} \sum_{g=1}^{k} \sum_{n=1}^{k} l_{mcgn} x_{c,si+g-1,sj+n-1}) + r_m \\ \forall (i,j) \in [1, d^{'}] \times [1, d^{'}], \forall m \in [1, h] \tag{2.2}$$

where l_m represents filter (kernel) m with C channels and width k, whereas r_m is the corresponding bias. $d^{'}$ is the width of convolutional output, s is the stride and h is the number of kernels.

The pooling layer is optionally added after the convolution layer, and its main focus is to reduce further the number of parameters and computation in the network. The most common pooling technique is Max pooling and Average pooling. Literally, it operates as choosing either the maximal value or average value in a particular area since neighboring pixel values are often highly correlated.

Other architectures, like RNN [77][85] is designed for sequential data, as the generalization of feed-forward neural network with internal memory, but it suffers from gradients vanishing and exploding. An improved version LSTM [37] is invented by explicitly introducing the memory units. For the completeness, we shortly mention them, and for the discussion with depth, we refer to [101].

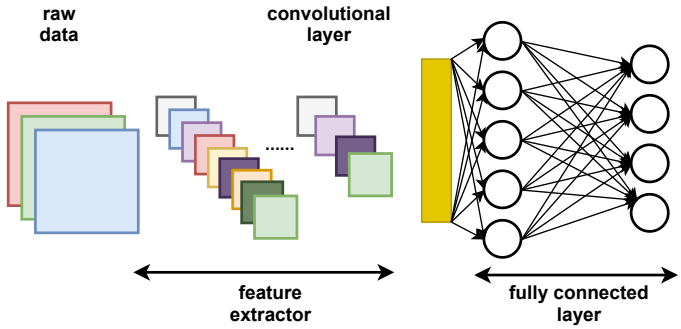


Figure 2.2: Convolutional Neural Network.

2.1.2 Error Propagation

Model training aims to find the model parameters to fit the data perfectly, and hopefully not only the training data but also generalize to the unseen data by minimizing the cost function.

For the regression task, often we use the Mean Square Error (MSE), or negative log-likelihood for the stochastic models. Instead, we use cross-entropy as the cost function for multi-classification after applying the softmax function, which normalizes the output between zero and one. The cost function with respect to the neural network's weights is neither convex nor concave; thus, there are multiple local minimums or maximums in the landscape. Furthermore, for a given local minimum, there will be multiple points that have the same values in the parameter space [9]. Empirically, even a local minimum may be sufficient to provide a good solution. Different from the pure optimization problem, which converges to the point whose gradient is small or zero, the training process in the neural network sometimes stops at the point whose gradient is big, and we refer to it as *early stopping* [30].

It is impossible to derive the analytical form with respect to all the model parameters to have $\nabla \ell = 0$. Generally, gradient-based iterative optimization is applied to address it. We define the iterative update as $w^{t+1} = w^t - \eta \times p^t$, where η is the learning rate (called step length in optimization) and $-p^t$ is the direction to move downhill in the current iteration, which can be computed by the first-order methods, second-order methods, or quasi second-order methods.

We define the gradient descent (a first-order method) as

$$w^{t+1} = w^t - \eta \times \frac{1}{N} \sum_{i=1}^{N} \nabla_w \ell(f(x_i; w^t), y_i) \tag{2.3}$$

where N is the size of training data, $f(\cdot; w)$ is the neural network parameterized by w and $\ell()$ is the cost function taking prediction and ground truth as the inputs. When

N is large, it is expensive to compute the expected gradient over all the instances. Instead, we can apply Stochastic Gradient Descent (SGD) for a batch size $B \ll N$, defined as

$$w^{t+1} = w^t - \eta \times \frac{1}{B} \sum_{i=1}^{B} \nabla_w \ell(f(x_i; w^t), y_i). \tag{2.4}$$

For the second-order methods, we compute the direction by $p^t = H^{-1}\Delta w^t$ where H is the Hessian matrix, whereas, for the first-order method, H is the identity matrix. Hessian matrix contains the curvature information of the function, but it is not commonly used in the neural network since it is computationally infeasible for deep neural networks containing millions of parameters.

Both learning rate and direction play an important role in navigating to the local optimal area. Another category of methods aim to use adaptive learning rate to improve first-order methods, like Adagrad [18], RMSProp [36], Adam [43]. For more information, we recommend the readers to the survey work [89].

2.2 Federated Learning

FL is a decentralized ML framework that aims to train a global model by periodically collecting gradients or model parameter updates from distributed devices (workers). It fits the scenario where the data is produced by multiple sources that are geographically distributed. FL helps to reduce the risk of sensitive information leakage caused by data transmission and may save bandwidth with the aid of compression techniques [45].

Say we have n active workers in each round and the local cost functions of the workers are defined as $\hat{\ell}_i \; \forall 1 \leq i \leq n$. Data owned by all workers are assumed to be generated from their underlying data distribution $\mathcal{D}_i$, and the empirical distribution is denoted as $\hat{\mathcal{D}}_i$. We define the global cost function $\hat{L}$ as the weighted combination of local cost functions:

$$\hat{L} = p_i \sum_{i=1}^{n} \hat{\ell}_i(w) \tag{2.5}$$

where $\hat{\ell}_i$ is

$$\hat{\ell}_i(w) = \mathbb{E}_{(x,y)\sim\hat{\mathcal{D}}_i}[\ell_i(f_w(x), y)]. \tag{2.6}$$

The joint training aims to find w^* by solving the optimization problem of the global empirical loss function, which can be seen through the lens of distributed optimization. p_i is equal to m_i/m where m_i is defined as the amount of data on device i, and m is total amount of data $m = \sum_{i=1}^{n} m_i$. If $m_i = m_j \forall i, j$ and each $\hat{\mathcal{D}}_i$ is a sub-distribution sampled from the same empirical distribution $\hat{\mathcal{D}}$, then $p_i = 1/n$. Often we name it as *AveFL*.

We show *AveFL* in Algorithm 1. The whole algorithm repeats T iterations and each iteration is composed of the actions from the server and the distributed workers. Each worker implements the local training and the server carries out the aggregation step [29]. More specifically, after each worker complete the local training (Algorithm 1, Line 5), they share the corresponding gradients with the server (Algorithm 1 Line 7). Given the criterion, the server aggregates the gradients (Algorithm 1 Line 10) and updates the global model. Finally, the server sends the updated global model back to the workers for the next round, which completes one iteration. Such iteration can repeat multiple times till it meets the goal.

2.2.1 Challenges in Federated Learning

Besides these advantages mentioned before, FL also has several core challenges associated with the distributed optimization (equation 2.5) [54]. We will briefly present them below.

Expensive communication. Frequent communication is required periodically to collect gradient information from workers to optimize the global objective function.

Algorithm 1 Average Federated Learning [73]

Initialization: w^0
for t=1,...T **do**
  =>**workers:**
  for j=1,2,..,n **do** n devices (in Parallel)
    1) $v_j^t = \nabla \hat{\ell}_j(f(X_j^t; w^t), Y_j^t)$
    with $(X_j^t = \{x_{jk}^t\}_{k=1,..B}, Y_j^t = \{y_{jk}^t\}_{k=1,...,B}) \sim \hat{\mathcal{D}}$
    2) share v_j^t with server
  end for
  =>**server:**
  $w^{t+1} = w^t - \eta \times \frac{1}{n} \sum_{j=1}^{n} v_j^t$
  share w^{t+1} with devices for next round
end for

To reduce communication expense (bandwidth and energy), we can either reduce the communication frequencies or compress the information transmitted in each round [41, 96]. To minimize communication frequency, often local SGD or local mini-batch SGD is applied, i.e. workers perform a sequential training before communication with the server. However, the large number of local training iterations or big batch size will probably lead to the divergence of aggregation or at least slow down the convergence rate. In other words, the performance of some devices degraded after aggregation.

Systems heterogeneity. The distributed devices might be equipped with variant computation, communication, and storage capabilities, which induces the constraints in a federated network. It introduces the scenario where a subset of devices take a much longer time to complete the local computation (a.k.a straggler) such that the other devices have to wait for them. In some cases, the active participants even drop out or remain inactive due to the limited resources. Thus, it requires the system to be able to schedule according to the resource distribution on devices, be robust to straggler mitigation, and be somewhat tolerant to the fault [53, 87].

Data heterogeneity. In practice, the locally generated data across devices is likely to be statistically different, which violates the Independent and Identically Distributed (i.i.d.) assumption. It increases the difficulty of problems formulating, model training, analysis, and evaluation. In particular, the optimization problem becomes more complex when minimizing the cost functions due to the considerable variance of gradients among workers.

Data privacy and security. FL shares the intermediate data with server or neighbor devices, e.g. model parameters or gradients. It largely decreases the risk of direct user data disclosure comparing with traditional Cloud Computing, where raw data is

shared. However, the sensitive information is still possible to reveal in other ways, i.e. membership attack [63], adversary attack [32] or input data reconstruction [106]. Thus, a secure-ensured mechanism is required when performing FL.

In addition to those aforementioned challenges, we are also interested in the feasibility of employing the agent like AL or Semi-supervised Learning (SSL) under a federated framework. They are important in the conventional centralized ML framework due to the practical consideration - expensive labeling. In Chapter 2.3 and 2.4 we will discuss data heterogeneity and data privacy, and in Chapter 2.5 we will introduce AL applied on neural network.

2.3 Data Heterogeneity on Federated Learning

AveFl assumes data owned by distributed workers are uniformly sampled from the same underlying distribution. However, such an ideal assumption may not be realistic in many contexts. In reality, data generated from workers are heterogeneous. Also, the biased sampling scheme, such as active sampler, may lead to highly biased sampling even though the dataset per sè is i.i.d.

Variance reduction has been widely studied in distributed optimization field. Mostly, they try to gradually reduce the learning rate or introduce a carefully designed gradient estimator to mitigate the big variance [16, 49, 50, 65, 76]. To reduce the communication frequency for FL, local SGD is often applied, i.e. the workers implement a sequence of local updates between communications with the server. I roughly categorize them into several groups: 1) empirical methods, 2) variance reduction methods, 3) learning rate decay, 4) seeking optimal combination.

Notations. Let's first formulate this problem. Given n workers, we define the local cost function for each worker i as f_i. We assume a subset of workers $\mathcal{S}$ with size K participate in each round if the straggler's effect exists; otherwise, all n workers participate. The global cost function F is the combination of local cost functions. If the server's learning rate is distinct from the local learning rate, we define η_g and η_l respectively, otherwise η.

Empirical methods. It empirically found the data heterogeneity issue and addressed it by sharing a small balanced subset that captures the full statistical characters among all the participants [104]. Locally, the subset is combined with the non-i.i.d. data in each batch to alleviate the statistical heterogeneity in the learning procedure. The subset size is relevant to the batch size as the subset should be sufficiently significant to generate the average gradient of each batch that has less variance. It requires no heavy computation; however, the behavior of sharing subset across workers subtly breaches privacy. Another work [29] groups edge devices into clusters according to the similarity of their data, which is similar to a clustering algorithm that alternatively optimize the cluster membership and optimize the cluster-based models.

Variance reduction. They [53] introduce *FedProx*, which includes an Euclidean regularization term in local cost functions:

$$h_k(w_k^t) = f_k(w_k^t) + \frac{\mu}{2} \left\| w_k^t - w^t \right\|^2 \tag{2.7}$$

to pull the distance between current local update w_k^t and global parameters w^t in the previous round to avoid the big variance among workers. Different from *AveFL*, they iteratively update model parameters according to $w_k^{t+1} = w^t - \eta(\nabla f_k(w_k^t) + \mu(w_k^t - w^t))$. They introduce an additive term $\eta\mu(w_k^t - w^t)$ where the hyperparameter μ may control the convexity of function h_k. Namely, if μ is chosen accordingly, Hessian matrix of

function h_k could be positive (semi-) definite. It is the first theoretical paper to analyze convergence rate under non-i.i.d. assumption in FL. They define an inexact solution γ_t^k to quantify the distance between current gradient and the optimal solution such that we may know how much local computation (on device k at round t) is required to converge to the preset inexact value. Another work [34] proposes the same cost function, but they address it by a randomized optimizer Loopless Local GD (L2GD) that enables the unbiased estimator and limit the step size. They update model according to $w^{t+1} = w^t + \eta G(w)$ where $G(w)$ is

$$G(w) = \begin{cases} \frac{\nabla_w F(w)}{1-p} & \text{with probability 1-p} \\ \frac{\mu(w-w^t)}{p} & \text{with probability p.} \end{cases} \tag{2.8}$$

The global model is defined as $F(w) := \frac{1}{n}\sum_{i=1}^{n} f_i(w_i)$. Thus, with probability $1-p$ they update it as $w_i^{t+1} = w_i^t - \frac{\eta}{n(1-p)}\nabla f_i(w_i^t)$ and with probability p they update it as $w_i^{t+1} = w_i^t - \frac{\eta\mu w_i^t}{np} + \frac{\eta\mu w^t}{np}$.

A follow-up work [42] defines a control variate to mitigate the drift. The trick of introducing a new correlated variable enables an unbiased estimator and at the same time reducing the variance, i.e. to estimate variance of variable X, another correlated variable Z is introduced such that $\mathrm{Var}(X - Z + \mathbb{E}[Z]) = \mathrm{Var}(X) + \mathrm{Var}(Z) - 2\mathrm{Cov}(X, Z) \leq \mathrm{Var}(X)$ if $\mathrm{Var}(Z) - 2\mathrm{Cov}(X, Z) \leq 0$. c_i is defined as the local control variable of device i and c as the global control variable. The local model update is $w_i^{t+1} = w_i^t - \eta(\nabla_w f_i(w^t) + c - c_i)$ where the distributed workers and server both carry out two steps in each round. For the workers: 1) local updates of client model parameters, 2) local updates of client control variate, which can be considered as a corrector. We define the local updates as

$$\text{workers updates}: \begin{cases} w_i^{t+1} = w^t - \eta_l \nabla f_i(w^t) \\ c_i^{t+1} = \nabla_w f_i(w) \text{ or } c_i^{t+1} = c_i^t - c + \dfrac{1}{n\eta_l}(w^t - w) \end{cases} \tag{2.9}$$

where η_l is the learning rate for workers, w^t is the global parameters updated at time t and w_i^{t+1} is the model parameters of worker i at time $t+1$. n is the number of workers in each round.

For the server: 1) aggregate the model update, 2) aggregate control variate. Note that we distinguish the learning rate of workers η_l from server η_g for flexibility. We define the server updates as

$$\text{server updates}: \begin{cases} w^{t+1} = w^t + \dfrac{\eta_g}{n}\displaystyle\sum_{i=1}^{n}(w_i^t - w^t), \\ c^{t+1} = c^t + \dfrac{1}{n}\displaystyle\sum_{i=1}^{n}(c_i^{t+1} - c_i^t). \end{cases} \tag{2.10}$$

Learning rate decay. They specifically study the local SGD on non-i.i.d., i.e. each local worker carries out multiple sequential update (rather than one) [55]. The authors propose to use a small learning rate such that the drift effect among workers introduced by multiple local updates will be reduced by the multiplication of a small learning rate. They show the convergence is bounded as $\mathcal{O}(T)$ for strongly convex and smooth function, and it will not converge to the optimal solution if the learning rate is fixed.

Optimal combination. They suggest considering the data heterogeneity as the multi-task problem [87], i.e. optimizing the local functions as solving the tasks. In addition to local cost functions, they include another term that quantifies the similarity between tasks. For instance, a bi-convex formula is defined as:

$$R(w, \Omega) = \lambda_1 \mathrm{tr}(w \Omega w^\intercal) + \lambda_2 \left\| w \right\|_F^2 . \tag{2.11}$$

One more variable Ω is required to optimize besides model parameters w, and the second term is the Frobenius norm. λ_1 and λ_2 decide the importance of these two parts. They address the problem by solving the conjugate dual function that enables the separation of global cost function into sub-problems solving on local devices. It actually studies the best combination of gradients across the workers with a regularizer. Another work [62] generalizes it and proposes a so-called agnostic FL framework that can learn any objective distribution as the mixture of multiple local loss functions. Moreover, they present data-dependent Rademacher complexity to ensure learning this objective. They aim to optimize a new cost function defined as

$$\min_{h \in \mathcal{H}} \max_{\lambda \in \mathrm{conv}(\Lambda)} \mathcal{L}_{\overline{\mathcal{D}}_\lambda}(h) + \gamma \left\| h \right\| - \mu \mathcal{X}^2(\lambda || \overline{m}) \tag{2.12}$$

where $\overline{m} = (\frac{m_1}{m}, \frac{m_2}{m}, .., \frac{m_n}{m})$, $m = \sum_i^n m_i$ and $\mathcal{X}$ defines the chi-squared divergence between two distributions $\mathcal{X}(p||q) = \sum_{i=1}^n \frac{(p_i - q_i)^2}{q_i}$. The first part is loss function w.r.t model h, and the second term controls the complexity of function h. The last term pulls λ (defined as a vector with length equal to number of classes C and $\sum_i^C \lambda_i = 1$) close to $\overline{m}$ as maximizing the function w.r.t λ is identical to minimizing $\mathcal{X}^2(\lambda||\overline{m})$ when $\lambda_i \geq 0, 1 \leq i \leq C$.

Most of them are based on the assumptions of (strongly) convexity, smoothness, and variance of estimated gradients to analyze the convergence rate. Empirically we test the robustness of local mini-batch SGD, only with the tweak of local iterations and communication frequency. We demonstrate that data heterogeneity does not significantly hinder the joint learning of a global model as long as the local data has certain common features and when the communication frequency and local training iterations are appropriately selected. Our conclusion is based on two simulations: 1) data heterogeneity and 2) biased data sampler. In the first case, we generate the scenario where edge devices have access only to a subset of the classification classes (no overlap between them). In the second case, we employ AL as an active sampler to sample the most representative instances using the acquisition function on edge devices, rather than uniform sampling [72].

2.4 Data Privacy and Security

FL is well known for the property that user data keeps locally, not sharing with the server or any neighboring device. However, there is no formal privacy guarantee, and potential security and privacy issues still exist. Accordingly, we will discuss them based on three possible attacks: membership attack, adversarial attack, and input reconstruction attack.

2.4.1 Membership Inference and Differential Privacy

Given a model M trained by specific training data X, the membership inference reveals whether a particular data instance y appears in training set X. In many cases, the attacker can put on this attack using the answers of the queries, without access to the parameters of ML models, which is called a black-box attack. Whereas the attacker has full access to the model, we call it a white-box attack.

A study called Differential Privacy (DP) [19] defines the privacy loss and offers the general defensive mechanisms for membership inference attacks. It describes a promise to the customers (data owners) "You will not be affected, adversely or otherwise, by allowing your data to be used in any study or analysis, no matter what other studies, data sets, or information sources are available." [19]. It quantifies the probability of a certain amount of information leakage and gives the general defense mechanism, e.g. Laplace Mechanism, Gaussian Mechanism. DP promises that the harm introduced by users' choice to participate in training is not significantly increased, which convinces the users to share their data for analysis. From the attacker's perspective, the attacker knows no more after the user shares data than before.

We will first introduce the definitions of DP based on [19]. Note [61] proposed a Rényi differential privacy as a variation of standard definition by [19].

Definition 1 (distance between datasets [19]). *The ℓ_1 norm of a dataset is denoted $\|x\|_1$ and is defined to be:*

$$\|x\|_1 = \sum_{i=1}^{|\mathcal{X}|} |x_i|$$

then ℓ_1 distance between two datasets D and D' is $\|D - D'\|_1$.

Supposed $\mathcal{X}$ is an universe dataset, and two datasets $D, D' \subset \mathcal{X}$ are neighbours (adjacent) if they differ in the data of a single individual, i.e. $\|D - D']\|_1 \leq 1$.

Definition 2 (relaxed DP [19]). *A random algorithm $\mathcal{M}$ with domain $\mathbb{N}^{|\mathcal{X}|}$ is (ϵ, δ)-differentially private if for all $\mathcal{S} \subseteq Range(\mathcal{M})$ (output space) and for all $D, D' \subset \mathcal{X}$ such that $\|D - D'\|_1 \leq 1$:*

$$P[\mathcal{M}(D) \in \mathcal{S}] \leq \exp(\epsilon) P[\mathcal{M}(D') \in \mathcal{S}] + \delta. \tag{2.13}$$

In equation 2.13, e^{ϵ} quantifies the privacy loss if no breach event occurs and δ upper bounds the probability of uncontrolled breach event, thus it should be preset as a small number. If $\delta = 0$, we say that $\mathcal{M}$ is $\epsilon-$differentially private or restrict DP. Equation 2.13 can be rewritten as $e^{-\epsilon} \leq \frac{P[\mathcal{M}(D)\in\mathcal{S}]}{P[\mathcal{M}(D')\in\mathcal{S}]} \leq e^{\epsilon}$. Note that $e^{-\epsilon}$ is approximated to $1-\epsilon$ and e^{ϵ} is approximated to $1+\epsilon$ if $|\epsilon| \approx 0$. Often we define *privacy loss* $\mathcal{L}^{\xi}_{\mathcal{M}(D)||\mathcal{M}(D')}$ as

$$\mathcal{L}^{\xi}_{\mathcal{M}(D)||\mathcal{M}(D')} = \ln \frac{p[\mathcal{M}(D) = \xi]}{p[\mathcal{M}(D') = \xi]}. \tag{2.14}$$

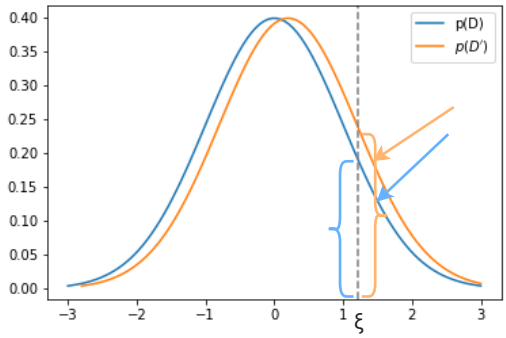


Figure 2.3: Two adjacent datasets D and D' with difference 1. Blue curve represents the probability density function of dataset D and similarly orange curve for D'. For an output is equal to ξ (grey dash line), the rate of the two probabilities are shown by the blue and orange vertical curly bracket.

One straightforward instance is shown in Figure 2.3, and the blue curves represents the density function of dataset D containing Alice's data, whereas the orange is the density function of dataset D' without Alice's data ($\|D - D'\|_1 = 1$). The difference of the intersections between the blue (orange) curve and grey dashed line indicates the ratio of two density functions. We wish the ratio close to one, which implies that including Alice's data will not change the statistics significantly and at the same time, we want the ML model to learn from the data. An extreme case is that the statistics do not change with more data added (the ratio is exactly one), which means no privacy loss occurs, but the model does not learn at all. This is not what we want from the perspective of training the model. Thus, it is a tradeoff between model learning and privacy loss in some sense, but they are also not completely opposite, and we will discuss it in Chapter 2.4.4.

To introduce the defense mechanism, we need first to present ℓ_1 sensitivity that captures the maximal difference of the given function between two neighbor datasets.

Definition 3 (ℓ_1 sensitivity, [19])**.** *The ℓ_1 sensitivity of a function f that maps from $\mathbb{N}^{|\mathcal{X}|} \to \mathbb{N}^k$ is:*

$$\Delta f = \max_{\substack{D,D' \subset \mathcal{X} \\ \|D-D'\|_1 = 1}} \|f(D) - f(D')\|_1 . \tag{2.15}$$

ℓ_1 sensitivity captures the maximal change of all possible individuals in the set. It tells how much one instance can change the results of function f in the worst case. It gives us the upper bound of how much perturbation we need to introduce in order to cover each individual's participation.

2.4.2 Defence Mechanisms

In this part, we will introduce three commonly used mechanisms: *Laplace Mechanism*, *Gaussian Mechanism* and *Exponential Mechanism*.

Laplace Mechanism. It samples noise from Laplace distribution for the perturbations on each coordinate of the outputs. First we define the Laplace distribution with mean μ and scale b as $\text{Lap}(x|\mu, b) = \frac{1}{2b} \exp\left(-\frac{|x-\mu|}{b}\right)$. If it is centered around zero (mean is 0), we write it as $\text{Lap}(x|b) = \frac{1}{2b} \exp\left(-\frac{|x|}{b}\right)$.

Definition 4 (Laplace Mechanism [19])**.** *Given any function f that maps from $\mathbb{N}^{|\mathcal{X}|} \to \mathbb{R}^k$, the Laplace Mechanism is defined as:*

$$\mathcal{M}_{Lap}(x, f(\cdot), \epsilon) = f(x) + (y_1, ..., y_k) \tag{2.16}$$

where y_i are i.i.d. random variables sampled from Lap$(\frac{\Delta f}{\epsilon})$.

Theorem 5. *The Laplace mechanism preserves $(\epsilon, 0)$-differential privacy [19].*

Proof. We will prove it by the privacy loss of two adjacent dataset D and D' where $\|D - D'\|_1 = 1$. Let P_D denote the probability density function of $M_{\text{Lap}}(D, f, \epsilon)$ and $P_{D'}$ denotes the probability density function of $M_{\text{Lap}}(D', f, \epsilon)$. We compute the privacy loss at any random value $z \in \mathbb{R}^k$. Thus, we have

$$\begin{aligned}
\frac{P_D(z)}{P_{D'}(z)} &= \Pi_{i=1}^k \frac{\exp\left\{-\frac{\epsilon|f(D)_i - z_i|}{\Delta f}\right\}}{\exp\left\{-\frac{\epsilon|f(D')_i - z_i|}{\Delta f}\right\}} \\
&= \Pi_{i=1}^k \exp\left\{\frac{\epsilon(|f(D')_i - z_i| - |f(D)_i - z_i|)}{\Delta f}\right\} \\
&\overset{(i)}{\leq} \Pi_{i=1}^k \exp\left\{\frac{\epsilon(|f(D')_i - f(D)_i|)}{\Delta f}\right\} \\
&\overset{(ii)}{\leq} \exp(\epsilon).
\end{aligned}$$

The first inequality (i) is due to the triangle inequality $|x| - |y| \leq |x - y|$ and the second inequality (ii) is due to the definition of Δf. By symmetry, $\frac{p_{D'}(z)}{p_D(z)} \geq \exp(-\epsilon)$. □

Exponential Mechanism. It was designed for the scenario where adding noise directly to the computed quantity will destroy the value completely; instead, we want to maximize the venue. Thus we will introduce an utility function u. Here the sensitivity is defined as the maximal value over any adjacent datasets D, D' and output space Θ.

Definition 6 (Exponential Mechanism [19])**.** *Given an utility function u and the global sensitivity is $\Delta u := \max_{\theta \in \Theta} \max_{\substack{\|D-D'\|=1 \\ D,D' \subset \mathcal{X}}} |u(D,\theta) - u(D',\theta)|$. Exponential Mechanism generated private $\theta \sim \exp\{\frac{\epsilon u(x,\theta)}{2\Delta_u}\}$ where θ is $(\epsilon, 0)$-differentially private.*

Theorem 7. *The exponential mechanism preserves $(\epsilon, 0)$-differential privacy [19].*

Proof. First we assume the range Θ is finite and the ratio of the exponential mechanism outputs some element $\theta \in \Theta$. Say we have two neighbour datasets D, D' and assume the sensitivity is Δu such that $u(D',\theta') \leq u(D,\theta') + \Delta u \, \forall \theta' \in \Theta$. The ratio of the two probabilities is given as

$$
\begin{aligned}
\frac{P[M_{\text{Exp}}(D,u,\Theta)=\theta]}{P[M_{\text{Exp}}(D',u,\Theta)=\theta]} &= \frac{\frac{\exp\{\frac{\epsilon u(D,\theta)}{2\Delta u}\}}{\sum_{\theta' \in \Theta} \exp\{\frac{\epsilon u(D,\theta')}{2\Delta u}\}}}{\frac{\exp\{\frac{\epsilon u(D',\theta)}{2\Delta u}\}}{\sum_{\theta' \in \Theta} \exp\{\frac{\epsilon u(D',\theta'\}}{2\Delta u})}} \\
&= (\frac{\exp\{\frac{\epsilon u(D,\theta)}{2\Delta u}\}}{\exp\{\frac{\epsilon u(D',\theta)}{2\Delta u}\}})(\frac{\sum_{\theta' \in \Theta} \exp\{\frac{\epsilon u(D',\theta')}{2\Delta u}\}}{\sum_{\theta' \in \Theta} \exp\{\frac{\epsilon u(D,\theta')}{2\Delta u}\}}) \\
&\overset{(i)}{\leq} \exp\{\frac{\epsilon}{2}\} \frac{\sum_{\theta' \in \Theta} \exp\{\frac{\epsilon u(D',\theta')}{2\Delta u}\}}{\sum_{\theta' \in \Theta} \exp\{\frac{\epsilon u(D,\theta')}{2\Delta u}\}} \\
&\overset{(ii)}{\leq} \exp\{\frac{\epsilon}{2}\} \frac{\sum_{\theta' \in \Theta} \exp\{\frac{\epsilon (u(D,\theta')+\Delta u)}{2\Delta u}\}}{\sum_{\theta' \in \Theta} \exp\{\frac{\epsilon u(D,\theta')}{2\Delta u}\}} \\
&= \exp\{\frac{\epsilon}{2}\} \frac{\exp\{\frac{\epsilon}{2}\} \sum_{\theta' \in \Theta} \exp\{\frac{\epsilon u(D,\theta')}{2\Delta u}\}}{\sum_{\theta' \in \Theta} \exp\{\frac{\epsilon u(D,\theta')}{2\Delta u}\}} \\
&= \exp\{\frac{\epsilon}{2}\} \exp\{\frac{\epsilon}{2}\} \\
&= \exp(\epsilon).
\end{aligned}
$$

The first inequality (i) is the definition of global sensitivity Δu and the second inequality (ii) is due to our assumption $u(D',\theta') \leq u(D,\theta') + \Delta u \, \forall \theta' \in \Theta$. Symmetrically $\frac{p[M_{\text{Exp}}(D',u,\Theta)=\theta]}{p[M_{\text{Exp}}(D,u,\Theta)=\theta]} \geq \exp(-\epsilon)$. □

Gaussian Mechanism. It is similar with Laplace Mechanism, which adds perturbation to the output of function f and the noise is instead sampled from the Gaussian distribution.

Definition 8 (Gaussian Mechanism [19]). *Given function $f : \mathcal{X}^n \to \mathbb{R}^k$ with global sensitivity $\Delta f = \max |f(D) - f(D')|$ where $\|D - D'\|_1 = 1$ and returning z, then output $f(x) + z$, where*

$$z \sim \mathcal{N}(0, \frac{2(\Delta^2 f) \ln \frac{1.25}{\delta}}{\epsilon^2}) \tag{2.17}$$

is (ϵ, δ)-differentially private.

For the proof, we refer to the appendix A of [19].

Both Laplace and Exponential Mechanisms are $(\epsilon, 0)$-differential private, instead Gaussian Mechanism is (ϵ, δ)-differential private. Often for Gaussian Mechanism, we use L2 distance to measure the global sensitivity, which introduces less noise than Laplace Mechanism where L1 distance is used to quantify the sensitivity. Laplace mechanism works well with numerical query and Exponential mechanism is applicable with both numerical and categorical query.

2.4.3 Composition Theorem

We will introduce composition theorem in this section, which is applied in the cases like combining two different differentially privates algorithms together or applying the same algorithm on the dataset more than once. This is a very significant property of DP.

Theorem 9. *Say $A_1 : \mathbb{N}^{|\mathcal{X}|} \to \mathcal{R}_1$ is an ϵ_1-differential private algorithm and $A_2 : \mathbb{N}^{|\mathcal{X}|} \to \mathcal{R}_2$ is an ϵ_2-differential private algorithm. Then their combination $A_{1,2}(x) = (A_1(x), A_2(x))$ is an $(\epsilon_1 + \epsilon_2)$-differential private algorithm [19].*

Proof. Let $D, D' \in \mathbb{N}^{|\mathcal{X}|}$ and $\|D - D'\|_1 \leq 1$. For any $(r_1, r_2) \in (\mathcal{R}_1, \mathcal{R}_2)$, we have

$$\begin{aligned} \frac{P[A_{1,2}(D) = (r_1, r_2)]}{P[A_{1,2}(D') = (r_1, r_2)]} &= \frac{P[A_1(D) = r_1]P[A_2(D) = r_2]}{P[A_1(D') = r_1]P[A_2(D') = r_2]} \\ &\overset{(i)}{\leq} \exp(\epsilon_1) \exp(\epsilon_2) \\ &= \exp(\epsilon_1 + \epsilon_2). \end{aligned}$$

Symmetrically, $\frac{p[M_{1,2}(D') = (r_1, r_2)]}{p[M_{1,2}(D) = (r_1, r_2)]} \geq \exp(-(\epsilon_1 + \epsilon_2))$. □

It can be generalized to k (ϵ_i, δ_i)-differential private algorithms combining together to form a new algorithm $\mathcal{A}_{[k]}(x) = (\mathcal{A}_1(x), \mathcal{A}_2(x), ..., \mathcal{A}_k(x))$ that preserves the $(\sum_i^k \epsilon_i, \sum_i^k \delta_i)$ differential privacy. For the proof, we refer to appendix B of [19]. This virtue of DP allows the combination of different basic private algorithms for the broad utilization in practice. Another situation is the multiple access of the same dataset, and it will make ϵ and δ degrade, which means we cannot offer a tight guarantee with repeated use.

2.4.4 Machine Learning and Differential Privacy

DP is not necessarily incompatible with ML algorithms [19]. The good learning algorithm aims to learn the generalized rule that describes the data at hand and the unseen data from the same distribution. The learner is expected to generalize well to the population distribution instead of depending too specifically on individuals (overfit in ML jargon). It is also the goal of private data analysis, learning, and inferring from data without disclosing too much information about any single data point in the set.

Often we can apply the defense mechanism directly on input data, output predictions (answers to queries), or the cost function based on which we find the optimal model to fit the data. [17], [11], [20] studied output perturbations, masking the sensitive information based on the quantification of the global sensitivity. They showed that output perturbation is productive in some cases, e.g. sum queries, Principle Component Analysis (PCA), k-means, statistical learning, etc. [66] proposed a new framework to address the high-value perturbation problem, which drives too much compromise of model accuracy. Instead, the instance-based additive noise is introduced since the noise is not only determined by the released function (global sensitivity) but also the data per sé. [13] works on the cost function, bounding the sensitivity of a regularized logistic regression function and afterward introducing the noise proportional to the sensitivity. [14] extends [13] by generalizing the cost function and shows the theoretical guarantee that it is a ϵ-differential private as long as the cost function and the regularizer satisfy certain convexity and differentiability criteria. In addition, they empirically and theoretically showed that the objective perturbation method outperforms the sensitivity method to manage the tradeoff between privacy and learning performance. [2] made the breakthrough, applying DP on the non-convex neural networks. They invented a stronger composition method - moments accountant, which led to a tighter privacy cost for determining the other parameters, e.g. noise scale and clipping threshold of gradients.

Under FL framework, we are no longer focused on a single data point. Instead, we protect the participation information of individual workers during federated optimization. It [28] introduces a client level protection by introducing noise sampled from Gaussian distribution to the normalized update of gradients from each local worker. To track the privacy loss, they apply the same method as [2] to produce a tighter bound than the standard composition theorem.

2.4.5 Adversarial attack

Another significant threat in FL (not limited to FL) is adversarial attack, the crafted inputs adding to the learning process to prevent the model from learning. Typically, the adversary imposes noise to input to lead the misclassification during learning or prediction, where the noise is called *adversarial perturbation*. We define the adversarial sample $\widetilde{x}$ as

$$\widetilde{x} := x + \eta \tag{2.18}$$

where $x \in \mathbb{R}^n$ is a clean input and $\eta \in \mathbb{R}^n$ is the added perturbation. The cost function is defined as $L(w, x, y)$, and w is the model parameter to optimize during the training process. It aims to introduce a small perturbation onto clean input while causes the model misclassifying with strong confidence.

