Imperial College London

MEng Individual Project

Imperial College London

Department of Computing

Exploring the Effect of Non-Identically Distributed Data on Federated Learning for Next-Word Prediction

Author:
Shashwat Pritesh Dalal

Supervisor:
Dr. Jonathan Passerat-Palmbach

Second Marker:
Dr. Bernhard Kainz

June 15, 2020

Abstract

Federated Learning is a promising new technology that enables clients to jointly train a machine learning model without revealing their own data. This approach makes machine learning on sensitive client data possible. For example, Google has been experimenting with Federated Learning to train next-word prediction models on mobile keyboard logs.

`FedAvg`, the state-of-the-art technique, achieves this goal by periodically synchronizing the parameters of a model that is locally trained on clients. As learning and data are silo-ed amongst participating clients, a central learner cannot draw samples that uniformly represent the underlying data distribution. As such, `FedAvg` training is assumed to be on non-identically and independently distributed (NIID) data.

`FedAvg` performance has been shown in the literature to deteriorate as client data becomes increasingly NIID. This is largely analysis on convergence bounds in the loss-space rather than an explanation. As privacy and NIID data are related concepts, Federated Learning unlocks the most potential in settings that strongly exhibit NIID data. Next-word prediction is one such setting. As such, it is crucial to understand why `FedAvg` performance degrades in the face of NIID data.

This report aims to explain the effect of deteriorating performance by exploring the divergence between the parameters of `FedAvg` models and their centrally trained counterparts. The main contributions of this report are the analysis of model parameters divergence in `FedAvg` for linear regression, a mathematical exploration on the bounds of model divergence under assumptions about client loss-surfaces, and how `FedAvg` behaves on next-word like prediction tasks when the amount of NIID-ness is varied.

Acknowledgements

I would first like to thank my supervisor, Dr. Passerat-Palmbach, for sparking my interest in privacy-preserving machine learning. Thick-and-thin, your feedback and direction helped me see the clarity and synergy between otherwise complex and unrelated concepts.

I would like to thank the Federated Learning reading group (support group?) for the many illuminating discussions that helped me appreciate the many facets and nuances of Federated Learning.

I would like to thank my closest friends for their support across the four years. I have deeply valued the diverse perspectives you have brought to the discussions and debates, however pseudo-intellectual.

Lastly, and most importantly, I would like to thank my parents for always believing in me and providing me with the opportunities to follow my dreams. I could not have done this without your continual guidance and unconditional support throughout the course of my education.

Contents

List of Figures

Chapter 1

Introduction

1.1 Federated Learning

The current age of humanity is often alluded to as the "Information Age". This should come as no surprise when estimates gauge that an average person today creates 1.7MB of data every second.[32] The availability of this scale of data coupled with the exponential increase in compute power has created the conditions necessary for machine learning, a field of Artificial Intelligence that uncovers patterns from raw data to make predictions and decisions, to take center stage.

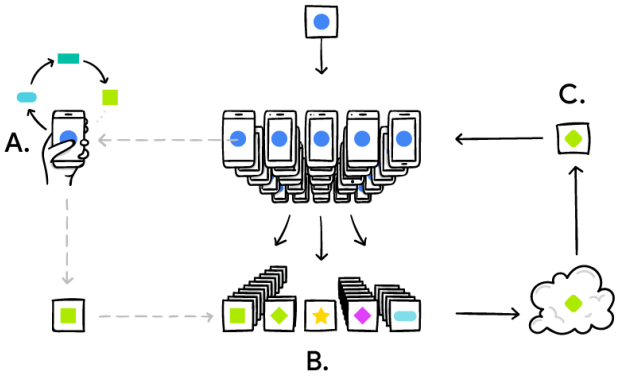


Figure 1.1: Federated Learning at a high level [16]

As machine learning continues to solve tasks of greater difficulty and variety, the natural progression is to apply this technology from toy problems to problems that improve quality of life. As legislation, however, slowly converges to viewing the privacy of an individual as a fundamental right, it has become increasingly difficult to centrally curate a database of personal data.[12][19][27] As problems that improve quality of life has traditionally involve curating a database of personal data, it has been increasingly difficult to broaden the application of machine learning. In other words, there is a trade-off between the scope of traditional machine learning and the privacy of clients.

Federated Learning (FL) is a promising solution to tackle the issue of machine learning utility without foregoing privacy. FL does not solve this trade-off entirely, but rather is a technology with the potential to push the privacy-utility Pareto frontier outward (figure 1.2).

FL, first coined by McMahan et al. at Google in 2016, describes a setting where edge clients, ranging on the scale of a few independent servers to millions of mobile devices, collaboratively train a machine learning model under the supervision of a central server.[13] Formally, *2019's Advances and Open Problems in Federated Learning*, a recent cross-discipline survey, defines FL as

> *"a machine learning setting where multiple entities (clients) collaborate in solving a machine learning problem, under the coordination of a central server or service provider.*

Each client's raw data is stored locally and not exchanged or transferred; instead, focused updates intended for immediate aggregation are used to achieve the learning objective."

[26]

At a very high level, figure 1.1 explains how `FedAvg`, the current state-of-the-art approach, achieves this. A client personalizes a shared model locally based on their local data **(A)**. Many clients' personalized models are aggregated **(B)** to form a new shared model **(C)**, after which the procedure is repeated. `FedAvg` is discussed in detail in section 2.5.

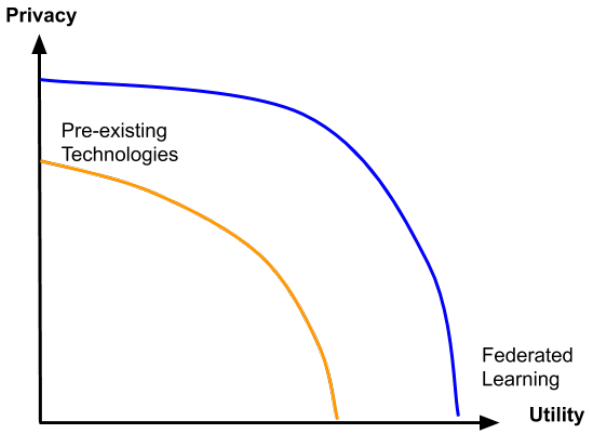


Figure 1.2: Privacy-Utility Pareto frontier

As this nascent setting of machine learning matures, consideration has gone into putting this solution in production.[23] Notably Google has used FL to build their model for next word prediction on Gboard, and Apple has used this technique to build their model for emoji prediction on IOS keyboards.[20][15] Nonetheless practical use of FL is polluted with issues such as non-identically distributed client data, communication bottlenecks, and high unreliability. As such, FL still remains an open research problem.[26]

1.2 Motivation

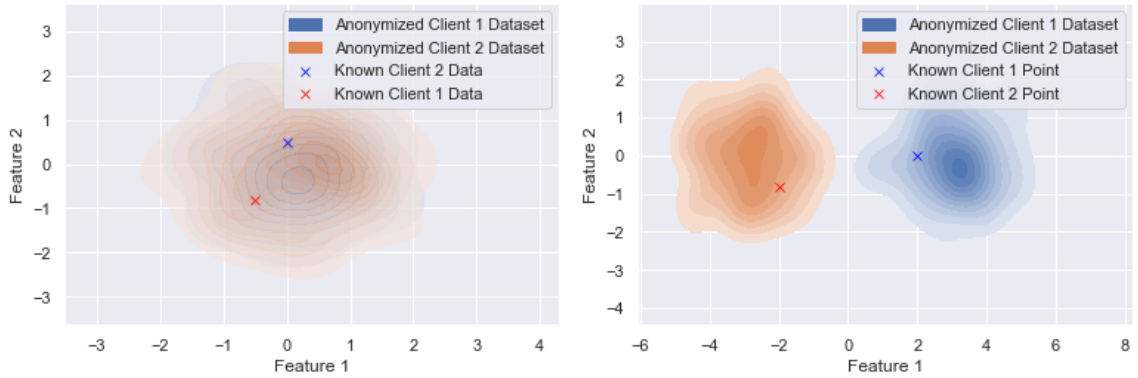
In this report I aim to explore how existing FL algorithms, namely `FedAvg`, handle non-identically and independently distributed (NIID) client data.

To explore this effect, this report first classifies the causes of NIID data as "feature-shift" and "concept-drift" (section2.2.3, and section 2.2.4). The effects of feature-shift and concept-drift are independently explored in the context of next-word prediction models. The problem of next-word prediction is interesting as it is one of the problems studied in FL that is most exposed to NIID data.[13][20][34]

Existing work shows theoretically and empirically that FL algorithms can perform well for next-word prediction under IID data, but no work compares the behavior of FL algorithms for next-word prediction when NIID-ness is controlled.[13][21][26] I outline below my motivation for exploring this behavior.

1.2.1 Realistic Assumption

As data are generated on different edge devices, usually representing individual clients, it is unreasonable to assume these clients produce data in a uniform way. This is especially true for the case of next-word prediction as each client's use of language is diverse and molded to their culture and the world they live in. Studying how FL algorithms behaves under different NIID scenarios can outline how these algorithms tolerate client diversity, and how FL model performance varies as client diversity increases.



(a) Two users with IID data

(b) Two users with NIID data

Figure 1.3: Centralized collection of user data under IID and NIID scenarios

1.2.2 Heightened Privacy Risk

If a client's data are used to train a machine learning model, they are exposed to privacy risk. The privacy risk of a participating client can be defined by how much of the client's data sent to the server can be de-anonymized in the presence of other data about the client, also known as a linkage attack.[7] Under this definition, FL promises significantly better privacy guarantees compared to traditional machine learning when client data are NIID.

To show why NIID client data has high privacy risk in traditional machine learning (even when data are noised and stripped of identifiers), I present two scenarios. In the first scenario two participating clients have similar data distributions (represented with two overlapping distributions in figure 1.3 forming a single cluster). If an adversary knows that a data point belongs to a particular client, it is very difficult to use this information to de-anonymized the dataset collected. This is because the data across all clients form one cluster, hence finding out that a particular client belongs to this cluster revels nothing new.

Consider however the second scenario where two clients participate where their data form two clear clusters. In this case, an adversary a priori can assume each cluster is attributed to a client, but does not know the mapping. If the adversary then finds out that a client belongs to a particular cluster, perhaps through a single data point as shown in figure 1.3, a posteriori every point in the data collection, even though noised and anonymized, becomes identifiable.[7]

FL assumes such linkage attacks are more difficult as data across clients are never aggregated, and client data are assumed to be secure locally.[26] Even though the examples above outlines two extreme scenarios, it motivates FL when a model needs to be trained on data with high privacy requirements, but when also client distributions vary. Next-word prediction trained on mobile keyboard logs falls into this category as illustrated by figure 1.4.

If the motivating reason for FL is to protect NIID client data from linkage attacks, FL algorithms must in-fact be robust to NIID client data. Analyzing FL under different NIID scenario can show the feasibility of FL under this attack vector.

1.2.3 Conditions for Personalization

Given the distributed nature of FL, the assumption that there should only be one authoritative model can be relaxed. A proposed strategy in the FL literature to deal with NIID data is model personalization. [33][25][24][17] The simplest personalization mechanism is for clients to perform additional steps of gradient descent on the final FL model using their own local data.[25]. Other more sophisticated mechanisms are also proposed. Personalization, in a way, turns the problem of NIID data from a bug into a feature!

However, personalization adds another step in the FL workflow and another layer of complexity from the perspective of a machine learning engineer in evaluating the success of a model.[23] Analyzing how FL algorithms behave under different NIID settings highlights in what circumstances personalization can be used to salvage a deteriorating model. As the amount of personalization is based on the difference of client data, the exploration thus highlights how much personalization can actually benefit client performance.

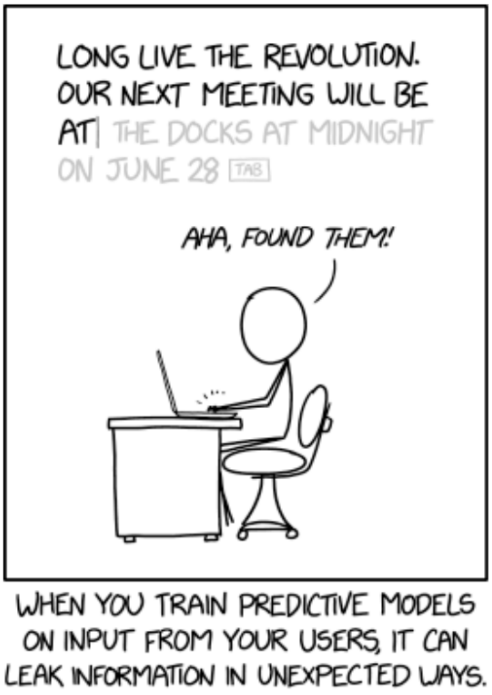


Figure 1.4: XKCD illustration of privacy risk due to NIID data (https://xkcd.com/2169/)

1.3 Contributions

Given how this is a novel research field where the theory and best-practices have not been established yet, a large part of this project was to establish the correct framework to formulate this problem and to consequently design the experiments.[26] Additionally due to the lack of theory with regards to NIID data in FL, partly due to a lack of formal definition of the probleme, a large part of the project also consisted of building the intuition to understand the results of the experimentation. The following thus summarize the main contributions of this report.

1.3.1 Analysis of the Effect of NIID data in One-Shot Averaging for Linear Regression

To explain why jointly training a model without exchanging data is non-trivial in the case of NIID data, section 2.4 demonstrates how averaging model-parameters for linear regression, a convex optimization problem, becomes increasingly worse as the amount of feature-shift and concept-drift increases.

1.3.2 Mathematical Bounds on Model Divergence due to `FedAvg`

Chapter 3 builds upon the empirical results presented in section 2.4 and presents mathematical bounds for how much worse `FedAvg` can be in comparison to traditional machine learning. This is done by making assumptions about the loss-surfaces across clients.

1.3.3 Novel Method to Generate Synthetic Sequential Data whilst controlling NIID-ness

In order to study the effect of feature-shift and concept-drift on next-word prediction. `SyntheticSeq`, a novel process to produce synthetic sequential data, is presented (Algorithm 3). This process has the property that the pattern of predicting the next word can be learnt through a single-layer recurrent neural network. Additionally the parameters of this process can be tuned to observe the required amounts of feature-shift and concept-drift.

1.3.4 Analysis of the Effect of NIID data in Next-Word Prediction like problems

Finally chapter 4 demonstrates empirically how `FedAvg` behaves on next-word prediction like problems where the NIID-ness between clients is varied. This is done by evaluating `FedAvg` when trained on various configuration of `SyntheticSeq`.

Chapter 2

Background

It is important to clarify that FL is neither a machine learning model nor is it a machine learning algorithm. FL is rather a broad term for a machine learning setting where a loose federation of participating clients, coordinated by a central server, work together to train a model over a succession of rounds. This setting invites a landscape of algorithms for achieving this goal. These class of algorithms embody principles of user anonymization, decentralized data, and focused collection; principles that parallel modern data-protection legislation.[27] Proposed algorithms for Federated Learning use a combination of techniques borrowed from Differential Privacy, Secure Multiparty Computation, and Homomorphic Encryption to uphold these principles.[7][4][3]

To motivate the exploration in chapter 3 and 4, the following background is required. It is important to discuss first why having NIID client data results in performance degradation in FL settings. In order to do this I first contrast traditional machine learning with FL in section 2.1. I then describe why NIID data arises in FL settings in section 2.2. Section 2.2 also categories two sources of NIID data: features shift and concept drift. Section 2.3 discusses how NIID-ness can be measured. Section 2.4 describes why under NIID data, learning between two clients is not trivial. Finally section 2.5 describes `FedAvg`, the state-of-the-art FL algorithm and discusses previous work demonstrating under what conditions `FedAvg` is robust to NIID data. Section 2.6, using linear regression as an example, reconciles why naive one-shot model averaging is not robust to NIID data, but periodic averaging, such as in `FedAvg`, is robust to NIID data.

2.1 Federated Learning Settings and Assumptions

2019's Advances and Open Problems in Federated Learning broadly categorizes FL into two settings each with it's own sets of assumption.[26] FL can also be defined as a spectrum of settings for distributed machine learning with associated assumptions.

This idea is depicted in figure 2.1. Each client's data is depicted as a different color, and the coordination service is depicted by a red circle. Further each box depicts a single compute resource abstraction (this could be an entire data-center or a single mobile phone). Communication within a compute resource is cheap whilst communication between them is expensive. Further to maintain privacy, raw data should not be transferred between compute resources.

2.1.1 Distributed Data-Center Learning

On the far left of the spectrum in figure 2.1 is distributed data-center learning. In this setting, the single compute resource is assumed to be highly available and reliable. This could for example be an entire data-center or a cluster. Clients are nodes within the compute resource. As each node belongs to the same compute resource, data can be arbitrarily shuffled and distributed amongst different nodes cheaply and without privacy concerns. Due to this, the distribution of data across nodes is assumed to be independently and identically distributed (IID), defined in section 2.2. As the coordination service is housed within the compute resource, computation is considered to be the bottleneck in this setting.

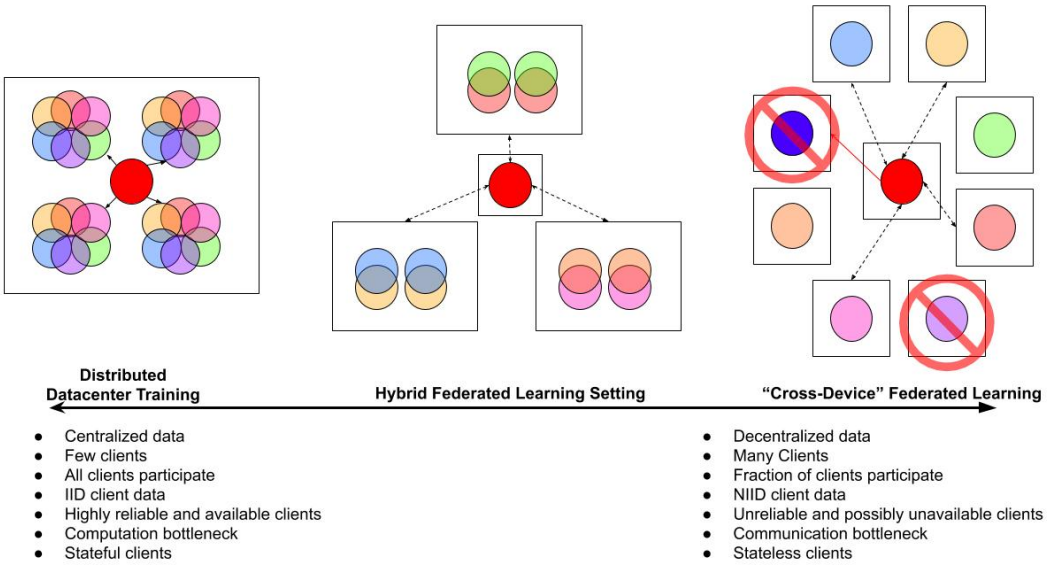


Figure 2.1: Spectrum of FL Settings with Associated Assumptions

2.1.2 "Cross Device" Federated Learning

The other end of the spectrum in figure 2.1 is what 2019's Advances and Open Problems in Federated calls cross-device FL, and is the original setting that inspired FL. [26] [13] In this setting, there are a large number of clients each its own compute resource. These can be assumed to be mobile phones or other low availability and low reliability edge devices. In this setting, the data across clients is assumed to be NIID because each client is assumed to generate data based on an independent arbitrary process. [13] A machine learning model is jointly trained with the help of a coordination server that belongs to a non-client compute resource. This coordination server can be assumed to be highly reliable and available. [23][15] In this setting, only a faction of all clients participate in each "roun" of training (a concept further explained in section 2.5). As such, clients in this setting are considered to be stateless. Next-word prediction on mobile keyboard comes under this setting. [20]

2.1.3 "Cross Silo" Federated Learning

In between these two extremes lies hybrid FL settings such as, what Advances and Open Problems in Federated calls "Cross-Silo" FL. [26] Here data is kept decentralized, but the assumption on client unreliability and statelessness are relaxed.

2.1.4 What Constitutes Federated Learning

Defining FL as a spectrum, blurs the line between traditional machine learning and this *"decentralized privacy preserving thing"*. Resorting to the definition in section 1.1, however, the distinguishing feature of FL is that *"raw data is stored locally and not exchanged or transferred"*. Thus any setting where this is satisfied can be labelled as FL, albeit given the spectrum above, some settings may be more decentralized than others, and some settings may follow stronger privacy principles than others. Irregardless of definition, across all FL settings, client data is uniformly assumed to be NIID. The rest of the report is dedicated to understanding the challenges that accompany this assumption.

2.2 Why Client Data are NIID

Section 2.1 claimed data across all settings of FL are assumed to be NIID. This section explains why the assumption is made by answering:

1. What does NIID data mean?

2. What are the causes of NIID data?

As NIID is the antithesis to IID, explaining the specific scenario of when data are IID will explain the general scenario of when data are NIID. Then to explain what may cause NIID data, the generative process of how individual clients might generate their data is explored.

2.2.1 IID Data

$$\begin{gathered} X_1, \ldots, X_n \in I \subseteq \mathbb{R} \text{ are i.i.d. if and only if} \\ (1) \quad \forall k \in \{1, \ldots, n\} \text{ and } \forall x \in I \quad F_{X_1}(\mathcal{X}) = F_{X_k}(\mathcal{X}) \\ (2) \quad \forall x_1, \ldots, x_n \in I \quad F_{X_1,\ldots,X_n}(\mathcal{X}_1, \ldots, x_n) = F_{X_1}(\mathcal{X}_1) \ldots F_{X_n}(\mathcal{X}_n) \\ \text{where} \quad F_X(\mathcal{X}) = Pr[X \leq x] \end{gathered} \tag{2.1}$$

IID data is mathematically defined by 2.1. In words, this means that every data point must come from the same distribution, and must be independent from every other data point. This is easy to establish in a centralized machine learning approach by drawing batches from a shuffled dataset. [13] In the case where client data remains local however, data across clients cannot be uniformly shuffled. As a result, data across clients can no longer be assumed to be identically distributed. Independence can be forced through shuffling data on every client, and randomly sampling clients.

In some "cross-device" settings where clients are globally distributed, it can be argued that even independence cannot be guaranteed. This is because different groups of clients become available at different times of the day hence there is strong correlation between the succession of clients sampled. [24] In this report, it is assumed that all clients participate and hence data independence can be assumed throghout this report. As such, technically, this reports explores the effects of non-identically distributed data (NID), not NIID data.

In the case of supervised learning, the data on each client can be characterized as a joint distribution across their features and labels. Considering this for two participating clients we have:

$$\begin{gathered} Client_i, Client_j \in Clients \\ (\mathcal{X}_i, y_i) \sim P_i(\mathcal{X}, \mathcal{Y}) \text{ and } (\mathcal{X}_j, y_j) \sim P_j(\mathcal{X}, \mathcal{Y}) \end{gathered} \tag{2.2}$$

Two clients in the supervised setting are hence defined to have NIID data if:

$$P_i(\mathcal{X}, \mathcal{Y}) \neq P_j(\mathcal{X}, \mathcal{Y}) \tag{2.3}$$

2.2.2 $\mathcal{X} \to \mathcal{Y}$ Problems vs $\mathcal{Y} \to \mathcal{X}$ Problems

Jose et Al's, categorization of dataset shifts can be adapted to categorize the different reasons why NIID data might arise.[5]

Supervised tasks can be categorized into two sets of problems.

1. $\mathcal{X} \implies \mathcal{Y}$ problems, where the values of the features determines the label. Examples include next-word prediction and linear regression.

2. $\mathcal{Y} \implies \mathcal{X}$ problems, where we are interested in learning the features given a label. Examples include medical diagnosis.

Each set of problems has its own independent causes of NIID, but given how the focus of this report is on next-word prediction $\mathcal{X} \implies \mathcal{Y}$ problems are considered. When features drives the label prediction, the joint probability distribution for any client can be decomposed into:

$$P_i(\mathcal{X}, \mathcal{Y}) = P_i(\mathcal{Y}|\mathcal{X}) P_i(\mathcal{X}) \tag{2.4}$$

This factored distribution can be interpreted as the latent generative process of how a particular client might be generating their data. Reasoning about why each factored distribution may vary across clients explains why the manifested joint distribution varies across clients.

2.2.3 Feature Shift

Consider a situation where the marginal probability, $P(\mathcal{Y}|\mathcal{X})$, is the same across clients, but the feature probability is not.

$$\begin{aligned} P_i(\mathcal{Y}|\mathcal{X}) &= P_j(\mathcal{Y}|\mathcal{X}) \\ \text{but } P_i(\mathcal{X}) &\neq P_j(\mathcal{X}) \end{aligned} \tag{2.5}$$

In the context of next-word predictions, this could represent two clients who construct their sentences in a comparable way, perhaps due to a similar upbringing and education. Nonetheless they both have different interests and hence talk about different things consequently logging a different frequency of words.

2.2.4 Concept Drift

Considering the converse, consider a situation where the feature probability is the same, but the marginal probability is not.

$$\begin{aligned} P_i(\mathcal{X}) &= P_j(\mathcal{X}) \\ \text{but } P_i(\mathcal{Y}|\mathcal{X}) &\neq P_j(\mathcal{Y}|\mathcal{X}) \end{aligned} \tag{2.6}$$

In the context of next-word predictions, this could represent two clients who have the same interests and talk about the same things consequently logging the same frequency of words. However given the flexibility of English sentence structure, the two might have different preferences as to how they compose sentences. Thus the most likely word to follow a given word would be different between the two.

As shown through the two examples, next-word prediction is a complex task that is exposed to a combination of sources of NIID-ness.

2.3 Measuring Difference between Probability Distributions

Having gone through the different possible sources of NIID data, the next step is to quantify NIID-ness. This section describes a method to measure the difference in probability distributions in the context of next-word prediction. This method can be adapted to be used in other discrete domain problems.

2.3.1 Next-Word Prediction

Thus far for our supervised learning problem, we have used $\mathcal{X}$ to denote the features we are predicting on and $\mathcal{Y}$ to denote the associated labels. Further we have used $\mathcal{P}_i$ to denote the partition of data at client i.

In the case of next-word prediction, $\mathcal{P}_i$ refers to the collection of words a client has typed on their mobile keyboard. This is shown in figure 2.2.

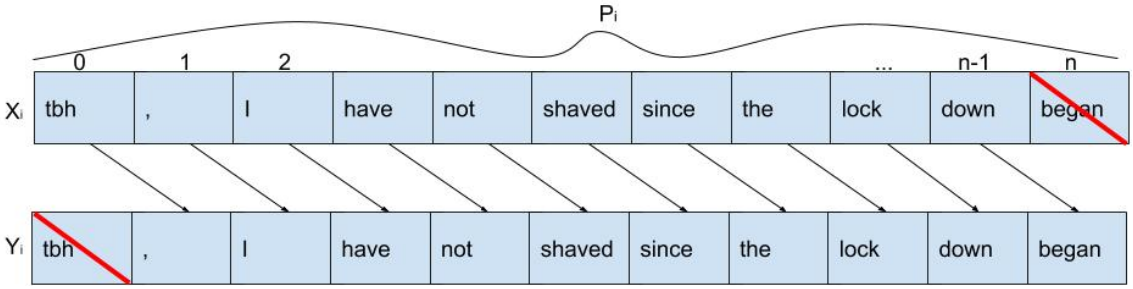


Figure 2.2: Notation for Client i's data partition for next-word prediction

As described in section 2.2, we assume that each client produces their data partition based on a latent generative process $P(X, Y) = P(Y|X)P(X)$. In the case of next-word prediction, a human considers everything that is typed so far as well as what is required to complete the thought to predict what next word to use. This potentially means there are an infinite number of features $\mathcal{X}$ that need to be considered to predict a single $\mathcal{Y}$. To simply this, the proposed method makes a

memoryless approximation and assumes that the only feature a human considers when predicting the next word is the previous word. This idea is illustrated by the arrows between $\mathcal{X}_i$ and $\mathcal{Y}_i$, in figure 2.2

If we let $\mathcal{P}_i$ be an array of tokens indexed from $[0:n]$, following the memoryless assumption the set of features on client i, $\mathcal{X}_i$, is defined as the slice $\mathcal{P}_i[0:n-1]$, and the set of labels on client i, $\mathcal{Y}_i$, is defined by the slice $\mathcal{P}_i[1:n]$.