It explains why small perturbation may cause the misclassification using the linear explanation [31]. Considering the linear product of adversarial sample as $w^\intercal \widetilde{x} = w^\intercal x + w^\intercal \eta$, $w^\intercal \eta$ is the term causes activation grows. If we set $\eta = \text{sign}(w)$ such that the multiplications $w^\intercal \eta$ is equal to $\|w\|_1$ where all the elements to be summed up are positive. They show a small perturbation η such that $\|\eta\|_\infty < \epsilon$ (ϵ very small) can cause change to output for the high-dimensional input as the change is linearly proportional to input dimension. Furthermore, they suggest to employ Fast Gradient Sign Method (FGSM) to maximize the inner product $w^\intercal \eta = w^\intercal \epsilon \text{sign}(\nabla_x L(w, x, y))$. They conclude that more linear neural network, e.g. activation function using ReLU, Maxout network, or softmax regressor is easier to apply adversarial attacks.

The work [58] considers the adversarial attacks through the lens of robust optimization and attempts to bound the strongest adversarial samples that the adversary may find for the sake of defense. They propose to modify the cost function as

$$\min_w \rho(w) = \mathbb{E}_{(x,y)\sim\mathcal{D}}[\max_{\eta\in\mathcal{S}} L(w, x + \eta, y)] \tag{2.19}$$

such that they first maximize the loss w.r.t the perturbation parameter η, and then they minimize the maximized function to train the model. They aim to find the saddle point in the landscape to maximize loss raised from adversarial perturbation and minimize loss through seeking optimal model parameters. From a defensive perspective, they want first to find the strongest adversarial samples, i.e. makes the model misclassifying with the highest confidence, such that any perturbation less than it the model would be "immune" to it. They propose to use Projected Gradient Descent (PGD) that is defined as $y = \Pi_{x+\mathcal{S}}(x + \epsilon \text{sign}(\nabla_x L(w, x, y)))$, thus inner product becomes $w^\intercal \Pi_{x+\mathcal{S}}(x + \epsilon \text{sign}(\nabla_x L(w, x, y)))$. There is no guarantee for finding the global optimal solution that maximizes the inner non-concave function and minimizes the outer non-convex function.

However, for the maximization issue, they empirically observe that local maximum found by PGD all have similar loss values for both normally trained and adversarially trained networks. Namely, they empirically guarantee that as long as the adversary uses the first-order gradient of input point to generate adversarial examples, they cannot find a significantly better local maximum than PGD, which means PGD can give a valid upper bound. For the outer minimization problem, their experiments show that SGD optimizer can consistently reduce loss at the maximal point of the inner part.

It proposes a new perturbation rectifying network where an additive layer is embedded before the targeted network such that the prediction of perturbed input is the same as the clean input [3]. The rectifier is defined as $\mathcal{R}(\widetilde{x}; \theta) : \widetilde{x} \to x$ and θ is trained according

to cost function $P_{x\sim\mathcal{D}}[f(x) = f(\mathcal{R}(\widetilde{x};\theta))] \approx 1$ where f is the classifier (targeted model). In addition, the network has a binary classifier to recognize whether the sample is adversarial, which is similar to the discriminator of Generative Adversarial Network (GAN). The binary classifier is defined as $\mathbb{B}(z) : z \to [0, 1]$ where z is sampled from clean samples and adversarial samples $z \sim \{x\} \cup \{\widetilde{x}\}$. It works on the model architecture instead of attempting to explicitly upper bounding the strongest adversarial samples in the set.

We introduced a few critical works in this section, and we suggest [82, 90, 93] for more reading. The adversarial attack is more likely to happen and easier to apply in the FL diagram when distributed customers share their gradients with the server constantly. The attacker can easily impersonate an honest participant and share the perturbed gradients with either server or neighbor to harm the global model learning. Often the magnitude of perturbation ϵ is small such that it is difficult to recognize the difference between a clean sample and the corresponding adversarial sample with the bare eyes. However, such constraint is released in FL since only gradients are communicated between workers and server. For instance, [10] shows that no linear aggregation rule is resilient to a single adversary. It may accurately decide the gradient vector after aggregation. Consider an aggregation rule A as $A(g_1, g_2, ...g_n) = \sum_{i=1}^{n} \lambda_i g_i$ where g_i is the gradient shared by worker i, $\lambda_i \geq 0$ and $\sum_i^n \lambda_i = 1$. Let G be any target gradient, if a single adversary proposes the gradient vector as $g_i = \frac{1}{\lambda_i} G - \sum_{j=1,j\neq i}^{n} \frac{\lambda_j}{\lambda_i} g_j$, then the aggregated result is G.

2.4.6 Reconstruction attack

One virtue of FL is to keep training data on the generated device, which avoids sensitive information leakage due to the direct transmission. However, recovery of the input image based on gradient information was empirically shown to be possible using a fully-connected neural network [5]. Similarly, [97] showed that reconstruction of input image is feasible on CNN.

Input reconstruction is actually equal to function inverse. Say, we have a model f parameterized by θ with input $X \in \mathbb{R}^{B\times d}$ where B is batch size and d is dimension of input. A function G_θ that takes input X and outputs the gradients vector V, $G_\theta : \mathbb{R}^d \to \mathbb{R}^p$ where p is the number of model parameters. For the batch case, we average the output of function G_θ for each instance $\overline{v} = \frac{1}{B} \sum_{i=1}^{B} G_\theta(x_i, y_i)$. If G_θ^{-1} exists or G_θ is a bijective function, then we have the analytical form to compute input $X = G_\theta^{-1}(V)$. However, in most cases, the neural network model is not reversible due to its non-linearity. Also, when $B > 1$ there never exists any analytical form since the mean gradient of the corresponding batch is computed, and all the possible perturbations give the same result. Thus, an iterative optimization method is suggested applying to solve this problem.

It is a relatively new research topic and has caught more and more attention recently due to the popularity of FL. The recent work [106] demonstrates the possibility of the

input reconstruction using LeNet5 [48]. However, it does not give experimental results on big batch reconstruction. Another work [103] expands on [106] to improve the label prediction accuracy. They [24] propose a new cost function and reconstruct the input based on deep neural networks like ResNet [35]. It [98] introduces a framework for the evaluation of privacy leakage and relation to FL hyperparameters. [67] also focuses on deep neural network like VGG [86], GoogLeNet [91], which are normally are more informative for the reconstruction.

Our work [73] instead provides the analysis of lower-bound requirements for the input reconstruction for the consideration of defense. We mainly study the reconstruction condition on two modules: the fully-connected and convolutional layers, and decompose the whole neural network as modules to analyze them independently. The inverse step of the fully-connected layer and convolutional layer in the manuscript is named as *demixing* and *deconvolution* in the manuscript. We will summarize the key observations in Chapter 3.4.

2.5 Active Learning

AL is practically required in the modern machine learning applications where many data points are unlabeled, and to manually label those data is costly. For instance speech recognition, [107] reports that annotation at the word level can take ten times longer than the actual audio. AL is typically applied when one set of data is labeled, and another set is unlabeled, and manually annotating the whole unlabeled set is expensive. According to the format of arrival unlabeled data, AL can be grouped as pool-based and stream-based. It is stream-based AL if data arrives in the stream, and pool-based otherwise [78]. In the stream-based AL scenario, whenever one instance arrives, the oracle decides whether discard this instance or keep and label it.

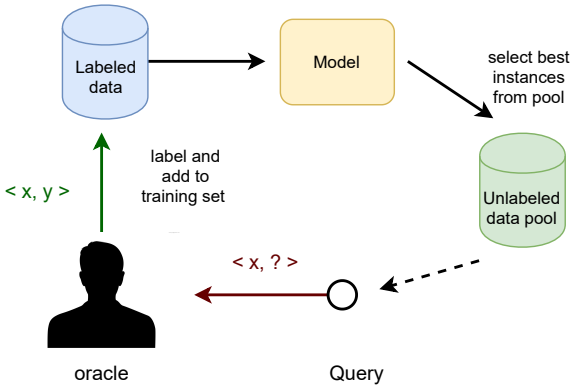


Figure 2.4: Pool-based Active Learning Scheme. The model is initially trained by the small labeled dataset $\mathcal{L}$, and then it samples the unlabeled instances $\mathcal{X}$ from the unlabeled data pool $\mathcal{U}$ based on the acquisition function and ask the oracle to label them. The model is further trained by either the recently labeled instances $(\mathcal{X}, \mathcal{Y})$ or the recently labeled instances along with the labeled dataset $(\mathcal{X}, \mathcal{Y}) \cup \mathcal{L}$. The figure is from appendix A.

For the convenience, we study pool-based AL in this context. Pool-based AL is composed of a model trained by the initial labeled dataset, an unlabeled data pool, a small labeled dataset and an "Oracle", as illustrated in Figure 2.4. Then, it selects the most representative instances from the unlabeled dataset based on the *acquisition function*. After that, it asks the oracle to label the chosen instances and update the labeled dataset by including them for future training. Such operations can be repeated for several times. Note here that we can also further train the model by labeled dataset in the current round to save the computation cost, discarding the training data in the previous rounds, which is often called *online AL*. However, if no regularizer is present in the cost function, it might suffer from the catastrophic forgetting.

The critical part of AL is the acquisition function, which aims to find the "good" samples such that we can use fewer data points to reach higher accuracy. It gauges

the uncertainties of all available data in the unlabeled set to guide the exploration in the parameter space during training. One illustration is shown in Figure 2.5 where the "good" instances to determine the decision boundary are marked by red circles. Instead some points are close to each other within two clusters, which are useful to learn the full data distribution, but they are redundant for the classification task, i.e. less critical to learn the decision boundary.

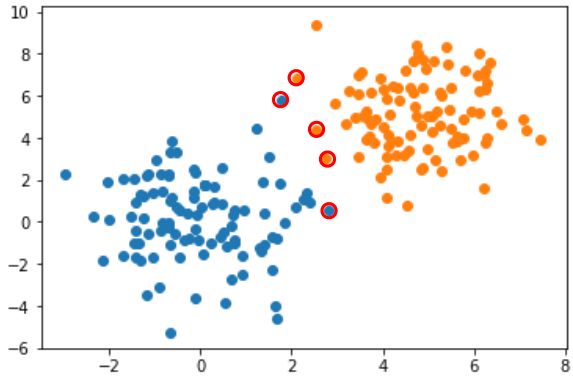


Figure 2.5: Classification task of blue and orange dots. The dots marked by red circles are critical to determine the decision boundary rather than those dots are far away from the critical area.

Let's first form the problem. Say we have two sets of data U and L, indicating unlabeled data and labeled data accordingly. We define the acquisition function as A based on which we aim to find the given number of instances from set U such that A is maximized or minimized. The acquisition function in fact defines a non-uniform sampling scheme.

The most common sampling framework is *uncertainty sampling* [51]. As the name suggested, the acquisition function samples the instances that the predictor is most uncertain about their predictions. It is simple to apply it to the probabilistic model since we can directly use the posterior function to quantify it, e.g. entropy [83]. We define it as

$$x^*_{\text{Entropy}} := \underset{x \in U}{\text{argmax}} - \sum_{i=1}^{C} p(y_i|x;\theta) \log p(y_i|x;\theta) \tag{2.20}$$

where x is the input and y is a vector with a length equal to the number of classes C and θ is the model parameters. The optimal instance is x^* by maximizing the entropy of all unlabeled instances in set U. For the non-probabilistic model, we can also use the votes to decide on which label(s) the committee is most likely disagreed, which is called *Query-by-committee* [81].

Another method measures the maximal gradient change after adding new instances into the training set [80]. It was referred to as *Expected Gradient Length* (EGL) and defined as:

$$x^*_{\mathrm{EGL}} := \underset{x \in U}{\operatorname{argmax}} \sum_{i=1}^{C} p(y_i|x;\theta) \left\|\nabla \ell(L \cup \{x, y_i\}; \theta)\right\|_2 . \tag{2.21}$$

Similarly, x is the input and y is a vector with length equal to the number of classes C and θ is the model parameters. $\|\cdot\|$ is the Euclidean norm of each resulting gradient after adding one instance from the unlabeled set U. L is the labeled set. If the instances are assumed to be independent, we can only consider $\nabla \ell(x, y_i; \theta)$ to reduce the computational cost.

Another type of method that aims to find the most representative instances that may generally capture the character of the other instances in the unlabeled pool was proposed by [79]. It is literally a density estimated method where the "good" samples are often found in the high-density area. Officially, it is called *information density* (ID) method. We define it as

$$x^*_{\mathrm{ID}} := \underset{x \in U}{\operatorname{argmax}}\, \Psi(x) \times \Big(\frac{1}{|U|} \sum_{x' \in U} \mathrm{sim}(x, x')\Big)^{\beta}. \tag{2.22}$$

We first compute the average of similarity between instance x and all the rest instances in set U and function $\mathrm{sim}(\cdot)$ could be for instance cosine function and β controls the importance of this term. $\Psi(\cdot)$ is the function indicating the representativeness of x, e.g. entropy.

Moreover, it [15] suggests a method aiming to minimize the variance, which is often used to solve the regression problem. An extended work [102] attempting to address the classification task, which suggests to sample the instances based on Fisher Information.

2.6 Active Learning on Neural Networks

AL is also applicable on DL, often referred to as Active Deep Learning (ADL). The posterior distribution is required for the quantification of uncertainty-based acquisition functions, however, the posterior of the neural network is intractable without any strong assumption. It can be addressed mainly by sampling or Variational Inference (VI) [40]. Sampling methods like Markov Chain Monte Carlo (MCMC), Metropolis-Hastings [59], Gibbs sampling [27] are often used to obtain samples from the target posterior after its convergence, which is sometimes ensured by the adjustment of the sampling process is [26].

2.6.1 Markov Chain Monte Carlo

We first define a posterior distribution as

$$p_\theta(z|x) = \frac{p_\theta(x|z)p(z)}{\int p_\theta(x,z)\,dz} \tag{2.23}$$

where x is the observation, and z is the latent variable and the probability is parameterized by θ. It is often infeasible to integral or sum over all latent variables; in particular for neural network it has millions of parameters. Thus exact inference is intractable. For the intractable posterior (or computationally expensive), we can employ a Markov chain simulation (a.k.a MCMC) to draw z from the approximated posterior distribution. The key of Markov chain simulation is to devise it such that it is not biased and its stationary distribution is target (posterior) distribution [26]. After the sampling process converges to the target distribution, all the subsequent samples can be used for target distribution. By construction, it ensures the convergence, but it might converge slowly.

2.6.1.1 Metropolis-Hastings method

The Metropolis-Hastings method is adapted to a random walk with acceptance and rejection rule so as to converge to the targeted distribution [59]. We presented in the followings, and step 3 means if the recently sampled z^* (corrected by $J(z^*|z^{t-1})$) gives higher posterior comparing with z^{t-1} (corrected by $J(z^{t-1}|z^*)$), we will update z^t by z^* with probability 1, otherwise with probability $\min(r, 1)$. It implies that the new sampled z^* will be more likely accepted if higher the posterior of the new sampled z^* improves.

- initialize z^0 randomly from the starting distribution.
- For $t = 1, 2, ..T$:
    1. at time t sample z^* from a proposed distribution $J(z^*|z^{t-1})$.
    2. calculate the ratio of densities $r = \frac{p(z^*|x)/J(z^*|z^{t-1})}{p(z^{t-1}|x)/J(z^{t-1}|z^*)}$

3. set

$$z^t = \begin{cases} z^* & \text{with probability } \min(r, 1), \\ z^{t-1} & \text{otherwise.} \end{cases} \tag{2.24}$$

2.6.1.2 Gibbs Sampling

Gibbs sampling is one particular Markov chain method for multidimensional variables, and it is also alternating conditional sampling since it samples one sub-dimension based on the subvector that contains the rest of the dimensions [25]. We define the multidimensional variable $z \in \mathbb{R}^d$ as $z = (z_1, z_2, ...z_d)$ and assume T iterations are required to reach the accurate sampling. The sampling algorithm is described below. Note that the superscript of z^t is the iteration, and for each iteration the sampler cycle through all dimensions in the vector. When all the dimension of z have been updated the superscripts are the same, i.e. after update the last dimension d, it becomes $(z_1^t, z_2^t, ...z_{j-1}^t, z_j^t, z_{j+1}^t, ..., z_d^t)$.

- initialize with z^t where $t = 0$.
- sample z^{t+1}, we need to cycle through all d dimensions in the vector z.

  for $j = 1, 2, ..., d$:

$$\begin{aligned} z_j^t &\sim p(z_j | z_{-j}^{t-1}, x) \\ z_{-j}^{t-1} &= (z_1^t, z_2^t, ...z_{j-1}^t, z_{j+1}^{t-1}, ..., z_d^{t-1}). \end{aligned} \tag{2.25}$$

- repeat it for T time and finally we have $z^T = (z_1^T, z_2^T, ..., z_d^T)$.

2.6.2 Variational Inference

Variational Inference often proposes a *surrogate* (*variational*) distribution $q_\gamma(z)$ (parameterized by γ) that approximates the real posterior distribution $p_\theta(z|x)$ (parameterized by θ) by minimizing the distance between them. Kullback–Leibler divergence (KL divergence) is commonly applied, which converts the inference problem into an optimization problem and the optimal $q_\gamma^*(z)$ is found by

$$q_\gamma^*(z) := \underset{q \in \mathcal{Q}}{\operatorname{argmin}} D_{KL}(q_\gamma(z) || p_\theta(z|x)). \tag{2.26}$$

For a given parametric family $\mathcal{Q}$ such as Normal distribution, an optimization problem is introduced over variational parameter space $\gamma \in \Gamma_\gamma$.

We can further write the KL divergence as:

$$\begin{aligned} D_{KL}(q_\gamma(z)||p_\theta(z|x)) &= \mathop{\mathbb{E}}_{q_\gamma(z)}[\log q_\gamma(z) - \log p_\theta(z|x)] \\ &= \mathop{\mathbb{E}}_{q_\gamma(z)}[\log q_\gamma(z) - \log \frac{p_\theta(x,z)}{p_\theta(x)}] \\ &= \mathop{\mathbb{E}}_{q_\gamma(z)}[\log q_\gamma(z) - \log p_\theta(x,z)] + \log p_\theta(x) \geqslant 0. \end{aligned} \tag{2.27}$$

The log likelihood $\log p_\theta(x)$ can be written as Evidence Lower Bound (ELBO) and approximation error in

$$\log p_\theta(x) = \underbrace{D_{KL}(q_\gamma(z)||p_\theta(z|x))}_{\text{approximation error}} + \underbrace{\mathop{\mathbb{E}}_{q_\gamma(z)}[\log p_\theta(x,z) - \log q_\gamma(z)]}_{\text{ELBO}}. \tag{2.28}$$

ELBO only requires the joint distribution of observation x and latent variables z, without the requirement of marginal distribution computed integrating over all latent variables. Since $D_{KL}(q_\gamma(z)||p_\theta(z|x))$ is greater or equal to zero, we have the lower bound of the log likelihood $\log p_\theta(x) \geq \mathbb{E}_{q_\gamma(z)}[\log p_\theta(x,z) - \log q_\gamma(z)]$.

Instead of estimating $\log p_\theta(x)$, we can maximize the ELBO, which is an optimization problem over parameters θ and γ. The challenge here is how tight the bound could be, which depends on whether $\mathcal{Q}$ covers the real posterior distribution $p_\theta(z|x)$. Note that KL divergence sometimes suffers from underestimating the variance of real posterior, and it is a comparably loose bound. To have more accurate posterior estimation, other divergence measurements are also studied, e.g. α divergence [56], f divergence [4].

2.6.3 Bayesian neural network approximating by dropout

The authors of [23] show that neural networks with dropout [88] as the regularizers approximate to VI. We reformulate ELBO as

$$\begin{aligned} \mathcal{L}_{VI} &:= \mathop{\mathbb{E}}_{q_\gamma(z)}[\log p_\theta(x,z) - \log q_\gamma(z)] \\ &= \mathop{\mathbb{E}}_{q_\gamma(z)}[\log p_\theta(x|z) + \log \frac{p_\theta(z)}{q_\gamma(z)}] \\ &= -D_{KL}(q_\gamma(z)||p_\theta(z)) + \mathop{\mathbb{E}}_{q_\gamma(z)}[\log p_\theta(x|z)]. \end{aligned} \tag{2.29}$$

We convert the KL divergence distance between real posterior and variational posterior to KL divergence distance between real prior and variational prior.

To be consistent with the notation in [71], we denote w as the model parameter to infer and rewrite the loss function as

$$\hat{\mathcal{L}}_{VI} := -D_{KL}(q(w)||p(w)) + \frac{1}{N}\sum_{i=1}^{N} \ell(y_i, \hat{f}(x_i, \hat{w})). \tag{2.30}$$

We ignore the subscripts γ and θ for the brevity and replace the second term with our cost function for the tasks, e.g. cross-entropy or mean square error.

Similarly, we derive a lower bound defined as $\hat{\mathcal{L}}_{VI}$ where $\hat{w} \sim q(w)$ and $\hat{f}(\cdot)$ is the neural network parameterized by $\hat{w}$, and $\ell(\cdot)$ is the loss function. The weights of all neurons in layer i is defined as w_i, and $W = \{w_i\}_{i=1}^{L}$ is the weights of the whole neural network with L layers. The elements of W are masked (set to zeros) according to the Bernoulli distribution, defined as

$$\widetilde{w}_i = w_i \times \text{diag}([z_{i,j}]_{j=1}^{K_i}) \tag{2.31}$$

where

$$z_{i,j} \sim \text{Bernoulli}(p_i) \quad i = 1, .., L. \tag{2.32}$$

The diag(.) is the operator that maps a vector to a diagonal matrix, whose diagonal are the elements of the vector. $z_{i,j}$ is the Bernoulli random variable and equal to either one or zero and K_i is the number of neurons in layer i. Given input x and weights $\hat{w}$ sampled from $q(w)$, the predictive distribution is defined as

$$\begin{aligned} p(y^*|x^*, D) &\approx \int p(y^*|x^*, w)q(w)\, dw \\ &\approx \frac{1}{T}\sum_{t=1}^{T} p(y^*|x^*, \hat{w}_t). \end{aligned} \tag{2.33}$$

After we manage to quantify the posterior distribution, we can employ the uncertainty-based acquisition functions to opt out the "good" instances from the pool.

Given C is the set of the unique labels, T is the sampling times and D is the dataset.

- **Maximal Entropy (ME):** $H[y|x, D]$ is the predictive entropy expectation defined as

$$H[y|x, D] := -\sum_{c \in C} p(y = c|x, D) \log p(y = c|x, D). \tag{2.34}$$

- **Bayesian Active Learning by Disagreement (BALD):** it seeks the examples that the different sampled models disagree the most [38]. It is defined as

$$I[y; w|x, D] := H[y|x, D] - \mathbb{E}_{p(w|D)}[H(y|x, w, D)] \tag{2.35}$$

  where $w \sim q(w)$. The first term is the entropy of the model prediction and the second term is the expected prediction over the posterior distribution. We want to find the points that maximize it, which means the first term is high and the second term is low. It happens when the model has different ways to explain the data with different samples from posterior distributions and they disagree among each other [44].

CHAPTER 3

Contributions

In this chapter, we describe the contributions to this thesis. For each paper, we will motivate the research question underlying the manuscript and briefly outline the contributions made by the paper. The manuscripts themselves are found in the appendix.

3.1 Distributed Active Learning Strategies on Edge Computing

To realize the data analytics on Edge Computing, we believe that FL is a promising framework, mainly considering the following aspects: data privacy, response latency and distributed data generation. We presented the relevant backgrounds in Chapter 1.1, 1.2 and covered the introduction of FL potentials and main challenges in Chapter 2.2.

Besides the challenges mentioned in Chapter 2.2, we are intrigued by applying AL under the federated framework for the practical consideration. For some applications, e.g. speed recognition [107], the manual annotation is time-consuming and expertise-demanding. Moreover, allowing the external party access to the local database system or transmit data away from the data-generating device compromises privacy with possible sensitive data leakage. In this study, we employed neural network as models and covered the preliminary introduction of neural networks in terms of architecture and training in Chapter 2.1.

We introduced AL in Chapter 2.5 and presented several acquisition functions, e.g. *Entropy*, *Expected Gradient length*, *Information Density*, etc. However, the estimation of posterior distribution on neural network is intractable as we explained in Chapter 2.6. We suggest to apply MC-dropout to approximate the posterior distribution in Chapter 2.6.3.

The experimental results show that AL works effectively in FL. It improves the model accuracy with the same amount of data compared with the uniform sampling. Given the target accuracy, AL requires less training data. We believe that the combination of AL and FL has the practical meaning, in particular, for applications that data

producers are close to users, we can directly ask the users to label some representative instances. To the best of our knowledge, we are the first to propose this idea.

3.2 Robustness Analytics to Data Heterogeneity in Edge Computing

As pointed out in Chapter 2.2, data heterogeneity is one of the challenges in FL. We simulated two scenarios to empirically study the impact of data heterogeneity on the model performance. First, we employed the active sampler to create mild heterogeneity. In this case, the data owned by the workers (devices) is homogeneous, but the sampling scheme is biased. Second, we assigned the subsets that only include part of the classes to the workers, and more importantly, there are no overlaps between them, which implies a higher heterogeneity level than the first case. For instance, given a dataset with 10 classes and 4 workers, we assign class 1 and 2 to the first worker, class 3 and 4 to the second worker, class 5, 6, 7 to the third worker and the rest to the last worker.

SGD requires frequent communications with the server to aggregate the model parameters. To reduce the communication frequency, we apply local SGD, i.e. a sequential of updates happened on workers before the communication with the server. The experiments in our work [72] show that the data heterogeneity will not necessarily lead to the divergence if the local SGD iterations and communication frequency well opted.

Problem formulation. First, let's formulate the problems. Presumably, N is the number of workers, and each worker k owns data $x_k = \{x_{k,1}, x_{k,2}, ..., x_{k,n_k}\}$ and n_k is the number of data possessed by worker k. p_k describes the weight for worker k when forming the global cost function and $p_k \geq 0$, $\sum_k^N p_k = 1$. We define the global cost function as

$$\min_w F(w) = \sum_{k=1}^{N} p_k F_k(w) \tag{3.1}$$

where F_k is defined as

$$F_k = \frac{1}{n_k} \sum_{i=1}^{n_k} \ell_k(w; x_{k,i}). \tag{3.2}$$

Every round, after the server aggregates the collected parameter information, it updates the model and shares it with all the workers. Then the workers start their local training for E iterations, and $\mathcal{I}_E$ is the set of round numbers for the communications with the server, i.e. if E is 5, then $\mathcal{I}_E = \{5, 10, 15, ...\}$. Then the local SGD update on worker k is defined as

$$w_{t+i+1}^k = w_{t+i}^k - \eta \nabla F_k(\xi_{t+i}^k; w_{t+i}^k) \quad \text{for } i = 0, 1, ...E-1 \tag{3.3}$$

where ξ_{t+i}^k is the data used for model training on worker k at $t+i$ and w_{t+i}^k is the model parameter of worker k at time $t+i$. The server only aggregates the model parameters after every E epochs by $\sum_k^N p_k w_{t+E}^k$.

Aggregation Strategies. We consider three aggregation strategies in [72]: *AveFL*, *OptFL* and *MixFL*. *AveFL* takes the averages of the parameters of local models during aggregation; *OptFL* selects the model with optimal performance; and *MixFL* is the mixture of *AveFL* and *OptFL*, in each iteration it chooses the better one between them.

Epochs choice. Here we call the number of local iterations before communication with the server as epochs. Note that the epoch number is the number of iterations on the same batch. The epoch number influences the model performance after aggregation.

Communication frequency. Say T is the total budget of iterations and F is the communication frequency; thus, the required communication number is T/F. Low communication frequency requires higher communication cost and vice verse.

Learning Rate. The small learning rate can mitigate the effect of the gradient drift caused by the non-i.i.d. data, but it takes longer to converge since it learns little from each model update. A small learning rate limits the quantity of each update to avoid generating divergence, but at the same time it also slows down the model training process.

The aggregation step is similar to "copy" the partial classification capability of each local classifier. However, *AveFL* "dilutes" it due to the average operation, and that is why we call it a partial copy. More sophisticated combinations can be studied, but it asks for more computational budget.

3.3 A Decomposed Deep Training Solution for Fog Computing Platforms

In chapter 1, we discussed the motivation and urgency to have a framework like FL to address some concerns of CC. As a summary, FL suggests implementing local training on distributed devices to avoid the direct data transmission from (massively) distributed devices. It also reduces the computational burden of the server by dividing the tasks among the workers. However, leaving all the training on the workers may not be suitable for applications where FNs have less computational budget.

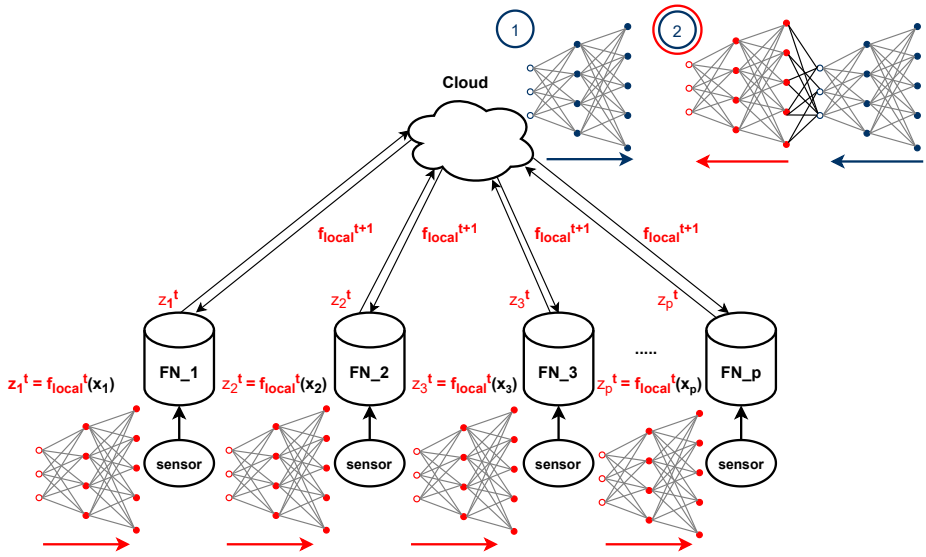


Figure 3.1: The figure is from appendix C. Fog Computing-based Decomposed Deep Training. The local workers own part of the full model, we call it f_{local}. The workers feed the input to the local model and share the intermediate result z_i with the server and the server completes the residual forward pass (step 1, marked by navy blue circle) and carry out the backward calculations (step2, marked by red and navy blue circles).

Unlike the CC solution that leaves all the computation on the Cloud nor FL that leaves most computation on the workers, we would like to propose a new approach in which the computation is split among distributed workers and the server. We name it as Fog Computing-based Decomposed Deep Training (FCDDT) and illustratively demonstrate it in Figure 3.1. Here we assume the most expensive computation of data analytics is model training and divide it into two parts: forward and backward (error propagation introduced in Chapter 2.1). In Figure 3.1, we illustrate one scenario where the workers carry out one part of the forward computation and then they share the intermediate results with the server. The server completes the rest of the forward

calculation and implements the overall error propagation (backward).

Presumably, we have p active workers and input $x \in \mathbb{R}^d$. The whole neural network model f is divided into two parts: local model f_{local} parameterized by w_{local} and remote model f_{remote} parameterized by w_{remote}. Local model f_{local} has m layers and remote model f_{remote} has n layers, and the comprehensive model f has $m+n$ layers. $|w_i|$ indicates the size of w_i and l_i is the number of nodes in layer i. We compared the communication cost for a single round and several other features in Table 3.1. To save the bandwidth, we can design the local model f_{local} with small m and l_m, e.g. one convolutional layer and one fully-connected layer with small number of nodes. In addition, we simulated and evaluated the effect of our method on the network traffic using OMNET++ in [70].

Table 3.1: Comparison of FCDDT with the related work. The table is from appendix C.

Feature	FCDDT	FL	CC
Compute (training) nodes	FNs & Cloud	FNs	Cloud
Raw data transmission	No	No	Yes
Communication cost	$p \times (B \times l_m + \sum_i^m \lvert w_i \rvert)$	$2 \times p \times (\sum_i^{m+n} \lvert w_i \rvert)$	$p \times B \times d$

Th limitation of this work is the expensive communication since the edge devices only implement SGD locally. We suggest to solve it by forming a local network where FN and edge devices are connected to finish the execution of local SGD. For applications with low-degree distributed edge devices, we can also borrow the idea of AlexNet [46] where the model is divided horizontally. We will explore further this idea in the future.

3.4 Minimal Conditions Analysis of Gradient-based Reconstruction in Federated Learning

Input reconstruction breaches the data-security promise of FL and demonstrates the feasibility of recovering input data with model parameters and the corresponding gradients. This realization is critical as FL is famous for no direct data leakage by keeping data on the generating devices. We show such an attack in Figure 3.2 where the server is the attacker. g_i indicates local gradients from worker i and w represents the model parameters after aggregation.

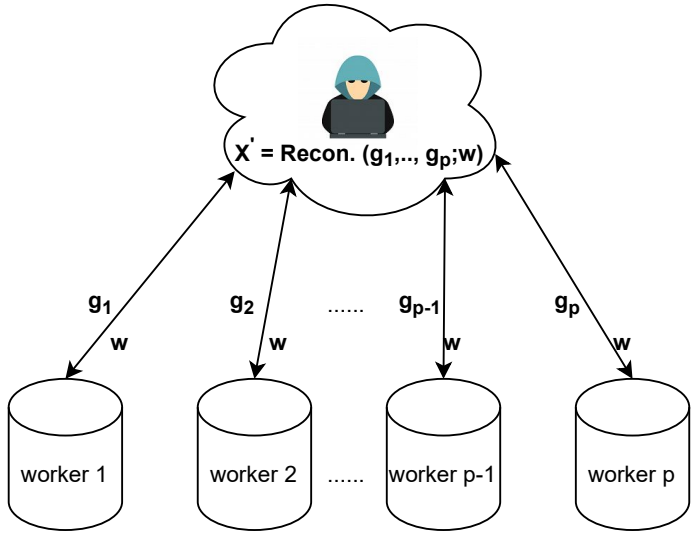


Figure 3.2: Reconstruction attack in FL. Presumably, p workers participate in each round and the server is the adversary. g_i is the gradients shared by worker i and w is the updated global parameters sent back from the server.

To better understand this attack, we analyze it from solving a linear system perspective regarding two types of modules in neural networks: convolutional layer and fully-connected layer (dense layer). Each element in the Jacobian matrix is computed according to the chain rule and we have the full access to partial derivatives in every layer of the neural network.

Our main focus in this work is to quantify the structural lower bound for the full reconstruction, i.e. the number of nodes and kernels required in each layer to completely reconstruction the original inputs. Several key observations from our work are listed below based on two cases: one-instance and batch reconstruction. B indicates the batch size and d is the input dimension, and n_i is the number of units (nodes) in hidden

layer i. For the convolutional layer, we define d as the input size of the convolutional layer and d' as the size after the convolutional operator.

Proposition 10 (ONE-INSTANCE MLP RECONSTRUCTION)**.** *To reconstruct one input based on MLP, we derive the analytical form to compute the (almost) lossless input, with only* ***single*** *unit in the first hidden layer as long as bias term exists, regardless how deep the network is.*

Proposition 11 (BATCH MLP RECONSTRUCTION)**.** *For batch reconstruction, the number of units in first hidden layer should meet $n_1 \geq B$, given the high-dimension input whose dimension $d \gg n_2, d \gg B$.*

Proposition 12 (SINGLE-LAYER CNN RECONSTRUCTION)**.** *To reconstruct a single-layer CNN immediately stacked by a fully-connected layer, $h \geq (\frac{d}{d'})^2 C$ kernels are required, where C is the channel number of input, d is the width of input, and d' is width after convolution layers.*

These observations also applicable to the big batch size, one instance for batch size 100 is shown in Figure 3.3 and 3.4.