Further the assumption is made that the the tokens are drawn from a finite set vocabulary V of size $vocab\_len$. Hence $P_i(\mathcal{X})$ denotes the frequency count of $\mathcal{P}[0:n-1]$ of all tokens in the vocabulary V used by client i. This can be easily computed.

2.3.2 Distance between Feature Probabilities

We define the distance between two client's feature probabilities as:

$$D[P_i(\mathcal{X})||P_j(\mathcal{X})] = \sum_{\forall v in V} |P_i(v) - P_j(v)| \tag{2.7}$$

Or more compactly, if $P_i(\mathcal{X})$ is considered to be a frequency vector where $P_i(\mathcal{X})_v$ denotes the frequency of observing v on client i,

$$D[P_i(\mathcal{X})||P_j(\mathcal{X})] = ||P_i(\mathcal{X}) - P_j(\mathcal{X})||_1 \tag{2.8}$$

2.3.3 Distance between Marginal Probabilities

Under the assumption of a fixed vocabulary, $\mathcal{Y}_i$ is also drawn from the same vocabulary set V. Hence $P_i(\mathcal{Y} = y|\mathcal{X} = x)$ characterizes the frequency of observing token y after token x normalized over the frequency of observing x for client i. This, assuming a limited sized vocabulary, is easy to calculate. For reference, FL next-word prediction models in the literature use a vocabulary size of $10,000$. [20] [30] Hence $P_i(\mathcal{Y}|\mathcal{X})$ can be thought of a matrix of size $|V| \times |V|$ where $P_i(\mathcal{Y}|\mathcal{X})_{v',v}$ is the observed frequency of client i transitioning from token v to v'. Hence the distance between two clients marginal probabilities can be defined as:

$$D[P_i((\mathcal{Y}|\mathcal{X})||P_j(\mathcal{Y}|\mathcal{X})] = ||P_i((\mathcal{Y}|\mathcal{X}) - P_j((\mathcal{Y}|\mathcal{X}))||_F \tag{2.9}$$

where $||.||_F$ is the Frobenius norm.

2.3.4 Measuring Distance between All Clients

The methods described in 2.8 and 2.9 provide measures for distance between two clients. In FL, however, an arbitrary number of clients participate. If n clients participate in jointly training a next-word prediction model, and each client has the same amount of data, client NIID-ness can be defined as the variance between:

1. $P(X)$, the feature distribution
2. $P(Y|X)$, the marginal distribution

Using the definitions of D in 2.8 and 2.9, we define the NIID measure on on the set of participating clients $\mathcal{C}$.

$$\begin{gathered}\Sigma : \text{Participating Clients} \to \mathbb{R}^2 \\ \Sigma(\mathcal{C}) = \begin{bmatrix} \sigma_{P(X),\mathcal{C}} \\ \sigma_{P(Y|X),\mathcal{C}} \end{bmatrix} \\ \text{where } \sigma_{P(\mathcal{K}),\mathcal{C}} = \frac{1}{|\mathcal{C}|} \sum_{\forall c \in \mathcal{C}} D[P_c(\mathcal{K}) \;||\; \frac{1}{|\mathcal{C}|} \sum_{\forall c' \in \mathcal{C}} P_{c'}(\mathcal{K})]\end{gathered} \tag{2.10}$$

The benefit of calculating the NIID-ness as two components rather than the average distance on the joint distribution is :

1. The divergence in these distributions are independent as explained above
2. By distinguishing the two components in the analysis, the impact of each component on FL performance can also be distinguished.

2.4 One Shot Averaging

An important feature of FL is to jointly train a model without the exchange of raw data. How is this achieved? This section considers the most basic approach that is, what happens if we naively average the parameters of the optimal models for every client that participates. According to the literature on one-shot averaging in a convex and IID setting, the average model performs, as well as as the locally trained model on any client.[6][10][13] This section discusses how one-shot averaging performs in the other settings. Namely:

1. convex but NIID setting
2. non-convex setting

This section also shows empirically that in the convex NIID setting, one-shot averaging worsens as features shift, and concepts drift.

2.4.1 Convex but NIID Setting

Problem Formulation

A machine learning problem is classified as a convex optimization problem if its loss surface exhibits convex properties. Convexity is defined as:

$$\begin{gathered}\text{A convex function is defined as any function } f \\ f : \mathbb{R}^n \to \mathbb{R} \\ \forall x, y \in \mathbb{R}^n, \forall \lambda \in [0,1] : f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda) f(y)\end{gathered} \tag{2.11}$$

[2] These functions are commonly used to derive guarantees in machine learning as they have the nice property of having a single minimum.[2][9][14].

Linear regression problems that use a mean-square loss are known to be a convex problems.[2] As such, the behavior of one-shot averaging in the case of linear regression is explored to understand how one-shot averaging behaves in a NIID convex setting.

Let us assume two clients are jointly training a linear regression model. Let each client have a data partition $\mathcal{P}_i = (\mathcal{X}_i, \mathcal{Y}_i)$ where the data is generated by the following process.

$$\begin{aligned}\mathcal{X}_i &\sim \mathcal{N}(\mu_i, \sigma_i) \\ \epsilon &\sim \mathcal{N}(0,1) \\ \mathcal{Y}_i &= \alpha_i \mathcal{X}_i + \beta_i + \epsilon\end{aligned} \tag{2.12}$$

If a linear regression model were to be trained locally on a client i, the mean-square loss, defined on α and β and computed on $\mathcal{P}_i$, would be a convex function.[2] Using gradient descent as an optimization method would converge on α_i and β_i with a rate of $\mathcal{O}(\frac{1}{k})$ where k is the number of iterations.[11]

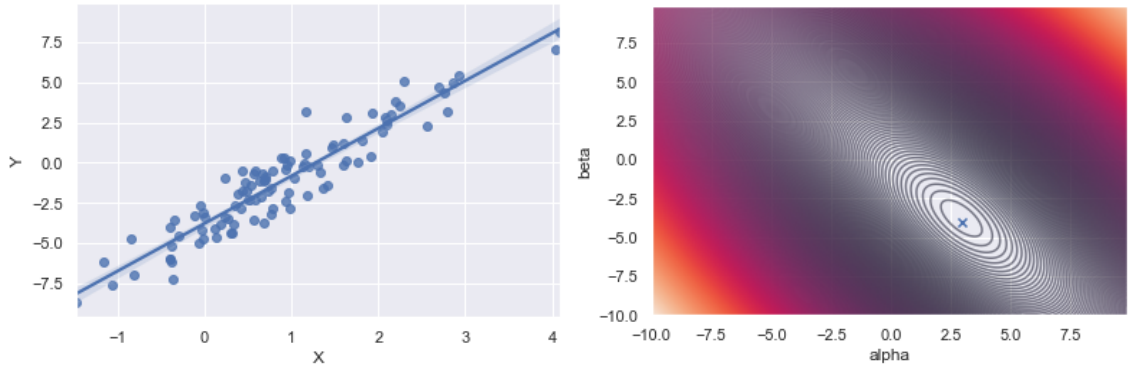
$$\begin{gathered}\alpha_i, \beta_i = \arg\min_{\alpha,\beta} \ell(\alpha, \beta; \mathcal{P}_i) \\ \text{where } \ell(\alpha, \beta; \mathcal{P}_i) = (\alpha \mathcal{X}_i + \beta - \mathcal{Y}_i)^2\end{gathered} \tag{2.13}$$

Given the convergence guarantee, if the average of each client's parameters is taken, provided enough iterations are performed, the average of the parameters should be the average of each client's optimal parameters (which is known as it is a parameter of the generative process 2.12)

$$\begin{aligned}\bar{\alpha} &= \frac{1}{2}\sum_{i=1}^{i=2} \alpha_i \\ \bar{\beta} &= \frac{1}{2}\sum_{i=1}^{i=2} \beta_i\end{aligned} \tag{2.14}$$

Let $\mathcal{P}$ define the union over the data sets of both clients.

$$\mathcal{P} = (\mathcal{X}, \mathcal{Y}) = \bigcup_{i=1}^{i=2} \mathcal{P}_i \tag{2.15}$$



(a) Line of best fit

(b) Loss surface $\ell(\alpha, \beta; \mathcal{P}_1)$

Figure 2.3: Linear Regression on Client 1 Partition

The mean-square loss of $\mathcal{P}$ with respect to α and β, $\ell(\alpha, \beta; \mathcal{P})$, can be defined as the average loss across each partition.

$$\begin{aligned} \alpha^*, \beta^* &= \arg\min_{\alpha,\beta} \ell(\alpha, \beta; \mathcal{P}) \\ \ell(\alpha, \beta; \mathcal{P}) &= \frac{1}{2} \sum_{i=1}^{i=2} \ell(\alpha, \beta; \mathcal{P}_i) \end{aligned} \tag{2.16}$$

As the loss of each partition is convex, the average loss of each partition is still a convex function.[2] This means 2.16 is still a convex problem and the guarantee that performing gradient descent will converge to some optimal parameters α^* and β^* asymptotically still holds.

Further, as linear regression can be calculated in closed form, α^* and β^* can be calculated without having to perform gradient descent if both clients' partitions are known.[31]

$$\begin{aligned} \text{let } X &= \begin{bmatrix} \mathcal{X}_1 & 1^T \\ \mathcal{X}_2 & 1^T \end{bmatrix}, Y = \begin{bmatrix} \mathcal{Y}_1 \\ \mathcal{Y}_2 \end{bmatrix} \\ \begin{bmatrix} \alpha^* \\ \beta^* \end{bmatrix} &= (X^T X)^{-1} X^T Y \end{aligned} \tag{2.17}$$

When the average of two convex functions is taken, even though the average is a convex function, the minimum of the average need not be the average of the minimum of each individual function.[2] Hence the interest lies in observing how different $\bar{\alpha}$ and $\bar{\beta}$ (the average of the minimum of each individual loss) are from α^* and β^* (the minimum of the average loss). If one-shot averaging is to yield good results, the difference must be negligible because otherwise the averaged model would perform poorly on any one client's partition.[10]

Specific Example of Unwanted Behavior

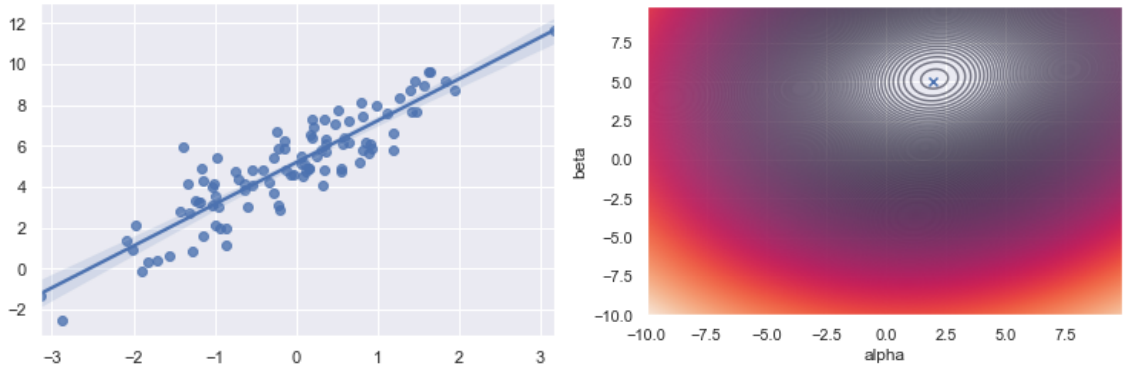
By taking arbitrary parameters, this section empirically shows that in the NIID convex setting, one-shot averaging *can* negatively impacts client performance. This is in contrast to the worst-case analysis highlighted in the literature for convex IID problems where averaging performs no worse than client models.

If the following parameters are used to generate the dataset of each client according to process defined in 2.12

$$\sigma_i = 1, \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} \alpha_1 \\ \beta_1 \end{bmatrix} = \begin{bmatrix} 3 \\ -4 \end{bmatrix}, \begin{bmatrix} \alpha_2 \\ \beta_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 5 \end{bmatrix} \tag{2.18}$$

and 100 data points are sampled for each client, the behavior of one-shot averaging is present in figure 2.3, 2.4, and 2.5.

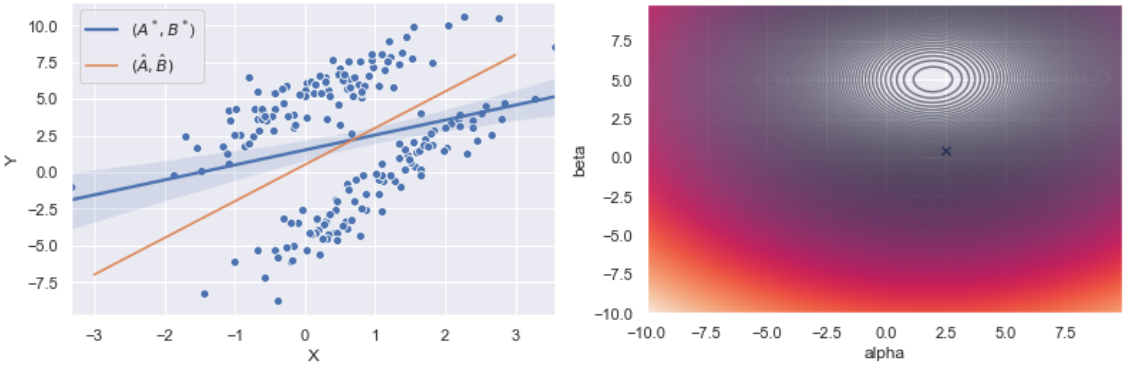
As seen across figures 2.3, 2.4, and 2.5, the loss function with respect to all three partitions is convex. Further for the linear regression on the full partition, $\bar{\alpha}$ and $\bar{\beta}$ does not generalize to the entire dataset. This is seen by the orange line in 2.5a which is parameterized by $\bar{\alpha}$ and $\bar{\beta}$. The '`x`' in 2.5b, marks $\bar{\alpha}$ and $\bar{\beta}$ on the loss with respect to the full partition. Even though $\bar{\alpha}$ and $\bar{\beta}$ is in proximity to α^* and β^*, the difference is significant enough to adversely affect client performance.



(a) Line of best fit

(b) Loss surface $\ell(\alpha, \beta; \mathcal{P}_2)$

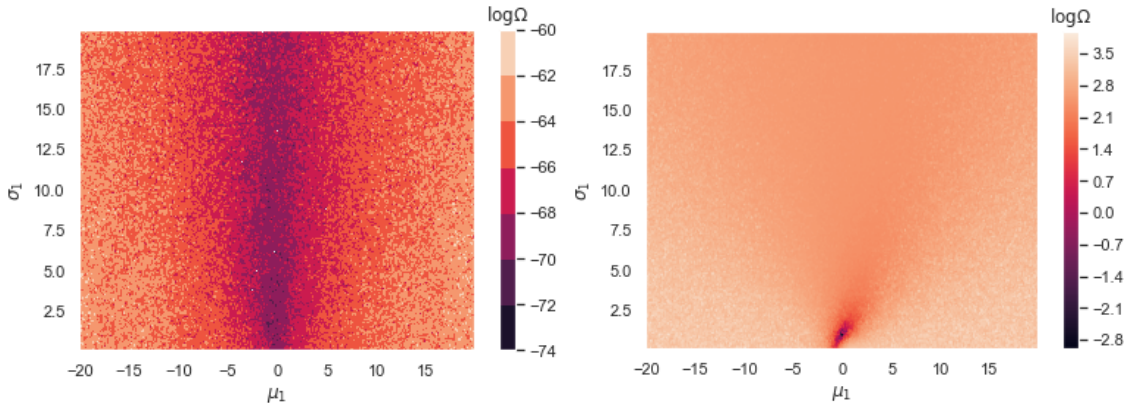
Figure 2.4: Linear Regression on Client 2 Partition



(a) Line of best fit

(b) Loss surface $\ell(\alpha, \beta; \mathcal{P})$

Figure 2.5: Linear Regression on Full Data-Set



(a) No concept drift

(b) Clients observe some concept drift

Figure 2.6: Effect of Features Shift on Convex NIID One-Shot Averaging

In this particular example, μ_i as well as α_i and β_i were varied across the clients. As such $P_i(X, Y)$ varied because both $P_i(X)$ and $P_i(Y|X)$ were not the same across clients. In other words, both clients observed some feature shift, and concept drift. What happens if one is varied at a time?

As $\mathcal{X}_i$ and $\mathcal{Y}_i$ is sampled from a defined process 2.12, the closed form expression 2.17 can be used to empirically estimate α^* and β^* as the parameters of process 2.12 are varied. The distance between $\bar{\alpha}$ and $\bar{\beta}$ and α^* and β^* can be hence calculated for the various parameters of 2.12. The distance between the two sets of parameters is denoted by Ω.

$$\Omega = || \begin{bmatrix} \alpha^* \\ \beta^* \end{bmatrix} - \begin{bmatrix} \bar{\alpha} \\ \bar{\beta} \end{bmatrix} ||_2 \tag{2.19}$$

Varying Feature Shift

This section evaluates the behavior of one-shot averaging when clients exhibit various amounts of feature shift, that is the marginal probability $P_i(\mathcal{Y}|\mathcal{X})$ stays constant whilst $P_i(\mathcal{X})$ varies. This can be done be holding α_i and β_i constant across client's and varying the difference between μ_i and σ_i. The feature drift is varied under two scenarios.

1. *No Concept Drift*

   No concept drift is observed when $P_i(\mathcal{Y}|\mathcal{X})$ is the same across clients. To do this the following parameters are used.

   $$\mu_2 = 0, \sigma_2 = 1, \begin{bmatrix} \alpha_i \\ \beta_i \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \tag{2.20}$$

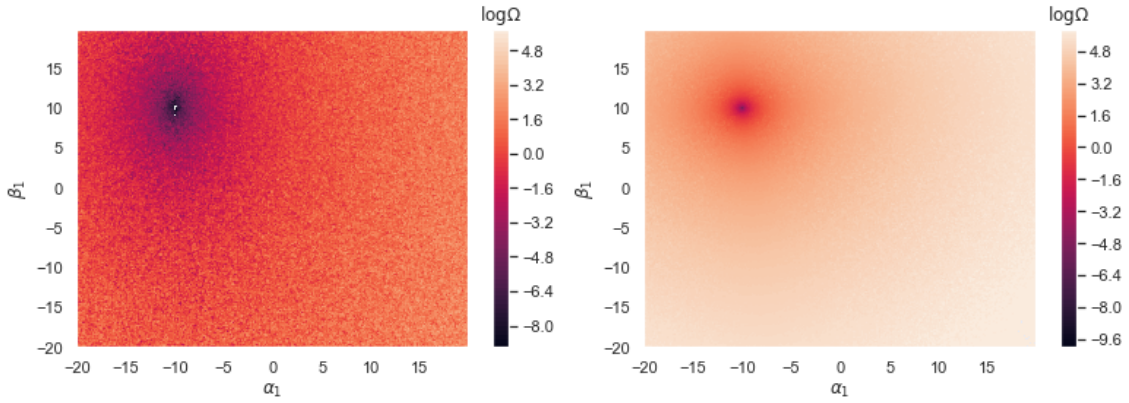
2. *Clients Observe some Concept Drift*

   To observe the effect of feature shift when clients observe some concept drift, the following parameters are used.

   $$\mu_2 = 0, \sigma_2 = 1, \begin{bmatrix} \alpha_1 \\ \beta_1 \end{bmatrix} = \begin{bmatrix} 5 \\ -2 \end{bmatrix}, \begin{bmatrix} \alpha_1 \\ \beta_2 \end{bmatrix} = \begin{bmatrix} 2 \\ -5 \end{bmatrix} \tag{2.21}$$

Figure 2.6 shows the effect of varying μ_1 and σ_1 on Ω in the two described cases. When there is no concept drift, Ω is negligibly small throughout the range μ_1 and σ_1 is varied. However when there is some concept drift, the experiments show that Ω differs drastically as μ_1 and σ_1 are varied.

It appears from the figure 2.6, that the surfaces exhibit a single minimum at the values where there is no feature shift. When clients exhibit concept shift, the gradients around the minimum become more intense, and Ω increases more quickly as features shift. Though not mathematically rigorous, it can be hypothesized that Ω, when paramaterized by μ_1 and σ_1, is convex, and the convexity increases as the concept shift increases.



(a) No Feature Shift

(b) Clients observe some Feature Shift

Figure 2.7: Effect of Concept Drift on Convex NIID One-Shot Averaging

Varying Concept Drift

This section explores the behavior of one-shot averaging when clients display a varying amount of concept shift, that is the feature probability $P_i(\mathcal{X})$ stays constant whilst the marginal probability $P_i(\mathcal{Y}|\mathcal{X})$ varies. This can be done be holding μ_i and σ_i constant across clients and varying the difference between α_i and β_i. Varying concept drift is considered under the following scenarios.

1. *Clients Observe no Feature Shift*

   No feature shift is observed when $P_i(\mathcal{X})$ is the same across clients. To do this the following parameters are used.

   $$\alpha_2 = -10, \beta_2 = 10, \begin{bmatrix}\mu_i \\ \sigma_i\end{bmatrix} = \begin{bmatrix}0 \\ 1\end{bmatrix} \tag{2.22}$$

2. *Clients Observe some Feature Shift*

   The behavior is observed when $P_i(\mathcal{X})$ is different, but constant across clients. To do this the following parameters are used.

   $$\alpha_2 = -10, \beta_2 = 10, \begin{bmatrix}\mu_1 \\ \sigma_1\end{bmatrix} = \begin{bmatrix}1 \\ 1\end{bmatrix}, \begin{bmatrix}\mu_2 \\ \sigma_2\end{bmatrix} = \begin{bmatrix}10 \\ 20\end{bmatrix} \tag{2.23}$$

Figure 2.7 shows the effect of varying α_1 and β_1 on Ω in the two cases. When there is no feature shift, Ω, unlike in the case of feature shift, is not negligible. Ω increases as α_1 and β_1 differ more and more from α_2 and β_2. When there is some feature shift, the same behavior is observed except this happens more rapidly. Once again it can be hypothesized that Ω, when paramaterized by α_1 and β_1, is convex, and the convexity increases as the feature drift increases

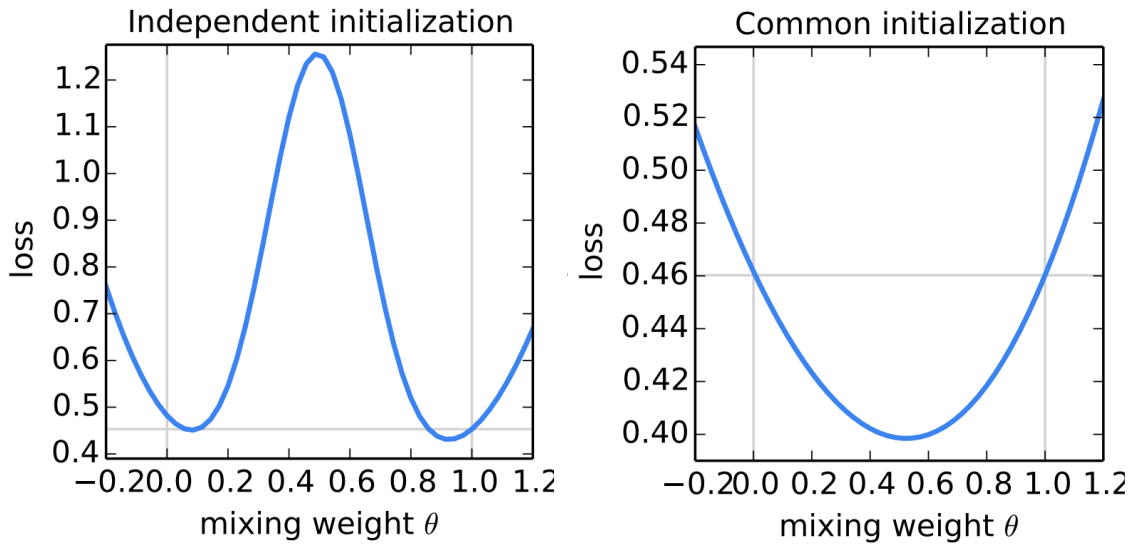
2.4.2 Non-Convex Setting

The behavior of one-shot averaging in a non-convex setting is considered.

IID Setting

If the data are IID, it implies that the loss surface for each client is the same and hence the average loss surface has no variance with each individual client's loss.

Even if the loss surfaces are identical, if the assumption of convexity is relaxed, there might be an arbitary number of minima. As such, different initializations of client models might result in reaching different minima. The average of these local minima might not be a minima on the average loss surface. This is shown when two MNIST digit classifiers are trained with different initializations. McMahan et. Al (2016) show that as these models are mixed, their performance becomes increasingly worse on client partitions. [13]. It has, however, been shown that the loss



(a) Different parameter initialization

(b) Same parameter initialization

Figure 2.8: Averaging two models that have non-convex losses trained on IID Data [13]

surfaces of over-parameterized networks are not actually not too prone to bad local minima.[9][8]. Hence if two models are trained from the same initialization, the average of the two models trained on two different IID data-sets actually improves performance. [13]. This is illustrated in figure 2.8.

NIID Setting

The literature states that if the data are NIID, the loss surface between clients can be arbitrarily different hence the optimal parameters between clients can be arbitrarily different. Thus one-shot averaging can result in an arbitrarily bad model even if the same initialization is used.[28][6][10].

This is problematic as given a moderately complex problem, the loss surface cannot be assumed to be convex. Further, as stated in section 2.1, data is assumed to be NIID in FL.

The lack of existing algorithms in this setting was the motivation behind the development of `FedAvg`. `FedAvg` periodically averages the model parameters across clients instead of only averaging the parameters at the very end. If this is done under the right hyperparamters, this algorithm can be used to jointly train a model with a non-convex loss across clients with NIID data. [13][24]

2.5 `FedAvg`

2.5.1 Motivation

`FederatedAveraging` or `FedAvg` was introduced by McMahan et al, as a communication-efficient protocol to train deep networks on decentralized data.

They propose a method where a federated network of clients with the help of a coordination server, collaboratively train a machine learning model. With considerations of privacy, limited communication, Non-IID data, and massive distribution, McMahan et al. coin the term Federated Learning in their seminal paper. [13]

As stochastic gradient descent (SGD) is the bedrock on which deep learning is built, `FedAvg` distributes SGD to solve FL. The paper shows that periodic averaging and synchronizing of local models, trains a model that generalizes across all clients. Further given the expensive communication channel between clients and the coordination server, in some cases, frequency of synchronization can be tuned to improve efficiency.

2.5.2 Loss to Minimize

The algorithm aims to train a model whose loss is the weighted average of the loss at each participating client. The following notation is introduced.

$$\begin{gathered}\mathcal{C} \text{ is the sets of clients that participate where each client } c \in \mathcal{C} \text{ has data partition } \mathcal{P}_c \\ \text{the number of data points on each client } c \text{ is given by } n_c = |\mathcal{P}_c| \\ \text{the size of the full partition is given by} n = \sum_{\forall c \in \mathcal{C}} n_c\end{gathered} \tag{2.24}$$

The algorithm attempts to return θ^*, the optimal set of parameters that minimize the average loss across all participating clients. Mathematically:

$$\begin{gathered}\theta^* = \arg\min_{\theta} F(\theta) \\ \text{where} \quad F(\theta) = \sum_{k=1}^{k=K} \frac{n_c}{n} f_c(\theta) \\ f_c(\theta) = \ell(\theta; \mathcal{P}_c)\end{gathered} \tag{2.25}$$

$f_c(\theta)$ can be replaced by an expectation on the samples from $\mathcal{D}_c$, the distribution over $\mathcal{P}_c$, if stochastic gradient descent is used.