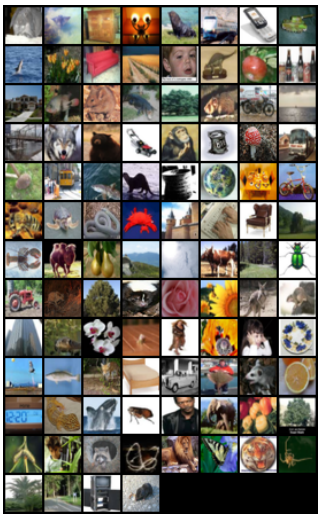


Figure 3.3: The figure is from appendix D. CIFAR100 original inputs with natch size 100.

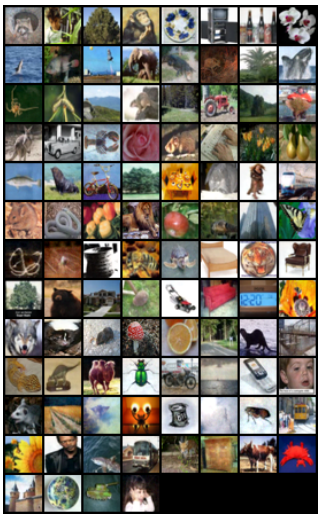


Figure 3.4: The figure is from appendix D. Reconstruction using MLP with 100 nodes.

CHAPTER 4
Conclusions and Outlook

This chapter summarizes the work presented before, reflecting on the doctoral studies and general progress in the topic, and gives an outlook for potential future work.

4.1 Conclusions

The primary goal of this thesis is to implement data analytics and explore the relevant challenges in Fog Computing. In Chapter 1, we presented the motivation for Fog Computing, the working mechanism, advantages, and challenges. By processing data on the data-generating devices instead of forwarding data to a Cloud-based server we enable shorter response times by saving time consumed for transmitting data from devices to the server.

We suggest FL as a possible solution for distributed data analytics. It provides the opportunity to train a global model that learns from local devices without sharing training data among devices. In Chapter 2.2, we presented FL and its related challenges: expensive communication, system heterogeneity, data heterogeneity, and data privacy. We also proposed to apply AL in the distributed setup to reduce the heavy annotation workload. Addition, we highlighted that FL lacks the flexibility of handling scenarios where computational resources are insufficient on the edge device.

We are particularly interested in the data heterogeneity issue due to its theoretical and practical value. It hinders the distributed optimization arising from the significant variance of gradients on different devices, which challenges aggregation. We empirically studied the relationship between model performance and factors such as communication frequency and number of local iterations, which play critical roles in the data-heterogeneity environment. Also, we related the observation of degraded performance to the divergence of model weights when training is becoming divergent.

Also, we studied another challenge - data privacy and security. FL is well known for the virtue that no direct data transmission occurs from the data-generating

devices to Cloud, making it popular for medical, financial applications. However, we demonstrated that the input could be reconstructed in FL with only knowledge of the model parameters and the corresponding Jacobian matrix. We demonstrated this on various datasets: benchmark image dataset (MNIST, KMNIST, CIFAR100) and biomedical dataset (fMRI, X-ray, white blood cell). Critically, we derived the minimal conditions for full reconstruction using MLP and CNN, by solving the linear system perspectives.

To alleviate the expense of manual labeling, we investigate the feasibility of employing AL under the federated framework in Chapter 2.5. As the posterior distribution of neural networks is intractable, we suggest using MC-dropout [23] to approximate the posterior distribution in Chapter 2.6.3. This applies a Monte Carlo sampler to the neural network with dropout several times to approximate the mean and variance of the posterior distribution. Thus, the uncertainty-based acquisition function can be applied to select the most representative instances. The experimental result shows that AL is effective in the distributed environment, i.e. given a target accuracy, AL requires less data or reaches higher accuracy for a given amount of data.

In FL, all the computational tasks are pushed towards the edge devices, and only the simple arithmetic operations are carried out on the server, which corresponds to aggregating the model parameters. The advantage of this approach is to avoid direct data transmission; however, the limited resources on edge devices may prevent local application to more complex problems requiring modeling and analysis. We propose to divide the training process between the edge devices and the server to allow greater flexibility in light of varying task complexity and computational resource availability.

4.2 Reflection and Outlook

Edge Computing (EC) transforms the way we handle and process data, which responds to the decentralized fashion of data generation, i.e. data is generated from millions of devices/machines. The explosive growth of internet-connected devices exacerbates the demand for the new computational scheme. EC enables personalized data analysis on devices, which vastly reduces the volume of data for transmission and traffic, lowering the risk of sensitive information leakage, and shortening the response time [100].

Companies like Siemens promoted the idea of Industrial Edge, integrating the industrial edge into the automation system to make better use of machine data. However, this new computational scheme also introduces challenges, e.g. it exposes a new way for attackers to exploit vulnerabilities. In the near future, we envision that EC-based data analytics will be used mainly in our life, gradually compensating CC or even replacing CC, with the fast development of internet technologies and hardware.

Bibliography

[1] Apple support. http://https://support.apple.com/en-us/HT212025, 2021 (accessed September 6, 2021).

[2] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 308–318, 2016.

[3] Naveed Akhtar, Jian Liu, and Ajmal Mian. Defense against universal adversarial perturbations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3389–3398, 2018.

[4] Syed Mumtaz Ali and Samuel D Silvey. A general class of coefficients of divergence of one distribution from another. *Journal of the Royal Statistical Society: Series B (Methodological)*, 28(1):131–142, 1966.

[5] Yoshinori Aono, Takuya Hayashi, Lihua Wang, Shiho Moriai, et al. Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Transactions on Information Forensics and Security*, 13(5):1333–1345, 2017.

[6] Mohammadreza Barzegaran, Nitin Desai, Jia Qian, Koen Tange, Bahram Zarrin, Paul Pop, and Juha Kuusela. Fogification of electric drives: An industrial use case. In *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 77–84. IEEE, 2020.

[7] Mohammadreza Barzegaran, Nitin Desai, Jia Qian, and Paul Pop. Electric drives as fog nodes in a fog computing-based industrial use case. *The Journal of Engineering*, 2021.

[8] Andrew L Beam and Isaac S Kohane. Big data and machine learning in health care. *Jama*, 319(13):1317–1318, 2018.

[9] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[10] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Machine learning with adversaries: Byzantine tolerant gradient descent. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 118–128, 2017.

[11] Avrim Blum, Cynthia Dwork, Frank McSherry, and Kobbi Nissim. Practical privacy: the sulq framework. In *Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 128–138, 2005.

[12] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16, 2012.

[13] Kamalika Chaudhuri and Claire Monteleoni. Privacy-preserving logistic regression. In *NIPS*, volume 8, pages 289–296. Citeseer, 2008.

[14] Kamalika Chaudhuri, Claire Monteleoni, and Anand D Sarwate. Differentially private empirical risk minimization. *Journal of Machine Learning Research*, 12 (3), 2011.

[15] David A Cohn, Zoubin Ghahramani, and Michael I Jordan. Active learning with statistical models. *Journal of artificial intelligence research*, 4:129–145, 1996.

[16] Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. *arXiv preprint arXiv:1407.0202*, 2014.

[17] Irit Dinur and Kobbi Nissim. Revealing information while preserving privacy. In *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 202–210, 2003.

[18] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

[19] Cynthia Dwork. Differential privacy: A survey of results. In *International conference on theory and applications of models of computation*, pages 1–19. Springer, 2008.

[20] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*, pages 265–284. Springer, 2006.

[21] Fog Computing and Networking Architecture Framework. IEEE 1934-2018 - IEEE Standard for Adoption of OpenFog Reference Architecture for Fog Computing. https://standards.ieee.org/standard/1934-2018.html, 2018 (accessed April 1, 2021).

[22] Ken-Ichi Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural networks*, 2(3):183–192, 1989.

[23] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR, 2016.

[24] Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. Inverting gradients–how easy is it to break privacy in federated learning? *arXiv preprint arXiv:2003.14053*, 2020.

[25] Andrew Gelman, John B Carlin, Hal S Stern, and Donald B Rubin. *Bayesian data analysis*. Chapman and Hall/CRC, 1995.

[26] Andrew Gelman, John B Carlin, Hal S Stern, David B Dunson, Aki Vehtari, and Donald B Rubin. *Bayesian data analysis*. CRC press, 2013.

[27] Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on pattern analysis and machine intelligence*, (6):721–741, 1984.

[28] Robin C Geyer, Tassilo Klein, and Moin Nabi. Differentially private federated learning: A client level perspective. *arXiv preprint arXiv:1712.07557*, 2017.

[29] Avishek Ghosh, Jichan Chung, Dong Yin, and Kannan Ramchandran. An efficient framework for clustered federated learning. *arXiv preprint arXiv:2006.04088*, 2020.

[30] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[31] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[32] Shixiang Gu and Luca Rigazio. Towards deep neural network architectures robust to adversarial examples. *arXiv preprint arXiv:1412.5068*, 2014.

[33] Farzin Haddadpour, Mohammad Mahdi Kamani, Aryan Mokhtari, and Mehrdad Mahdavi. Federated learning with compression: Unified analysis and sharp guarantees. In *International Conference on Artificial Intelligence and Statistics*, pages 2350–2358. PMLR, 2021.

[34] Filip Hanzely and Peter Richtárik. Federated learning of a mixture of global and local models. *arXiv preprint arXiv:2002.05516*, 2020.

[35] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[36] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, 14(8), 2012.

[37] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[38] Neil Houlsby, Ferenc Huszár, Zoubin Ghahramani, and Máté Lengyel. Bayesian active learning for classification and preference learning. *arXiv preprint arXiv:1112.5745*, 2011.

[39] Bunpei Irie and Sei Miyake. Capabilities of three-layered perceptrons. In *ICNN*, pages 641–648, 1988.

[40] Michael I Jordan, Zoubin Ghahramani, Tommi S Jaakkola, and Lawrence K Saul. An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233, 1999.

[41] Sai Praneeth Karimireddy, Quentin Rebjock, Sebastian Stich, and Martin Jaggi. Error feedback fixes signsgd and other gradient compression schemes. In *International Conference on Machine Learning*, pages 3252–3261. PMLR, 2019.

[42] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh. Scaffold: Stochastic controlled averaging for federated learning. In *International Conference on Machine Learning*, pages 5132–5143. PMLR, 2020.

[43] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[44] Andreas Kirsch, Joost Van Amersfoort, and Yarin Gal. Batchbald: Efficient and diverse batch acquisition for deep bayesian active learning. *Advances in neural information processing systems*, 32:7026–7037, 2019.

[45] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[46] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[47] Alexandros Labrinidis and Hosagrahar V Jagadish. Challenges and opportunities with big data. *Proceedings of the VLDB Endowment*, 5(12):2032–2033, 2012.

[48] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521 (7553):436–444, 2015.

[49] Lihua Lei and Michael Jordan. Less than a single pass: Stochastically controlled stochastic gradient. In *Artificial Intelligence and Statistics*, pages 148–156. PMLR, 2017.

[50] Lihua Lei, Cheng Ju, Jianbo Chen, and Michael I Jordan. Non-convex finite-sum optimization via scsg methods. *arXiv preprint arXiv:1706.09156*, 2017.

[51] David D Lewis and William A Gale. A sequential algorithm for training text classifiers. In *SIGIR'94*, pages 3–12. Springer, 1994.

[52] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, pages 583–598, 2014.

[53] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. *arXiv preprint arXiv:1812.06127*, 2018.

[54] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3):50–60, 2020.

[55] Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. On the convergence of fedavg on non-iid data. *arXiv preprint arXiv:1907.02189*, 2019.

[56] Yingzhen Li and Richard E Turner. R\'enyi divergence variational inference. *arXiv preprint arXiv:1602.02311*, 2016.

[57] Shaoshan Liu, Liangkai Liu, Jie Tang, Bo Yu, Yifan Wang, and Weisong Shi. Edge computing for autonomous driving: Opportunities and challenges. *Proceedings of the IEEE*, 107(8):1697–1716, 2019.

[58] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[59] Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

[60] Seonwoo Min, Byunghan Lee, and Sungroh Yoon. Deep learning in bioinformatics. *Briefings in bioinformatics*, 18(5):851–869, 2017.

[61] Ilya Mironov. Rényi differential privacy. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, pages 263–275. IEEE, 2017.

[62] Mehryar Mohri, Gary Sivek, and Ananda Theertha Suresh. Agnostic federated learning. In *International Conference on Machine Learning*, pages 4615–4625. PMLR, 2019.

[63] Milad Nasr, Reza Shokri, and Amir Houmansadr. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *2019 IEEE symposium on security and privacy (SP)*, pages 739–753. IEEE, 2019.

[64] Nebbiolo Technologies, Inc. Nebbiolo. https://www.nebbiolo.tech/, 2021 (accessed April 1, 2021).

[65] Lam M Nguyen, Jie Liu, Katya Scheinberg, and Martin Takáč. Sarah: A novel method for machine learning problems using stochastic recursive gradient. In *International Conference on Machine Learning*, pages 2613–2621. PMLR, 2017.

[66] Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. Smooth sensitivity and sampling in private data analysis. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 75–84, 2007.

[67] Xudong Pan, Mi Zhang, Yifan Yan, Jiaming Zhu, and Min Yang. Theory-oriented deep leakage from gradients via linear equation solver. *arXiv preprint arXiv:2010.13356*, 2020.

[68] Paul Pop, Bahram Zarrin, Mohammadreza Barzegaran, Stefan Schulte, Sasikumar Punnekkat, Jan Ruh, and Wilfried Steiner. The fora fog computing platform for industrial iot. *Information Systems*, 98:101727, 2021.

[69] Carlo Puliafito, Enzo Mingozzi, Francesco Longo, Antonio Puliafito, and Omer Rana. Fog computing for the Internet of Things: A Survey. *ACM Transactions on Internet Technology (TOIT)*, 19(2):1–41, 2019.

[70] Jia Qian and Mohammadreza Barzegaran. A decomposed deep training solution for fog computing platforms (submitted to tec2021 workshop). 2021.

[71] Jia Qian, Sarada Prasad Gochhayat, and Lars Kai Hansen. Distributed active learning strategies on edge computing. In *2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*, pages 221–226. IEEE, 2019.

[72] Jia Qian, Lars Kai Hansen, Xenofon Fafoutis, Prayag Tiwari, and Hari Mohan Pandey. Robustness analytics to data heterogeneity in edge computing. *Computer Communications*, 164:229–239, 2020.

[73] Jia Qian, Hiba Nassar, and Lars Kai Hansen. Minimal conditions analysis of gradient-based reconstruction in federated learning (submitted to neurips workshop). *arXiv preprint arXiv:2010.15718*, 2020.

[74] Jia Qian, Prayag Tiwari, Sarada Prasad Gochhayat, and Hari Mohan Pandey. A noble double-dictionary-based ecg compression technique for ioth. *IEEE Internet of Things Journal*, 7(10):10160–10170, 2020.

[75] Junfei Qiu, Qihui Wu, Guoru Ding, Yuhua Xu, and Shuo Feng. A survey of machine learning for big data processing. *EURASIP Journal on Advances in Signal Processing*, 2016(1):1–16, 2016.

[76] Mark Schmidt, Nicolas Le Roux, and Francis Bach. Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 162(1-2):83–112, 2017.

[77] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681, 1997.

[78] Burr Settles. Active learning. *Synthesis lectures on artificial intelligence and machine learning*, 6(1):1–114, 2012.

[79] Burr Settles and Mark Craven. An analysis of active learning strategies for sequence labeling tasks. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 1070–1079, 2008.

[80] Burr Settles, Mark Craven, and Soumya Ray. Multiple-instance active learning. *Advances in neural information processing systems*, 20:1289–1296, 2007.

[81] H Sebastian Seung, Manfred Opper, and Haim Sompolinsky. Query by committee. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 287–294, 1992.

[82] Uri Shaham, Yutaro Yamada, and Sahand Negahban. Understanding adversarial training: Increasing local stability of neural nets through robust optimization. *arXiv preprint arXiv:1511.05432*, 2015.

[83] Claude Elwood Shannon. A mathematical theory of communication. *ACM SIGMOBILE mobile computing and communications review*, 5(1):3–55, 2001.

[84] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE internet of things journal*, 3(5):637–646, 2016.

[85] Hava T Siegelmann and Eduardo D Sontag. Turing computability with neural nets. *Applied Mathematics Letters*, 4(6):77–80, 1991.

[86] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[87] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet Talwalkar. Federated multi-task learning. *arXiv preprint arXiv:1705.10467*, 2017.

[88] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[89] Shiliang Sun, Zehui Cao, Han Zhu, and Jing Zhao. A survey of optimization methods from a machine learning perspective. *IEEE transactions on cybernetics*, 50(8):3668–3681, 2019.

[90] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[91] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[92] Prayag Tiwari, Jia Qian, Qiuchi Li, Benyou Wang, Deepak Gupta, Ashish Khanna, Joel JPC Rodrigues, and Victor Hugo C de Albuquerque. Detection of subtype blood cells using deep learning. *Cognitive Systems Research*, 52: 1036–1044, 2018.

[93] Florian Tramèr, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. The space of transferable adversarial examples. *arXiv preprint arXiv:1704.03453*, 2017.

[94] TTTech Computertechnik AG. Nerve. `http://tttech.com/products/industrial/industrial-iot/nerve`, 2019 (accessed October 7, 2019).

[95] Paul Voigt and Axel Von dem Bussche. The eu general data protection regulation (gdpr). *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, 10:3152676, 2017.

[96] Hongyi Wang, Scott Sievert, Zachary Charles, Shengchao Liu, Stephen Wright, and Dimitris Papailiopoulos. Atomo: Communication-efficient learning via atomic sparsification. *arXiv preprint arXiv:1806.04090*, 2018.

[97] Zhibo Wang, Mengkai Song, Zhifei Zhang, Yang Song, Qian Wang, and Hairong Qi. Beyond inferring class representatives: User-level privacy leakage from federated learning. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 2512–2520. IEEE, 2019.

[98] Wenqi Wei, Ling Liu, Margaret Loper, Ka-Ho Chow, Mehmet Emre Gursoy, Stacey Truex, and Yanzhao Wu. A framework for evaluating gradient leakage attacks in federated learning. *arXiv preprint arXiv:2004.10397*, 2020.

[99] Zehui Xiong, Yang Zhang, Dusit Niyato, Ping Wang, and Zhu Han. When mobile blockchain meets edge computing. *IEEE Communications Magazine*, 56 (8):33–39, 2018.

[100] Shanhe Yi, Zijiang Hao, Zhengrui Qin, and Qun Li. Fog computing: Platform and applications. In *2015 Third IEEE workshop on hot topics in web systems and technologies (HotWeb)*, pages 73–78. IEEE, 2015.

[101] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

[102] T Zhang and FJ Oles. A probability analysis on the value of unlabeled data for classification problems. 17th icml (pp. 1191–1198), 2000.

[103] Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. idlg: Improved deep leakage from gradients. *arXiv preprint arXiv:2001.02610*, 2020.

[104] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582*, 2018.

[105] Zhi Zhou, Xu Chen, En Li, Liekang Zeng, Ke Luo, and Junshan Zhang. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *Proceedings of the IEEE*, 107(8):1738–1762, 2019.

[106] Ligeng Zhu and Song Han. Deep leakage from gradients. In *Federated Learning*, pages 17–31. Springer, 2020.

[107] Xiaojin Zhu, John Lafferty, and Ronald Rosenfeld. *Semi-supervised learning with graphs*. PhD thesis, Carnegie Mellon University, language technologies institute, school of . . . , 2005.

CONTRIBUTION A

Distributed Active Learning Strategies on Edge Computing

Jia Qian, Sarada Prasad Gochhayat, and Lars Kai Hansen. Distributed active learning strategies on edge computing. In *2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*, pages 221–226. IEEE, 2019

2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/ 2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)

Distributed Active Learning Strategies on Edge Computing

1st Jia Qian, 3rd Lars Kai Hansen
Department of Applied Mathematics and Computer Science
Technical University of Denmark
Lyngby, Denmark
{Jiaq, lkai}@dtu.dk

2nd Sarada Prasad Gochhayat
Virginia Modeling Analysis and Simulation Center
Old Dominion University
Norfolk, VA, USA
Sarada1987@gmail.com

***Abstract*—Fog platform brings the computing power from the remote cloud-side closer to the edge devices to reduce latency, as the unprecedented generation of data causes ineligible latency to process the data in a centralized fashion at the Cloud. In this new setting, edge devices with distributed computing capability, such as sensors, surveillance camera, can communicate with fog nodes with less latency. Furthermore, local computing (at edge side) may improve privacy and trust. In this paper, we present a new method, in which, we decompose the data processing, by dividing them between edge devices and fog nodes, intelligently. We apply active learning on edge devices; and federated learning on the fog node which significantly reduces the data samples to train the model as well as the communication cost. To show the effectiveness of the proposed method, we implemented and evaluated its performance on a benchmark images data set.**

***Index Terms*—Fog Computing, Edge Computing, Active Learning, Federated Learning, Bayesian Neural Network**

I. Introduction

Internet of Thing (IoT) is growing in adoption to become an essential element for the evolution of connected products and related services. To enlarge IoT adoption and its applicability in many different contexts, there is a need for the shift from the cloud-based paradigm towards a fog computing paradigm. In cloud-based paradigm, the devices send all the information to a centralized authority, which processes the data. However, in Fog Computing (FC, [1]) or Edge Computing (EC, [37]) the data processing computation will be distributed among edge devices, fog devices, and the cloud server. FC and EC are interchangeable. This emergence of EC is mainly in response to the heavy workload at the cloud side and the significant latency at the user side. To reduce the delay in fog computing, the concept of fog node is introduced. The Fog Node (FN) [2] is essentially a platform placed between Cloud and Edge Devices (ED) as middleware, and it will further facilitate the 'things' to realize their potentials [3]. This change of paradigm will help application domains, such as industrial automation, robotics or autonomous vehicles, where real-time decision making by using machine learning approaches is crucial.

The research leading to these results has received funding from the European Unions Horizon 2020 research and innovation program under the Marie Sklodowska-Curie grant agreement No. 764785, FORA-Fog Computing for Robotics and Industrial Automation. This work was further supported by the Danish Innovation Foundation through the DanishCenter for Big Data Analytics and Innovation (DABAI).

Convolutional Neural Network (CNN), which is one subtype of the artificial neural network and models the system by observing all the data during training at a single place. Notably, the emergence of CNN [27] introduces the common usage of the neural network as it is more efficient concerning computation cost comparing with (fully connected) neural network. However, to train the model, we need to access all the data, which may create communication bottlenecks and privacy issues [28]. Generally, to train a CNN model, users send all the data to a single machine or a cluster of machines in a data center. However, this sharing operation with the central authority in data centers costs both network usages and breaching of user privacy, since the user doesn't want to share personal, sensitive information, e.g., legally protected medical data.

FL [6] allows the centralized server training models without moving the data to a central location. In particular, FL is used in a distributed way, in which the model is built without direct access to training data, and indeed the data remains in its original location, which provides both data locality and privacy. In the beginning, a server coordinates a set of nodes, each with training data that cannot be accessed by server directly. These nodes train a local model and share individual models with the server. The server uses these individual models to create a federated model and sends the model back to the nodes. Then, another round of local training takes place, and the process continues. Nevertheless, this extra work on edge devices has to be minimized by selecting the most important data samples needed to build the local model. In this context, we want to use Active Learning (AL) as a more effective learning framework. AL chooses training data efficiently when labeling data becomes drastically expensive.

Motivated by the above mentioned, the research issues and possible direction, we propose a new scheme. In literature, there exist some papers that discuss the application of machine-learning algorithm directly on the Fog node platform [30] [31]. As we have already discussed, to efficiently use cloud and fog infrastructure, we need to delegate the work among them. Hence, in this paper, we propose a new efficient privacy-preserving data analytic scheme in Fog environment. We offer using federated learning at the centralized fog devices to create the initial training model. To improve the perfor-

978-1-7281-1661-7/19/$31.00 ©2019 IEEE
DOI 10.1109/CSCloud/EdgeCom.2019.00029

mance further, we recommend using Active Learning at the edge devices, by selecting the sample points effectively. All in all, we propose a possible solution in Edge Computing setting, where the user privacy, training cost, and upload bottleneck are the main issues to address. Our strategy may reduce the training cost by applying AL and preserve the user privacy and reduce the communication by using FL. Moreover, the proof of concept is demonstrated by applying the method on a benchmark dataset.

The remainder of this paper is organized as follows. Section II explains preliminary concepts, in particular, CNN, FL, AL framework, model uncertainty measurement, and previous studies related to our work. In Section III, we introduce the proposed scheme. Section IV covers the details of our experimental design and data collection strategy followed by a discussion on our results. Section V concludes the paper.

II. Preliminary Concepts and Related Work

In this section, we discuss the different techniques that are essential for our scheme. We also present a brief overview of the major research studies related to our work.

A. Convolutional Neural Network

Convolutional Neural Network (CNN) is proposed in [9] for the first time, which addressed the computational problems imposed by the fully-connected neural network, in particular, deep neural network. It's composed by the following layers:

- **Convolutional Layer:** This layer helps in largely reducing the dimension by applying the kernel to the input. For example, if we have images with size m*n as input, with a stride equal to one and no padding, after applying the kernel with size k*k, it ends up a matrix with size $[m-(k-1)]*[n-(k-1)]$.
- **Pooling Layer:** This layer performs non-linear down-sampling. Maximal and average pooling is the most commonly used in this layer.
- **Activation Layer:** The active layer is the non-linear transformation layer, whe e bothrSigmoid aHd nyperbolic can be used in this layer. We use ReLu in our work, which is the abbreviation of rectified linear unit, expressed as:

$$R(x) = \max(0, x) \tag{1}$$

- **Fully-connected Layer:** Here, all the neurons in a fully connected layer connect to all activations in the previous layer. We compute the output for node i at layer l, denoted as p_i^l, using its weight as $w_{.i}^l$ and input from the previous layer as o_j^{l-1}, i.e.,

$$p_i^l = \sum_j o_j^{l-1} * w_{ji}^l \tag{2}$$

CNNs commonly have a huge number of parameters, and it will lead to huge communication costs by sending updates for these many values to a server leads. Thus, a simple approach to sharing weight updates is not feasible for larger models. Since uploads are typically much slower than downloads, it is acceptable that users have to download the current model, while compression methods should be applied to the uploaded data.

B. Federated Machine Learning

Federated learning (FL) is a collaborative form of machine learning where the training process is distributed among many users; this enables to build machine learning systems without complete access to training data [6]. In FL, the data remains in its original location, which helps to ensure privacy and reduces communication costs. The server or the central entity does not do most of the work but only coordinates everything by a federation of users. In principle, this idea can be applied to any model for which the criterion of updates can be defined, which naturally includes the methods based on gradient descent, which nowadays most of the popular models do. For instance, linear regression, logistic regression, neural networks, and linear support vector machines can all be used for FL by letting users compute gradients [33] [34].

Concerning data, FL is especially useful in situations where users generate and label data by themselves implicitly. In such a situation, Federated Learning is very powerful since models can be trained with the massive data that actually is not stored and not directly shared with the server at all. We can thus make use of the massive data that we could otherwise not have used without violating the users' privacy. FL aims to improve communication efficiency and train a high-quality centralized model. The centralized model is trained over the distributed client nodes, which we refer to edge devices in the FC setting. The model is locally trained on every device, and the devices update the refined models to the Fog Node (server node) by sending the parameters of the models. The FC might aggregate the parameters in different ways, for instance, average parameters, choose the best-performed model or sum the weighted parameters.

We define the goal of federate learning to learn a model with parameters embodied in matrix from data stored across a large number of clients (Edge Devices). Suppose the server (FN) distributes the model (at round t) W_t to N clients for further updating, and the updated models are denoted as $W_t^1, W_t^2, W_t^3, ...W_t^N$. Then, the clients send the updated models back to the server, and the server updates the model W according to the aggregated information.

$$W_{t+1} := \sum_i^N \alpha_i * W_t^i \tag{3}$$

Where α can be uniformly distributed or according to the $t-1$ round performance, we use the former one in our work, namely, average the parameters. The learning process can be iteratively carried out.

C. Active Machine Learning

Active learning (AL) is a particular case of machine learning in which a learning algorithm interactively queries the user to obtain the desired outputs at new data points. Typically, AL achieves higher performance with the same number of data samples, using the same method, for example, support

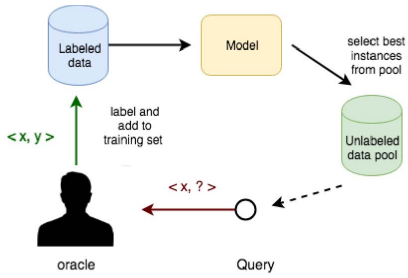


Fig. 1. Pool-based Active Learning Framework.

vector machine, neural network, while with a sophisticated data acquisition [4]. In other words, AL achieves a given performance using less data. Active learning can be divided into two categories: pool-based and stream-based [29]. Pool-based active learning queries the most informative instance from a large pool (See Fig 1), whereas the stream-based one typically draws one at a time from the input source, and the learner must decide whether to query or discard it. In this paper we consider pool-based active learning. The critical point is the way of choosing training data, which is carried out by an interaction between data pool and model: the model strategically picks the new training data according to some specific criteria (refer to Acquisition Function in subsection II-E). The authors in [12] showed that a pool-based support vector machine classifier significantly reduces the needed data to reach some particular level of accuracy in text classification, analogously, [13] for image retrieval application.

Active learning is an appropriate choice when i) labeling data is expensive, or ii) limited data collection. Initially, the researchers fit the machine learning algorithms that mostly work for tabular data to the active learning framework. Recently, it starts registering with the deep neural network, though, it seemingly contradicts with each other as deep neural network typically requires large training data. In the next subsection II-D, we will introduce a vital concept, Bayesian neural network, which is the foundation that AL can work appropriately on image processing by using a neural network.

D. Bayesian Neural Network approximating by Dropout

In this subsection, we will briefly introduce the Bayesian neural network, from which the dropout is applied to approximate the variational inference, proposed by [32]. Bayesian network is defined as placing a prior distribution over the weights of the neural network. Let's define the weights of neurons as $W = (w_i)_{i=1}^{L}$, we are interested in the posterior $p(W|X, Y)$, given all the observations $\langle X, Y\rangle$. As we know, the posterior is intractable integral from [35]. In [32], it is solved by approximation of the real distribution and Monte Carlo. Thus, we define the approximating variational distribution $q(W_i)$ at layer i as:

$$W_i = M_i * \text{diag}([Z_{i,j}]_{j=1}^{K_i}), \tag{4}$$

where

$$Z_{i,j} \sim Bernoulli(p_i, dropout), \text{for} \quad i = 1, ..., L, j = 1, ..K_{i-1} \tag{5}$$

and M_i are variational parameters to be optimized. The diag(.) maps vector to diagonal matrices, whose diagonal are the elements of the vector. K_i indicates the number of neurons in layer i. By given input x and the weights w, which can be sampled from $q(w)$, the predictive distribution of our interest is defined as:

$$p(y^*|x^*, w)q(w)dw \approx \int p(y^*|x^*, w)q(w)dw \approx \frac{1}{T}\sum_{t=1}^{T} p(y^*|x^*, \widehat{w}_t) \tag{6}$$

with $\widehat{w}_t \sim q(w)$, and it is sampled by applying dropout on the corresponding layer. This is referred to as Monte Carlo dropout (MC- dropout).

E. Acquisition Function

The acquisition function is a measure of how desirable the given data point is for minimizing the loss or maximizing the likelihood. In this paper, we are going to use MC-dropout to sample weights and a particular type of uncertainty-based method called *Maximal Entropy* to measure the uncertainty, as it outperforms the others reported in [16].

Uncertainty-based methods aim to use uncertain information to enhance the model during the re-training process. It plays the role of the exploitation while acts as the exploration part. We will introduce three different ways to estimate uncertainty.

– Maximal Entropy: $H[y|x, D_{train}]$ is the predictive entropy expectation as defined in [10].

$$\begin{aligned} H[y|x, D_{train}] := -\sum_c (\frac{1}{T}\sum_{t=1}^{T} p(y = c|x, w_t)) * \\ \log(\frac{1}{T}\sum_{t=1}^{T} p(y = c|x, w_t)) \end{aligned} \tag{7}$$

– BALD: (Bayesian Active Learning by Disagreement [14]) measures the mutual information between data and weights, and it can be interpreted as seeking data points for which the parameters (weights) under the posterior disagrees the most.

$$\begin{aligned} I[y; w|x, D_{train}] := H[y|x, D_{train}] - \frac{1}{T}\sum_t \sum_c \\ - p(y = c|x, w_t) \log p(y = c|x, wu_t) \end{aligned} \tag{8}$$

– Variational ratios: it maximises the variational ratio by considering the followings [15]:

$$V[x] := 1 - \max_y p(y|x, D_{train}) \tag{9}$$

It is similar to Maximal Entropy, but less effective as reported in [16].

III. Proposed scheme

In this section, we discuss the functioning of the different components of the proposed scheme. The primary objective of the proposed scheme is to generate the model by an iterative and distributed way using the fog nodes and edge devices. The overall scheme is shown in Fig. 2, where between edge devices and the cloud server, the fog nodes work as a middleware. Here, a fog node is connected to the edge devices which have a similar task to implement the specific application. For instance, a fog node might be linked to all the surveillance cameras to detect the particular object. Every camera possesses a trained model dispatched by the FN, and it keeps training by the images generated locally under Active Learning framework. It is followed by Federated Learning, namely, the models individually are trained by every device uploading the weights of the (refined) models to FN. The whole process is sketched as follows:

- Firstly, FN trains an initial model "M" using "m" data samples, where "m" is very small which barely helps the model to learn. To generalize, we denote model as M^t, where t is the round.
- FN dispatches the model M_t to the edge devices. For example, let's say, we have four devices, called E_1, E_2, E_3, E_4, and each receives M_t from FN.
- All edge devices implement Active Learning locally with **Maximal Entropy** acquisition function. More specifically, during every acquisition (totally R acquisitions), edge devices train M^t by another "N" $(N > m)$ data samples.
- Next, Edge devices label the new models. In the example, E_1, E_2, E_3, E_4 label their local model as $M_1^t, M_2^t, M_3^t, M_4^t$ respectively.
- Then, the edge devices upload the weights of models $M_1^t, M_2^t, M_3^t, M_4^t$ to the centralized FN.
- FN aggregates the weights either by averaging or choosing the best-trained model, and pass it to next round $t+1$ if necessary.

In our experiments, we set m equal to 20, and we only consider one round. Moreover, we can update the model on the fog node side during every acquisition.

IV. Experiments and Results

In this section, we discuss the experimental setup along with the data set used for our evaluation, and the results of these experiments.

Experiment Setup:
Initially, we trained the CNN model by 20 images at the centralized node (FN), and then send the model to the edge devices. On the devices side, we further trained the model by additional data points. They are acquired by choosing top 10 data samples that have the highest entropy, and this operation will iterate several times. Notably, the model is independently trained by the edge devices. Then it is followed by updating the refined models from all the devices to the centralized FN. The FN will average the parameters of the models, for the future data analysis.

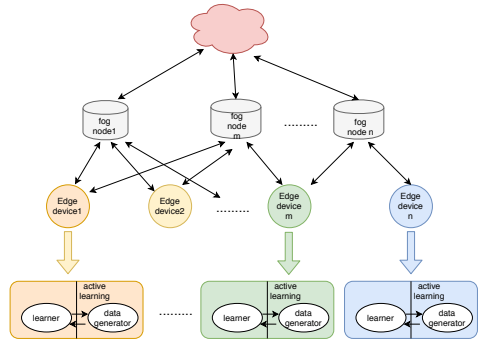


Fig. 2. Overview of the scheme.

All the experiments are implemented by Python language, more specifically, the Pytorch package [36]. The codes are run in Mac Os system (High Sierra) with version 10.13.6, with RAM 16 GB.

Data Set:
We implemented the methods on MNIST dataset [8], which is a real data set of handwritten images, with 60000 for training data and 10000 for the test, totally ten classes. All the images have already been pre-processed, built with the size as $28*28$. It is the basic benchmark to test the performance of approaches in machine learning.

A. Experiment I: random sample vs active learning on edge device

To demonstrate the effectiveness of Active Learning, We compare its performance with randomly chosen data, the experiments are carried out on edge devices, as shown in Fig. 3 and 4 for 10 acquisitions and 20 acquisitions. The outcome of every device with active learning outperforms randomly chosen data points. Notably, here we only want to show that AL has better performance than randomly choosing training samples, not for the sake of the state-of-the-art result.

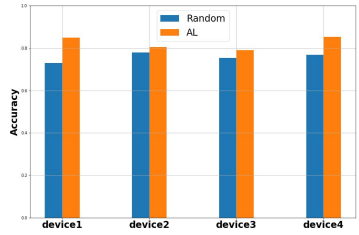


Fig. 3. Active Learning Vs Random Sample (10 Acquisitions).

B. Experiment II: AL acquisition number

In this series of experiment, we study how does the number of training data influence the performance by plotting the

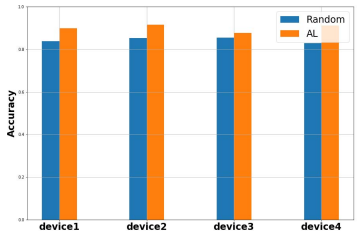


Fig. 4. Active Learning Vs Random Sample (20 Acquisitions).

learning curve. Recall that during every data acquisition, we include ten additional images for further training. Fig. 5, 6, 7 illustrate the learning curve of edge devices for 10, 20, 40 acquisitions accordingly. Here, we run the experiments for five times and we also plot the standard deviations. Again, the training on edge devices are independent; namely, with the different dataset, which conform to the practical situation, the data is generated locally and independently.

Assume that all the data generated from edge devices are from the same distribution. The first observation is every device has its own learning curve as the data at every device is different though they are from the equal distribution. In addition, we build the data pool by randomly choosing 200 images at every iteration of acquisition, from the whole dataset (10000), in order to reduce the computing cost as all the data in the pool are being measured the uncertainty. It is the main reason we run the experiments several rounds to test the real performance of our method.

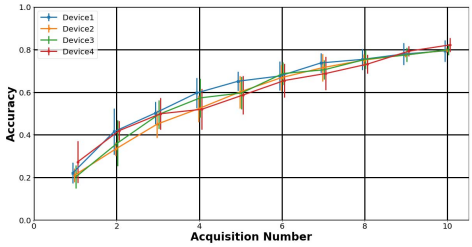


Fig. 5. Learning Curve of Edge Devices for 10 acquisitions of data.

This learning curves of devices are highly related to the aggregating strategy on the FN side. As before we have already discussed the options: choose the optimal model, average the parameters of models from diverse devices or place the weights on the models. Heuristically, when training data size is small, the accuracy between devices vary, thus, picking the optimal model leads a higher accuracy than averaging parameters, and our method mainly shows its strength when the dataset is small. Instead, when the training size is large, though, the learning curves are not necessary the same, they end up with the similar performance, shown in Fig. 7.

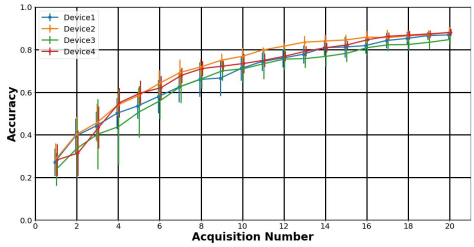


Fig. 6. Learning Curve of Edge Devices for 20 acquisitions of data.

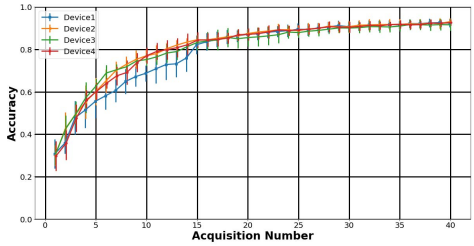


Fig. 7. Learning Curve of Edge Devices for 40 acquisitions of data.

Moreover, We compared the accuracy on FN by applying our approach, with the result obtained by training a dataset with the size triple bigger (4*N) than on every edge device (N) as we have four edge devices in our experiment. For instance, if we train the model by 100 data points on the edge device, then we compare it with the result directly training 400 data samples on FN. The details are shown in Table I, and the columns indicate the different number of acquisitions from data pool, during every acquisition we pick ten images, Acq 10 means the model is further trained by another 100 images on every edge device. For the sake of comparison, we directly trained the initial model with 400 images since we have four devices, every device is trained by 100 images. And then we compared the accuracy with two aggregation strategies: average and optimal model. Note that it is arguable that how many images we train directly on FN to compare since the model is not directly trained by 400 images when we apply FL, we train the model by 100 images on every device. Nevertheless, here we train the model by training data with the size equal to the number locally trained on every device time the number of devices, considering the worst case.
Notably, when the number of edge devices is large, the advantage of AL is not as obvious as the case (4 edge devices) we demonstrated before. Assuming with 1000 data points, if we have 4 edge devices, every one is trained by 250 images, while if we have 20 devices, then every device 'sees' 50 images. As we can guess, in the second case, the centralized device that uses the averaged weights works worse than one machine trained directly by 1000 images ($50 << 1000$). It can be solved by the communication between devices, cascading the

training process, namely, after one device completes training, shares the weights with the close device.

V. Conclusion

In this paper, we for the first time discussed Active Learning in a distributed setting tailored to the so-called Fog platform consisting of distributed edge devices and a centralized fog node. We implemented active learning in edge devices to down scale the necessary training set and reduce the label cost. We presented evidence that it performs similarly to centralized computing with a reduced communication overhead, latency and harvesting the potential privacy benefits. In the future, we will do more experiments on different setting (large number of edge devices) and offering the corresponding solution. In addition, we will study the additional acquisition functions and also address privacy issues in more details.

TABLE I
Fog Node Performance with/without Federated Learning

	Acq 10	**Acq 20**	**Acq 30**	**Acq 40**
FN without FL (No. of train data)	400	800	1200	1600
FN with FL (No. of train data)	100	200	300	400
Accuracy (FN without FL)	0.73	0.882	0.811	0.901
Accuracy (FN with FL (ave))	0.8	0.909	0.881	0.918
Accuracy (FN with FL (opt))	0.854	0.915	0.944	0.963

References

[1] Bonomi, Flavio, et al. "Fog computing and its role in the internet of things." Proceedings of the first edition of the MCC workshop on Mobile cloud computing. ACM, 2012.

[2] Bonomi, Flavio, et al. "Fog computing: A platform for internet of things and analytics." Big data and internet of things: A roadmap for smart environments. Springer, Cham, 2014. 169-186.

[3] Dastjerdi, Amir Vahid, and Rajkumar Buyya. "Fog computing: Helping the Internet of Things realize its potential." Computer 49.8 (2016): 112-116.

[4] Settles, Burr. "Active learning literature survey. 2010." Computer Sciences Technical Report 1648 (2014).

[5] Gal, Yarin, and Zoubin Ghahramani. "Dropout as a Bayesian approximation: Representing model uncertainty in deep learning." international conference on machine learning. 2016.

[6] Konen, Jakub, et al. "Federated learning: Strategies for improving communication efficiency." arXiv preprint arXiv:1610.05492 (2016).

[7] Rasmussen, Carl Edward. "Gaussian processes in machine learning." Advanced lectures on machine learning. Springer, Berlin, Heidelberg, 2004. 63-71.

[8] LeCun, Yann, Corinna Cortes, and C. J. Burges. "MNIST handwritten digit database." $AT\&T$ Labs [Online]. Available: http://yann. lecun. com/exdb/mnist 2 (2010).

[9] LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86.11 (1998): 2278-2324.

[10] Shannon, Claude Elwood. "A mathematical theory of communication." Bell system technical journal 27.3 (1948): 379-423.

[11] Wang, Keze, et al. "Cost-effective active learning for deep image classification." IEEE Transactions on Circuits and Systems for Video Technology 27.12 (2017): 2591-2600.

[12] Tong, Simon, and Daphne Koller. "Support vector machine active learning with applications to text classification." Journal of machine learning research 2.Nov (2001): 45-66.

[13] Tong, Simon, and Edward Chang. "Support vector machine active learning for image retrieval." Proceedings of the ninth ACM international conference on Multimedia. ACM, 2001.

[14] Houlsby, Neil, et al. "Bayesian active learning for classification and preference learning." arXiv preprint arXiv:1112.5745 (2011).

[15] Freeman, Linton C. Elementary applied statistics: for students in behavioral science. John Wiley and Sons, 1965.

[16] Gal, Yarin, Riashat Islam, and Zoubin Ghahramani. "Deep bayesian active learning with image data." arXiv preprint arXiv:1703.02910 (2017).

[17] McMahan, H. Brendan, et al. "Communication-efficient learning of deep networks from decentralized data." arXiv preprint arXiv:1602.05629 (2016).

[18] Konen, Jakub, Brendan McMahan, and Daniel Ramage. "Federated optimization: Distributed optimization beyond the datacenter." arXiv preprint arXiv:1511.03575 (2015).

[19] Konecn, Jakub, et al. "Federated optimization: Distributed machine learning for on-device intelligence." arXiv preprint arXiv:1610.02527 (2016).

[20] Hoi, Steven CH, et al. "Batch mode active learning and its application to medical image classification." Proceedings of the 23rd international conference on Machine learning. ACM, 2006.

[21] Liu, Ying. "Active learning with support vector machine applied to gene expression data for cancer classification." Journal of chemical information and computer sciences 44.6 (2004): 1936-1941.

[22] Thompson, Cynthia A., Mary Elaine Califf, and Raymond J. Mooney. "Active learning for natural language parsing and information extraction." ICML. 1999.

[23] Osugi, Thomas, Deng Kim, and Stephen Scott. "Balancing exploration and exploitation: A new algorithm for active machine learning." Data Mining, Fifth IEEE International Conference on. IEEE, 2005.

[24] Lakshminarayanan, B., Pritzel, A. and Blundell, C., 2017. Simple and scalable predictive uncertainty estimation using deep ensembles. In Advances in Neural Information Processing Systems (pp. 6402-6413).

[25] Osband, I., Blundell, C., Pritzel, A. and Van Roy, B., 2016. Deep exploration via bootstrapped DQN. In Advances in neural information processing systems (pp. 4026-4034).

[26] Gal, Y. and Ghahramani, Z., 2016, June. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In international conference on machine learning (pp. 1050-1059).

[27] LeCun, Y., Haffner, P., Bottou, L. and Bengio, Y., 1999. Object recognition with gradient-based learning. In Shape, contour and grouping in computer vision (pp. 319-345). Springer, Berlin, Heidelberg.

[28] Dwork, C., 2008, April. Differential privacy: A survey of results. In International Conference on Theory and Applications of Models of Computation (pp. 1-19). Springer, Berlin, Heidelberg.

[29] Settles, B., 2012. Active learning. Synthesis Lectures on Artificial Intelligence and Machine Learning, 6(1), pp.1-114.

[30] Tang, B., Chen, Z., Hefferman, G., Wei, T., He, H. and Yang, Q., 2015, October. A hierarchical distributed fog computing architecture for big data analysis in smart cities. In Proceedings of the ASE BigData and SocialInformatics 2015 (p. 28). ACM.

[31] Diro, A.A. and Chilamkurti, N., 2018. Distributed attack detection scheme using deep learning approach for Internet of Things. Future Generation Computer Systems, 82, pp.761-768.

[32] Gal, Y. and Ghahramani, Z., 2015. Bayesian convolutional neural networks with Bernoulli approximate variational inference. arXiv preprint arXiv:1506.02158.

[33] Konecn, J., McMahan, H.B., Ramage, D. and Richtrik, P., 2016. Federated optimization: Distributed machine learning for on-device intelligence. arXiv preprint arXiv:1610.02527.

[34] Hong, D. and Si, L., 2012, August. Mixture model with multiple centralized retrieval algorithms for result merging in federated search. In Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval (pp. 821-830). ACM.

[35] Blei, D.M., Kucukelbir, A. and McAuliffe, J.D., 2017. Variational inference: A review for statisticians. Journal of the American Statistical Association, 112(518), pp.859-877.

[36] Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L. and Lerer, A., 2017. Automatic differentiation in pytorch.

[37] Shi, W., Cao, J., Zhang, Q., Li, Y. and Xu, L., 2016. Edge computing: Vision and challenges. IEEE Internet of Things Journal, 3(5), pp.637-646.

CONTRIBUTION B

Robustness Analytics to Data Heterogeneity in Edge Computing

Jia Qian, Lars Kai Hansen, Xenofon Fafoutis, Prayag Tiwari, and Hari Mohan Pandey. Robustness analytics to data heterogeneity in edge computing. *Computer Communications*, 164:229–239, 2020

Computer Communications 164 (2020) 229–239

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Robustness analytics to data heterogeneity in edge computing

Jia Qian [a], Lars Kai Hansen [a], Xenofon Fafoutis [a], Prayag Tiwari [b], Hari Mohan Pandey [c,*]

[a] Department of Applied Mathematics and Computer Science, Technical University of Denmark, 2800 Lyngby, Denmark
[b] Department of Information Engineering, University of Padova, Italy
[c] Department of Computer Science, Edge Hill University, Ormskirk, United Kingdom

ARTICLE INFO

Keywords:
Intelligent edge computing
Fog computing
Active learning
Federated learning
Distributed machine learning
User data privacy

ABSTRACT

Federated Learning is a framework that jointly trains a model *with* complete knowledge on a remotely placed centralized server, but *without* the requirement of accessing the data stored in distributed machines. Some work assumes that the data generated from edge devices are identically and independently sampled from a common population distribution. However, such ideal sampling may not be realistic in many contexts. Also, models based on intrinsic agency, such as active sampling schemes, may lead to highly biased sampling. So an imminent question is how robust Federated Learning is to biased sampling? In this work[1], we experimentally investigate two such scenarios. First, we study a centralized classifier aggregated from a collection of local classifiers trained with data having categorical heterogeneity. Second, we study a classifier aggregated from a collection of local classifiers trained by data through active sampling at the edge. We present evidence in both scenarios that Federated Learning is robust to data heterogeneity when local training iterations and communication frequency are appropriately chosen.

1. Introduction

Federated Learning [1] is a promising method to enable edge Intelligence and data protection at the same time. FL is of significant theoretical and practical interest. From a theoretical point of view, Federated Learning poses challenges in terms of, e.g., consistency (do distributed learning lead to the same result as centralized learning) and complexity (how much of the potential parallelism gain is realized). From a practical point of view, Federated Learning offers unique opportunities for data protection. In particular, Federated Learning can be realized without "touching" the training data, but rather the data remains in its generation location, which provides the opportunity to secure user privacy. It is very intrinsic to bring it to IoT application, in particular, when 5G is arriving. For instance, FL is exerted in industrial IoT (IIoT) to predict electric drivers' maintenance in the fog computing platform [2]. The medical data collected from distributed individuals can be processed locally and share the metadata with the central server at some point to protect personal privacy [3]. Extending FL to other machine learning paradigms, including reinforcement learning, semi-supervised and unsupervised learning, active learning, and online learning [4,5] all present interesting and open challenges. Some works assume that data is **Independent** and **Identically Distributed** (IID) on the edge devices, which is evidently a strong assumption, say in a privacy-focused application. Users are not identical; hence, we expect locally generated dataset to be the result of idiosyncratic sampling, namely, biased. We believe that data diversity is not necessarily harmful in terms of performance, which mainly attributes to the aggregation step of FL, with the condition that local training iterations and batch size are appropriately opting. A high-level depiction of this scenario is presented in Fig. 1.

To investigate the robustness of FL, we consider two types of Non-IID cases: Type i we will simulate a highly biased data-generation environment, edge devices have access only to a subset of the classification classes (no overlap between them); Type ii on the edge devices, we employ AL as an active sampler to sample the most representative instances, rather than uniform sampling.

1.1. Contribution

Our contribution can be summarized as:

- In general, we aim to investigate the relationship between distributed data diversity and centralized server performance in the edge computing environment.
- More specifically, we simulate two types of biased data generation to study the robustness of FL to different unbalanced data generation level.

* Corresponding author.
E-mail addresses: jiaq@dtu.dk (J. Qian), lkai@dtu.dk (L.K. Hansen), xefa@dtu.dk (X. Fafoutis), prayag.tiwari@dei.unipd.it (P. Tiwari), pandeyh@edgehill.ac.uk (H.M. Pandey).

[1] https://github.com/jiaqian/robustness_of_FL

https://doi.org/10.1016/j.comcom.2020.10.020
Received 20 May 2020; Received in revised form 16 September 2020; Accepted 27 October 2020
Available online 2 November 2020

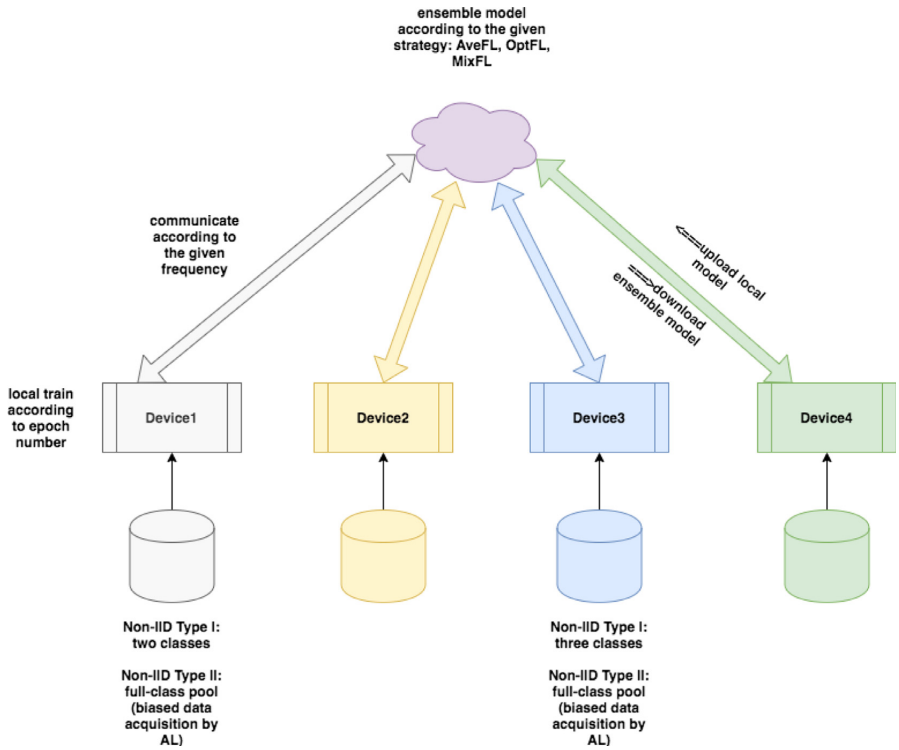


Fig. 1. Federated learning scheme.

- Our experiments show that centralized server performance is highly correlated to the local training time and communication frequency. The divergent aggregation might happen if they are not appropriately chosen.
- Finally, we investigate the effects of parameter (gradient) aggregation by comparing local neural networks activation patterns and aggregated neural networks, which shows the evidence that the server's classification capability is "inherited" from distributed devices through aggregation.

1.2. Organization

The remainder of this paper is organized as follows: Section 2 we will explain the preliminary concepts and introduce the related work, in Section 3, we will give the specific introduction of our scheme. In Section 4 the details of our experimental results will be recovered. Section 5 we will conclude the paper.

For the convenience of readers, we list all the abbreviations and annotations (see Table 1).

Table 1
Abbreviations & Annotations.

Abbreviations	Full Name
FL	Federated learning
Al	Active Learning
IID	Independent Identical Distributed
AveFL	Average Federated Learning
OptFL	Optimal Federated Learning
MixFL	Mixed Federated Learning
Mc-drop	Monte Carlo dropout
Non-IID Type i	Active Learning sampler
Non-IID Type ii	Non overlap between categories

2. Preliminaries and related work

2.1. Federated learning

FL uses a coordinated fashion to train a global model by dynamically collecting models from distributed devices for some rounds. It was first proposed by [1] for the user privacy consideration in mobile networks, and it is a very practical framework in edge computing. [6] employs FL to detect attacks in a distributed system, [7] predicts model uncertainty by a deep aggregated model, and [8] aims to optimize the structure of neural network in FL. Some FL-based applications assume the data is IID on edge devices. [9] considers Non-IID data, but it focuses on the observation that accuracy reduction caused by Non-IID is correlated to weight diversity. Our work extends it, studying two types of Non-IID data: (i) Type i we will simulate a highly biased data-generation environment, whereby edge devices can only generate their categories without any overlap between devices, (ii) Type ii whereby we employ AL on the edge devices as an active sampler to simulate a slightly biased data generation.

2.2. Active learning of neural networks

Labeling is challenging and expensive when data generation increases exponentially. Thus, when intelligence sits close to edge users, it is therefore natural to utilize the interaction between machines and users/humans. We combine Federated Learning (FL) and Active Learning (AL) as Non-IID Type ii , and we reported the prototype

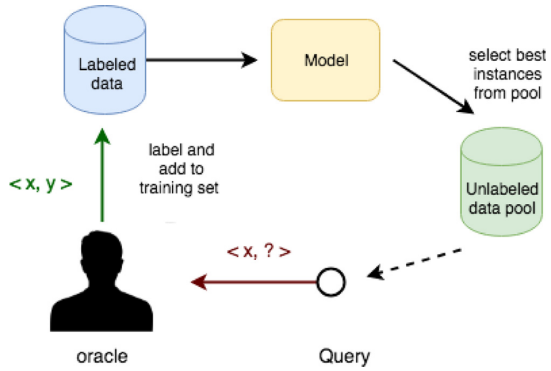


Fig. 2. Pool-based active learning scheme.

in [10]. Theoretically, AL may achieve one of the following situations: higher accuracy with the same amount of data or with a given performance using fewer data. According to the formula of incoming data, it can be grouped as pool-based and stream-based. The stream-based AL approach is used when the data arrives in a stream way, and the model must decide whether to query from the "Oracle" or discard it.

The pool-based approach (Fig. 2) is composed of an initially trained model, an "Oracle", an unlabeled data pool, and a small labeled dataset. More specifically, the initially trained model elaborately opts for some representative samples out from the unlabeled pool based on the acquisition function. After that, it asks the oracle to label them and includes the labeled ones to the training set for future training. We can repeat such operations for several times. In the previous work [10,11], we train our model whenever lately-labeled data is added, along with the old (labeled) data. In this paper, we consider Online Active Learning, which means we immediately discard the data after training (more details in Section 3.3), but with a negligible small subset shared across the devices. To apply AL on a neural network, we firstly build a Bayesian Neural Network (BNN), which can be considered as a model that outputs different values for the same input fed in the model several times. There is no analytical form of the posterior distribution in the neural network; typically a surrogate distribution q is introduced to approximate it by minimizing the distance between them. We implement it through a free ride Dropout [12], feeding the same input multiple times to approximate a distribution with certain mean and variance. It can be proved that running dropout is approximated to apply Bernoulli prior on the model parameters. More details can be found in [10].

2.3. Boosting approach

Boosting is designed to improve any machine learning method, e.g., tree-like classifiers, by aggregating many weak learners through bias and variance reduction [13,14]. The approach of the present work can be related to boosting by viewing the aggregation as a combination of 'weak' (specialized) edge models in repeated steps of the federation. In [15], the authors proposed the Boosting Gradient Classifier, which has a set of weak learners and sets off by creating a weak learner, and it keeps increasing after every iteration. The set of learners is built by randomly combining features. It seeks an appropriate combination $\hat{F}$ of f_i such that approximates the true F, expressed as $\hat{F}(x) = \sum_{i=1} \beta_i f_i(x)$. Apart from computing gradients during training, it also computes the second-order derivative to decide the learning rate. Instead, our method keeps the number of weak learners constant, which is the number of edge devices. Analogously, we can also make it dynamic, like boosting gradient classifiers. Another difference is that we do not compute the second-order derivatives to decide the learning rate; instead, we empirically choose one as the Neural Network has a large amount of parameters. The Boosting Gradient classifier typically has a good performance in conventional machine learning application [16–18]. The main steps of Boosting Gradient are as follows:

- if $m = 0$, we output the prediction by average the outcomes from the weak learners $\hat{F}(x) = \frac{1}{n}\sum_{i=1}^{n} y_i$.
- For iteration m from 1 to M (the case $m \neq 0$):
  - model in iteration m defines as

    $$W_m = W_{m-1} + \gamma_m g_m(x)$$

  - γ_m and $g_m(.)$ are computed separately by the first and second order of loss function L.

    $$g_m(x) = -[\frac{\partial L(y_i, \hat{F}(x, W_m))}{\partial W_m}]_{W=W_{m-1}}$$

    $$\gamma_m = [\frac{\partial^2 L(y_i, \hat{F}(x_i, W_m))}{\partial W_m^2}]_{W=W_{m-1}}$$

  F_{m-1}: the collection of learners up to stage m-1.
  γ_m: the learning rate in iteration m.
  $g_m(x)$: gradient in stage m.

In summary, both Boosting Gradient classifier and FL attempt to improve the performance by assembling a set of weak models. The Boosting Gradient classifier works on a dataset with extracted features, specifically, optimize the learning rate and keeps the number of weak learners increasing; whereas we design federated learning for neural network, the learning rate is empirically decided due to the computation problem and the size of models is constant, we can make it dynamic though.

2.4. Other works

Data non-IID was introduced in [9], and they tackle it by introducing a relatively small global subset that may somehow capture the whole distribution, shared across all devices. Similarly, [19] suggests using data distillation to extract a low-dimension (or sparse) representation of the original data. However, it is computationally expensive; in particular, it is typically carried out at the edge side where only little computation resources can be offered. [20] converts non-IID data distribution as an advantage by considering it as a multi-task optimization, which conforms to our conclusion. Furthermore, [21] utilizes distributionally robust optimization to minimize the worst-case risk over all the distributions close to the empirical distribution.

3. Proposed scheme

3.1. Federated learning aggregation strategies

More specifically, let us assume the server shares the model (at round t) W_t with n devices for their local updating, and the updated models are denoted as $W_t^1, W_t^2, W_t^3, \ldots, W_t^n$. Then, the devices upload the improved models to the server, and the server outputs the aggregated model according to the following criterion:

$$W_{t+1} := \sum_{i}^{n} \alpha_i * W_t^i \tag{1}$$

The combination weights α_is can be uniformly distributed or determined to reflect network performance. The former is referred to as *AveFL* (Algorithm 1). The learning process is iterative. We also consider the second scheme, where we opt for the highest-accuracy model, namely, set α^* of the best model equal to one, and the rest to zero, labeled as *OptFL* (Algorithm 2). In Section 4, we evaluate the schemes and a combination of *AveFL* and *OptFL*, named as *MixFL*. The latter selects the best model of the former two (Algorithm 3).

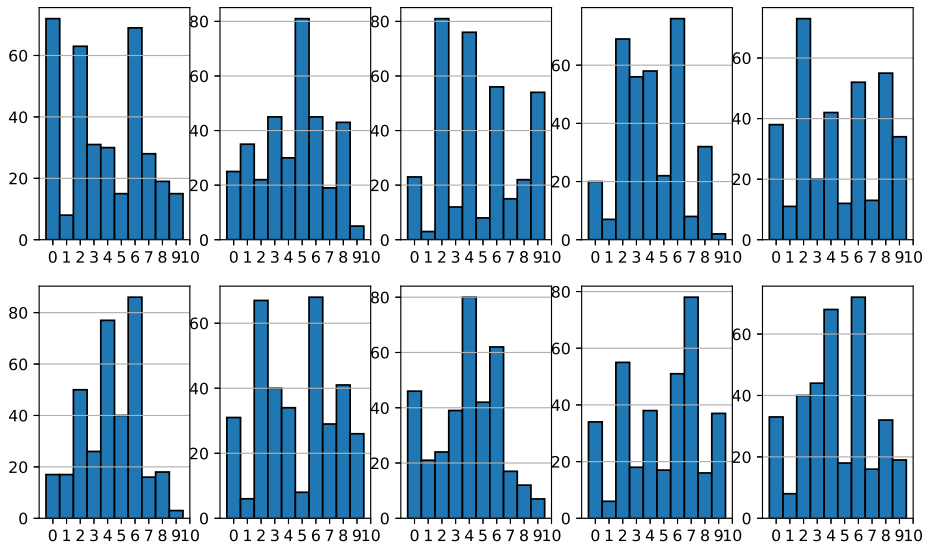


Fig. 3. Biased Data Acquisition by Active Learning: we demonstrate the distribution of data acquired by AL for 10 acquisitions. They are unbalanced in different ways for every acquisition.

Algorithm 1 *AveFL*

1: Input: W_j^t: local models at round t
2: Output: aggregated model W^t
3: $W^t = \frac{1}{n}\sum_{j=1}^{n} W_j^t$

Algorithm 2 *OptFL*

1: Input: W_j^t: local models at round t, $A(.)$: measure accuracy
2: X: test dataset
3: Output: aggregated model W^t
4: $W^t = \text{argmax}\ A(\text{BNN}(X, W_j^t))$ for $j = 1, 2, .., n$

Algorithm 3 *MixFL*

1: Input: local models at round t W_j^t
2: Output: aggregated model W^t
3: $W_{\text{ave}}^t = AveFL(W_j^t)$
4: $W_{\text{opt}}^t = OptFL(W_j^t)$
5: $\text{acc}_{\text{ave}} = A(\text{BNN}(X, W_{\text{ave}}^t))$
6: $\text{acc}_{\text{opt}} = A(\text{BNN}(X, W_{\text{opt}}^t))$
7: **if** $\text{acc}_{ave} >= \text{acc}_{opt}$ **then**
8: Return W_{ave}^t
9: **else**
10: Return W_{opt}^t

Rather than aggregating the weights of models in Eq. (1), we can also work on the gradients. We conclude that one-batch weight average is equal to gradient average. Suppose we have n devices, and training data D ($|D| = N$) is sectioned into n parts as $D_1, D_2, \ldots, D_n$, $|D_1| = N_1, |D_2| = N_2, \ldots |D_n| = N_n$. The corresponding weights inferred from D_i is W_i. Then we define a cost function $G(D) = \sum_{i=1}^{N} g(\widetilde{y}_i, y_i, w)$ and the initial model is W_0, β is the learning rate. We first define average one-batch weights of models as shown in Eq. (3), notably, the local update of edge devices is after one batch (no iteration of the batch), otherwise it is **not** W_0 in cost function $g(.)$. The gradient aggregation is defined in Eq. (4).

$$\sum_{i=1}^{n} \alpha_i = 1 \tag{2}$$

Aggregation Weights:

$$\begin{aligned} W &:= \sum_{i=1}^{n} \alpha_i (W_0 + \beta G(D_i)) \\ &= \sum_{i=1}^{n} \alpha_i \underbrace{(W_0 + \beta \frac{1}{N_i} \sum_{j=1}^{N_i} g(\widetilde{y}_j, y_j, \mathbf{W_0}))}_{\text{updated W from device i}} \\ &= W_0 \sum_{i=1}^{n} \alpha_i + \beta \sum_{i=1}^{n} \alpha_i \frac{1}{N_i} \sum_{j=1}^{N_i} g(\widetilde{y}_j, y_j, W_0) \\ &= W_0 + \beta \sum_{i=1}^{n} \alpha_i \frac{1}{N_i} \sum_{j=1}^{N_i} g(\widetilde{y}_j, y_j, W_0) \end{aligned} \tag{3}$$

Aggregation Gradients:

$$\begin{aligned} W &:= W_0 + \beta * (\sum_{i=1}^{n} \alpha_i G(D_i)) \\ &= W_0 + \beta (\sum_{i=1}^{n} \alpha_i \underbrace{\frac{1}{N_i} \sum_{j=1}^{N_i} g(\widetilde{y}_j, y_j, W_0)}_{\text{gradient of device i}}) \end{aligned} \tag{4}$$

In each iteration, keeping W_0 the same for all edge devices is a mandatory step; otherwise, weight divergence can occur. If initial models are different, they might be placed in a different low-cost region of the cost landscape. Thus, after the average aggregation step, it might be sub-optimal. In this paper, we also aim to investigate how the number of local training influences the result, and we decide to work on the weights aggregation for the sake of convenience.

3.2. Method for non-IID type i

Our approach can be divided into two stages: local learning and aggregation. The two stages will be iterated in one round.

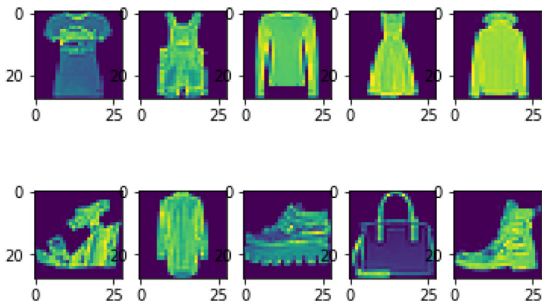


Fig. 4. Fashion MNIST dataset: 10 classes, every image has 28 × 28 pixels.

Algorithm 4 Non-IID Type i

Input: $X_1 \cup X_2 \cup X_3, .. \cup X_n$, X_{ini}, W^0
Return: model $f(W)$
if t==0 **then**
  $W^1 \leftarrow W^0 - \alpha \nabla f(X_{\text{ini}}; W^0)$
else
  for t=1,2,..T **do**
    =>**Devices:**
    for j=1,2,..,n **do** n devices (in Parallel)
      random sample from X_j: $D_j^t \sim X_j$
      $W_j^{t+1} = W_j^t - \alpha \times \nabla f(D_j^t; W_j^t)$
    end
    =>**Server:**
    Aggregation: $W^{t+1} = AveFL(W_j^{t+1})$ for j=1,2,...,n
  end
end

1. **Initialization:** At initialization, the centralized server trains an initial model W^0 with m data samples. More general, we define the model as W^t, where t indicates the current round number.
2. **Sharing:** Server shares the model W^t with n activated edge devices $d_1, d_2, \ldots, d_n$.
3. **Local Training:** All edge devices implement local training and update their models $W_1^t, W_2^t, \ldots, W_n^t$. This step incorporates one or multiple cycles of data acquisition.
4. **Aggregation:** Edge devices transmit their corresponding models to server and the server aggregates $W_i^t, i = 1, 2.., n$ to get W^{t+1}. The aggregation could be *AveFL*, *OptFL* or *MixFL*.
5. Repeat steps 2–4 if necessary.

Algorithmically, it is described in Algorithm 4.

3.3. Method for non-IID type ii

We consider FL with AL as Non-IID Type ii since AL samples a subset of data with higher uncertainty, which leads to biased samples. First, we divide the whole training set into four parts, one part for one edge device. Then we build a pool with 4000 images randomly sampled from one part for the computation consideration since we need to measure the uncertainty of every data point in the pool. The pool is almost balanced; however, the batches generated by the active sampler is unbalanced, one example of 10 acquisitions shown in Fig. 3. For scalability, in this paper, we perform an online AL. Namely, the model is further trained only by the new batch, without the access of the old data (except small subset with 50 images), which is different from the previous work that we train all the data from scratch whenever new data is coming. We try to alleviate the forgetting problem of online learning by a cheap trick, storing 50 images, a balanced set (5 images per class) and will be combined with a new batch to train the model. After completing current-round training, we dump the new batch and only keep 50 images in the labeled set. Moreover, we also use weight decay [22] as a regularizer that prevents the model from changing too much. We define it in Eq. (5), $E(.)$ is the cost function, w^t is the model parameter at round t and λ is a parameter governing how strongly large

Table 2
Neural Network Architecture.