2.5.3 Algorithm

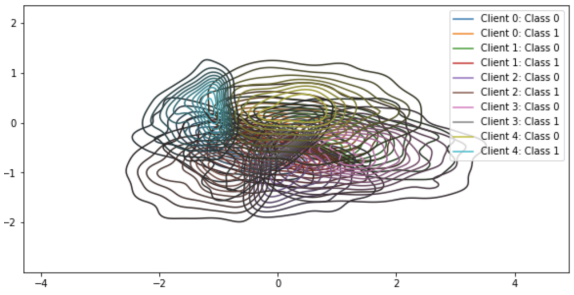
Algorithm 1: `FedAvg`. where

- $\mathcal{C}$ is the set of clients
- T is the number of rounds the algorithm runs for
- q is the number of clients selected per round
- E is the number of local epochs
- B is the local minibatch size
- η is the learning rate

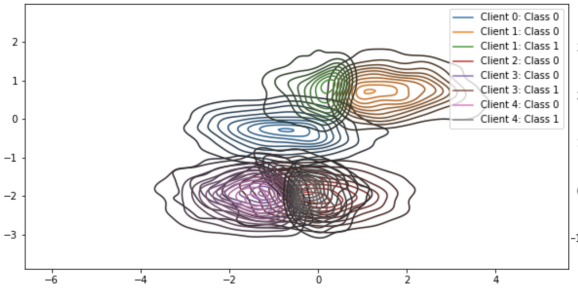
def *Server*() :
  initialize Θ_1
  for each Round $t = 1, ..., T$ **do**
    $\mathcal{C}_t \leftarrow$ Randomly sample q clients from $\mathcal{C}$
    for each client $c \in \mathcal{C}_t$ **in parallel do**
      $\theta_t^c \leftarrow \mathit{ClientUpdate}(\Theta_t, c)$
    end
    $\Theta_{t+1} = \sum_{\forall c \in \mathcal{C}_t} \frac{n_c}{n} \theta_t^c$
  end
end
def *ClientUpdate*(Θ_t, c) : *(Run on client c*
  $i \leftarrow 0$
  $\theta_t^{c,i} = \Theta_t$
  $\mathcal{B} \leftarrow$ Split $\mathcal{P}_c$ into batches of size B
  for each local epoch $e = 1, 2, .., E$ **do**
    for batch $b \in \mathcal{B}$ **do**
      $\theta_t^{c,i+1} \leftarrow \theta_t^{c,i} + \eta \nabla \ell(\theta_t^{c,i}; b)$
      $i \leftarrow i + 1$
    end
  end
  return $\theta_t^{c,i}$ to server
end

Algorithm 1, is largely lifted from McMahan et al. (2016) where the notation is adapted to 2.24 and to the notation used in the mathematical exploration in 3.[13]

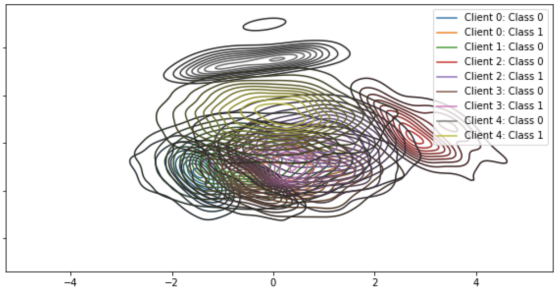
If the algorithm is run with $T = 1$, `FedAvg` becomes the one-shot averaging approach discussed in section 2.4. Thus from one perspective, `FedAvg` can be thought as a generalization of one-shot averaging. As `FedAvg` does not transfer any raw data, $\mathcal{P}_c$ is never transferred between clients nor with the server, `FedAvg` respects the privacy principles described in section 2.1.



(a) `Synthetic`$(0, 0)$



(b) `Synthetic`$(0.5, 0.5)$



(c) `Synthetic`$(1, 1)$

Figure 2.9: Visualization of data sampled from `Synthetic` under different values of γ and ϵ

2.5.4 Previous Experiments on NIID-ness

As mentioned in section 2.4.2, the reason why `FedAvg` has brought decentralized machine learning into the limelight is because it is the first algorithm proposed that allows a federation of clients to jointly train a model that

1. has a non-convex loss
2. can tolerate, up to some extent, NIID data.

[13]

Tian Li et al. give a generative process, `Synthetic`, for producing NIID data for a multi-class classification problem. This process, has the property that the classfication pattern can be learnt by a single-layer perceptron. Further the amount of feature shift and concept drift between the clients can be controlled.[21] The generative process `Synthetic`(γ, ϵ) is described in algorithm 2.

Algorithm 2: `Synthetic`(γ, ϵ) where
- γ controls amount of feature shift
- ϵ controls concept drift
- N number of data points generated
- F dimension size of features
- L is the number of features. Hence $\mathcal{L} = [1, ..., L]$

def *Synthetic*(γ, ϵ)
  Initialize $u \sim \mathcal{N}(0, \gamma)$
  Initialize $\epsilon \sim \mathcal{N}(0, \epsilon)$
  Initialize $\mathbf{\Sigma} \in \mathbb{R}^{F \times F}$ *is diagonal where* $\mathbf{\Sigma}_{f,f} = f^{-1.2}$
  Initialize $\mathbf{X} \in \mathbb{R}^{N \times F} \sim \mathcal{N}(v, \mathbf{\Sigma})$
  Initialize $b \in \mathbb{R}^{L} \sim \mathcal{N}(u, 1)$
  Initialize $\mathbf{W} \in \mathbb{R}^{F \times L} \sim \mathcal{N}(u, 1)$
  $Y \in \mathcal{L}^{N} \leftarrow \arg\max(softmax(\mathbf{X} \cdot \mathbf{W} + b))$
  return $(\mathbf{X}, Y)$
end

The data generated by algorithm 2 is visualized in figure 2.9.

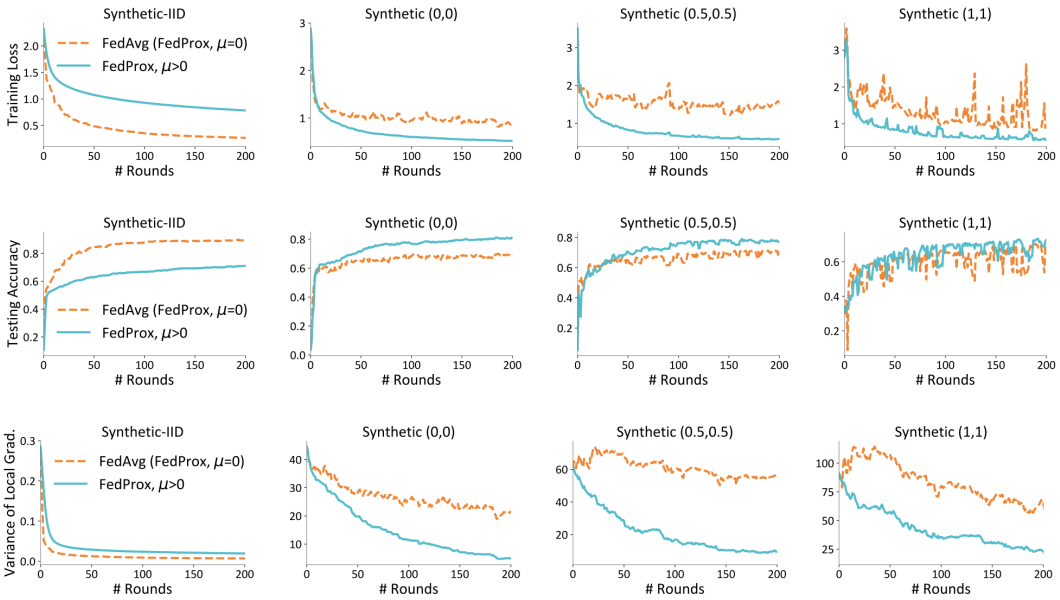


Figure 2.10: Li et al (2018)s exploration of `FedAvg` on various synthetic classification tasks [21]

Li et al (2018) perform `FedAvg` with three different combinations of γ and ϵ where the number of data points per client is distributed according to a power law. The results are recorded for 30 clients, with $E = 20$, and every client participates in each round.

As the data becomes increasingly NIID, the variance between local clients increases by a factor of 100 in the experiment shown in figure 2.10. This results in a decrease in test accuracy and an increase in training loss. Even though between `Synthetic(0,0)` and `Synthetic(1,1)` the accuracy and loss does not change, the training becomes increasingly unstable.

2.6 `FedAvg` on NIID Linear Regression

Section 2.4 explains that one-shot averaging when client data is NIID, even in the convex case of linear regression, does not work. Section 2.5.4, however, shows that the periodical averaging of client models such as in `FedAvg` can, to some extent, be robust to NIID data. To show why more frequent synchronization is beneficial, this section shows empirically on linear regression why `FedAvg` can potentially lift the curse of NIID data. This section further discusses choices for hyperparamters defined in algorithm 1.

Throughout this section, `FedAvg` is performed on the dataset generated by process 2.12 with the parameters chosen in 2.18.

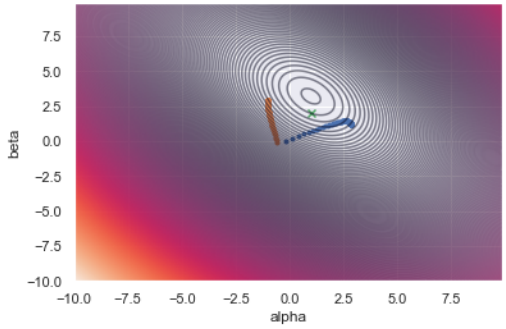
2.6.1 The Two Extremes of FedAvg

One-Shot Averaging

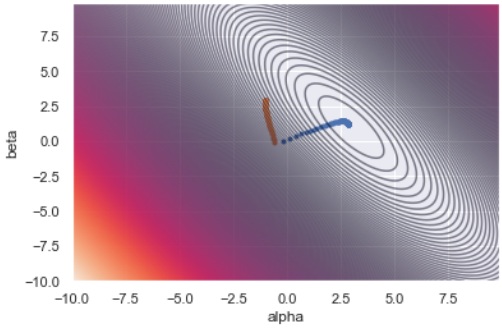
As mentioned in section 2.5.3, when the only one round with a large number of client steps is considered, `FedAvg` is effectively one-shot averaging. This is one extreme configuration. 2.11 shows `FedAvg` on NIID linear regression for hyperparameters that resemble one-shot averaging. Both clients make complete progress towards their own local minimum, but as shown in section 2.4, this need not be the local of the average loss surface.

Averaging after Every Step

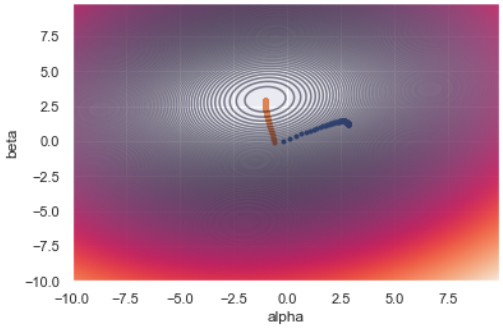
The other extreme configuration of `FedAvg` is when averaging is performed after every client step. Chapter 3 shows that theoretically in a one-epoch, full-batch setting where all clients participate, $(E = 1, B = \infty, q = |\mathcal{C}|)$, this is equivalent to performing gradient descent on the union of all client datasets. Figure 2.12 shows `FedAvg` on NIID linear regression for hyperparameters that resemble



(a) With respect $\ell(\alpha, \beta; \mathcal{D})$

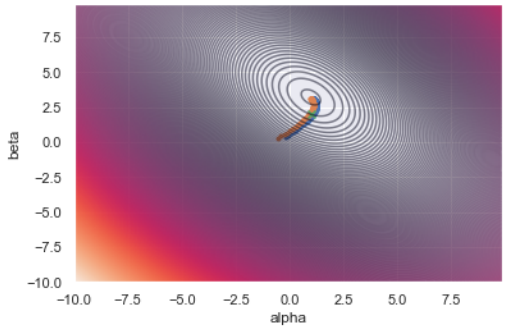


(b) With respect to $\ell(\alpha, \beta; \mathcal{P}_1)$

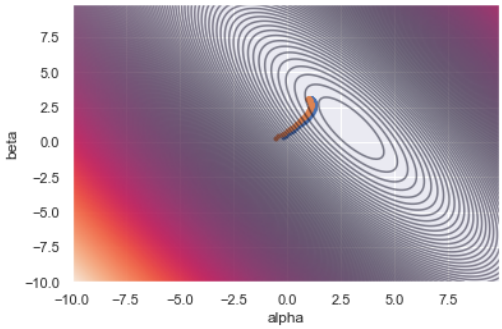


(c) With respect to $\ell(\alpha, \beta; \mathcal{P}_2)$

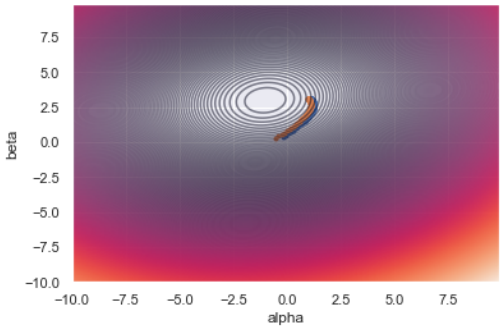
Figure 2.11: Steps of `FedAvg Lr=0.02, n_rounds=1, n_steps=100` on multiple loss surfaces



(a) With respect $\ell(\alpha, \beta; \mathcal{D})$



(b) With respect to $\ell(\alpha, \beta; \mathcal{P}_1)$



(c) With respect to $\ell(\alpha, \beta; \mathcal{P}_2)$

Figure 2.12: Steps of `FedAvg Lr=0.02, n_rounds=100, n_steps=1` on multiple loss surfaces

this extreme. After 100 rounds, very little progress is made with respect to each clients local minima. The averaged model has, however, converged on the global minimum.

2.6.2 A Balance of Two Extremes

Intuitively, when the learning rate is high, and when a large number of local steps is performed, at least in the convex setting, clients make large local progress. However if client distributions are different this may not result in progress towards the minimum of the average loss. On the other hand, when the learning rate is small, and only a few local steps are performed, clients make incomplete progress towards their respective minima. If this incomplete local progress is averaged and fed into client training for the next round, the learning mimics gradient descent performed on the union of all client datasets. Thus the averaged model approaches the minimum of the average loss.

Unfortunately in practice the theoretical perspective above needs to be balanced with the reality that is: the communication between clients is assumed to be the bottleneck (section 2.1. Thus if allowing clients to perform more local steps does not result in a divergence from the optimal parameters (rigorously defined in chapter 3), allowing clients to perform more than one local step reduces the convergence time of training.[13] However chapter 3 shows this lack of divergence can only be made in the case when client distributions are similar and local minima across clients are more or less the same. It is hence crucial to tune `FedAvg`'s hyperparameters to balance between the communication bottleneck and the statistical heterogeneity.