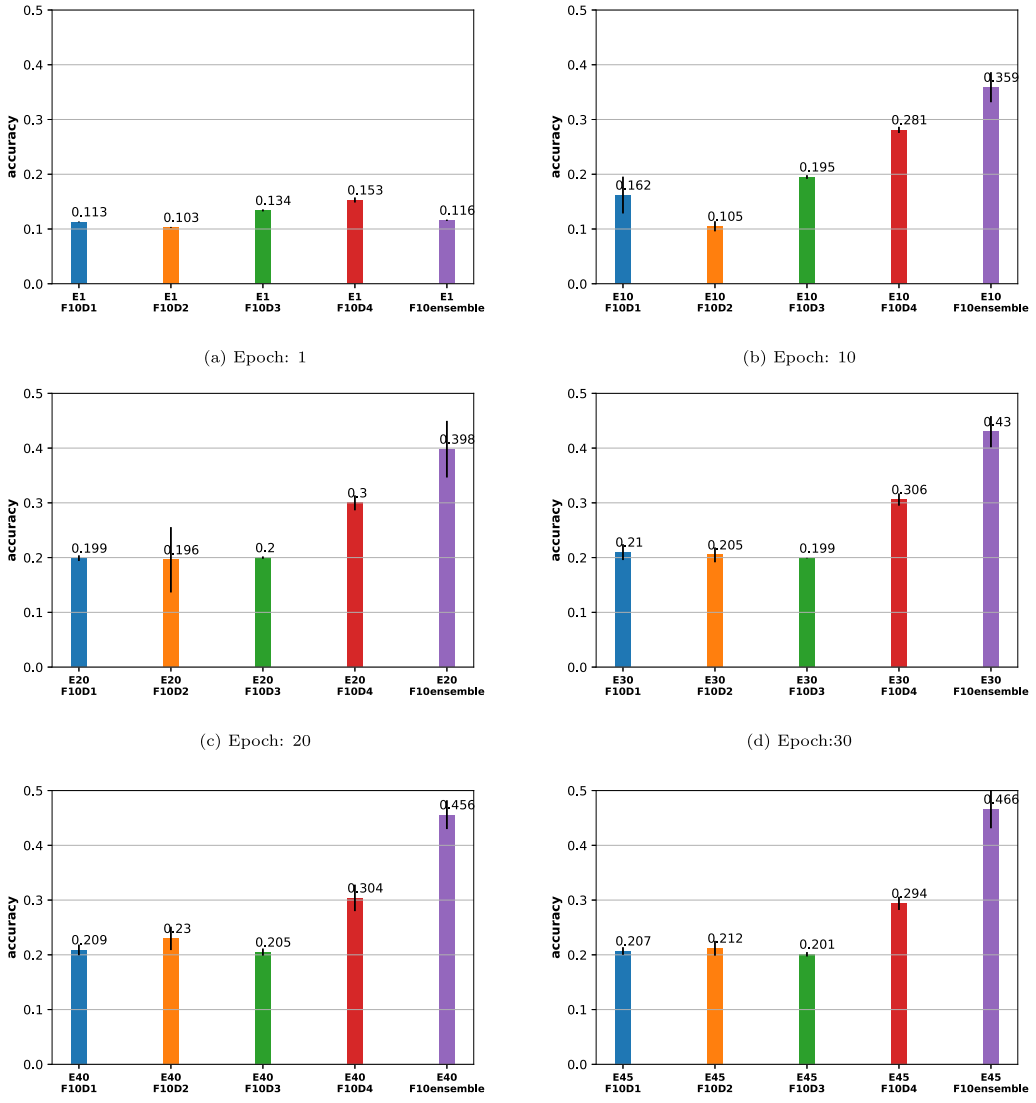
Layer	Layer name	Output channels or number of nodes	Kernel size
1	Conv2d	64	4 × 4
2	ReLu	–	–
3	Conv2d	16	5 × 5
4	ReLu	–	–
5	Max Pooling	–	2 × 2
6	Dropout	–	0.25
7	conv2d	32	4 × 4
8	ReLu	–	–
9	conv2d	16	4 × 4
10	ReLu	–	–
11	Max Pooling	–	2 × 2
12	Dropout	–	0.25
13	Linear	128	–
14	ReLu	–	–
15	Dropout	–	0.5
16	Output	10	–

Algorithm 5 Non-IID Type ii

Input: $X_1 \cup X_2 \cup X_3, .. \cup X_n$, X_{ini}, W^0, k
Return: model $f(W)$
if t==0 **then**
  $W^1 \leftarrow W^0 - \alpha \nabla f(X_{\text{ini}}; W^0)$
else
  for t=1,2,..T **do**
    =>**Devices:**
    for j=1,2,...,n **do** n devices (in Parallel)
      $\log p_j, p_j = \text{BNN}(f_j(W^t), x_j)$
      compute entropy: $S_j = -p_j \times \log p_j$
      sort in descending order and pick top k: $D_j^t = \text{sort}(S_j)[k]$
      $W_j^{t+1} = W_j^t - \alpha \times \nabla f(D_j^t; W_j^t)$
    end
    =>**Server:**
    Aggregation: $W^{t+1} = AveFL(W_j^{t+1})$ for j=1,2,...,n
  end
end

Algorithm 6 *Bayesian Neural network* (BNN)

Input: $f_i(W^t), x_i$
Return: $\log \overline{p}, p$
$s = 0$
for g = 1,2,..r **do**
  $p = f_i(x_i; W^t)$
  $s += p$
end
$\overline{p} = \frac{1}{r} \times s$



(a) Epoch: 1

(b) Epoch: 10

(c) Epoch: 20

(d) Epoch:30

(e) Epoch: 40

(f) Epoch: 45

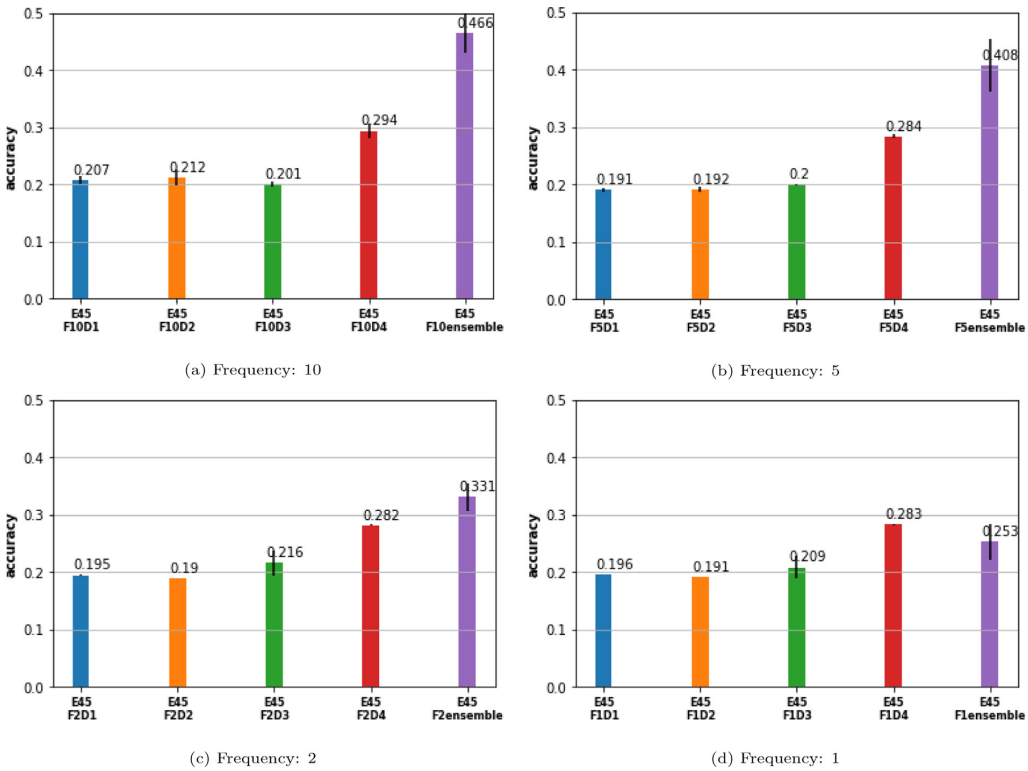
Fig. 5. Epoch Analysis: The various cases are labeled using the format 'ExFyDz', where 'E' is the epoch, 'F' is the aggregation frequency, and 'D' is the device identifier or the aggregated model respectively. For a given batch, the number of epochs during local training highly influences the aggregation performance. The experimental results show that we should ensure sufficient difference between local models to enable the aggregation effect, which is also related to divergence study in the following experiment.

weights are penalized.

$$E(w^{t+1}) = E(w^t) + \frac{1}{2}\lambda \sum_i (w_i^t)^2 \tag{5}$$

[23] learns the weights that can mostly approximate the distribution of the data from the pool by solving an optimization problem. It is highly computation-demanding and not suitable for edge devices. Another work [24] attempts to avoid forgetting by dividing the NN architecture into parts and assigning them to different edge devices, but it requires restricting synchronization during aggregation. Our Non-IID Type ii method is sketched as:

1. **Initialization:** In the beginning, a centralized server trains an initial model W^0 using m data samples. Without the loss of generality, we denote the model by W^t, where t is the current round.
2. **Sharing:** The central server shares the model W^t to n activated edge devices $d_1, d_2, \ldots, d_n$.
3. **Local Training:** All edge devices implement AL on a Bayesian Neural Network approximated by Dropout [12], locally train and update their models $W_1^t, W_2^t, \ldots, W_n^t$. This step incorporates one or multiple cycles of data acquisition.
4. **Aggregation:** Edge devices transmit their corresponding models to server and the server aggregates $W_i^t, i = 1, 2.., n$ to get W^{t+1}.



(a) Frequency: 10 (b) Frequency: 5

(c) Frequency: 2 (d) Frequency: 1

Fig. 6. Aggregation Frequency Analysis: The various cases are labeled using the format 'ExFyDz', where 'E' is the epoch, 'F' is the aggregation frequency, and 'D' is the device or the aggregated model respectively. For a given Epoch 45 and 10 acquisitions of data, we plot the performance with respect to different aggregation frequencies High aggregation frequency has higher accuracy, increasing the cost of communication, and vice versa.

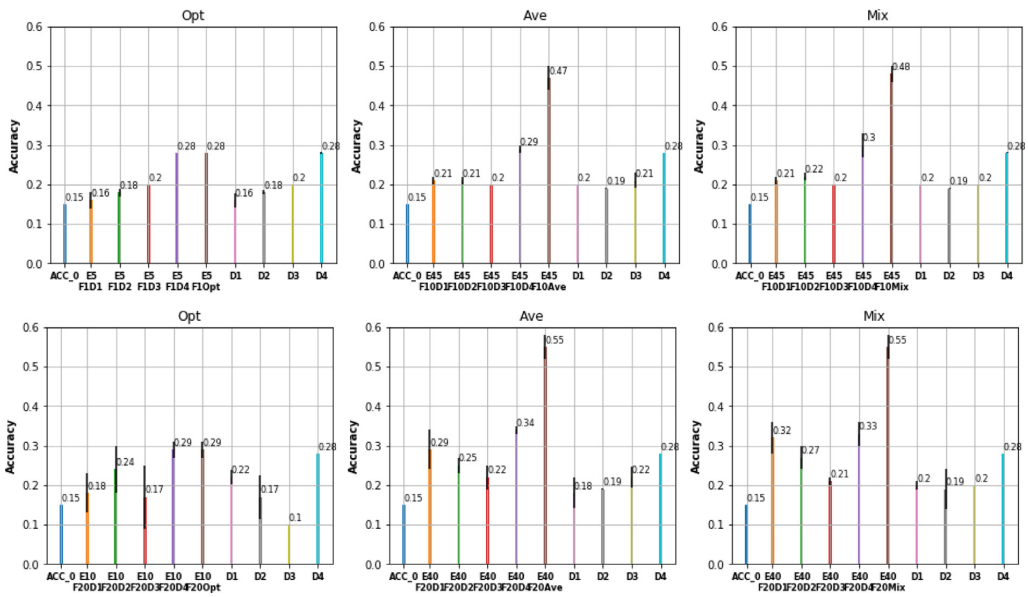


Fig. 7. Aggregation strategies: We compare accuracy with different aggregation strategies, namely *AveFL*, *OptFL* and *MixFL* (top: 10 acquisitions, bottom: 20 acquisitions). The various cases are labeled using the format 'ExFyDz', where 'E' is the epoch, 'F' is the aggregation frequency, and 'D' is the device or the aggregation model respectively. ACC_0 is the initial accuracy. The rightmost four bars are the performance by the independent local models without considering aggregation.

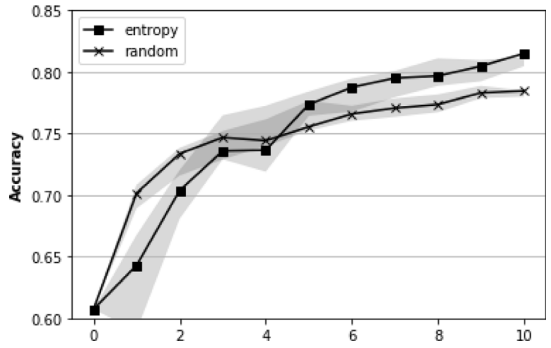


Fig. 8. Given the same number of data, AL outperforms random choice in terms of accuracy.

The aggregation step could entail the average, performance-based or mixed mechanisms.

5. Repeat steps 2–4 if necessary.

The specific algorithm is described in Algorithm 5.

3.4. Architecture

Our model consists of four convolutional layers, one fully-connected layer and a softmax layer shown in Table 2. Note that we did not use batch normalization [25] in the architecture since the biased batch normalization has a deleterious effect on the aggregation performance. Mathematically, it defines as shown in Eqs. (6) and (7). Suppose we have a batch $B = \{x_j\}_{j=1,2..,m}$, then it is normalized by its mean μ_B and variance σ_B (computed in Eq. (6)), and then we infer a new mean (β) and variance (γ) during training process. It may reduce the internal covariate shift and speed up the training procedure to form new representation of data (Eq. (7)).

In the Non-IID case, the means (β) and variances (γ) optimized in the local training stage are decided by their biased data, and it is not beneficial during the aggregation stage in our experiments. It is very critical to enable aggregation effect in highly biased data generation; otherwise, the aggregated model performs very poorly (e.g., 20% accuracy with batch normalization and 47% otherwise).

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i, \sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2 \tag{6}$$

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \eta}}, y_i = \gamma\hat{x}_i + \beta \tag{7}$$

4. Experimental results

4.1. Real dataset

Fashion-MNIST (shown in Fig. 4) is one benchmark image dataset published by Zalando, as the alternative of MNIST dataset. It is formed by a training set with 60,000 examples and a test set with 10,000 examples and 10 classes. One image has 28 × 28 pixels for width and height and one channel. Each pixel value ranges between 0 and 255, indicating the shades of gray.

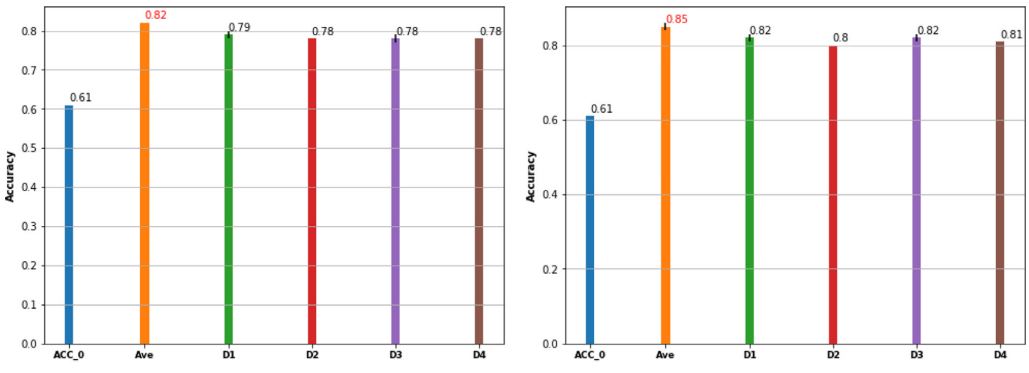
4.2. Non-IID type i

We first evaluate the case of Non-IID Type i : we have a ten-classes dataset and four edge devices (D1, D2, D3, and D4), randomly assign two classes to two devices and three classes to another two devices without overlap. More specifically, class 0 and 1 were assigned to D1, class 2 and 3 to D2, class 4, 5, and 6 to D3, and 7, 8, 9 to D4. Note, if we train a single neural network sequentially: first on the subset of classes 0 and 1, then on classes 2,3, next 4, 5, 6, and finally 7, 8, 9, the model would suffer catastrophic forgetting. It will forget most of the patterns learned before, and capable of classifying the class corresponding to the last subset (around 28%).

4.2.1. Epochs

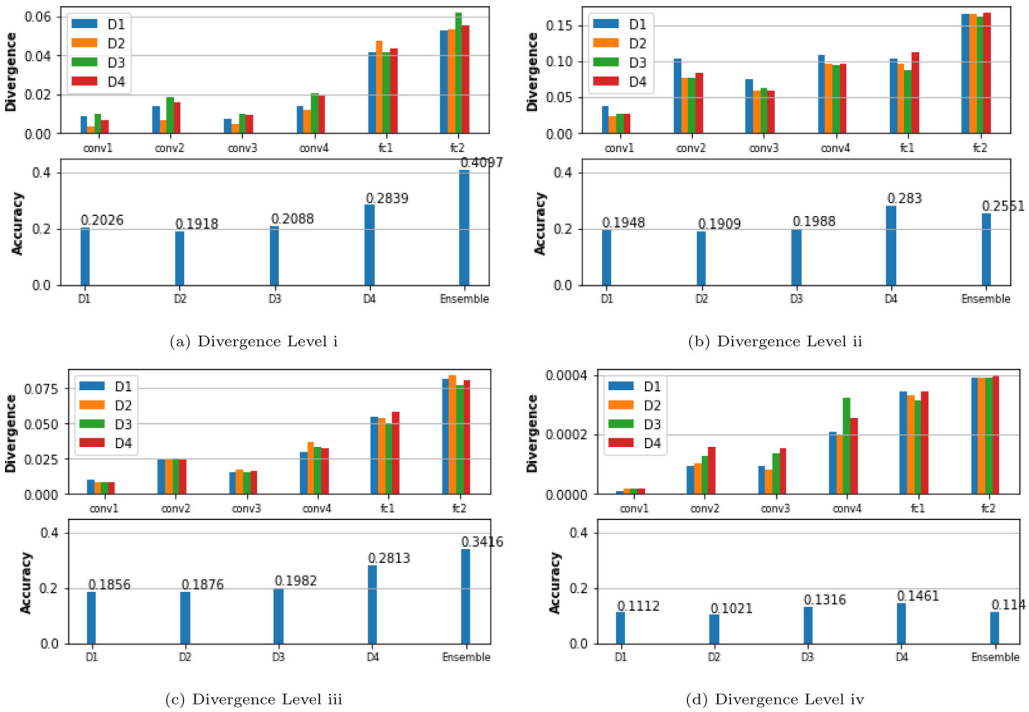
One of the most critical hyper-parameters is the amount of local training before aggregation on the centralized server. In this work, we redefine the concept of 'epoch' since it usually refers to the number of times the learning algorithm will work through the whole training dataset. Here we consider mini-batch gradient descent; thus, 'epoch' refers to the number of times the algorithm goes through the mini-batch.

As shown in the Algorithm 4, at the beginning of every round, all the devices have the same model W^t, the number of epochs will decide how much variance between updated models $W_1^t, W_2^t, \ldots, W_i^t$, produced by one batch (or multiple batches, determined by aggregation frequency that we will discuss later). If the epoch number is not big enough, the performance after aggregation will not be improved significantly or even be worse. In Fig. 5, we plot the accuracy of four distributed models and the aggregated model. Among them, the leftmost four bars represent the accuracy of local models, and the rightmost one is the accuracy of the aggregated model. Note the initial accuracy is 15%.



(a) Random (b) Entropy

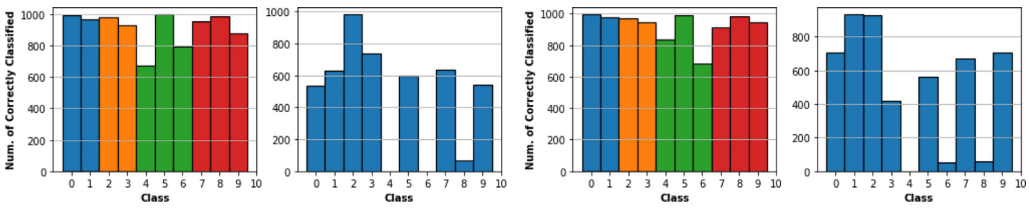
Fig. 9. From left to right we plot initial accuracy, aggregation accuracy and four model accuracy without aggregation step. First of all, no matter random or AL, the result of aggregated model has higher accuracy compared with no aggregation. Overall, AL has better performance with respect to random choice (from the second bar to the last bar).



(a) Divergence Level i (b) Divergence Level ii

(c) Divergence Level iii (d) Divergence Level iv

Fig. 10. Plot above represent the divergence and plot below is the corresponding accuracy of local models and aggregated model. X coordinator of plot (above) indicates the layers of model and bars with different colors represent the different devices. Aggregation Effect has high correlation with divergence grade. Here we compared four level of divergence, From Figs. 10(a) to 10(d) the divergence decreases, Fig. 10(c) corresponds to best aggregation performance. **Note that y coordinators are not aligned due to the different magnitudes**. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)



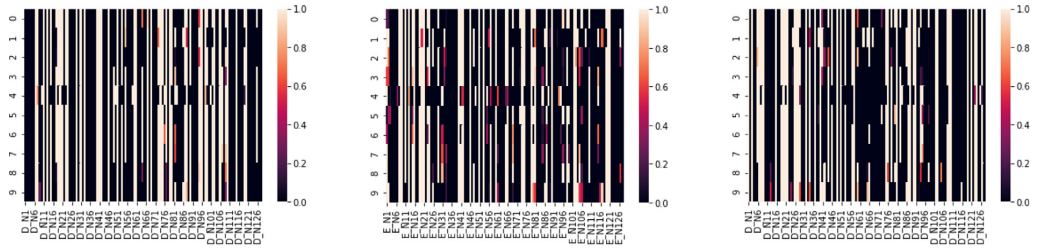
(a) *AveFL* (b) *MixFL*

Fig. 11. Fig. 11(a) is the correctly classified histogram of Average aggregation case, in the left plot the four different color bars correspond to four models of edge devices without aggregation. They can only correctly classify their own categories. While the right plot in Fig. 11(a) has a better comprehensive capability of classification, it has the difficulty of distinguishing classes 4 and 6. In Fig. 11(b) we plot the same results for the mix aggregated method. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Figs. 5(a) to 5(f) correspond to different epoch numbers, the accuracy almost monotonically increases with the increment of epoch number, not only for aggregation performance but also for local models. In Fig. 5(f), after enough training, three local models reach the highest accuracy they can, $20\%, 20\%, 30\%$ as they own two, two, and three classes correspondingly.

4.2.2. Aggregation frequency

In [10], we only consider one-shot FL (aggregate only once), here we also study the aggregation frequency, which defines the number of acquisitions to train during local training. For instance, assume that we have 10 acquisitions (fixed budget, 400 data for every acquisition), if aggregation frequency is 5, it means that every $10/5 = 2$ acquisitions we aggregate. Note that aggregation frequency is different from epoch: epoch defines the number of repetitions given the acquisition number (training data size), whereas aggregation frequency decides the number of acquisitions, though, both of them are critical factors to enable the performance. If the aggregation frequency is low, we aggregate after a relatively large number of training data, it reduces the communication cost and takes the risk of severe divergence. Instead, if the aggregation frequency is high, we aggregate after a small amount of data, we can avoid the divergence problem, but with increasing the cost of communication. For a given epoch number 45, in Fig. 6 we demonstrate the results corresponding to different aggregation frequencies. Correspondingly, we plot the performance concerning different aggregation frequencies (10, 5, 2, and 1). From Figs. 6(a) to 6(d), the aggregated accuracy decreases with the decrements of aggregation frequency. We



(a) Activation of Model from d_1 (b) Activation of Ensem. Model (c) Activation of Model from d_3

Fig. 12. We choose 10 test images from class one and feed into local model from d_1 in Fig. 12(a), aggregated model in Fig. 12(b) and local model from d_3 in 12(c). The x coordinator indicates the nodes of the fully connected layer right before the softmax layer, and the heatmap values are the output of the nodes. The activation of pattern of d_1 is similar to the aggregated model, fairly different from d_3 where class 1 is not generated.

will look at this problem from analyzing the weight divergence in Section 4.5. Both the epoch and the aggregation frequency cause divergence, but aggregation has more significant impact than the epoch number.

4.3. Aggregation strategies

We consider three aggregation strategies in this paper: *AveFL*, *OptFL* and *MixFL*. As we discussed in the previous section, *AveFL* averages the parameters of models during aggregation; *OptFL* opts for the model with optimal performance; and *MixFL* is the mixture of *AveFL* and *OptFL*, in each iteration it chooses the better one between them. The result (shown in Fig. 7) demonstrates that there is no big difference between *OptFL* and *MixFL* for both cases of 10 and 20 acquisitions. However, the whole distribution they learned is different, as we discuss in Section 4.6.

4.4. Non-IID data type ii (FL with AL)

For Non-IID Type ii , we simulate it by applying AL on the four edge devices. AL can be considered an effective way of choosing data than random sampling, and this behavior causes a slightly biased data generation. For instance, in [10], we select the data with maximal entropy (uncertainty) to train our model. The method is shown in Algorithm 5. In Fig. 8, we firstly show that AL outperforms random choice in terms of prediction accuracy. Also, in Fig. 9 shows how aggregation affects the performance when FL combines AL.

4.5. Weights divergence and aggregation performance

In this subsection, we quantitatively investigate the correlation between weight divergence and aggregation performance. Firstly, let us define the divergence of layer l of device i as follows:

$$\text{divergence}_i^l = \frac{|W_i^{tl} - W_{\text{aggregated}}^{tl}|}{|W_i^{tl}|}$$

Where W_i^{tl} is the layer l of model (device) i at iteration t and $W_{\text{aggregated}}^{tl}$ is the layer l of aggregated model at iteration t.

In Fig. 10, we plot the divergence of all layers in the network (above) and the corresponding aggregated result (below). The first coordinate represents the network layers, and the four colored bars represent the different devices. From Figs. 10(a) to 10(d), the divergence decreases for all the layers; however, the aggregation increases in the beginning and stops increasing at some point. It could indicate that if the divergence value is too large (Fig. 10(a)), the aggregation effect is not fully enabled, and on the other hand, if it is too small (Fig. 10(d)), it may disable the aggregation effect. Our result is consistent with [9], where they also showed the accuracy reduction is significantly correlated with weight divergence.

4.6. Correlation between local models and aggregated model

We can also consider the aggregated model as the Gaussian Mixture Model (GMM) [26]. Suppose we have C classes, which correspond to C models $M_1, M_2, \ldots, M_C$. We define GMM as $M_{GMM} = \sum_{i=1}^{C} \alpha_i M_i$ and $\sum_{i=1}^{C} \alpha_i = 1$. In our case, we consider α uniformly distributed since we do not have prior knowledge and do not want to solve the optimization problem to compute α_i. It implies an assumption that the ten classes share some common features. Averaging weights is like partially 'copying' the classification capability of different classes from their related edge devices. We call it 'partially' because models from other categories will dilute the effect. The experimental evidence is shown in Fig. 11. For *AveFL* (Fig. 11(a)), we plot the histogram of correctly classified classes for four edge devices (corresponds to four colors) in the left figure and the histogram of correctly classified classes for the average model in the right figure. As we can see, without aggregation the local model can **only** predict their corresponding categories. For instance, d_1 generates class 0 and 1, the trained model on d_1 can only predict 0 and 1. However, the prediction by aggregated model can cover most of the classes, except with difficulty in classifying 4 and 6. In Fig. 4, we can see class 4 is 'coat' and class 6 corresponds to 'shirt'(label starts from 0). These two classes are very similar, and it is not easy to distinguish. While, *MixFL* (Fig. 11(b)) has different behavior: it learns different distributions from aggregation, though, their overall accuracy is similar (shown in Fig. 7).

To further study how local models benefit the aggregated model, we analyze the neuron activation patterns. We choose 10 test images from class 1 and feed them into local model trained by d_1 in Fig. 12(a), aggregated model in Fig. 12(b) and local model from d_3 in Fig. 12(c). The x coordinator indicates that the nodes of the fully connected layer right before the softmax layer, and the heatmap values are the output of the nodes. The activation of the pattern of d_1 is similar to the aggregated model, fairly different from d_3 that does not have any information about class 1.

5. Conclusion and future work

Distributed machine learning has several virtues, including the potential to reduce data aggregation and thus improved privacy. However, this virtue poses a potential challenge, namely that the edge devices are set to learn from Non-IID data. Hence, to investigate the robustness of Federated Learning to Non-IID data, we simulate two scenarios. Furthermore we analyze and compare different aggregation strategies: *AveFL*, *OptFL* and *MixFL*. We presented evidence that federated learning is robust to sampling bias, and also we found that the epoch (amount of local learning) and the aggregation frequency are important parameters for Federated Learning. In the end, we also post-process the prediction performance to understand the correlation between local models and the aggregated model.

CRediT authorship contribution statement

Jia Qian: Concept, Design, Analysis, Writing or revision of the manuscript. **Lars Kai Hansen:** Concept, Design, Analysis, Writing or revision of the manuscript. **Xenofon Fafoutis:** Concept, Design, Analysis, Writing or revision of the manuscript. **Prayag Tiwari:** Concept, Design, Analysis, Writing or revision of the manuscript. **Hari Mohan Pandey:** Concept, Design, Analysis, Writing or revision of the manuscript.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation program under the Marie Sklodowska-Curie grant agreement No. 764785, FORA-Fog Computing for Robotics and Industrial Automation.

References

[1] J. Konečnỳ, H. McMahan, F.X. Yu, P. Richtárik, A.T. Suresh, D. Bacon, Federated learning: Strategies for improving communication efficiency, 2016, arXiv:1610.05492, arXiv preprint.

[2] M. Barzegaran, N. Desai, J. Qian, K. Tange, B. Zarrin, P. Pop, J. Kuusela, Fogification of electric drives: An industrial use case, in: The 25th International Conference on Emerging Technologies and Factory Automation ETFA2020, 08 Sep 2020, Vienna, Austria, 2020.

[3] J. Qian, P. Tiwari, S.P. Gochhayat, H.M. Pandey, A noble double dictionary based ecg compression technique for ioth, IEEE Internet Things J. (2020).

[4] C. He, C. Tan, H. Tang, S. Qiu, J. Liu, Central server free federated learning over single-sided trust social networks, 2019, arXiv:1910.04956, arXiv preprint.

[5] Y. Zhao, C. Yu, P. Zhao, H. Tang, S. Qiu, J. Liu, Decentralized online learning: Take benefits from others' data without sharing your own to track global trend, 2019, arXiv:1901.10593, arXiv preprint.

[6] A.A. Diro, N. Chilamkurti, Distributed attack detection scheme using deep learning approach for Internet of Things, Future Gener. Comput. Syst. 82 (2018) 761–768.

[7] B. Lakshminarayanan, A. Pritzel, C. Blundell, Simple and scalable predictive uncertainty estimation using deep ensembles, in: Advances in Neural Information Processing Systems, 2017, pp. 6402–6413.

[8] H. Zhu, Y. Jin, Multi-objective evolutionary federated learning, IEEE Trans. Neural Netw. Learn. Syst. (2019).

[9] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, V. Chandra, Federated learning with non-iid data, 2018, arXiv:1806.00582, arXiv preprint.

[10] J. Qian, S. Sengupta, L.K. Hansen, Active learning solution on distributed edge computing, 2019, arXiv:1906.10718, arXiv preprint.

[11] J. Qian, S.P. Gochhayat, L.K. Hansen, Distributed active learning strategies on edge computing, in: 2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom), IEEE, 2019, pp. 221–226.

[12] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, J. Mach. Learn. Res. 15 (1) (2014) 1929–1958.

[13] L. Breiman, Bias, Variance, and Arcing Classifiers, , Tech. Rep., Tech. Rep. 460, Statistics Department, University of California, Berkeley . . . , 1996.

[14] Z.-H. Zhou, Ensemble Methods: Foundations and Algorithms, CRC press, 2012.

[15] T. Chen, T. He, M. Benesty, V. Khotilovich, Y. Tang, Xgboost: extreme gradient boosting, 2015, pp. 1–4, R package version 0.4-2.

[16] Z. Xu, G. Huang, K.Q. Weinberger, A.X. Zheng, Gradient boosted feature selection, in: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2014, pp. 522–531.

[17] D.A. Klein, A.B. Cremers, Boosting scalable gradient features for adaptive real-time tracking, in: 2011 IEEE International Conference on Robotics and Automation, IEEE, 2011, pp. 4411–4416.

[18] J. Son, I. Jung, K. Park, B. Han, Tracking-by-segmentation with online gradient boosting decision tree, in: Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 3056–3064.

[19] T. Nishio, R. Yonetani, Client selection for federated learning with heterogeneous resources in mobile edge, in: ICC 2019-2019 IEEE International Conference on Communications, ICC, IEEE, 2019, pp. 1–7.

[20] P. Kairouz, H.B. McMahan, B. Avent, A. Bellet, M. Bennis, A.N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, et al., Advances and open problems in federated learning, 2019, arXiv:1912.04977, arXiv preprint.

[21] T.B. Hashimoto, M. Srivastava, H. Namkoong, P. Liang, Fairness without demographics in repeated loss minimization, 2018, arXiv:1806.08010, arXiv preprint.

[22] A. Krogh, J.A. Hertz, A simple weight decay can improve generalization, in: Advances in Neural Information Processing Systems, 1992, pp. 950–957.

[23] R. Pinsler, J. Gordon, E. Nalisnick, J.M. Hernández-Lobato, Bayesian batch active learning as sparse subset approximation, in: Advances in Neural Information Processing Systems, 2019, pp. 6356–6367.

[24] S.S. Sarwar, A. Ankit, K. Roy, Incremental learning in deep convolutional neural networks using partial network sharing, IEEE Access (2019).

[25] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015, arXiv:1502.03167, arXiv preprint.

[26] J.J. Verbeek, N. Vlassis, B. Kröse, Efficient greedy learning of Gaussian mixture models, Neural Comput. 15 (2) (2003) 469–485.

CONTRIBUTION C

A Decomposed Deep Training Solution for Fog Computing Platforms

Jia Qian and Mohammadreza Barzegaran. A decomposed deep training solution for fog computing platforms (submitted to tec2021 workshop). 2021

A DECOMPOSED DEEP TRAINING SOLUTION FOR FOG COMPUTING PLATFORMS

A PREPRINT

Jia Qian, Mohammadreza Barzegaran
Department of Applied Mathematics and Computer Science,
Technical University of Denmark,
2800 Lyngby, Denmark.
{jiaq,mohba}@dtu.dk

September 30, 2021

ABSTRACT

Legacy machine learning solutions collect user data from data sources and place computation tasks in the Cloud. Such solutions eat communication capacity and compromise privacy with possible sensitive user data leakage. These concerns are resolved with Fog computing that integrates computation and communication in Fog nodes at the edge of the network enabling and pushing intelligence closer to the machines and devices. However, pushing computational tasks to the edge of the network requires high-end Fog nodes with powerful computation resources. This paper proposes a method whose computation tasks are decomposed and distributed among all available resources. The more resource-demanding computation is placed in the Cloud, and the remainder is mapped to the Fog nodes using migration mechanisms in Fog computing platforms. Our presented method makes use of all available resources in a Fog computing platform while protecting user privacy. Furthermore, the proposed method optimizes the network traffic such that the high-critical applications running on the Fog nodes are not negatively impacted. We have implemented the (deep) neural networks - using our proposed method and evaluated the method on MNIST and CIFAR100 as the data source for the test cases. The results show advantages of our proposed method comparing to other methods, i.e., Cloud computing and Federated Learning, with better data protection and higher flexibility.

1 Introduction

Big data has been expanded in all areas affecting business and research models [Qiu et al., 2016, Beam and Kohane, 2018, Labrinidis and Jagadish, 2012]. These new models range from data-oriented economies that supply and facilitate data, to the areas where data analytics help with improving productivity and reliability [Schroeder, 2016]. A challenge to realize this vision is the need for collecting a huge amount of data into data center facilities that are equipped with different communication solutions. However, the power of Big Data will be revealed with the use of Machine Learning (ML) [Sagiroglu and Sinanc, 2013] which requires powerful computation resources to perform [Zhou et al., 2017]. Such computation and communication capabilities are integrated in Cloud Computing (CC) which has gained significant popularity and success in the past decade [Marston et al., 2011].

With CC providing cost-efficient, powerful computation and storage capabilities [Marston et al., 2011], Big data and ML have been spread in different application areas [Marston et al., 2011]. Companies such as Google, Microsoft, and Amazon have been providing different solutions for collecting data from producers, e.g., sensors, to their Cloud facilities, and for data processing to analyze the data. Collecting all raw data from producers to data centers compromises the data privacy, is more vulnerable to security attacks, and is forbidden according to General Data Protection Regulation (GDPR) in Europe. It is also massive and often repetitive that eats network bandwidth. Nevertheless, CC having non-deterministic behavior, does not fit the applications that are latency-sensitive. An example of such area is the industrial applications that are high-critical, i.e., they have stringent timing requirements.

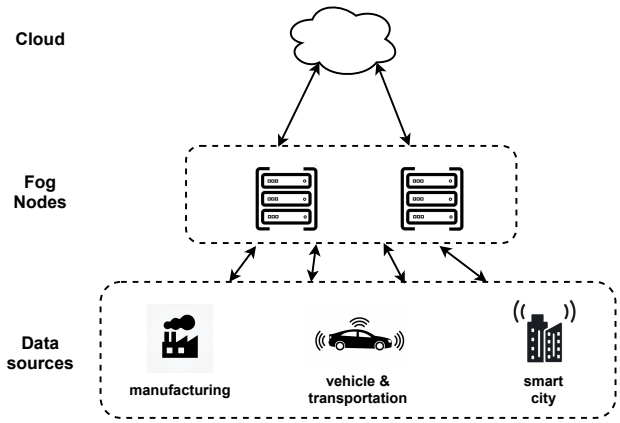


Figure 1: Overview of Fog Computing Platform

An architectural means to solve such issues is the Fog Computing (FC) that is defined as a "system-level architecture that distributes resources and services of computing, storage, control and networking anywhere along the continuum from Cloud to Things" [Fog Computing and Networking Architecture Framework, 2018 (accessed April 1, 2021]. An interchangeable term is Edge Computing (EC) [Shi et al., 2016] with the difference that FC emphasizes more about the infrastructures and EC focuses more on the computation part. A fog computing platform (FCP) brings computation and storage resources closer to the edge of the network and it is composed of several interconnected fog nodes (FNs). The vicinity property of FC outcasts CC by lower latency, local computation and high bandwidth for clients [Yi et al., 2015]. As shown in Figure 1, FNs are placed at the edge of the network and in the proximity of data sources. FC is literally pushing computation, data storage, data analytics, and service away from centralized Cloud to FNs. FC has been envisioned to implement different application areas such as industrial Internet of Things Pop et al. [2021], self-driving vehicles [Chiang et al., 2017], and smart healthcare [Qian et al., 2020a] and so on [Lee et al., 2015] in the near future. Several types of FNs from powerful high-end FNs to low-end FNs with limited resources have been proposed by researchers [Bonomi et al., 2012, Puliafito et al., 2019] and have been developed by companies [TTTech Computertechnik AG, 2019 (accessed October 7, 2019, Nebbiolo Technologies, Inc, 2021 (accessed April 1, 2021].

FC enables performing intelligence at the edge of the network using the integrated computation resources and data storage of FNs. It also avoids the raw data transmission between FNs and the Cloud avoiding the privacy leakage and the bandwidth overuse. To this end, a favorable computation framework - Federated Learning (FL) [Konečnỳ et al., 2016] is proposed by Google where they suggest to exchange gradients between FNs and the Cloud to jointly train a global model that may capture the information of whole environment. However, FL may not be suitable in applications where FNs have less computation budget and performing computation for FL may compromise the performance of high-critical applications.

We are interested in using benefits of FC for ML and specially Deep Learning (DL) which typically needs more computation power, without compromising the performance of high-critical applications running on FNs. We focuses on avoiding to send all raw data to the Cloud by performing local analytics on the FNs. Unlike the CC method that does all the computation on the Cloud and FL that does all the computation on the FNs, we would like to propose a new approach in which the computation is split among FNs and the Cloud. Thus, we propose decomposing data training into several tasks to reduce both the computation workload and data exchange size. We consider two steps for the learning process of neural network (a.k.a training): forward and backward. We feed inputs to model during forward step and use chain rule to compute gradients in backward step. As pointed out by [Huo et al., 2018], in most cases backward computation is much more expensive than forward computation, therefore, we assign a part of forward computation to the FN, and it's remainder together with the backward computation to the Cloud.

Our main contribution are as follows. We motivate the demand of addressing the computation decomposition between FCP and server, for applications that require more computational resources that could not be met by edge devices. We propose an approach that decomposes the training process and test the proposed approach on neural network models since it often requires more computational expense than other ML models [Goodfellow et al., 2016]. We compare

our proposed methods with other methods in the literature and evaluate the performance of our proposed methods on several test cases. We compare the model performance, communication cost and also simulate it on OMNET++.

The remainder of this paper is structured as follows. We present the theory and concepts for CC-based and FL approaches in Section 2. Afterwards, Section 3 presents the details of our proposed method. We evaluate our proposed approach on test cases from MNIST and CIFAR100 considering high-critical applications running on FNs. The related work is presented in Section 5 and Section 6 concludes the paper.

2 Background and Related Concepts

Deep Learning (DL) [LeCun et al., 2015a] has gained significant success and beat the state-of-the-art in many fields such as object detection and recognition, speech recognition, medical diagnosis [Tiwari et al., 2018], and many other domains, e.g., genomics [Qian and Comin, 2019]. The presence of neural network avoids the non-trivial feature extraction work, which was required in conventional ML methods. Instead the model takes raw data as the input and the model automatically carries out the feature extraction work. There are several types of neural network architectures such as Multilayer Perceptron (MLP), Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), and Long short-term memory (LSTM). However, RNN and LSTM are out of the scope in this work, thus, we ignore their details, we just introduce MLP and CNN. Additionally, in Section 4, we give information on LeNet5 and VGG16 that are used in the experiments.

MLP [Rosenblatt, 1958] was firstly invented and intended to model how the human brain processed visual data and learned to recognize objects. It is a set of layers stacking together with both linear and non-linear transformations in the interleaving way, see Figure 2. CNN [LeCun et al., 1999] is probably the most widely used network since it largely reduced the number of model parameters comparing with MLP. Instead of one unit connecting with all units in the previous layer, it only connects to the receptive field where convolution kernel is applied, see Figure 3. The convolution kernel is shared in the convolution layer, which made CNN shift-invariant, namely, certain features can be detected regardless of its location in the input image. Moreover, RNN [Schuster and Paliwal, 1997] and LSTM Hochreiter and Schmidhuber [1997] are often used for speech recognition or any sequential data. In this work we use CNN and MLP in our proposed implementation method.

2.1 Cloud-based Deep Learning

In CC-based DL, the FNs collect data from the sensors connected to them and transfer the raw data to the Could where DL model is implemented as computation tasks for training. Figure 4 shows the demonstration of the CC-based DL which consists of p FNs involved via transmitting data to the Cloud.

Each FN receives the input data produced by the connected sensors. We denote the input data as $x \in \mathbb{R}^d$. The FN integrates B instances of data to form one batch. Typically the number of data instances is much smaller than the size of each data instance, $d \gg B$. To this end, each batch transmits $B \times d$ data. Since all the p FNs transmit data to the Cloud, the size of total data transmitted in each iteration is $p \times B \times d$. We only consider one iteration of data transmission for the comparison with all the other methods. The communication cost is fixed in CC regardless of the ML models

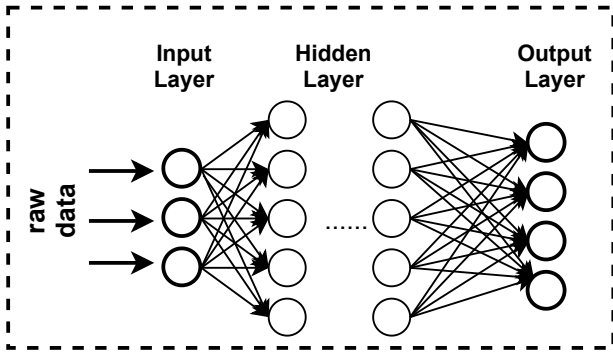


Figure 2: Multilayer Perception

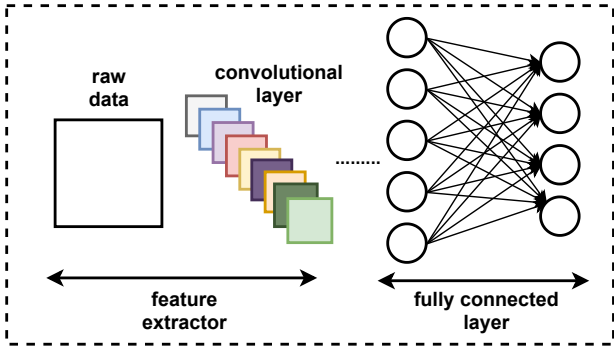


Figure 3: Convolutional Neural Network

since only training data is considered. Moreover, in some applications, the data storage is not permanent on Cloud and the same data is transmitted for multiple times, which is another downside of CC. For instance, using GPU in Colab [Bisong, 2019], it requires transmitting data whenever it reconnects.

2.2 Federated Deep Learning

Federated Learning (FL) was first proposed by Google as a solution for large-scale mobile network training [Konečnỳ et al., 2016]. It suggests to distribute computation workload among nodes connected via network, whereas the server in the network maintains globally the shared parameters of models, which are in the form of dense or sparse matrices. There is a similar work in the literature titled as "parameter server" [Li et al., 2014], which focuses on the system design and implementation. FL caught more attention in the ML community, comparing with "parameter server".

In FL diagram, the FNs or data sensor (if applicable) perform most of the computation, and the Cloud aggregates and shares the parameters, shown in Figure 5. Each FN implements local training using its own data (from its connected sensor), and sends the gradient information to the Cloud. On the other hand, the Cloud aggregates all the gradient information from all FNs, and sends the aggregated gradient information back to the FNs that uses the information for the next round of local training. The overall view of the FL method is introduced in Algorithm 1.

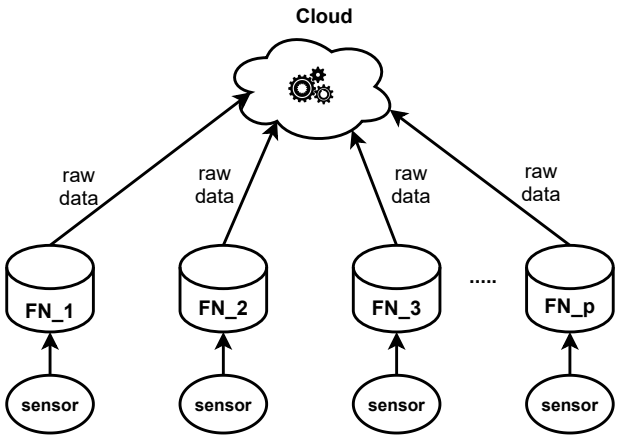


Figure 4: Cloud Computing-based ML method

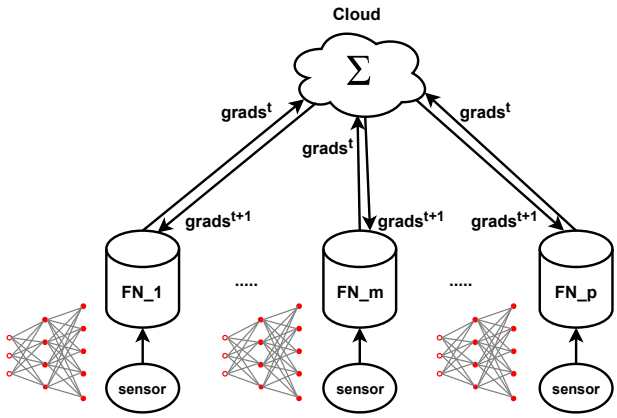


Figure 5: Federated Deep Learning at round t.

At each round, every FN receives B instances of raw data (each instance has size d) from sensors connected to it and take the raw data as the input to perform the local computation. Presumably, the data denoted as (X, Y) is generated from the empirical data distribution $\hat{\mathcal{D}}$. ℓ_j is defined as the local loss function in FN j, v_j^t is the Jacobian matrix (containing all the partial derivatives) of FN j at round t, and η is the learning rate. Cloud instead performs the simple arithmetic calculations that aggregate the model parameters from all the active FNs.

Leaving most computation on local devices constrains the range of applications that FL may be applicable. For instance, it is not suitable for the scenario that the computational resources on device are insufficient to carry out all the tasks. On the other hand, the available assets on Cloud are not fully utilized neither, e.g. computation power and storage.

Regarding to the communication cost, say we have a neural network f with $m+n$ layers and $|v_j^t|$ represents the number of elements in Jacobian matrix v_j^t. At each round t, FNs and the Cloud transmit data with total size of $2 \times p \times \sum_j^{m+n} |v_j^t|$ or $2 \times p \times \sum_j^{m+n} |w_j^t|$ for the two-way transmission (from devices to server and from server to devices).

3 Solution

In this section, we present our method that decomposes the computational tasks among the distributed FNs and the Cloud. More specifically, the overall model is divided into two parts: local model and remote model. The local model is

Algorithm 1 Federated Learning

Input: w^0, X, Y
for t=1,...T **do**
    =>**workers:**
    for j=1,2,..,p **do** p devices (in Parallel)
        1) $v_j^t = \nabla \hat{\ell}_j(f(X_j^t; w^t), Y_j^t)$
        with $((X_j^t = \{x_{jk}^t\}_{k=1,..B}, Y_j^t = \{y_{jk}^t\}_{k=1,...,B}) \sim \hat{\mathcal{D}}$
        2) share v_j^t with server
    end
    =>**server:**
    $\overline{v}^t = \frac{1}{p} \sum_{j=1}^{p} v_j^t$
    $w^{(t+1)} = w^t - \eta \times \overline{v}^t$
    share w^{t+1} with devices for next round
end
Return w^T

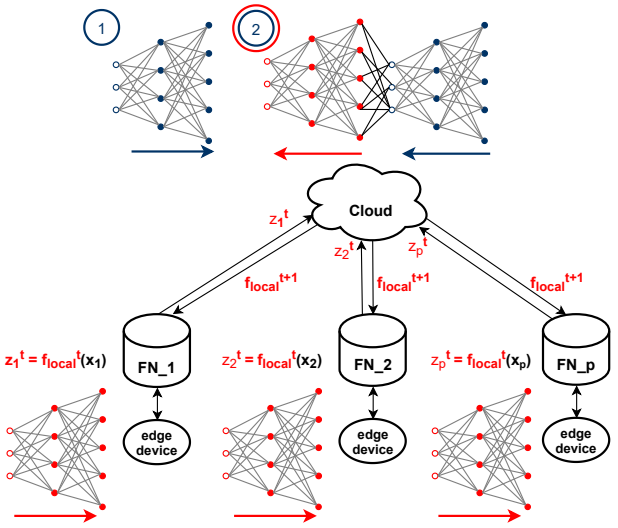


Figure 6: Fog Computing-based Decomposed Deep Training. FNs own a part of the full model, we call it f_{local}. FNs feed the input to the local model and share the intermediate result z_i with the server and the server completes the residual forward pass (step 1, marked by navy blue circle) and carry out the backward calculations (step2, marked by red and navy blue circles).

Algorithm 2 Fog Computing-based Decomposed Deep Training (FCDDT)

Input: $X, Y, [w^1_{\textbf{local}}, w^1_{\textbf{remote}}], \eta$
for t=1,...,T **do**
  for j=1,...,p **do** p devices (in Parallel)
    $z_j^t = \textbf{FogCom}(X_j^t; w_{\text{local}}^t)$ ▷ Computation on fog nodes (FNs)
    with $((X_j^t = \{x_{jk}^t\}_{k=1,..B}, Y_j^t = \{y_{jk}^t\}_{k=1,...,B}) \sim \hat{\mathcal{D}}$ ▷ j indicates data on worker j
  end
  $z^t = [z_1^t, z_2^t, .., z_p^t]$ ▷ Cloud collects the outputs from FNs
  $y^t = [y_1^t, y_2^t, ..., y_p^t]$ ▷ Cloud collects the labels from FNs
  $w_{\text{local}}^{t+1} = \textbf{CloudCom}(z^t, y^t, \eta)$ ▷ Computation on Cloud
end
Return $[w^T_{\textbf{local}}, w^T_{\textbf{remote}}]$

stored on FNs and Cloud, whereas the remote model is only stored on Cloud. Note the Cloud also owns a copy of local model for the error propagation step. **On the FNs side**, first they collect data from sensors connected to them and the data is used as the input to the local models (named as *local computation*). *Local computation* is one part of the overall foreward step. Second, all FNs send the intermediate result (output of local model) to the Cloud. **On the Cloud side**, it first finishes the rest of the forward task on the remote model and then implements the overall backward computation (error propagation) for training using the combo of the local and remote models. The proposed method is illustrated in Figure 6 where the local models and the remote model are shown in red and navy blue circles, respectively.

An overview of our proposed method FCDDT is presented in Algorithm 2 where two main functions FogCom and CloudCom are introduced for all the computational tasks on FNs and Cloud. At round t, FogCom takes X_j^t as the input and returns the intermediate result z_j^t where X_j^t represents the training data of FN j at round t. After the active FNs shared the intermediate output and labels with the server, the server implements CloudCom, which outputs the updated parameters of the local model w_{local}^{t+1}. This procedure repeats T times and finally it returns the complete model parameterized by $[w^T_{\text{local}}, w^T_{\text{remote}}]$. The unbalanced computation resources on FNs might delay the computation on

Algorithm 3 CloudCom

1: **Input:** z^t, η, y^t▷ z^t is the combination of all outputs from FNs, η is the learning rate and y^t is the labels at round t
2: $\hat{y}^t = f_{\text{remote}}(z^t; w_{\text{remote}})$ ▷ f_{remote} is the remote model on server, parameterized by w^t_{remote}
3: $w^{t+1} = w^t - \eta \times \nabla_w \hat{\ell}(\hat{y}^t, y^t)$ ▷ $\nabla_w \hat{\ell}(\hat{y}^t, y^t)$ is gradients w.r.t. all the parameters
4: **Return** $w^{t+1}_{\textbf{local}}$ ▷ w^{t+1}_{local} is one part of w^{t+1}

Cloud. However, the synchronization is not necessarily required, in particular, when the number of FNs or batch size is big, see Section 4 for the related observation.

We present the details of computation on the Cloud (CloudCom) in Algorithm 3. CloudCom takes all the outputs from local computation and the labels at the round t, and the learning rate η as the inputs. CloudCom first completes the forward step computation using the remote model f_{remote}, and then implements the backward step computation, and shares the updated parameters of local model with FNs.

Implementation: We assume that the remote process (CloudCom) is implemented as an application on the Cloud that is connected to the FCP. However, implementing the local process (FogComp) requires mechanism for application deployment. Thus, we assume that FogComp is implemented as dedicated applications for each FN, and they are submitted to the FCP via the Cloud. Once submitted, the *Fog Controller* FN, which is determined at runtime using mechanisms such as [Karagiannis and Papageorgiou, 2017], receives the submission request. Since the Fog controller has knowledge of the available resources on the FNs using the resource discovery algorithms such as [Karagiannis and Papageorgiou, 2017, OpenStack, 2020 (accessed May 7, 2021, European Telecommunications Standards Institute, 2016 (accessed May 7, 2021] at runtime, it can decide the placement of applications on the FNs using a decentralized resource allocation technique [Avasalcai et al., 2019, Skarlat et al., 2018]. In the case of an FN leaving the FCP, Fog controller is able to migrate application from the leaving FN to a next available FN.

Communication Expense: The network traffic in FCDDT consists of exchanging messages that enable the decomposed training. The message exchanging can be split into two parts: (1) the messages uploading intermediate results and labels to the Cloud at each round and (2) the messages downloading updated model parameters from the Cloud to the FNs. We define a scenario where the p active FNs in the FCP that collects B instances as a batch, and l_m nodes sit in layer m. Thus, the total size of uploading messages is equal to $p \times B \times l_m + B$. Since $p \times B \times l_m \gg B$, we can easily omit B. Similarly, the size of downloading messages is equal to $p \times \sum_i^m |w_i|$.

4 Experiments and the Results

The structure of this section is as follows. We first describe our test setup and the test cases we used for the evaluations in Section 4.1. We compare our proposed method FCDDT with the related works (FL and CC) in Section 4.2. Afterwards, we measure the accuracy of our proposed method on MNIST and CIFAR10 in Section 4.3. Section 4.4 presents the OMNET++ simulation and evaluation results on the traffic latency.

4.1 Test Cases and Setup

We implemented our method in Python and run it on a Titan X GPU with Architecture Maxwell. The performance of FCDDT is evaluated on test cases that are derived from two sets of images: MNIST and CIFAR10. Each image in MNIST has the size of 28×28 pixels and represents gray-scale handwritten digits. MNIST consists of 50K images for training and 10K images for test that are grouped to 10 classes corresponding to digits from zero to nine. Each image in CIFAR10 has the size of $3 \times 32 \times 32$ pixels (i.e., images are RGB) and represents an object of 10 classes such as airplanes, automobiles, etc. CIFAR10 consists of 50K images for training and 10K images for test. FCDDT is applicable to different ML methods, thus, we used a two-layer CNN, two-layer MLP for MNIST, LeNet5 [LeCun et al., 2015b], and VGG [Simonyan and Zisserman, 2014] for CIFAR10. The introduction of CNN and MLP are already

Table 1: Comparison of FCDDT with the related work

Feature	FCDDT	FL	CC
Compute nodes	FNs & Cloud	FNs	Cloud
Raw data exchange	No	No	Yes
Communication cost	$p \times (B \times l_m + \sum_i^m \lvert w_i \rvert)$	$2 \times p \times (\sum_i^{m+n} \lvert w_i \rvert)$	$p \times B \times d$
Implementation	via Cloud	on each FN	via Cloud

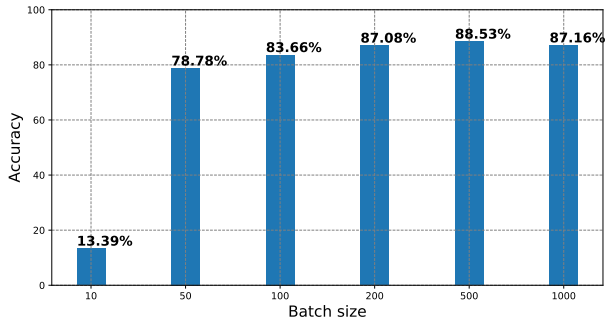


Figure 7: batch size comparison

presented in Section 2. LeNet5 [LeCun et al., 2015a] is a commonly used convolutional neural network, composed of 5 convolutional layers and one dense layer (fully connected). VGG was designed for large-scale images classification tasks. It uses the small kernel with size 3×3, for more details we refer the readers to [Simonyan and Zisserman, 2014].

We also evaluated the effect of FCDDT on the network traffic using simulation in OMNET++. The simulation is carried out on 6 test cases with increasing number of FNs in the FCP. We assumed that the FNs in each test case exchange both critical and ML-related messages. The critical message exchanging between an FN and the Cloud is modeled as a periodic request with the size of 1400 bytes and a periodic reply with the size of 5000 bytes. The periods of the request and reply messages are equal to 400 ms. Each FN exchanges critical messages with two of its neighbors, which are similarly modeled as periodic request and reply messages that have periods are randomly chosen among 200, 300, and 400 ms and sizes equal to 1400 and 4000 bytes, respectively.

Moreover, ML messages are also modeled as periodic request and reply messages that have period equal to 1000 ms. The size of request messages is 1200 bytes and the size of reply messages is equal to the size of training data. We simulate the network traffic for a duration of hyperperiod, i.e., the least common divisor of all message periods, and evaluate FCDDT for its effect on the mean bandwidth usage and mean imposed delay on the critical messages.

4.2 Comparison with the related work

We compare the features of FCDDT with the related work in Table 1. The related work consists of FL [Konečnỳ et al., 2016] and CC [Marston et al., 2011] solutions. The first feature in the table represents the computation platform where each solution runs. As shown in the table, FCDDT is the only solution that uses all the resources in the FCP and Cloud.

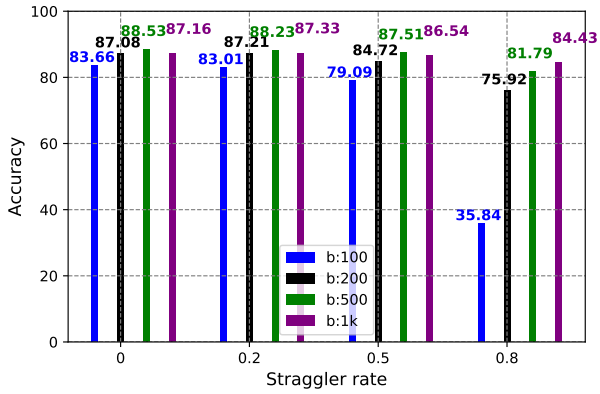


Figure 8: Straggler rate comparison (CIFAR10). Percentage sign is omitted. $b : x$ represents the batch size.

Note here in FL, the Cloud only carries out easy the arithmetic calculation, we ignore it in Table 1. Regarding privacy, both FL and FCDDT avoid sending raw data and only transmit the model parameters or other intermediate results. However, as shown in the table, FCDDT is more efficient comparing with FL in communication cost since it uses less bandwidth $Bl_m + \sum_i^m |w_i| < 2\sum_i^{m+n} |w_i|$. Note we can further save bandwidth by allocating a small local model to FNs (in terms of depth and width of neural network) such that $\sum_i^m |w_i|$ is relatively small. CC requires least bandwidth without considering re-connection problem that we discussed in Section 2.1, but it compromises the sensitive data privacy.

Table 2: MNIST performance

Name	CC	FL	FCDDT
MLP (Two layers)	0.9 ± 0.007	0.9 ± 0.01	0.89 ± 0.003
CNN (Two layers)	0.96 ± 0.0014	0.96 ± 0.003	0.96 ± 0.002

Table 3: CIFAR10 performance

Name	CC	FL	FCDDT
LeNet5	0.445 ± 0.01	0.45 ± 0.05	0.45 ± 0.003
VGG16	0.93 ± 0.074	0.92 ± 0.13	0.885 ± 0.297

CC solution is suitable for the implementation on large scale FCP since the model training process is only implemented to run on the Cloud, thus it is scalable concerning the network size. Our proposed solution FCDDT uses a similar approach for the application deployment, automatically placing the applications on the FNs, which makes the solution scalable as well. However, FL requires all nodes to be programmed individually, unless a specific mechanism is introduced for a large scale FCP.

4.3 Performance evaluation

The accuracy evaluations of FCDDT on datasets MNIST and CIFAR10 using different ML methods are shown in Table 2 and Table 3 accordingly. As the results show, our method FCDDT either has identical performance or mildly degraded comparing with CC and FL. Note that we assume local training happened on FNs in FL framework for only one iteration and we repeat the experiments for 5 times to evaluate the standard deviation.

Moreover, we explore the impact of batch size on accuracy using FCDDT. We test it on CIFAR10 in Figure 7 where VGG16 is applied as the model. As we mentioned before, the synchronization between FNs is not necessarily required. Here we introduce the *straggler rate* α to simulate the scenario where the Cloud aggregates the transmission from top $1 - \alpha$ FNs. Figure 8 shows that the different straggler rates have various impact on the model performance when batch size is different. The four colors represent batch size equal to 100, 200, 500, 1K accordingly. As we can see the accuracy drops drastically when batch size is 100 and straggler rate decreases to 0.8. However, it has no significant influence on the performance when batch is equal to 500 or 1K even with four FNs. We suggest to use big batch size in the environment where computational resources are not evenly distributed among FNs and the *straggler* phenomenon is likely to happen.

4.4 Network traffic evaluation

We simulate the network traffic for FCDDT and the related work for the 8 test cases in Table 4. It includes the mean end-to-end delay for all messages in μs and the mean bandwidth usage for the traffic. More specifically, the simulated results show that FCDDT is able to reach 16% less bandwidth usage comparing to FL and 19% less than CC. The saved bandwidth usage of FCDDT contributes to decreasing the average end-to-end delay to 59 μs, which is 8 μs and 18 μs less than FL and CC, respectively.

A key observation is that in the case where no ML service is running on the FCP, i.e. ignoring the ML messages, the average bandwidth usage is 20% and the average end-to-end delay is 55 μs. Taking this as the baseline, our method FCDDT introduces only 1% and 4 μs for the average bandwidth usage and the average end-to-end delay in addition, with ML service.

Table 4: Simulation results on six test cases

#	No. of FNs	FCDDT		FL		CC	
		E2E delay (μs)	bandwidth usage	E2E delay (μs)	bandwidth usage	E2E delay (μs)	bandwidth usage
1	4	49	18%	52	21%	57	22%
2	7	52	19%	56	23%	65	24%
3	10	56	21%	63	25%	75	26%
4	12	62	22%	69	25%	83	27%
5	15	66	24%	77	28%	89	29%
6	18	71	25%	84	29%	96	31%
mean		59	21%	67	25%	77	26%

5 Related Work

It's impossible to train the machine learning model on one device with the exponential growth of data and the increasing number of connected devices that generate data. Two types of parallel technologies are often used: *data parallelism* and *model parallelism*. Data parallelism is commonly used where each processor (GPU) owns one complete copy of the model and process one subset of the data. It is effective in some applications when data is naturally distributed across multiple devices and processing data in parallel way saves the cost of data transmission to one centralized machine. However, it requires the frequent communications between processors for the synchronization of models trained separately. It was considered as the main drawback of data parallelism [Seide et al., 2014, Strom, 2015, Lin et al., 2017]. It affects both statistical and hardware efficiency. After one processor finished computing the gradients, it has to wait for the other processors to complete their computation, which decreases the efficiency of hardware. Since it is a cross-device pattern learning, there might be the statistical heterogeneity among devices, such joint learning might have low statistical efficiency. [Tarditi et al., 2006] demonstrate how to translate high-level data parallel to GPU programs. [Shallue et al., 2018] experimentally studies the effect of increasing batch size during training, measured by the number of steps required to reach a preset error and how does this relationship changes with respect to different training methods, network architecture and dataset.

Another category is model parallelism. Increasing size of neural network (layers and parameters) can no surprise result in higher accuracy, however there is a limit to the maximum model size fit in a single processor (GPU). Thus, the model is partitioned into several parts and each part is processed on one GPU. Each GPU is responsible for their weight updates accordingly. The main drawback of model parallelism techniques is the dependence between processors, namely, the processor has to wait for the previous processor to finish its work and take the output of previous processor as the input. [Harlap et al., 2018] introduced a parallelizes computation by pipelining execution across multiple machines, which showed 95% for large deep neural networks relative to data-parallel training. They literally generate multiple batches of data and the processor processes the next-to-process batch in the waiting queue instead of waiting. [Krizhevsky et al., 2012] is a special case of model parallelism, instead of divide model into consecutive layers it partitions model horizontally. They design a model that is horizontally symmetric, so it can be separated into two parts and train independently on two processors and combine at the end of each backward.

Our method is the mixture of data and model parallelism. The data is intrinsically generated and distributed on multiple devices and we partition the model into two parts, one part for edge devices and another part for the server. Different from traditional model parallel methods, the edge devices only implement the forward step of the local model and the server carries out the rest including the residual forward step and whole backward step.

6 Conclusion and future work

Fog Computing as an enabler for Industry 4.0 which integrates mixed-criticality applications on a shared computing platform, envisions to run data analysis at the edge of the network. This capability prevents sending all data to the Cloud, thus prevents user data leakage and eating communication capacity. Although Fog Computing Platforms employ separation mechanisms to protect critical applications, the resource-demanding data analysis computation may debilitate Fog nodes from providing critical applications sufficient resources. We propose a decomposed deep training solution that distributes its workload among the Fog nodes and the Cloud. With such a solution, less resource-demanding computation is assigned to the Fog nodes and the remainder of computation is assigned to the Cloud, which exchange messages for performing data analytics. To further enhance the data security, we refer to another work [Qian et al., 2020b], which studies the lower bound of model structure to avoid the possible input reconstruction. The simulated network traffic in OMNET++ shows that our method is more suitable for mixed-critical systems comparing to the

literature. In our future work, we will consider the optimal decomposition of the computation that will use the maximum available resources on the Fog nodes.

The limitation of our method is that only one iteration is allowed to implement locally before the communication with the server. We believe that it can be addressed by a hierarchical FL architecture where the sub-network is introduced and composed of a set of devices and a FN. The sub-network may approximate the local Stochastic Gradient Desceent (SGD) with the assumption that the devices and FN are connected via an intranet. Or we can explore another way of model decomposition.

References

Junfei Qiu, Qihui Wu, Guoru Ding, Yuhua Xu, and Shuo Feng. A survey of machine learning for big data processing. *EURASIP Journal on Advances in Signal Processing*, 2016(1):1–16, 2016.

Andrew L Beam and Isaac S Kohane. Big data and machine learning in health care. *Jama*, 319(13):1317–1318, 2018.

Alexandros Labrinidis and Hosagrahar V Jagadish. Challenges and opportunities with big data. *Proceedings of the VLDB Endowment*, 5(12):2032–2033, 2012.

Ralph Schroeder. Big data business models: Challenges and opportunities. *Cogent Social Sciences*, 2(1):1166924, 2016.

Seref Sagiroglu and Duygu Sinanc. Big data: A review. In *2013 international conference on collaboration technologies and systems (CTS)*, pages 42–47. IEEE, 2013.

Lina Zhou, Shimei Pan, Jianwu Wang, and Athanasios V Vasilakos. Machine learning on big data: Opportunities and challenges. *Neurocomputing*, 237:350–361, 2017.

Sean Marston, Zhi Li, Subhajyoti Bandyopadhyay, Juheng Zhang, and Anand Ghalsasi. Cloud computing—the business perspective. *Decision support systems*, 51(1):176–189, 2011.

Fog Computing and Networking Architecture Framework. IEEE 1934-2018 - IEEE Standard for Adoption of OpenFog Reference Architecture for Fog Computing. `https://standards.ieee.org/standard/1934-2018.html`, 2018 (accessed April 1, 2021).

Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE internet of things journal*, 3(5):637–646, 2016.

Shanhe Yi, Zijiang Hao, Zhengrui Qin, and Qun Li. Fog computing: Platform and applications. In *2015 Third IEEE workshop on hot topics in web systems and technologies (HotWeb)*, pages 73–78. IEEE, 2015.

Paul Pop, Bahram Zarrin, Mohammadreza Barzegaran, Stefan Schulte, Sasikumar Punnekkat, Jan Ruh, and Wilfried Steiner. The fora fog computing platform for industrial iot. *Information Systems*, 98:101727, 2021.

Mung Chiang, Bharath Balasubramanian, and Flavio Bonomi. *Fog for 5G and IoT*, volume 288. Wiley Online Library, 2017.