Given this discussion, two questions naturally arise:

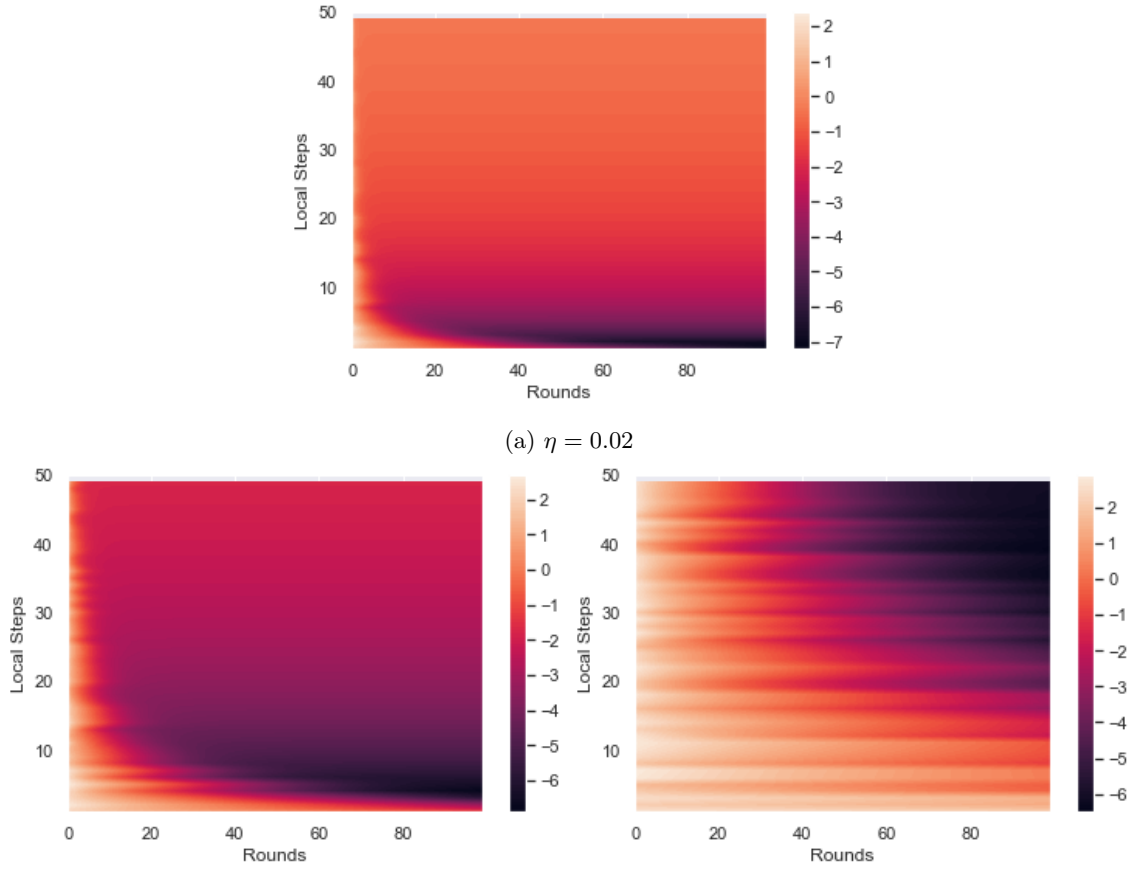
1. How much local progress can be tolerated before `FedAvg` observes too much divergence?
2. What is the optimal amount of local progress per round to reach the global minimum in the least amount of rounds?

Although these largely remain open questions as the answer fully depend on the specifics of the loss surfaces, chapter 3, attempts to mathematically phrase these questions. By bounding properties of the loss surfaces across clients, chapter 3 partially gives an upper bound to the first question.

2.6.3 Simulation Results

Without the mathematical rigour, for the simple case of linear regression, we can visualize this divergence in parameters by observing Ω (2.19) under different hyperparameters.

The experiments shown in figure 2.13 corroborate the intuition described in section 2.6.2 in the case of linear regression. As seen in 2.13, for larger learning rates, `FedAvg` only reaches the global minimum when the number of local steps is limited to one or two. As the learning rate is decreased, more steps per round are required to reach the same Ω. Across the different learning rates, the experiments show that as the number of rounds increases, the number of local steps performed per round must also decrease to reach the same Ω. Lastly, the experiments show that increasing the learning rate does not reduce the number of rounds required to reach a certain Ω. If, however, the number of rounds is constrained, depending on the learning rate, increasing local progression is observed to be helpful.



(a) $\eta = 0.02$

(b) $\eta = 0.01$

(c) $\eta = 0.001$

Figure 2.13: Ω for linear regression optimized through `FedAvg` under different learning rates and client steps

Chapter 3

Bounding FedAvg Parameter Divergence

This section aims to rigorously define why under NIID client data, global progress is impeded as the number of local steps in FedAvg increases.

3.1 Problem Formulation

In neural network based supervised learning, we are interesting in minimizing the empirical risk, i.e. the performance of a model on a known training set $\mathcal{D}$. We use the notion of a loss ℓ to encapsulate this risk for a given set of network parameters θ. This loss is calculated with respect to $\mathcal{D}$. We use ℓ loosely to represent the *negative* average pair-wise prediction loss.

$$\ell(\theta; \mathcal{D}) = \frac{-1}{|\mathcal{D}|} \sum_{\forall (x_i, y_i) \in \mathcal{D}} \ell(f_\theta(x_i), y_i) \tag{3.1}$$

Throughout the analysis, just like in 3.1, we assume full-batches are used whenever calculating the loss or the gradient of the loss. This is to simply analysis as it ensures no stochasticity.

3.1.1 Gradient Descent

The goal of any learning algorithm is to return an optimal parameter θ^* that minimizes the loss with respect to some training set.

$$\theta^* = \arg\min_\theta \ell(\theta; \mathcal{D}) \tag{3.2}$$

Gradient descent is a popular optimization method that iteratively reaches θ^* by taking steps in the negative direction of the loss gradient. One step of gradient descent is described as:

$$\theta_{t+1} \leftarrow \theta_t + \eta \nabla \ell(\theta_t; \mathcal{D}) \tag{3.3}$$

We henceforth refer to gradient descent on a single node with full access to $\mathcal{D}$ as *centralized gradient descent*.

3.1.2 Partitioned Gradient Descent

When the training set is very large, the dataset can be partitioned across multiple nodes to allow computation to scale out. Data across nodes are partitioned such that each partition has the same amount of data points.

$$\forall p_1, p_2 \in \mathcal{P} : |p_1| = |p_2|, p_1 \cap p_2 = \emptyset \tag{3.4}$$

$$\mathcal{D} = \bigcup_{\forall p \in \mathcal{P}} p \tag{3.5}$$

Parameters updates are now in the direction of the average gradient.

$$\theta_{t+1} \leftarrow \theta_t + \frac{\eta}{|\mathcal{P}|} \sum_{\forall p \in \mathcal{P}} \nabla \ell(\theta; p) \tag{3.6}$$

Due to our assumption of full-batches, 3.6 is mathematically equivalent to 3.3. In other words, partitioned gradient descent can be considered to be equivalent to

$$\nabla \ell(\theta_t; \mathcal{D}) = \frac{1}{|\mathcal{P}|} \sum_{\forall p \in \mathcal{P}} \nabla \ell(\theta; p) \tag{3.7}$$

3.7 will show to be a useful conceptual reformulation later in the analysis.

3.1.3 `FedAvg`

As described in algorithm 1, `FedAvg` allows us to vary the batch size, the number of clients sampled per round, and the number of epochs each client cycles through per round. As such it is important to differentiate between the model parameters after a server performs averaging and the model parameters after a client performs gradient descent on their partition. The following notation is introduced to distinguish between the two.

Notation

$$\Theta_t^{(N)} \text{- Server model after } t \text{ rounds where each round consists of averaging client models after } N \text{ client steps} \tag{3.8}$$

$$\mathcal{C} \text{ - the set of participating clients, where client } c \text{ has data partition } \mathcal{P}_c \tag{3.9}$$

We define the full dataset $\mathcal{D}$ as the union of all the partitions. In a centralized setting, a single entity would have access to the entirety of $\mathcal{D}$.

$$\mathcal{D} = \bigcup_{\forall c \in \mathcal{C}} \mathcal{P}_c \tag{3.10}$$

$$\theta_t^{c,n} \text{- Client model after client } c \text{ performs } n \text{ out of } N \text{ steps of gradient descent on } \Theta_t^{(N)} \tag{3.11}$$

Further Constraints

In the following analysis the following constraints are put on `FedAvg` to remove stochasticity and to simply the analysis.

1. Every client has the same amount of data. $\forall c, c' \in \mathcal{C} : |\mathcal{P}_c| = |\mathcal{P}_{c'}|$
2. Every client participates per round $\forall t, N : \Theta_{t+1}^{(N)} = \frac{1}{|\mathcal{C}|} \sum_{\forall c \in \mathcal{C}} \theta_{t-1}^{c,N}$
3. Full batches are always used, i.e. 3.1 is the definition of ℓ. As full batches are used, E defined in algorithm 1, is equivalent to the number of steps N a client performs.

3.2 Defining `FedAvg` Divergence

The notion of divergence can now also be defined. We say a `FedAvg` model $\Theta_t^{(N)}$ has diverged by d if the `FedAvg` model differs by distance d to a centralized gradient descent model that has made the same number of passes over the full dataset.

$$\begin{gathered} div(\Theta_t^{(N)}) = ||\Theta_t^{(N)} - \theta_{Nt}||_2 \\ \text{where } \forall n \in [1, ...nT], \theta_{Nt} \leftarrow \theta_{Nt-n} + \eta \nabla \ell(\theta_{Nt-n}; \mathcal{D}) \end{gathered} \tag{3.12}$$

3.3 `FedAvg` N=1

When each client only takes one full-batch step, averaging client parameters is identical to one step of full-batch over $\mathcal{D}$. This is because the server model after one round is simply one gradient descent step in the direction of the average loss across the whole dataset.

3.3.1 One Round

As a round of `FedAvg` involves the averaging of client models,

$$\Theta_{t+1}^1 \leftarrow \frac{1}{|\mathcal{C}|} \sum_{\forall c \in \mathcal{C}} \theta_t^{c,1} \tag{3.13}$$

where each client step is defined as:

$$\theta_t^{c,1} \leftarrow \Theta_t + \eta \nabla \ell(\Theta_t; \mathcal{P}_c) \tag{3.14}$$

Rewriting Θ_{t+1}^1 in terms of Θ_t^1:

$$\Theta_{t+1}^1 \leftarrow \Theta_t + \frac{\eta}{|\mathcal{C}|} \sum_{\forall c \in \mathcal{C}} \nabla \ell(\Theta_t; \mathcal{P}_c) \tag{3.15}$$

Using 3.7, this can be rewritten as:

$$\Theta_{t+1}^1 \leftarrow \Theta_t + \eta \nabla \ell(\Theta_t; \mathcal{D}) \tag{3.16}$$

Thus performing one step of `FedAvg` $N = 1$ is identical to centralized gradient descent. As such, a step of `FedAvg` under this setting progresses towards Θ^*, i.e. makes global progress.

3.3.2 Two Rounds

After two rounds, the server model reaches:

$$\begin{aligned} \Theta_{t+2}^1 &\leftarrow \Theta_{t+1}^1 + \frac{\eta}{|\mathcal{C}|} \sum_{\forall c \in \mathcal{C}} \nabla \ell(\Theta_{t+1}^1; \mathcal{P}_c) \\ \Theta_{t+2}^1 &\leftarrow \Theta_t + \frac{\eta}{|\mathcal{C}|} \sum_{\forall c \in \mathcal{C}} \nabla \ell(\Theta_t; \mathcal{P}_c) + \nabla \ell(\Theta_{t+1}^1; \mathcal{P}_c) \end{aligned} \tag{3.17}$$

Observing 3.7 and 3.16, we can rewrite this as:

$$\Theta_{t+2}^1 \leftarrow \Theta_t + \eta(\nabla \ell(\Theta_t; \mathcal{D}) + \nabla \ell(\Theta_{t+1}^1; \mathcal{D})) \tag{3.18}$$

We see that consequent rounds of `FedAvg` $N = 1$ behave identically to centralized gradient descent. We have hence shown that this argument holds for an arbitrary number of rounds. In other words, `FedAvg` $N = 1$ continues to make global progress across the rounds, as long as centralized gradient descent approaches Θ^* across the iterations.

As this is the case, we can define the divergence defined in 3.12, in terms of `FedAvg` $N = 1$.

$$div(\Theta_t^{(N)}) = ||\Theta_t^{(N)} - \Theta_{Nt}^1||_2 \tag{3.19}$$

3.4 FedAvg N=2

As now models are averaged after two client steps:

$$\Theta_{t+1}^2 \leftarrow \frac{\eta}{|\mathcal{C}|} \sum_{\forall c \in \mathcal{C}} \theta_t^{c,2} \tag{3.20}$$

where:

$$\begin{aligned} \theta_t^{c,1} &\leftarrow \Theta_t + \eta \nabla \ell(\Theta_t; \mathcal{P}_c) \\ \theta_t^{c,2} &\leftarrow \theta_t^{c,1} + \eta \nabla \ell(\theta_t^{c,1}; \mathcal{P}_c) \end{aligned} \tag{3.21}$$

Rewriting Θ_{t+1}^2 in terms of Θ_t^2:

$$\begin{aligned} \theta_t^{c,2} &\leftarrow \Theta_t + \eta(\nabla \ell(\Theta_t; \mathcal{P}_c) + \nabla \ell(\theta_t^{c,1}; \mathcal{P}_c)) \\ \Theta_{t+1}^2 &\leftarrow \Theta_t + \frac{\eta}{|\mathcal{C}|} \sum_{\forall c \in \mathcal{C}} \nabla \ell(\Theta_t; \mathcal{P}_c) + \nabla \ell(\theta_t^{c,1}; \mathcal{P}_c) \end{aligned} \tag{3.22}$$

This almost looks identical to Θ^1_{t+2} in 3.17, except here the second term in the summation is. $\nabla\ell(\theta^{c,1}_t; \mathcal{P}_c)$, while in 3.17, the term is $\nabla\ell(\Theta^1_{t+1}; \mathcal{P}_c)$. Hence Θ^2_{t+1} is equivalent to Θ^1_{t+2} if for all clients c, $\theta^{c,1}_t$ is equivalent to Θ^1_{t+1}. Comparing 3.14 and 3.16, we see that this can only be true if, $\forall c \in \mathcal{C}, \nabla\ell(\Theta_t; \mathcal{P}_c) = \nabla\ell(\Theta_t; \mathcal{D})$, or in other words true if the loss evaluated at any client is the same if the loss had been evaluated on the whole dataset.