Jia Qian, Prayag Tiwari, Sarada Prasad Gochhayat, and Hari Mohan Pandey. A noble double-dictionary-based ecg compression technique for ioth. *IEEE Internet of Things Journal*, 7(10):10160–10170, 2020a.

Kanghyo Lee, Donghyun Kim, Dongsoo Ha, Ubaidullah Rajput, and Heekuck Oh. On security and privacy issues of fog computing supported internet of things environment. In *2015 6th International Conference on the Network of the Future (NOF)*, pages 1–3. IEEE, 2015.

Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16, 2012.

Carlo Puliafito, Enzo Mingozzi, Francesco Longo, Antonio Puliafito, and Omer Rana. Fog computing for the Internet of Things: A Survey. *ACM Transactions on Internet Technology (TOIT)*, 19(2):1–41, 2019.

TTTech Computertechnik AG. Nerve. `http://tttech.com/products/industrial/industrial-iot/nerve`, 2019 (accessed October 7, 2019).

Nebbiolo Technologies, Inc. Nebbiolo. `https://www.nebbiolo.tech/`, 2021 (accessed April 1, 2021).

Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

Zhouyuan Huo, Bin Gu, Heng Huang, et al. Decoupled parallel backpropagation with convergence guarantee. In *International Conference on Machine Learning*, pages 2098–2106. PMLR, 2018.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015a.

Prayag Tiwari, Jia Qian, Qiuchi Li, Benyou Wang, Deepak Gupta, Ashish Khanna, Joel JPC Rodrigues, and Victor Hugo C de Albuquerque. Detection of subtype blood cells using deep learning. *Cognitive Systems Research*, 52: 1036–1044, 2018.

Jia Qian and Matteo Comin. Metacon: unsupervised clustering of metagenomic contigs with probabilistic k-mers statistics and coverage. *BMC bioinformatics*, 20(9):1–12, 2019.

Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. Object recognition with gradient-based learning. In *Shape, contour and grouping in computer vision*, pages 319–345. Springer, 1999.

Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681, 1997.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Ekaba Bisong. Google colaboratory. In *Building Machine Learning and Deep Learning Models on Google Cloud Platform*, pages 59–64. Springer, 2019.

Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, pages 583–598, 2014.

Vasileios Karagiannis and Apostolos Papageorgiou. Network-integrated edge computing orchestrator for application placement. In *IEEE International Conference on Network and Service Management*, pages 1–5, 2017.

OpenStack. Documentation on Nova Scheduler. `https://docs.openstack.org/developer/nova/filter_scheduler.html`, 2020 (accessed May 7, 2021).

European Telecommunications Standards Institute. Mobile edge computing (MEC) framework and reference architecture (GS MEC 003 V1.1.1). `http://www.etsi.org/deliver/etsi_gs/MEC/001_099/003/01.01.01_60/gs_MEC003v010101p.pdf`, 2016 (accessed May 7, 2021).

Cosmin Avasalcai, Christos Tsigkanos, and Schahram Dustdar. Decentralized resource auctioning for latency-sensitive edge computing. In *IEEE International Conference on Edge Computing*, pages 72–76, 2019.

Olena Skarlat, Vasileios Karagiannis, Thomas Rausch, Kevin Bachmann, and Stefan Schulte. A framework for optimization, service placement, and runtime operation in the fog. In *IEEE International Conference on Utility and Cloud Computing*, pages 164–173, 2018.

Yann LeCun et al. Lenet-5, convolutional neural networks. *URL: http://yann. lecun. com/exdb/lenet*, 20(5):14, 2015b.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns. In *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

Nikko Strom. Scalable distributed dnn training using commodity gpu cloud computing. In *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. *arXiv preprint arXiv:1712.01887*, 2017.

David Tarditi, Sidd Puri, and Jose Oglesby. Accelerator: using data parallelism to program gpus for general-purpose uses. *ACM SIGPLAN Notices*, 41(11):325–335, 2006.

Christopher J Shallue, Jaehoon Lee, Joseph Antognini, Jascha Sohl-Dickstein, Roy Frostig, and George E Dahl. Measuring the effects of data parallelism on neural network training. *arXiv preprint arXiv:1811.03600*, 2018.

Aaron Harlap, Deepak Narayanan, Amar Phanishayee, Vivek Seshadri, Nikhil Devanur, Greg Ganger, and Phil Gibbons. Pipedream: Fast and efficient pipeline parallel dnn training. *arXiv preprint arXiv:1806.03377*, 2018.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

Jia Qian, Hiba Nassar, and Lars Kai Hansen. Minimal conditions analysis of gradient-based reconstruction in federated learning. *arXiv preprint arXiv:2010.15718*, 2020b.

CONTRIBUTION D

Minimal Conditions Analysis of Gradient-based Reconstruction in Federated Learning

Jia Qian, Hiba Nassar, and Lars Kai Hansen. Minimal conditions analysis of gradient-based reconstruction in federated learning (submitted to neurips workshop). *arXiv preprint arXiv:2010.15718*, 2020

MINIMAL CONDITIONS ANALYSIS OF GRADIENT-BASED RECONSTRUCTION IN FEDERATED LEARNING

Jia Qian, Hiba Nassar, Lars kai Hansen
Department of Applied Mathematics and Computer Science,
Technical University of Denmark,
2800 Lyngby, Denmark.
{jiaq,hibna,lkai}@dtu.dk

September 30, 2021

ABSTRACT

Federated Learning (FL) proposed a distributed Machine Learning (ML) framework where every distributed worker owns a complete copy of global model and their own data. The training is occurred locally, which assures no direct transmission of training data. However, the recent work Zhu et al. [2019] demonstrated that input data from a neural network may be reconstructed only using knowledge of gradients of that network, which completely breached the promise of FL and sabotaged the user privacy.

In this work, we aim to further explore the theoretical limits of reconstruction, speedup and stabilize the reconstruction procedure. We show that a single input may be reconstructed with the analytical form, regardless of network depth using a fully-connected neural network with one hidden node. Then we generalize this result to a gradient averaged over mini-batches of size B. In this case, the full mini-batch can be reconstructed if the number of hidden units exceeds B, with an orthogonality regularizer to improve the precision. For a Convolutional Neural Network (CNN), the number of required filters in convolutional layers is decided by multiple factors, e.g., padding, kernel and stride size, etc. We require the number of filters $h \geq (\frac{d}{d'})^2 C$, where we define d as input width, d' as output width after convolutional layer, and C as channel number of input. We validate our theoretical analysis and improvements using bio-medical (fMRI, white blood cell) and benchmark data (MNIST, Kuzushiji-MNIST, CIFAR100, ImageNet and face images).

1 Introduction

FL Konečnỳ et al. [2016] was proposed by Google for the mobile network training application and caught tremendous interests recently. One important virtue of FL is to keep data remaining in the generated machines to avoid direct leakage due to data transmission. However, the distributed setup opens the new exposure to the attackers, and sensitive information may be disclosed in other forms, e.g., yet the presence of specific data point in training set of a trained ML model can be revealed, using for example a member participation attack Kairouz et al. [2019], Shokri et al. [2017]. Namely, the attacker may know if certain data point is present in the training set by queries. The person who is linked to the disclosed data could be targeted against by the leakage of this confidential information. Differential privacy mechanisms provide a common defence Dwork [2008].

Alternatively, adversarial attacks attempt to "poison" models during training and it may prevent model from learning patterns to some extent Bagdasaryan et al. [2020], Sun et al. [2019]. It often uses the skewed statistic to trick the learner by crafting the adversarial data such that the learner misclassify a target subset of data with strong confidence, and the adversarial data often cannot be distinguished from correct data by bare eyes Bhagoji et al. [2019]. In federated setup, Wang et al. [2019] showed a malicious server that employs a model Generative Adversarial Network (GAN) to learn a generative model with the gradient updates collected from distributed workers, even without access to the input data. In some applications, such attack is deadly. For instance, Lin et al. [2018] demonstrate the possibility of an adversarial attack applied on intrusion detection such that the model may misclassify an actual intrusion as safe.

Recent work demonstrated another severe attack - reconstruction of input data within an FL environment Zhu et al. [2019]. This idea was similar with model inversion Fredrikson et al. [2015], however, it is easy and effective to apply this attack in federated setup. By imitating an honest server, data may be reconstructed with only knowledge of the gradient update and model parameters. It completely breaches the promise of FL - data is kept locally (on generation devices). Here our aim is to explore the limits and efficiency of this highly interesting attack. We offer theoretical analyses of the minimal reconstruction requirements from solving the linear equations perspectives, based on both a fully-connected Neural Network and CNN. Moreover, we propose a stage-wise reconstruction solution and a regularizer that increases numerical stability. Our main contributions are the follows.

- We show that reconstruction of input amounts to solving a linear set of equations, and based on which we give minimal structural conditions for reconstruction based on the fully-connected neural network (a.k.a.Multilayer Perceptron (MLP)) and CNN.
- We show that MLP only needs **a single** node in **one** hidden layer to reconstruct a single input image, regardless of the depth. Complementing Zhu et al. [2019], we derive a closed form for a lossless reconstruction in this case. Moreover, we generalize the result to mini-batch reconstruction and show that the number of hidden units has to exceed the size of mini-batch.
- We show that for CNN, the number of filters in the first convolutional layer should be such that the size of output after passing through the convolutional layers exceeds the size of original input. Specifically, number of filters should meet the requirement: $h \geq (\frac{d}{d'})^2 C$.
- We propose the reconstruction method in response to three cases: 1) single-instance reconstruction using MLP; 2) single-instance reconstruction using CNN; 3) batch reconstruction using both MLP and CNN.
- We suggest to include an orthogonality regularizer for mini-batch MLP reconstruction, and it increases the numerical stability during iterative optimization.

The paper is organized as follows: in Section 2, we briefly introduce FL and present the key related works and in Section 3 we focus on our reconstruction method and offer the theoretical analysis based on the model architecture. Finally, we provide numerical demonstrations in Section 4 and conclude in Section 5.

2 Background and Related Works

2.1 Federated Learning

FL is a distributed ML diagram that incorporates a set of distributed workers (customers) and server to jointly train a global model. Unlike traditional Cloud-based ML Yang et al. [2019], it has no (training) data transmission across workers and server. Instead, the set of workers collaborate to optimize the global cost function that is estimated by a collection of local cost functions. The workers own their data, the full copy of the global model, and implement local training (minimizing local cost functions) using their data and update the gradients with the server. The global empirical loss function $\hat{\ell}$ (defined in equation (1)) can be approximated, for instance by average of local loss functions $\hat{\ell}_i$. Data owned by all workers are assumed to be generated from the same underlying data distribution $\mathcal{D}$, and the empirical distribution is denoted as $\hat{\mathcal{D}}$. We define $m = \sum_{i=1}^{p} m_i$ where m_i is the amount of data on worker i. We assume that p active workers participate in each round, and all the workers share the same batch size B.

$$\begin{aligned} \hat{\ell}(x, y) &= \sum_{i=1}^{p} \frac{m_i}{m} \mathbb{E}_{(x,y)\sim\hat{\mathcal{D}}}[\hat{\ell}_i(f_w(x), y)] \\ &= \frac{1}{p} \sum_{i=1}^{p} \mathbb{E}_{(x,y)\sim\hat{\mathcal{D}}}[\hat{\ell}_i(f_w(x), y)] (\text{if } m_i = m_j, \forall i, j) \end{aligned} \tag{1}$$

Each iteration consists of two stages: local training (workers) and aggregation (server). More specifically, after distributed workers have completed local training, they share the corresponding gradients with the server, and the server aggregates the gradients based on the given criterion and finally it sends the updated global model (or aggregated gradients) back to the workers. The workers will use the updated model for the next iteration. This procedure can be repeated for multiple times.

2.2 Related Work

GAN-based attacks are proposed as a means to adversely train the global model in federated environment. Wang et al. [2019] employs a multi-task GAN whose discriminator is trained by the gradient update from the victim. Its discriminator simultaneously discriminates for multiple tasks: category, reality, and client identity of input samples. Melis et al. [2019] shows that it is possible to have the membership attack and also infer properties that hold only for a subset of the training data. Hitaj et al. [2017] assumes that one participant is an attacker, who owns a discriminator with the same architecture as the classifier. Moreover, the attacker generates fake images and adversely updates the global model to force the victim to reveal more information. GAN-based attacks are known to increase the complexity of training the model and partially disclosing the data distribution. Most of the attacks mentioned beforehand are white-box attacks, in which attackers have access to the complete model description. For a black-box attack, the attackers can typically ask for a prediction or other query. For instance, in Fredrikson et al. [2014], the attacker uses black-box access to the model to infer a sensitive feature x_i. More precisely, given joint distribution and marginal priors they employ Maximum a Posterior (MAP) estimator that may sample a promising x_i that maximizes the posterior probability.

The current state-of-the-art is Zhu et al. [2019]. It proposed to reconstruct input using the knowledge of gradients and model parameters by firstly giving the initial guess sampled from normal distribution and iteratively optimize it through minimizing the distance between gradients and guessed gradients. However, it does not give strong experimental results on big batch reconstruction. Zhao et al. [2020] expands based on the work of Zhu et al. [2019] to improve the label prediction accuracy. Geiping et al. [2020] proposed a new cost function and reconstruct the input based on deep neural networks like ResNet [He et al., 2016]. Wei et al. [2020] introduces a framework for evaluation of privacy leakage and relation to FL hyperparameters. Pan et al. [2020] also focus on deep neural network like VGG Simonyan and Zisserman [2014], GoogLeNet Szegedy et al. [2015], which are normally are more informative for the reconstruction. We are interested to provide the lower-bound requirements analysis of reconstruction based on the network structures since none of them did before.

3 Reconstruction Method and Theoretical analysis

In this section, we will first present the reconstruction method and then introduce the theoretical analysis of minimal reconstruction conditions mainly based on two architectures: MLP and CNN.

3.1 Reconstruction Method

Input reconstruction is essentially an inverse problem, without the loss of generality, say we have any function G parameterized by w, which maps from input $x \in R^d$ to a (gradient) vector v ($|v| = p$), thus $G : \mathbb{R}^d \to \mathbb{R}^p$. For the batch case, the output of function G is often the mean gradient $\overline{v} = \frac{1}{B}\sum_{i=1}^{B} G(x_i, y_i; w)$. We aim to compute x with the knowledge of model parameters w and output v (or $\overline{v}$). If G^{-1} exists, we can compute x directly, which is typically not the case. The intriguing question is that is it still possible to reconstruct x in the noninvertible case? We show that in Section 3.2 one-instance reconstruction based on MLP has an analytical form and can be computed directly. This is different from Zhu et al. [2019] where they consider it as noninvertible and address it as an optimization problem. For one-instance CNN reconstruction, we propose a two-step method where we first compute the output of convolutional layer using the closed-form mentioned before, and then we apply the optimization method. While, for batch reconstruction, there is no analytical form in most cases. We convert it to the optimization problem; more specifically, we start from a random guess $(\hat{x}, \hat{y})$, and gradually pull it to close to the original input x by minimizing the cost function, like proposed by Zhu et al. [2019]. For the batch reconstruction, we augment the cost function by an additive regularizer as $L(.) + \lambda R(.)$ where $L(.)$ is the distance between ground-truth gradients and guessed gradients. For $R(.)$ we suggest an orthogonality regularizer defined as $R = \lambda \sum_{k \neq k'=1}^{n} \left(\hat{x}_k^\intercal \hat{x}_{k'}\right)^2$, if $\hat{x}_k$ and $\hat{x}_{k'}$ are orthogonal the product is zero, which implies that the optimizer promotes solutions where the batch members are dissimilar. Or a L2 regularizer defined as $\lambda \sum_i \hat{x}_i^\intercal \hat{x}$, and we will experimentally explore the critical role of the regularizer, particularly when batch size is large in Section 4.

We show insecure FL with potential reconstruction attack in Algorithm 1 where the attacker could be the server who has the complete knowledge of gradients update and model parameters in each round. In Algorithm 2, we present the reconstruction approaches in response to three cases: 1) one-instance MLP reconstruction, 2) one-instance CNN reconstruction, and 3) batch reconstruction (using MLP and CNN). The pseudo code in Algorithm 3 demonstrates the iterative optimization step.

Algorithm 1 Insecure FL with reconstruction attack

Initialization: w^0
for t=1,...T **do**
  =>**workers:**
  for j=1,2,..,p **do** p workers (in Parallel)
    $v_j^t = \nabla \hat{\ell}_j(f(X_j^t; w^t), Y_j^t)$
    with $(X_j^t = \{x_{jk}^t\}_{k=1,..B}, Y_j^t = \{y_{jk}^t\}_{k=1,...,B}) \sim \hat{\mathcal{D}}$
    share v_j^t with server
  end
  =>**server(attacker):**
  $w^{t+1} = w^t - \eta \times \frac{1}{p} \sum_{j=1}^{p} v_j^t$
  share w^{t+1} with workers for next round
  $\hat{X}_j^t, \hat{Y}_j^t$ =*Reconstruction*$(v_j^t, w^t, m, \lambda,$ prior$)$(Algorithm 2)
end

Algorithm 2 *Reconstruction*

Input: $v, w, m, \lambda,$ prior
if 1) single recon. & mlp **then** ▷ case one
  Return $\hat{x} = \frac{\partial \ell}{\partial w_{1i}^1} / \frac{\partial \ell}{\partial b_1^1} \quad \forall i < d$ ▷ w_{1i}^1, b_1^1 are the weights and bias in 1st hidden layer, d is input dimension
End
if 2) single recon. & cnn **then** ▷ case two
  $\hat{z} = \frac{\partial \ell}{\partial w_{1i}^1} / \frac{\partial \ell}{\partial b_1^1} \quad \forall j < d'$ ▷ w_{1j}^1, b_1^1 are the weights and bias on 1st hidden layer after conv. layer, d' is dimension of output of conv. layer
  Return Itr_rec$(\hat{z}, w_{\text{partial}},$ "partial", $m, \lambda,$ prior$)$ ▷ w_{partial} refers to the params. of convol. layer
End
if 3) batch reconstruction **then** ▷ case three
  Return Itr_rec$(v, w,$ "all", $m, \lambda,$ prior$)$
End

Our method implementation deviates from the pioneering work Zhu et al. [2019] in a few ways. First, we derive a closed-form for one-instance MLP reconstruction, which leads to a faster and more accurate reconstruction (almost lossless). Second, we divide one-instance CNN reconstruction into two steps; first, we directly compute the output of convolutional layer using the advantage of closed-form and based on which we reconstruct the input (a.k.a deconvolution), which speeds up the overall reconstruction. Last, we expand the cost function with either an orthogonality regularizer or L2 regularizer for batch reconstruction. The orthogonality regularizer may penalize the similarities between reconstructed images (since we only know the average gradient of the batch), mainly when image patterns are similar (e.g., MNIST). In contrast, L2 regularizer offers faster convergence, in particular, when batch size is big.

3.2 Reconstruction with fully-connected neural network

Say we have a one-layer MLP f, with n_1 units in hidden layer and n_2 units in output layer (if classification task, n_2 is equal to the number of classes). Thus, we can define the output of model as $a_j = f_w(x) = \sum_{i=1}^{n_1} w_{ji}^2 \sigma(w_i^1 x + b_i^1) + b_j^2 \quad \forall j \in [1, n_2]$, where w^1 and b^1 are the weights and bias in hidden layer, and w^2, b^2 in output layer and $\sigma(x)$ is sigmoid (monotonic) activation function. For the classification task, we employ cross-entropy as the cost function $\ell(p_i, y_i) = -\sum_j^C y_{ij} \log p_{ij}$ where $p_{ij} = \frac{e^{a_j}}{\sum_k e^{a_k}}$ after softmax function ($|p_i| = C$), and y_i is the one-hot encoding vector with all zeros except the corresponding class indicating one.

Proposition 1 (ONE-INSTANCE MLP RECONSTRUCTION)**.** *To reconstruct one input based on MLP, we derive the analytical form to compute the (almost) lossless input, with only* ***single*** *unit in the first hidden layer as long as bias term exists, regardless how deep the network is.*

Algorithm 3 *Itr_rec*

Input: $v, w,$ flag, $m, \lambda,$ prior
for i=0,1,2,..,I **do** I iterations
  if i==0 **then**
    if prior==uniform **then**
      $\hat{X}_0, \hat{Y}_0 \sim \mathbb{U}(\mathbb{0}, \mathbb{1})$
    if prior==normal **then**
      $\hat{X}_0, \hat{Y}_0 \sim \mathbb{N}(\mathbb{0}, \mathbb{1})$
    $\hat{v}_1 = G(\hat{X}_0, \hat{Y}_0; w)$ ▷ G is function of computing gradients
  else
    $\hat{v}_i = G(\hat{X}_i, \hat{Y}_i; w)$
    if flag=="all" **then**
      $L = \|v - \hat{v}_i\|_2^2 + \lambda R(\hat{X}_i)$ ▷ R is the regularizer
    else
      $L = \|v - \hat{v}_i\|_2^2$
    end
    update: $\hat{X}_{i+1} = \hat{X}_i - \eta \times \nabla_{\hat{X}_i} L$
      $\hat{Y}_{i+1} = \hat{Y}_i - \eta \times \nabla_{\hat{Y}_i} L$
  if (i//m)==0 **then**
    $\lambda = 0.9 * \lambda$
  end
end
return $\hat{X}_I, \hat{Y}_I$

Proof. The derivative of loss function ℓ w.r.t a_j (output of network f_w) is $p_j - y_j$ by plugging eq. (2) into eq. (3).

$$\frac{\partial p_i}{\partial a_j} = \begin{cases} p_j(1-p_j), & i = j \\ -p_j p_i, & i \neq j \end{cases} \tag{2}$$

$$\begin{aligned} \frac{\partial \ell}{\partial a_j} &= -\sum_k y_k \frac{\partial \log p_k}{\partial a_j} = -\sum_k y_k \frac{1}{p_k} \frac{\partial p_k}{\partial a_j} \\ &= -[\frac{y_j}{p_j}(p_j(1-p_j)) - \sum_{k \neq j} \frac{y_k}{p_k} p_j p_k] \\ &= p_j - y_j \end{aligned} \tag{3}$$

Thus, all the partial derivatives of Jacobian matrix are as the followings:

$$\frac{\partial \ell}{\partial b_j^2} = p_{.j} - y_{.j} \quad \forall j \in [1, n_2] \tag{4}$$

$$\begin{aligned} \frac{\partial \ell}{\partial w_{ji}^2} = (p_{.j} - y_{.j})\sigma(w_i^1 x + b_i^1) = (p_{.j} - y_{.j})\sigma_i \\ \forall j \in [1, n_2], \forall i \in [1, n_1] \end{aligned} \tag{5}$$

$$\frac{\partial \ell}{\partial b_j^1} = \sum_i^{n_2} (p_{.i} - y_{.i}) w_{ij}^2 \sigma'(w_j^1 x + b_j^1) = \sum_i^{n_2} (p_{.i} - y_{.i}) w_{ij}^2 \sigma_j' \tag{6}$$

$$\frac{\partial \ell}{\partial w_{ji}^1} = \sum_k^{n_2} (p_{.k} - y_{.k}) w_{kj}^2 \sigma_j' x_i \tag{7}$$

We can directly compute $x_i = \frac{\partial \ell}{\partial w_{1i}^1} / \frac{\partial \ell}{\partial b_1^1}, \forall i \in [1, d]$ from eq. (6) and (7) where j is equal to one since only one unit is required in hidden layer. □

It can be generalized to deep fully-connected neural network (say last layer is L). We only need two ingredients in our recipe, the partial derivative w.r.t the bias term in the first hidden layer $\frac{\partial \ell}{\partial b_n^1} = \sum_h^{n_L} (p_{.h} -$

$y_{.h}) \sum_j^{n_{L-1}} w_{hj}^L (\sigma_j^{L-1})' \cdots w_{mn}^2 (\sigma_n^1)'$, and the partial derivative w.r.t weights in the first hidden layer $\frac{\partial \ell}{\partial w_n^1} = \sum_h^{n_L} (p_{.h} - y_{.h}) \sum_j^{n_{L-1}} w_{hj}^L (\sigma_j^{L-1})' \cdots w_{mn}^2 (\sigma_n^1)' x$. Note that n_L is the number of nodes in layer L. Training with Stochastic Gradient Descent (SGD) is very vulnerable to reconstruction attack in FL (distributed ML), in particular, on fully-connected neural network no matter how deep it is.

Proposition 2 (BATCH MLP RECONSTRUCTION)**.** *For batch reconstruction, the number of units in first hidden layer should meet $n_1 \geq B$, given the high-dimension input whose dimension $d \gg n_2, d \gg B$.*

Proof. (sketch:) Reconstructing $x_1, x_2, ..., x_B \in \mathbb{R}^d$ approximates to solving the linear equations. Given an invertible sigmoid function and a single output y_i, we may compute the unique $\sigma'(x_i)$. The number of equations exceeds the number of variables $n_2 + n_1 n_2 + n_1 + n_1 d \geq B n_2 + n_1 B + Bd$, thus $n_1 \geq \frac{B n_2 + Bd - n_2}{n_2 + 1 + d - B}$. Typically we have $d \gg n_2, d \gg B$, and n_1 is dominated by $\frac{B(d+n_2)}{d+n_2}$, therefore $n_1 \geq B$. □

In general, batch reconstruction is more challenging since we typically only know the average gradient or the sum of gradient and aim to reconstruct every individual instance. We refer to this procedure as *demixing* in the following context. Theoretically, if the number of equations is identical or greater than batch size and all the equations are independent, it is solvable. However, it is not easy to solve in practice, mainly when x is high-dimensional and the scales in the linear system are extremely small (close to zero, different though). We use iterative method to solve it. Besides, the to-be-optimized input value during optimization might introduce the saturation of sigmoid function, i.e., $\sigma(x) \approx 0, \sigma(x) \approx 1$ regardless the change of x outside the interval $[-4, 4]$. The choice of optimization method is important. For a high-dimensional (non-sparse) input, second-order or quasi-second-order methods sometimes fail since they are designed to search for the zero-gradient area. The number of saddle points exponentially increases with the number of input dimensions Dauphin et al. [2014]. The first-order method, e.g., Adam Kingma and Ba [2014] takes much more iterations to converge, but it is relatively more stable, likely to escape from the saddle points Goodfellow et al. [2016].

3.3 Reconstruction with convolutional neural network

Proposition 3 (SINGLE-LAYER CNN RECONSTRUCTION)**.** *To reconstruct a single-layer CNN immediately stacked by a fully-connected layer, $h \geq (\frac{d}{d'})^2 C$ kernels are required, where C is the channel number of input, d is the width of input, and d' is width after convolution and pooling layers.*

For a single convolutional layer CNN, the kernel parameters (kernel size k, padding size p, stride size s, and pooling stride q) determine the output size d' after convolutional layers. Say we have input $X \in \mathbb{R}^{B \times C \times d \times d}$, for the simplicity we assume height and width are identical, and C is the channel number and B is the batch size. We define a square kernel with width k (weights indicated as l^0), bias term r, and we have h kernels. After convolutional layer, the width of output is $d' = \frac{d+2p-k+s}{s \times q}$. The output of convolutional layer is shown in eq. (8), and more specifically the convolutional operator can be expressed in eq. (9) where $\hat{x}$ is the matrix after padding.

$$z_{\underline{m}} = \sum_{c=1}^{C} x_c * l_{\underline{m}c}^0 + r_{\underline{m}} \tag{8}$$

$$\begin{aligned} z_{\underline{m}ij} &= (\sum_{c=1}^{C} \sum_{g=1}^{k} \sum_{n=1}^{k} l_{\underline{m}cgn}^0 \hat{x}_{c,si+g-1,sj+n-1}) + r_{\underline{m}} \\ &\forall (i,j) \in [1, d'] \times [1, d'], \forall \underline{m} \in [1, h] \end{aligned} \tag{9}$$

Then we define $H = [\text{vec}(\hat{z}_{\underline{1}..}), \text{vec}(\hat{z}_{\underline{2}..}), ..., \text{vec}(\hat{z}_{\underline{h}..})]$, $\hat{z}_i$ is the output of convolutional layer, and $|H| = h(d')^2 = n_0$, after convolutional layer we have one hidden layer and an output layer. It is expressed as $p_j = \sum_{i=1}^{n_1} w_{ji}^2 \sigma(w_i^1 H + b_i^1) + b_j^2$, where $w^1 \in \mathbb{R}^{n_1 \times n_0}$, $w^2 \in \mathbb{R}^{n_2 \times n_1}$, $b^1 \in \mathbb{R}^{n_1}$ and $b^2 \in \mathbb{R}^{n_2}$ are the weights and bias in the hidden and output layer, $\sigma()$ is sigmoid function as defined before. Thus, we have the derivatives w.r.t w^1, b^1, w^2, b^2 shown in eq. (4), (5), (6) and (7). Moreover, the partial derivatives w.r.t r and l^0 are shown in eq. (10) and (11).

$$\frac{\partial \ell}{\partial r_{\underline{m}}} = \sum_{i=1}^{d'} \sum_{j=1}^{d'} \frac{\partial \ell}{\partial z_{\underline{m}ij}} \frac{\partial z_{\underline{m}ij}}{\partial b_{\underline{m}}^1} = \sum_{i=1}^{d'} \sum_{j=1}^{d'} \frac{\partial \ell}{\partial z_{\underline{m}ij}} \quad \forall \underline{m} \in [1, h] \tag{10}$$

$$\frac{\partial \ell}{\partial l_{\underline{m}cgh}^0} = \sum_{i=1}^{d'} \sum_{j=1}^{d'} \frac{\partial \ell}{\partial z_{\underline{m}ij}} \hat{x}_{c,si+g-1,sj+h-1} \quad \forall \underline{m} \in [1, h] \tag{11}$$

$$\frac{\partial \ell}{\partial \hat{z}_{\underline{m}ij}} = \frac{\partial \ell}{\partial H[(m-1) \times (d')^2 + (i-1) \times d' + j]} \tag{12}$$

The number of equations should be equal or greater than the number of unknowns. From eq. (9), we have $(d')^2Bh$ equations and d^2BC unknowns, thus we need $h \geq (\frac{d}{d'})^2C$, with the assumption that H is known, which can be solved by enough units in dense layer from Proposition 2. One special case is that convolutional layer is stacked by an output layer directly (no dense layer), then we need to meet $h \geq (\frac{d}{d'})^2CB$, $n_0 \geq \frac{n_1(B-1)}{n_1-B}$, and $1 < B < n_1$ to solve H since σ is monotonic activation function ($n_0 = h(d')^2$). Note here n_1 indicates the number of units in output layer (as no dense layer).

CNN reconstruction can be seen as a two-stage reconstruction: *demixing* and *deconvolution*. Namely, the demixing stage is essentially an inverse procedure of fully-connected network and we aim to reconstruct the output of convolutional layer (input of fully-connected layer). The deconvolution stage we want to reverse the convolutional step, starting from the output of convolutional layer to reconstruct the original input. For the demixing stage, we can apply the conclusion from MLP, as long as the number of hidden units is equal or greater than the batch size, then theoretically we may evaluate the individual instance.

Generalizing it to the multiple-convolutional-layer neural network, the output H^{i-1} of layer $i-1$ is the input of layer i, then we can express it as the following, where h^{i-1} and k^{i-1} are the number of kernels and kernel width in layer $i-1$.

$$H^i_{mij} = (\sum_{c=1}^{h^{i-1}} \sum_{g=1}^{k^{i-1}} \sum_{n=1}^{k^{i-1}} l^i_{mcgn} H^{i-1}_{c,si+g-1,sj+n-1}) + b^i_m \tag{13}$$

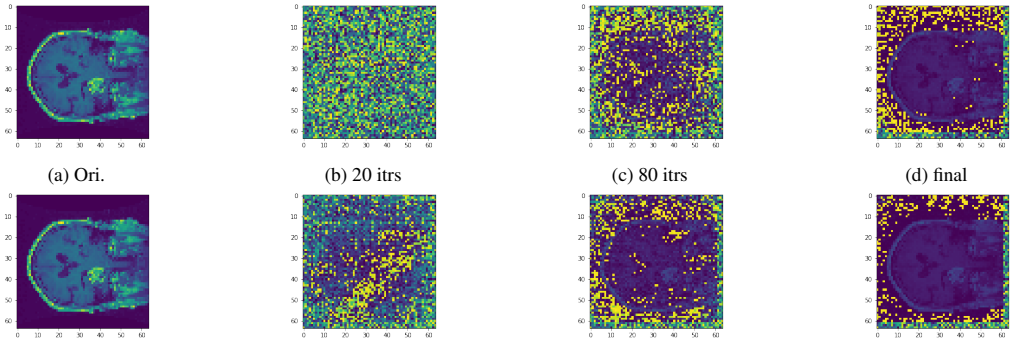
We can recursively calculate the number of kernels required in each convolutional layer from right to left order (from the last convolutional layer to the first one). One example of two-layer CNN is shown in Figure 13.

4 Experimental Results

Dataset and Setup. fMRI image (available here) set is the brain tumor dataset containing 3064 images from 233 patients with three types of brain tumor (meningioma, glioma, pituitary). Face dataset contains 40 individuals, and every image has size $1 \times 32 \times 32$. MNIST and KMNIST are the handwritten digits and Kuzushiji accordingly, with size $1 \times 28 \times 28$ and it totally contains 10 classes. CIFAR100 contains 100 classes and every class has 600 images with size $3 \times 32 \times 32$. ImageNet contains 1000 classes, and each image has size $3 \times 64 \times 64$. Every image in White Blood Cell (WBC) dataset has size $3 \times 240 \times 320$ and four classes (Eosinophi, Neutrophil, Lymphocyte and Monocyte). We have implemented our method in Python and run it on a Titan X GPU with Architecture Maxwell and 12 GB of Ram.

Prior Image Information. We suggest to use the initialization based on the prior knowledge of image preprocessing to fasten the convergence and increase the numerical stabilization. Zhu et al. [2019] normalizes data between zero and one and initialize the guess by normal distribution, instead we suggest to initialize it from an uniform distribution. We plot the single-instance reconstruction based on LeNet LeCun et al. [2015] with the optimizer L-BFGS Liu and Nocedal [1989] the same as Zhu et al. [2019] did for the fair comparison. The reconstruction error is shown in Figure 3, due to the improved initialization, our reconstruction turns out to have lower error. Moreover, we showed several reconstruction snapshots in Figure 1, 2 test on face dataset and fMRI dataset.

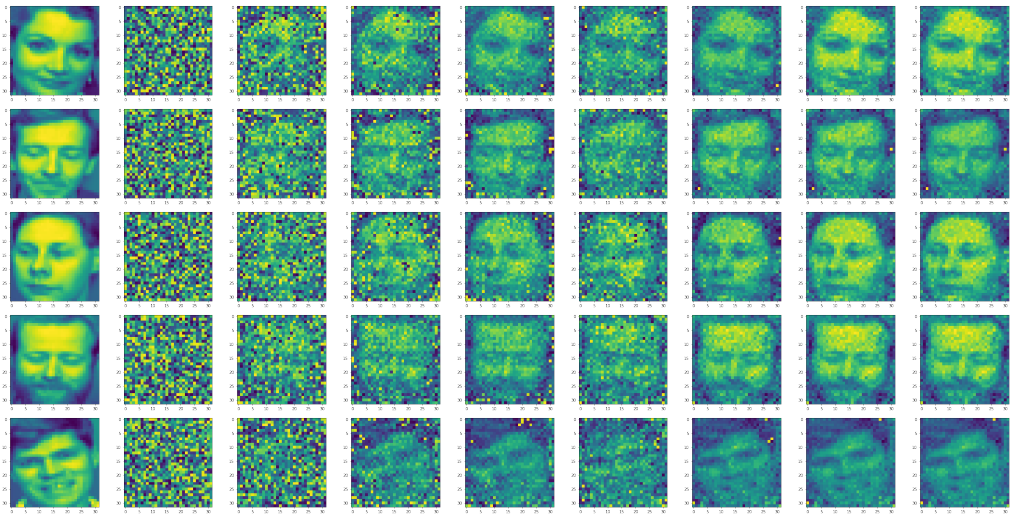
MLP Reconstruction Analysis. We implement one-instance reconstruction on MLP by the analytical form $x_i = \frac{\partial \ell}{\partial w^1_{1i}} / \frac{\partial \ell}{\partial b^1_1}, \forall i \in [1, d]$, using one-hidden layer MLP with only one unit in hidden layer. The outcome is demonstrated in Figure 4 using ImageNet and WBC dataset, the average L1 distance per pixel between original input and reconstruction is bounded below $1\mathrm{e}-8$, which is almost lossless with $\mathcal{O}(n)$ complexity. For MLP batch reconstruction, we experimentally set the batch size equal to four, identical to the number of units in hidden layer, as shown in Figure 5. We first give the final reconstruction of four inputs without regularizer in Figure 5a, whereas in Figure 5b the reconstruction quality is being improved significantly with the orthogonality regularizer. More specifically, we start λ from 0.1 and gradually decay after 200 epochs by 90%. Besides, from Figure 5c to 5f, we partially show the reconstruction procedure and we can see how the regularizer plays the key role during optimization procedure to hinder the similarities between instances. Moreover, we also show MLP reconstruction with batch size 16 in Figure 9a and 9b. Empirically, it shows that the optimization for high-dimension demixing is very challenging and the existence of orthogonality regularizer significantly improves the numerical stability and the reconstruction quality. Sometimes the choice of m (interval that orthogonality regularizer λ decays) is crucial, for instance in Figure 6b the reconstruction performance is significantly distinct with different values. While, in Figure 6a, the choice of m is insensitive.



(a) Ori. (b) 20 itrs (c) 80 itrs (d) final

(e) ori. (f) 20 itrs(our) (g) 80 itrs(our) (h) final(our)

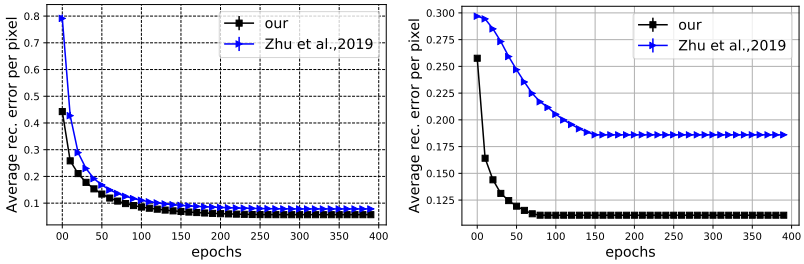
Figure 1: *fMRI*: The plots (first row) are produced by Zhu et al. [2019], and it shows the reconstructions after 20, 80, and final iteration accordingly, whereas the second row corresponds to our reconstruction (with L-BFGS optimizer).



(a) ori. (b) 10 (c) 50 (d) 90 (e) final (f) 10(our) (g) 50(our) (h) 90(our) (i) fi.(our)

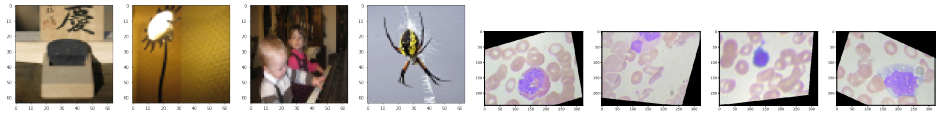
Figure 2: *Face* (batch reconstruction): Mini-batch contains 5 images, and we show partial reconstruction for 10, 50, 90, and final (400 iterations, with L-BFGS optimizer) iteration for Zhu et al. [2019] from the second to the fifth column. From the sixth to the last columns correspond to our method and the first column is the original input.

CNN Reconstruction Analysis. We divide one-instance CNN reconstruction into two steps: 1) we directly compute the output of convolutional layer; 2) we apply the iterative optimization. We test the reconstruction performance with different numbers of kernels. We set kernel size 5, padding size 2, stride size 2 and pooling size 1, thus at least 12 kernels are required according to Proposition 3. In Figure 8 we first visually show the reconstruction improvement with the incremental numbers of kernels, and from Table 8f we numerically show reconstruction error (L1 distance) decreasing with more kernels. For batch CNN reconstruction, we demonstrate it in Figure 9c and 9d where we have the similar convolutional setup with one-instance CNN, thus 12 kernels are applied in convolutional layer. Moreover, the experimental result in Figure 7 (appendix) shows that the two-step reconstruction enables a faster convergence and lower error compared with Zhu et al. [2019]. Note for the redundant architectures, i.e., with more kernels and units than required, we can easily mask the corresponding gradients during iterative optimization to fasten the reconstruction step.



(a) fMRI reconstruction error (b) Face reconstruction error

Figure 3: Reconstruction Loss with L-BFGS optimizer.



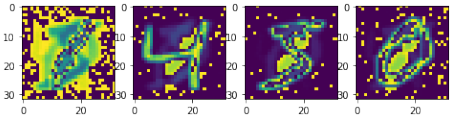
(a) (b) (c) (d) (e) (f) (g) (h)

Figure 4: ImageNet and White Blood Cell analytical reconstruction.

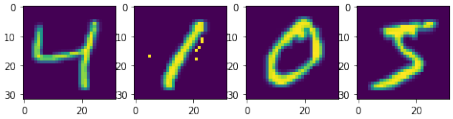
Large batch size. Last, we did the experiments with batch size equal to 100 using MLP in Figure 10. We test it on MLP with one hidden layer (100 units according to Proposition 2). As shown in Figure 10a and 10b, when batch size increases, the impact of the regularizer is fairly obvious; without the regularizer, some reconstructed images are difficult to recognize in Figure 10b. Note the order of images are changed comparing with the original inputs due to the random initialization. Similarly, we demonstrate the case of CNN with kernel size 5, padding size 2, stride 2 and pooling size 1. According to Proposition 3 and Proposition 2, 12 kernels and 100 nodes in hidden layer are required. We can tell that the reconstruction with regularizer is significantly improved.

5 Conclusion

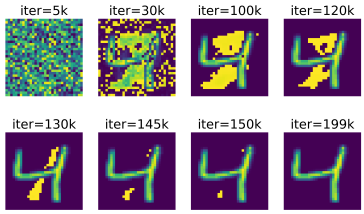
We theoretically studied the minimal structural requirements for reconstruction and analyzed the relations between network architecture, size, and reconstruction quality. We show that the number of units in first hidden layer should be equal to greater than batch size using MLP. It is worthwhile to mention that joint training using MLP (with bias) with batch size equal to one in FL is extremely venerable to the adversary. For CNN, the number of kernels along with the number of units in a fully-connected (dense) layer decides the quality of reconstruction. Specifically, number of units in hidden layer is determined by batch size and required number of kernels is decided by output dimension after convolutional layers. Our observations also apply to big batch size with the aid of the regularizer. We hope that the limits explored in the present work and conditions for reconstruction can aid the practitioners to choose network architecture and communication strategies when applying FL on sensitive information-related applications, e.g., medical data, financial data.



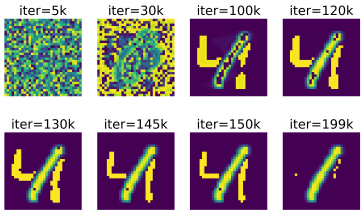
(a) No regularizer



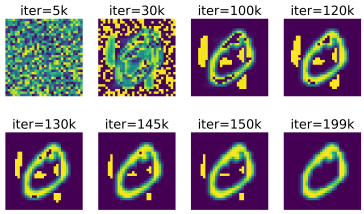
(b) with orthogonality regularizer



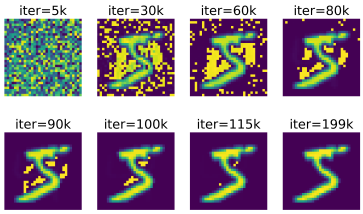
(c) 1/4 recon. procedure



(d) 2/4 recon. procedure

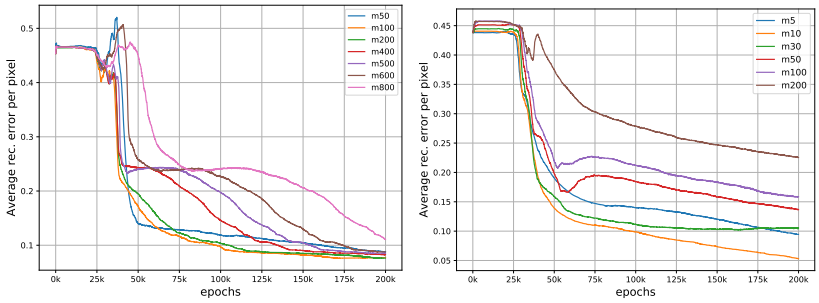


(e) 3/4 recon. procedure



(f) 4/4 recon. procedure

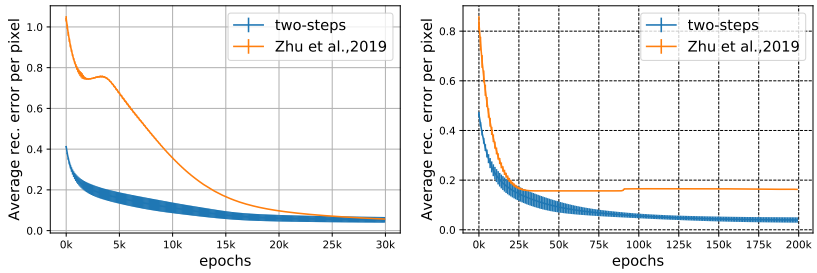
Figure 5: *MNIST*: 5a and 5b show the final reconstruction without and with regularizer accordingly. Moreover, the reconstructions procedures of 5b are shown in 5c, 5d, 5e and 5f.



(a) MNIST

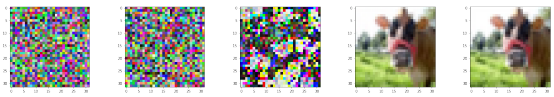
(b) KMNIST

Figure 6: Orthogonality regularizer with different interval values to decay.



(a) CIFAR100 (b) MNIST

Figure 7: CNN one-instance reconstruction (Adam optimizer): the blue curve shows the average distance per pixel of the overall iterative reconstruction, and the orange curve is the result of two-step reconstruction.

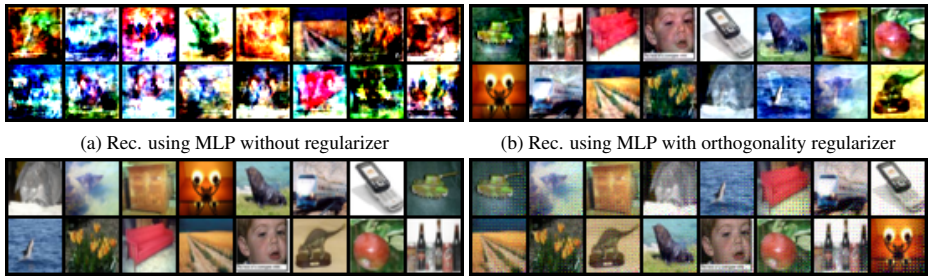


(a) 1 filter (b) 5 filt.s (c) 11 filt.s (d) 12 filt.s (e) ori.

Filters	**Params.** $((b')^2h)$	**Mean L1 error.**
1	64	2.4
5	1280	0.21
11	2816	0.04
12	3072	0.00019

(f) L1 distance

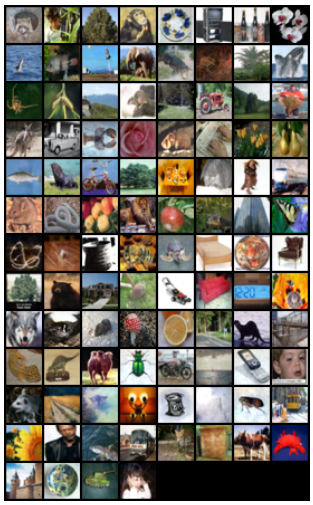
Figure 8: *One-layer CNN*: we show that the image reconstruction changes with increasing number of filters (numerical result is shown in Table 8f).



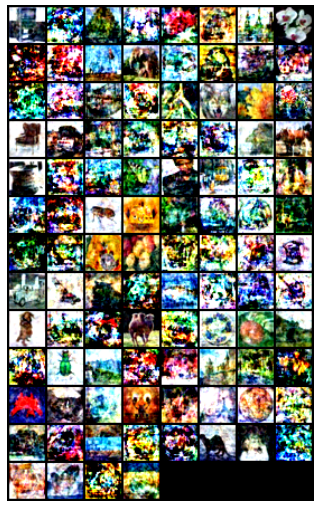
(a) Rec. using MLP without regularizer (b) Rec. using MLP with orthogonality regularizer

(c) original input (d) Rec. using CNN with L2 regularizer

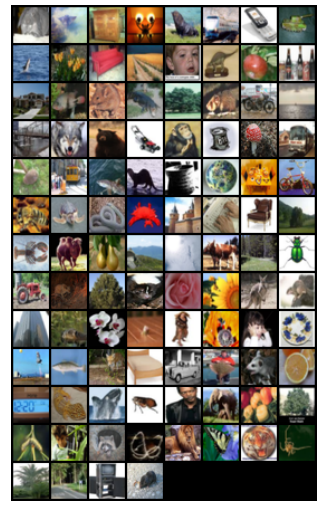
Figure 9: Batch size 16 (MLP and CNN).



(a) Reconstruction with L2 regularizer

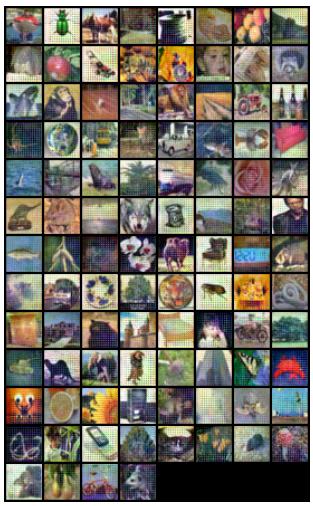


(b) Reconstruction without regularizer

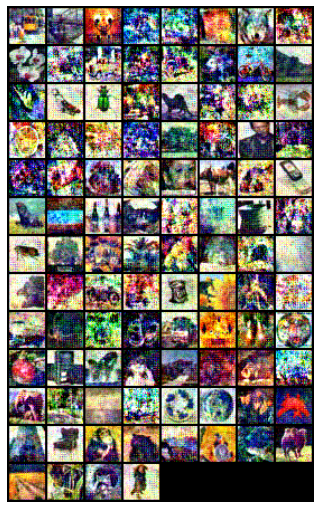


(c) Original input

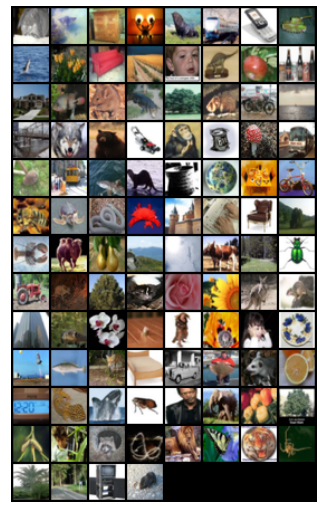
Figure 10: MLP reconstruction with batch size 100 (CIFAR100).



(a) Reconstruction with L2 regularizer



(b) Reconstruction without regularizer



(c) Original input

Figure 11: CNN reconstruction with batch size 100 (CIFAR100).

References

Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. In *Advances in Neural Information Processing Systems*, pages 14774–14784, 2019.

Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 2019.

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

Cynthia Dwork. Differential privacy: A survey of results. In *International conference on theory and applications of models of computation*, pages 1–19. Springer, 2008.

Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How to backdoor federated learning. In *International Conference on Artificial Intelligence and Statistics*, pages 2938–2948. PMLR, 2020.

Ziteng Sun, Peter Kairouz, Ananda Theertha Suresh, and H Brendan McMahan. Can you really backdoor federated learning? *arXiv preprint arXiv:1911.07963*, 2019.

Arjun Nitin Bhagoji, Supriyo Chakraborty, Prateek Mittal, and Seraphin Calo. Analyzing federated learning through an adversarial lens. In *International Conference on Machine Learning*, pages 634–643, 2019.

Zhibo Wang, Mengkai Song, Zhifei Zhang, Yang Song, Qian Wang, and Hairong Qi. Beyond inferring class representatives: User-level privacy leakage from federated learning. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 2512–2520. IEEE, 2019.

Zilong Lin, Yong Shi, and Zhi Xue. Idsgan: Generative adversarial networks for attack generation against intrusion detection. *arXiv preprint arXiv:1809.02077*, 2018.

Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1322–1333, 2015.

Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19, 2019.

Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. Exploiting unintended feature leakage in collaborative learning. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 691–706. IEEE, 2019.

Briland Hitaj, Giuseppe Ateniese, and Fernando Perez-Cruz. Deep models under the gan: information leakage from collaborative deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 603–618, 2017.

Matthew Fredrikson, Eric Lantz, Somesh Jha, Simon Lin, David Page, and Thomas Ristenpart. Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, pages 17–32, 2014.

Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. idlg: Improved deep leakage from gradients. *arXiv preprint arXiv:2001.02610*, 2020.

Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. Inverting gradients–how easy is it to break privacy in federated learning? *arXiv preprint arXiv:2003.14053*, 2020.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Wenqi Wei, Ling Liu, Margaret Loper, Ka-Ho Chow, Mehmet Emre Gursoy, Stacey Truex, and Yanzhao Wu. A framework for evaluating gradient leakage attacks in federated learning. *arXiv preprint arXiv:2004.10397*, 2020.

Xudong Pan, Mi Zhang, Yifan Yan, Jiaming Zhu, and Min Yang. Theory-oriented deep leakage from gradients via linear equation solver. *arXiv preprint arXiv:2010.13356*, 2020.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in neural information processing systems*, pages 2933–2941, 2014.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT Press, 2016.

Yann LeCun et al. Lenet-5, convolutional neural networks. *URL: http://yann. lecun. com/exdb/lenet*, 20(5):14, 2015.

Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1):503–528, 1989.

Hanie Sedghi, Vineet Gupta, and Philip M Long. The singular values of convolutional layers. *arXiv preprint arXiv:1805.10408*, 2018.

A Appendix

A.1 Inversibility of CNN

For simplicity we consider the case of one filter with stride equal to 1. Let a 2D image Y resulted from the convolution of a 2D image X with a 2D convolution kernel $\mathcal{K}$, thus $Y = \mathcal{K} * X$. The filter coefficients are simply a $k \times k$ matrix. The image X and the image Y are of size $n \times n$. The circulant matrix equivalent to the convolution filter, allows us to embed this $k \times k$ matrix into an $n \times n$ matrix, call it K, by padding with zeroes (which corresponds to the fact that the offsets with those indices are not used). Let A be a matrix which represents a convolutional layer. Matrix A is doubly block circulant $n^2 \times n^2$ matrix that is, A is a circulant matrix of $n \times n$ blocks that are in turn circulant. Let $\mathrm{vec}(X)$ be the vector obtained by stacking the columns of a matrix X.

Lemma 4 (Sedghi et al. [2018], Lemma 1)**.** *For any set of filter coefficients $\mathcal{K}$, the linear transform for the convolution by $\mathcal{K}$ is represented by the following doubly block circulant matrix:*

$$A = \begin{bmatrix} \mathrm{circ}\left(K_{0,:}\right) & \mathrm{circ}\left(K_{1,:}\right) & \dots & \mathrm{circ}\left(K_{n-1,:}\right) \\ \mathrm{circ}\left(K_{n-1,:}\right) & \mathrm{circ}\left(K_{0,:}\right) & \dots & \mathrm{circ}\left(K_{n-2,:}\right) \\ \vdots & \vdots & \vdots & \vdots \\ \mathrm{circ}\left(K_{1,:}\right) & \mathrm{circ}\left(K_{2,:}\right) & \dots & \mathrm{circ}\left(K_{0,:}\right) \end{bmatrix} \tag{14}$$

That is, if X is an $n \times n$ matrix, and

$$Y_{ij} = \sum_{p=1}^{n} \sum_{q=1}^{n} X_{i+p,j+q} K_{p,q} \quad \forall ij$$

then $\mathrm{vec}(Y) = A\,\mathrm{vec}(X)$

Define $\omega_n = \exp(2\pi\sqrt{-1}/n)$ be the primitive nth root of unity. Let F be a $n \times n$ complex matrix such that $F_{ij} = \omega_n^{ij} = \omega_n^i \omega_n^j = \exp(2\pi i\sqrt{-1}/n)\exp(2\pi j\sqrt{-1}/n) \in \mathbb{C}$. Matrix F represents the discrete Fourier transform where its columns are n Fourier basis functions with different frequencies. Then the eigenvectors of the doubly block circulant matrix A are the columns of $Q = \frac{1}{n}(F \bigotimes F)$, where $\bigotimes$ is a Kronecker product.

The following proposition determines the singular values of a doubly block circulant matrix.

Proposition 5 (Sedghi et al. [2018], Theorem 5)**.** *For the matrix A defined in (14), the eigenvalues of A are the entries of $F^T K F$, and its singular values are their magnitudes. That is, the singular values of A are*

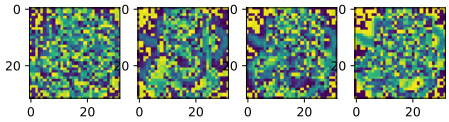
$$\left\{ \left| \left(F^T K F\right)_{u,v} \right| : u, v \in [n] \right\}$$

where $\left(F^T K F\right)_{u,v} = \sum_{p,q \in [n]} \omega^{up} \omega^{vq} K_{p,q}$, that is $F^T K F$ is the 2D Fourier transform of K.

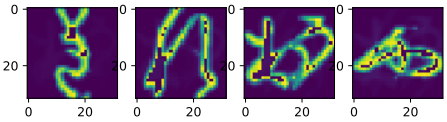
Theorem 6. *Let filter $\mathcal{K}$ be a Gaussian random matrix, then all the eigenvalues of A are non-zero with probability one.*

Proof. Under the assumption that the Filter $\mathcal{K}$ is a random matrix, i.e. all the entries of the matrix $\mathcal{K}$ are i.i.d. Gaussian random variables. The matrix K results from embedding the matrix $\mathcal{K}$ in an $n \times n$ matrix. Hence, for any $u, v \in [n]$, the distribution of $\left(F^T K F\right)_{u,v}$ is Gaussian since it is a sum of k^2 Gaussian distributions and $n^2 - k^2$ Degenerate distributions at zero (constant distribution). The previous result based on the two facts that are the sum of two independent normally distributed random variables is normal, and the sum of any distribution with degenerate distributions at zero is the same distribution. Hence, $P[\left(F^T K F\right)_{u,v} = 0] = 0$. Applying Proposition 5, the eigenvalues of A are non-zero with probability one. □

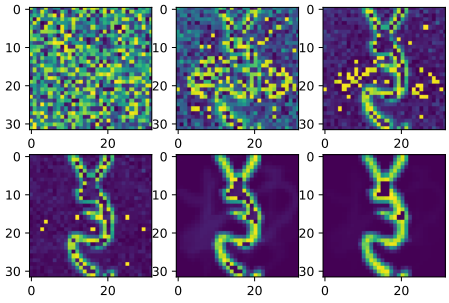
It follows from the previous results that the deconvolution step can be achieved by inverting the matrix A, which is invertible with probability one.



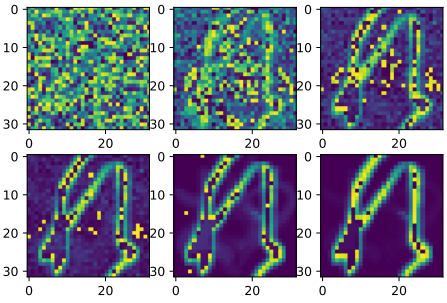
(a) No regularizer



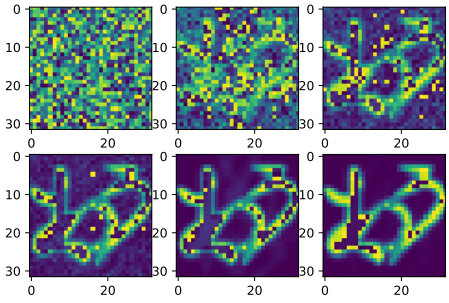
(b) with orthogonality regularizer



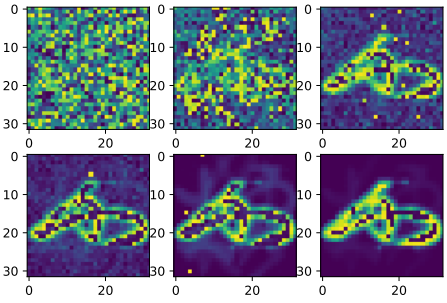
(c) 1/4 recon. procedure



(d) 2/4 recon. procedure



(e) 3/4 recon. procedure



(f) 4/4 recon. procedure

Figure 12: *KMNIST*: 12a and 12b show the final reconstruction without and with regularizer accordingly. Moreover, the reconstructions procedures of 12b are shown in 12c, 12d, 12e and 12f.

(a) 1 kernel

(b) 12 kernels

(c) 24 kernels

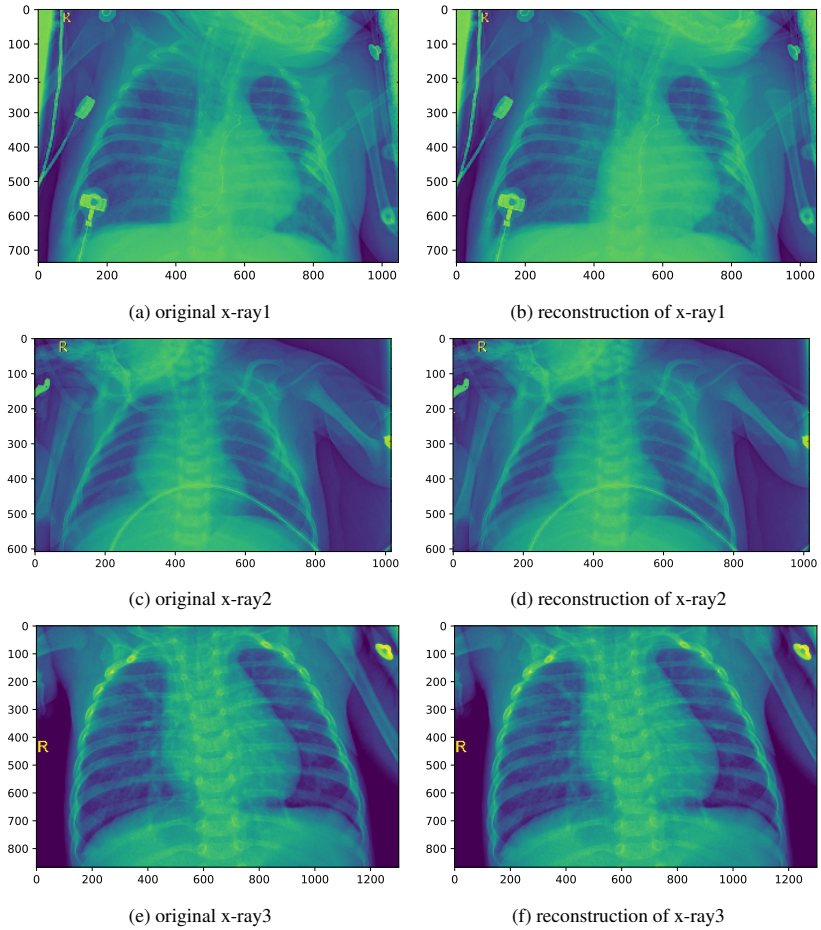
(d) 48 kernels

(e) ori.

Kernels	Params. $(b')^2h$	Mean L1 error
1	64	2.18
12	512	0.32
24	768	0.034
48	3072	0.0002

(f) Average L1 error per pixel.

Figure 13: *Two-layer CNN*: According to the Proposition 3, 12 and 48 kernels are required in the first layer and second convolutional layer, with kernel size 5 and padding size 2, stride size 2 in both the first and the second layer. We provide 12 kernels in the first layer and study the performance with varying kernels in the second layer. We separately show the final reconstruction for different numbers of kernels and the distance between original input and the reconstruction.



(a) original x-ray1

(b) reconstruction of x-ray1

(c) original x-ray2

(d) reconstruction of x-ray2

(e) original x-ray3

(f) reconstruction of x-ray3

Figure 14: X-ray reconstruction on MLP using the analytical form.

A.2 Convolutional neural network without hidden layer

One special case of CNN mentioned in the paper is no hidden layer after convolutional layer, which can be expressed as $p_j = \sum_{i=1}^{n_1} w_{ji}^2 \sigma(H_i) + b_j^2$, where H is output of convolutional layer, $w^2 \in \mathbb{R}^{n_2 * n_1}$ and $b^2 \in \mathbb{R}^{n_2}$ are the weights and bias in the output layer, n_1 is dimension of H, and $\sigma()$ is sigmoid function. Here the number of filters required for reconstruction is:

$$h \geq (\frac{d}{d^{'}})^2 BC \tag{15}$$

d is input width, $d^{'}$ is output width after convolution kernel, B is batch size and C is channel number of input. Equation (15) comes from equation (16) and (17) that we want the number of equations are equal or greater than dimension of unknowns (input), assuming H is known. Namely, we have $(d^{'})^2 h$ equations and $d^2 BC$ unknowns, thus $h \geq (\frac{d}{d^{'}})^2 C$. From equation (18) and (19), if $n_1 \geq \frac{n_2(B-1)}{n_2 - B}$, and $1 < B < n_2$, all $\sigma(H_i)$ are known, thus H_i are also known since $\sigma()$ is a monotonic function.

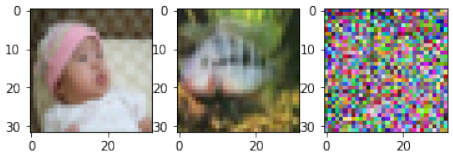
$$\frac{\partial \ell}{\partial l^1_{\underline{m}cgh}} = \sum_{i=1}^{d^{'}} \sum_{j=1}^{d^{'}} \frac{\partial \ell}{\partial z_{\underline{m}ij}} \hat{x}_{c,si+g-1,sj+h-1} \quad \forall \underline{m} \in [1, h] \tag{16}$$

$$\frac{\partial \ell}{\partial z_{\underline{m}ij}} = \frac{\partial \ell}{\partial H[(m-1) \times (d^{'})^2 + (i-1) \times d^{'} + j]} \tag{17}$$

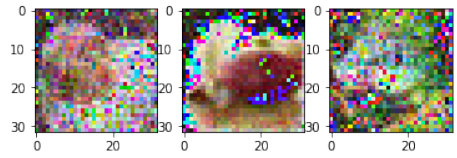
$$\frac{\partial \ell}{\partial b_j^2} = p_{\cdot j} - y_{\cdot j} \quad \forall j \in [1, n_2] \tag{18}$$

$$\frac{\partial \ell}{\partial w_{ji}^2} = (p_{\cdot j} - y_{\cdot j}) \sigma(H_i) \quad \forall j \in [1, n_2], \forall i \in [1, n_1] \tag{19}$$

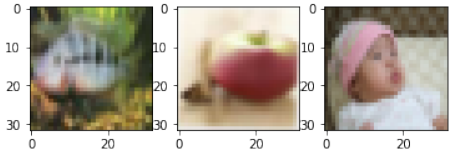
Known a batch of input with size 3, we set the number of nodes in output layer (n_2) as 3, 4 and 5. For the convolutional layer, we set kernel size equal to 5, padding size equal to 2, stride size equal to 2, thus at least 12 filters are required for the reconstruction. In Figure 15, when 3 nodes in output layer, we lose the information of one instance in batch, when 4 and 5 nodes in output layer we have better reconstruction performance as it satisfies $1 < B < n_2$.



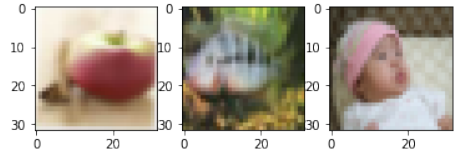
(a) 3 nodes in output layer



(b) 4 nodes in output layer



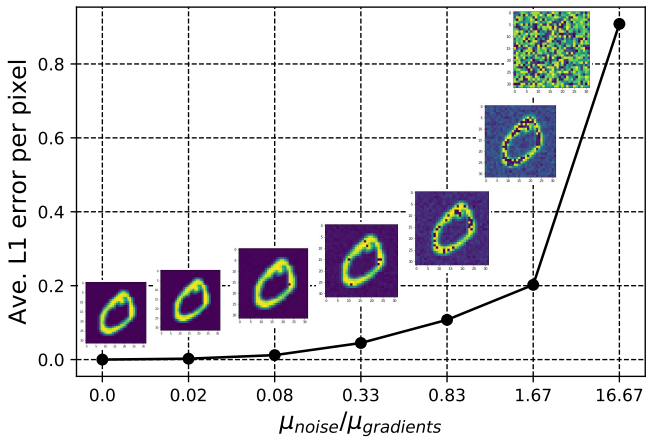
(c) 5 nodes in output layer

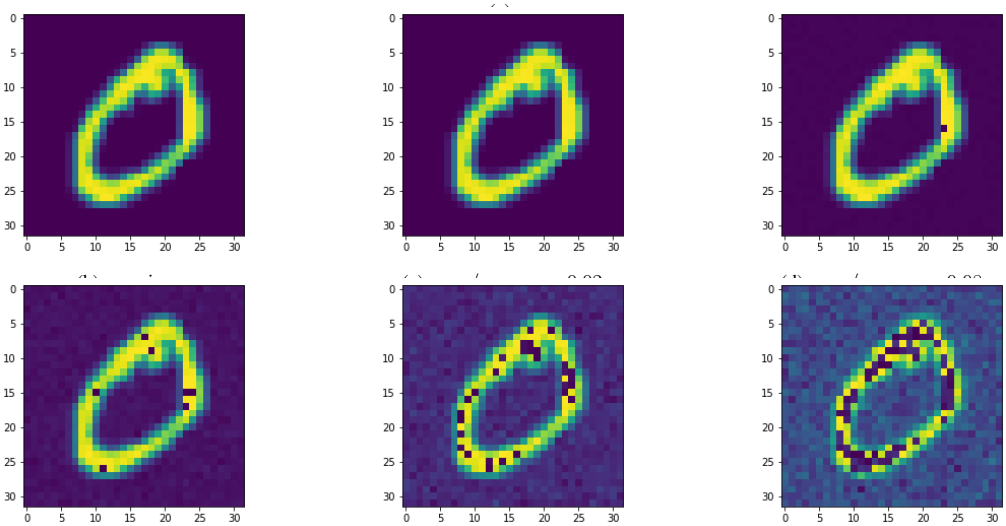


(d) original

Figure 15: CIFAR100 reconstruction of batch size 3 (three instances with different classes) with three, four and five nodes in output layer.

Sensitivity to Perturbation and Defense Noise injection is the classic defense, thus it's worthy investigating how sensitive the reconstruction responses to the model parameters and gradients perturbations. Considering the most vulnerable single-instance reconstruction on MLP, noise injection on the parameter has no influence on reconstruction as we directly compute it from $\frac{\partial \ell}{\partial (w_{1i}^1 + \sigma)} / \frac{\partial \ell}{\partial (b_1^1 + \sigma^{'})}$. However, gradient perturbation is sensitive when the noise-to-signal ratio of mean surpasses certain value as shown in Figure 16.





(e) $\mu_{\text{noise}}/\mu_{\text{gradients}} = 0.33$ (f) $\mu_{\text{noise}}/\mu_{\text{gradients}} = 0.87$ (g) $\mu_{\text{noise}}/\mu_{\text{gradients}} = 1.67$

(h) $\mu_{\text{noise}}/\mu_{\text{gradients}} = 16.67$

Figure 16: *MNIST:* Sensitivity Analysis in response to gradient perturbation on MLP. Note the values of x-axis are not equally scaled, and the distance between values is meaningless.

{2.7182818284}ϑφερτυθιοπσδφγηξκλ