If the data is identically distributed, the loss function at all clients would be the same. This means the loss function across all clients would be the same as the loss function of any one client. It follows that the gradient of the loss evaluated at all or one client would be identical. Thus averaging the client model after two local steps, when data is IID, would be identical to taking two steps of gradient descent with respect to $\mathcal{D}$. As argued in 3.3.2, this means global progress is made.

However if the data is NIID, the loss for the same parameter evaluated at each client would result in different values. Thus the loss at any one client is no longer the loss over all clients. Hence performing two local steps of gradient descent before averaging is not the same as performing two steps of gradient descent on $\mathcal{D}$).

3.5 Bounding the Divergence

3.5.1 Loss Surface Assumptions

To allow us to bound the divergence between different configurations of `FedAvg`, the following assumptions are made about the loss surfaces.

Bounded Variance

For any model parameter θ, a the loss gradient evaluated at any client differs at most by *sigma* when compared with the loss gradient evaluated across the entire dataset. This is inspired from [28].

$$\forall\theta, c : ||\nabla\ell(\theta; \mathcal{P}_c) - \nabla\ell(\theta; \mathcal{D}))||_2 \leq \sigma \tag{3.23}$$

Following 3.7 this can be rewritten as:

$$\forall\theta, c : ||\nabla\ell(\theta; \mathcal{P}_c) - \frac{1}{|\mathcal{C}|}\sum_{\forall c' \in \mathcal{C}} \nabla\ell(\theta; \mathcal{P}_{c'}))||_2 \leq \sigma \tag{3.24}$$

L-Smoothness

For any loss surface, the average gradient between two bounded points is also bounded.

$$\forall\theta, \theta' : ||\nabla\ell(\theta; \mathcal{P}_c) - \nabla\ell(\theta'; \mathcal{P}_c)||_2 \leq L||\theta - \theta'||_2 \tag{3.25}$$

Combining the Two

As each loss surface is L-smooth and the variance in loss gradients is bounded, the average gradient between two bounded points between two loss surfaces is also bounded.

$$\forall\theta, \theta' : ||\nabla\ell(\theta; \mathcal{P}_c) - \nabla\ell(\theta'; \mathcal{D})||_2 \leq L||\theta - \theta'||_2 + \sigma \tag{3.26}$$

This is because we can use the triangle equality with 3.23 and 3.25.

$$\begin{aligned}
&||\nabla\ell(\theta; \mathcal{P}_c) - \nabla\ell(\theta'; \mathcal{D})||_2 \\
&= ||\nabla\ell(\theta; \mathcal{P}_c) - \nabla\ell(\theta'; \mathcal{P}_c) + \nabla\ell(\theta'; \mathcal{P}_c) - \nabla\ell(\theta'; \mathcal{D})||_2 \\
&\leq ||\nabla\ell(\theta; \mathcal{P}_c) - \nabla\ell(\theta'; \mathcal{P}_c)||_2 + ||\nabla\ell(\theta'; \mathcal{P}_c) - \nabla\ell(\theta'; \mathcal{D})||_2 \\
&\leq L||\theta - \theta'||_2 + \sigma
\end{aligned} \tag{3.27}$$

3.5.2 Divergence after One Round of FedAvg N=2

As outlined in 3.19, we are interested in seeing the difference between Θ^2_{t+1} (the FedAvg model after one round where each client performs two local steps) and Θ^1_{t+2} (the FedAvg model after two rounds where each client performs one local step).

$$div(\Theta^2_{t+1}) = ||\Theta^2_{t+1} - \Theta^1_{t+2}||_2 \tag{3.28}$$

Substituting the values from 3.18 and 3.22, 3.28 can be rewritten as:

$$\frac{\eta}{|\mathcal{C}|}|| \sum_{\forall c \in \mathcal{C}} \nabla \ell(\theta^{c,1}_t; \mathcal{P}_c) - \nabla \ell(\Theta^1_{t+1}; \mathcal{P}_c)||_2 \tag{3.29}$$

To bound this entire term, we take an arbitrary client c and first bound the difference between $\theta^{c,1}_t$ and Θ^1_{t+1}. Using the bounds between these two model parameters, we then bound the gradient difference between them using the L-smoothness assumption.
Using definitions 3.13 and 3.14:

$$\begin{aligned} &\forall c \in \mathcal{C}, ||\theta^{c,1}_t - \Theta^1_{t+1}||_2 \\ &= ||\theta^{c,1}_t - \frac{1}{|\mathcal{C}|} \sum_{\forall c' \in \mathcal{C}} \theta^{c',1}_t||_2 \\ &= ||\nabla \ell(\Theta_t; \mathcal{P}_c) - \frac{1}{|\mathcal{C}|} \sum_{\forall c' \in \mathcal{C}} \nabla \ell(\Theta_t; \mathcal{P}_{c'})||_2 \end{aligned} \tag{3.30}$$

Following from the bounded variance assumption 3.24, we see that

$$\forall c \in \mathcal{C}, ||\theta^{c,1}_t - \Theta^1_{t+1}||_2 \leq \sigma \tag{3.31}$$

Now $\theta^{c,1}_t$ and Θ^1_{t+1} is bounded, using the L-smoothness assumption (3.25), we can bound the gradient difference.

$$\begin{aligned} &\forall c \in \mathcal{C}, ||\nabla \ell(\theta^{c,1}_t; \mathcal{P}_c) - \nabla \ell(\Theta^1_{t+1}; \mathcal{P}_c)||_2 \\ &\leq L||\theta^{c,1}_t - \Theta^1_{t+1}||_2 \\ &\leq L\sigma \end{aligned} \tag{3.32}$$

From the triangle inequality we can bound 3.29 by:

$$\frac{\eta}{|\mathcal{C}|}|| \sum_{\forall c \in \mathcal{C}} \nabla \ell(\theta^{c,1}_t; \mathcal{P}_c) - \nabla \ell(\Theta^1_{t+1}; \mathcal{P}_c)||_2 \leq \frac{\eta}{|\mathcal{C}|} \sum_{\forall c \in \mathcal{C}} ||\nabla \ell(\theta^{c,1}_t; \mathcal{P}_c) - \nabla \ell(\Theta^1_{t+1}; \mathcal{P}_c)||_2 \tag{3.33}$$

As we have the bounded the gradient difference, in 3.32

$$\frac{\eta}{|\mathcal{C}|} \sum_{\forall c \in \mathcal{C}} ||\nabla \ell(\theta^{c,1}_t; \mathcal{P}_c) - \nabla \ell(\Theta^1_{t+1}; \mathcal{P}_c)||_2 \leq \eta L\sigma \tag{3.34}$$

Hence putting everything together we have our first theorem.

Theorem 1 $div(\Theta^2_{t+1}) \leq \eta L\sigma$
The divergence of full-batch **FedAvg** $N = 2$ *after one round is bounded by* $\eta L\sigma$.

3.5.3 Divergence after One Round of FedAvg N=3

As outlined in 3.19, we are interested in seeing the difference between Θ^3_{t+1} (the FedAvg model after one round where each client performs three local steps) and Θ^1_{t+3} (the FedAvg model after three rounds where each client performs one local step).

$$div(\Theta^3_{t+1}) = ||\Theta^2_{t+1} - \Theta^1_{t+3}||_2 \tag{3.35}$$

Following the pattern in section 3.3 and 3.3.2, we define the following:

$$\Theta^3_{t+1} \leftarrow \Theta_t + \frac{\eta}{|\mathcal{C}|} \sum_{\forall c \in \mathcal{C}} \nabla \ell(\Theta_t; \mathcal{P}_c) + \nabla \ell(\theta^{c,1}_t; \mathcal{P}_c) + \nabla \ell(\theta^{c,2}_t; \mathcal{P}_c) \tag{3.36}$$

$$\Theta^1_{t+3} \leftarrow \Theta_t + \frac{\eta}{|\mathcal{C}|}\sum_{\forall c\in\mathcal{C}} \nabla\ell(\Theta_t;\mathcal{P}_c) + \nabla\ell(\Theta^1_{t+1};\mathcal{P}_c) + \nabla\ell(\Theta^1_{t+2};\mathcal{P}_c) \tag{3.37}$$

Substituting this into 3.35, we get:

$$\frac{\eta}{|\mathcal{C}|}||\sum_{\forall c\in\mathcal{C}} \nabla\ell(\theta^{c,1}_t;\mathcal{P}_c) + \nabla\ell(\theta^{c,2}_t;\mathcal{P}_c) - \nabla\ell(\Theta^1_{t+1};\mathcal{P}_c) - \nabla\ell(\Theta^1_{t+2};\mathcal{P}_c)||_2 \tag{3.38}$$

We break down 3.38 into the following two parts.

Part A

$$\frac{\eta}{|\mathcal{C}|}||\sum_{\forall c\in\mathcal{C}} \nabla\ell(\theta^{c,1}_t;\mathcal{P}_c) - \nabla\ell(\Theta^1_{t+1};\mathcal{P}_c)||_2 \leq \eta L\sigma \tag{3.39}$$

As 3.39 is identical to 3.29, we know from **Theorem 1** that 3.39 is bounded by $\eta L\sigma$.

Part B

$$\frac{\eta}{|\mathcal{C}|}||\sum_{\forall c\in\mathcal{C}} \nabla\ell(\theta^{c,2}_t;\mathcal{P}_c) - \nabla\ell(\Theta^1_{t+2};\mathcal{P}_c)||_2 \tag{3.40}$$

To bound 3.40, we first need to bound $\theta^{c,2}_t - \Theta^1_{t+2}$. If we take an arbitrary client c, and use definitions 3.21 and 3.17:

$$\begin{aligned}&\forall c \in \mathcal{C}, ||\theta^{c,2}_t - \Theta^1_{t+2}||_2\\ &= \eta||\nabla\ell(\Theta_t;\mathcal{P}_c) + \nabla\ell(\theta^{c,1}_t;\mathcal{P}_c) - \frac{1}{|\mathcal{C}|}\sum_{\forall c'\in\mathcal{C}} \nabla\ell(\Theta_t;\mathcal{P}_{c'}) + \nabla\ell(\Theta^1_{t+1};\mathcal{P}_{c'})||_2\end{aligned} \tag{3.41}$$

We break 3.41 into two parts.

1. $$||\nabla\ell(\Theta_t;\mathcal{P}_c) - \frac{1}{|\mathcal{C}|}\sum_{\forall c'\in\mathcal{C}} \nabla\ell(\Theta_t;\mathcal{P}_{c'})||_2 \leq \sigma \tag{3.42}$$

   We see this is bounded from our bounded variance assumption.

2. $$\begin{aligned}&||\nabla\ell(\theta^{c,1}_t;\mathcal{P}_c) - \frac{1}{|\mathcal{C}|}\sum_{\forall c'\in\mathcal{C}} \nabla\ell(\Theta^1_{t+1};\mathcal{P}_{c'})||_2\\ &= ||\nabla\ell(\theta^{c,1}_t;\mathcal{P}_c) - \nabla\ell(\Theta^1_{t+1};\mathcal{D})||_2\end{aligned} \tag{3.43}$$

   From 3.31, we know that $|\theta^{c,1}_t - \Theta^1_{t+1}||_2$ is bounded by σ. Hence using 3.26 we have:

$$||\nabla\ell(\theta^{c,1}_t;\mathcal{P}_c) - \nabla\ell(\Theta^1_{t+1};\mathcal{D})||_2 \leq L\sigma + \sigma \tag{3.44}$$

Combining the bounds given by 3.42 and 3.44, we can use the triangle inequality to get the following bound.

$$\forall c \in \mathcal{C}, ||\theta^{c,2}_t - \Theta^1_{t+2}||_2 \leq \eta\sigma(L+2) \tag{3.45}$$

Using the L-Smoothness assumption and pulling the sum out of the norm **Part B** hence is bounded by:

$$\frac{\eta}{|\mathcal{C}|}||\sum_{\forall c\in\mathcal{C}} \nabla\ell(\theta^{c,2}_t;\mathcal{P}_c) - \nabla\ell(\Theta^1_{t+2};\mathcal{P}_c)||_2 \ \leq \eta^2 L\sigma(L+2) \tag{3.46}$$

Thus putting **Part A** and **Part B** together, we have:

$$||\Theta^3_{t+1} - \Theta^1_{t+3}||_2 \leq \eta L\sigma + \eta^2 L\sigma(L+2) \tag{3.47}$$

Theorem 2 *$div(\Theta^3_{t+1}) \leq \eta L\sigma + \eta^2 L\sigma(L+2)$*
The divergence of full-batch `FedAvg` *$N = 3$ after one round is bounded by $\eta L\sigma + \eta^2 L\sigma(L+2)$.*

3.5.4 Divergence of FedAvg N=2 after Two Rounds

As outlined in 3.19, we are interested in seeing the difference between Θ^2_{t+2} (the FedAvg model after two round where each client performs three local steps) and Θ^1_{t+4} (the FedAvg model after four rounds where each client performs one local step).

$$div(\Theta^2_{t+2}) = ||\Theta^2_{t+2} - \Theta^1_{t+4}||_2 \tag{3.48}$$

Following the pattern in section 3.3 and 3.3.2, we define the following:

$$\Theta^2_{t+2} \leftarrow \Theta^2_{t+1} + \frac{\eta}{|\mathcal{C}|}\sum_{\forall c \in \mathcal{C}} \nabla\ell(\Theta^2_{t+1}; \mathcal{P}_c) + \nabla\ell(\theta^{c,1}_{t+1}; \mathcal{P}_c) \tag{3.49}$$

$$\Theta^1_{t+4} \leftarrow \Theta^1_{t+2} + \frac{\eta}{|\mathcal{C}|}\sum_{\forall c \in \mathcal{C}} \nabla\ell(\Theta^1_{t+2}; \mathcal{P}_c) + \nabla\ell(\Theta^1_{t+3}; \mathcal{P}_c) \tag{3.50}$$

Substituting this into 3.48, we get:

$$||\Theta^2_{t+1} - \Theta^1_{t+2} + \frac{\eta}{|\mathcal{C}|}\sum_{\forall c \in \mathcal{C}} \nabla\ell(\Theta^2_{t+1}; \mathcal{P}_c) + \nabla\ell(\theta^{c,1}_{t+1}; \mathcal{P}_c) - \nabla\ell(\Theta^1_{t+2}; \mathcal{P}_c) - \nabla\ell(\Theta^1_{t+3}; \mathcal{P}_c)||_2 \tag{3.51}$$

This can be broken down into three parts.

Part A

$$||\Theta^2_{t+1} - \Theta^1_{t+2}||_2 \leq \eta L\sigma \tag{3.52}$$

This bound is given by **Theorem 1**.

Part B

$$||\frac{\eta}{|\mathcal{C}|}\sum_{\forall c \in \mathcal{C}} \nabla\ell(\Theta^2_{t+1}; \mathcal{P}_c) - \nabla\ell(\Theta^1_{t+2}; \mathcal{P}_c)||_2 \tag{3.53}$$

Taking an arbitrary client c we attempt to bound we know from **Theorem 1** that $\Theta^2_{t+1} - \Theta^1_{t+2}$ is bounded by $\eta L\sigma$. Hence using the L-smooth assumption, we have:

$$\forall c \in \mathcal{C}, ||\nabla\ell(\Theta^2_{t+1}; \mathcal{P}_c) - \nabla\ell(\Theta^1_{t+2}; \mathcal{P}_c)||_2 \leq \eta L^2\sigma \tag{3.54}$$

Applying this to the whole sum, we have:

$$||\frac{\eta}{|\mathcal{C}|}\sum_{\forall c \in \mathcal{C}} \nabla\ell(\Theta^2_{t+1}; \mathcal{P}_c) - \nabla\ell(\Theta^1_{t+2}; \mathcal{P}_c)||_2 \leq \eta^2 L^2\sigma \tag{3.55}$$

Part C

$$||\frac{\eta}{|\mathcal{C}|}\sum_{\forall c \in \mathcal{C}} \nabla\ell(\theta^{c,1}_{t+1}; \mathcal{P}_c) - \nabla\ell(\Theta^1_{t+3}; \mathcal{P}_c)||_2 \tag{3.56}$$

Taking an arbitrary client c, we attempt to bound $||\theta^{c,1}_{t+1} - \Theta^1_{t+3}||_2$ where each model is defined as:

$$\begin{aligned} \theta^{c,1}_{t+1} &\leftarrow \Theta^2_{t+1} + \eta\nabla\ell(\Theta^2_{t+1}; \mathcal{P}_c) \\ \Theta^1_{t+3} &\leftarrow \Theta^1_{t+2} + \frac{\eta}{|\mathcal{C}|}\sum_{\forall c' \in \mathcal{C}} \nabla\ell(\Theta^1_{t+2}; \mathcal{P}_{c'}) \end{aligned} \tag{3.57}$$

Hence

$$||\theta^{c,1}_{t+1} - \Theta^1_{t+3}||_2 = ||\Theta^2_{t+1} - \Theta^1_{t+2} + \eta\nabla\ell(\Theta^2_{t+1}; \mathcal{P}_c) - \frac{\eta}{|\mathcal{C}|}\sum_{\forall c' \in \mathcal{C}} \nabla\ell(\Theta^1_{t+2}; \mathcal{P}_{c'})||_2 \tag{3.58}$$

Breaking this into two parts:

1. $||\Theta^2_{t+1} - \Theta^1_{t+2}||_2$ which according to **Theorem 1** is bounded by $\eta L\sigma$

2.

$$\begin{aligned}&\eta||\nabla\ell(\Theta^2_{t+1};\mathcal{P}_c) - \frac{1}{|\mathcal{C}|}\sum_{\forall c'\in\mathcal{C}}\nabla\ell(\Theta^1_{t+2};\mathcal{P}_{c'})||_2\\&= \eta||\nabla\ell(\Theta^2_{t+1};\mathcal{P}_c) - \nabla\ell(\Theta^1_{t+2};\mathcal{D})||_2\end{aligned} \tag{3.59}$$

As $||\Theta^2_{t+1} - \Theta^1_{t+2}||_2$ is bounded by $\eta L\sigma$, we can use 3.26 to show:

$$\eta||\nabla\ell(\Theta^2_{t+1};\mathcal{P}_c) - \nabla\ell(\Theta^1_{t+2};\mathcal{D})||_2 \leq \eta\sigma(\eta L^2 + 1) \tag{3.60}$$

Using these two parts:

$$||\theta^{c,1}_{t+1} - \Theta^1_{t+3}||_2 \leq \eta\sigma(\eta L^2 + L + 1) \tag{3.61}$$

Having now bounded $||\theta^{c,1}_{t+1} - \Theta^1_{t+3}||_2$, we can use the L-Smoothness assumption.

$$\forall c \in \mathcal{C}, ||\nabla\ell(\theta^{c,1}_{t+1};\mathcal{P}_c) - \nabla\ell(\Theta^1_{t+3};\mathcal{P}_c)||_2 \leq \eta L\sigma(\eta L^2 + L + 1) \tag{3.62}$$

As this bound exists for any client c,

$$||\frac{\eta}{|\mathcal{C}|}\sum_{\forall c\in\mathcal{C}}\nabla\ell(\theta^{c,1}_{t+1};\mathcal{P}_c) - \nabla\ell(\Theta^1_{t+3};\mathcal{P}_c)||_2 \leq \eta^2 L\sigma(\eta L^2 + L + 1) \tag{3.63}$$

Combining the three parts, we finally have:

$$||\Theta^2_{t+2} - \Theta^1_{t+4}||_2 \leq \eta L\sigma + \eta^2 L\sigma(\eta L^2 + 2L + 1) \tag{3.64}$$

Theorem 3 *$div(\Theta^2_{t+2}) \leq \eta L\sigma + \eta^2 L\sigma(\eta L^2 + 2L + 1)$*
The divergence of full-batch `FedAvg` $N = 2$ after two rounds is bounded by $\eta L\sigma + \eta^2 L\sigma(\eta L^2 + 2L + 1)$.

3.6 Discussion

3.6.1 Conjectured Generalization

Increasing Number of Local Steps

The implication of **Theorem** 1 **Theorem** 2, is that taking one more local step for one round, increases the divergence bound by $\eta^2 L\sigma(L + 2)$. This increase is proportional to the square of the learning rate, and to the square of the L-smoothness factor.

Intuitively this is because as more local steps are taken, each client makes more local progress independently and hence taking the average after three rounds could be increasingly different from the ideal case of three steps of centralized gradient descent.

Theorem 1 states any client parameter after two local steps is at most $\eta L\sigma$ different from the model parameter after two steps of centralized gradient descent. Under the L-smoothness assumption (3.25), the gradient difference between these two models between any two client is proportional to $\eta L^2\sigma$, taking a step in this direction of size of η would result in models that are different by a term proportional to $\eta^2 L^2\sigma$ under the L-Smoothness assumption.

Under this line of reasoning I hypothesize if four local steps are taken before averaging, the bounds increases to that stated in **theorem** 2 by a term that is proportional to $\eta^3 L^3\sigma$. This argument can extended for n local steps.

Increasing Number of Rounds

The implication of **Theorem** 1 and **Theorem** 3 is that performing one more round when client parameters are averaged after two local steps increases the divergence bounds by $\eta^2 L\sigma(\eta L^2 + 2L + 1)$. This term is proportional to L^3 and η^3.

The reason, intuitively, why the divergence increases is because even when the model are initially identical, after just one round the models diverge by $\eta L\sigma$. If one round was performed when the models had already diverged by $\eta L\sigma$ the divergence would only accumulate.

The reason why this additional term is proportional to L^3 and η^3 is because after one round according to the L smoothness assumption (3.25) and **Theorem** 1, the difference in gradient between the `FedAvg` model and the centrally trained model is proportional to $\eta L^2 \sigma$. Hence taking steps of size η would result in models whose difference is proportional to $\eta^2 L^2 \sigma$. **Theorem** 1 shows that when a client model and a central model differ by σ, taking one more step locally and then averaging, compared to performing another iteration of centralized gradient descent, results in models that differ by $\eta L \sigma$. Hence if the difference between the client model and the central model is proportional to $\eta^2 L^2 \sigma$, taking one more step locally and then averaging compared to another iteration of centralized gradient descent results in models that differ by $\eta^3 L^3 \sigma$.

Under this line of reasoning I hypothesize if three rounds are performed by a `FedAvg` that averages after 2 client steps, the divergence would further increase from that shown in **Theorem** 3 by a term that is proportional to $\eta^5 L^5 \sigma$. This argument can be extended for T rounds.

Putting these two conjectured generalizations together, the bounds for a `FedAvg` algorithm that averages after N local client steps at round T can be argued.

3.6.2 Connection to Feature Shift and Concept Drift

Throughout the mathematical demonstration, the assumption is made that for any model parameter, the difference in the gradient of the loss evaluated at the centralized dataset and the gradient of the loss evaluated at any client is bounded by σ (3.23). As shown in section 2.4.1, as the feature-shift and concept-drift independently increase, the loss-surfaces on individual clients and the average loss-surface becomes increasingly more different. As such, the bounds on the gradient difference between client loss-surfaces and the average loss-surface becomes larger. In other words, as NIID-ness increases, a larger σ needs to be assumed. This is also empirically observed by Li et al. in figure 2.9.

Theorem 2 and **Theorem 3** show that as the number of local computation and the number of rounds increases, in both cases the divergence bound increases by a factor that is proportional to σ. In section 3.6.1, I claim that as successive local steps or successive rounds are taken, the divergence increases by a term that is proportional to σ. Hence irregardless of the number of local computation and number or rounds, the divergence is always proportional to σ. As such, the divergence in `FedAvg` is proportional to both the amount feature-shift and the amount of concept-drift.

Chapter 4

Empirical Study of NIID-Data in Next-Word Prediction

Section 2.4 shows empirically that `FedAvg` performance is influenced by the amount of NIID-ness across clients in the case of linear regression. Chapter 3 builds upon this intuition by representing the amount of NIID-ness as the maximum gradient variance σ, and showing that `FedAvg` degrades degrades as σ increases. Following the discussion of the previous sections, this section explores how NIID-ness affects `FedAvg` in next-word prediction; a highly NIID, non-convex problem.

Inspired by the generative process `Synthetic` (algorithm 3) discussed in section 2.5.4, `FedAvg` is studied in next-word prediction by observing the behavior of `FedAvg` trained on sequential data where the feature-shift and concept-drift are controlled. This is done by generating data through `SyntheticSeq`.

4.1 `SyntheticSeq`

Similar to `Synthetic` the goal of `SyntheticSeq` is to generate a dataset whose pattern can be learnt through a simple model. `Synthetic` does this for classification by generating data through a process which mimics the forward pass of a single-layer perception with a Softmax activation. This idea is used for sequential data by instead generating the data through the forward passes of a single-layer recurrent neural network.[22] This can be thought of as an approximation to the latent generative process behind how users use their mobile keyboards. [1][20] This method is fully described in algorithm 3.

4.2 Methodology

4.2.1 Data

Throughout the experiments the following parameters of `SyntheticSeq` are kept constant across clients: $vocab\_len = 100, E = 5, H = 20, n\_seq = 40, seq\_len = 20$ These sets of constants produce a good variety of datasets when K and ϵ are varied.

Varying Feature-Shift

Just like in algorithm 2 the feature-shift is controlling by changing the variance of the process that generates the features. In `Synthetic`, as the features are in $\mathbb{R}^N$, this is done by varying the diagonals of the covariance of the Gaussian used to generate the features. As text data is discrete, as shown in section 2.3, each token is represented as the integer index of the vocabulary set. Further the token at time-step t affects the token at time-step $t + 1$. As such, there is no distribution whose variance can be controlled to generate the features. To achieve this controlled feature-shift, `SyntheticSeq` forces each client to begin every sequence with the same token. To vary feature-shift, the size of the start-token set is varied. This has the effect of creating client clusters of uniform size as seen in Figure 4.1, 4.2, and 4.3. To ensure that each client does not produce the same sequence of tokens if the first token is seeded, the token $t + 1$ is sampled from

Algorithm 3: `SyntheticSeq`(K, ϵ) where

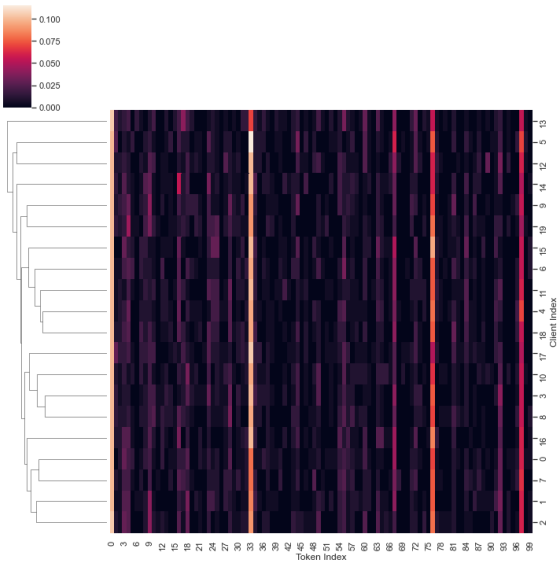
- K is the number of clusters hence controls feature-shift
- ϵ controls concept-drift
- $vocab\_len$ is the size of vocabulary $V, V = \{1, ..., vocab\_len\}$
- E is the embedding dimension size
- H is the hidden state dimension size
- n_seq is the number of sequences to generate,
- seq_len is the number of tokens in each sequence

def *SyntheticSeq(K,ϵ)*

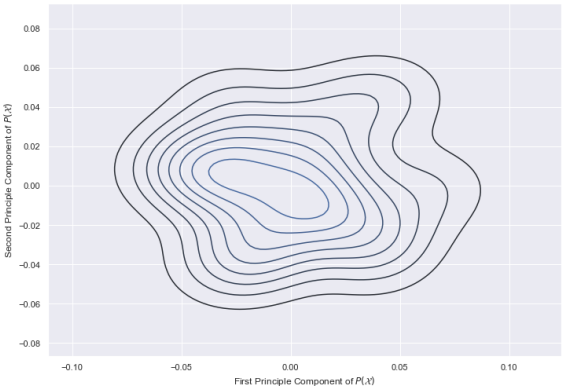
- *Initialize* $\mathbf{T} \in V^{n\_seq \times seq\_len}$
- *(if* $\epsilon = 0$*, the following weight matrices have non-zero deterministic values)*
- *Initialize* $\mathbf{W_{ei}} \in \mathbb{R}^{E \times vocab\_size} \sim \mathcal{N}(0, \epsilon)$
- *Initialize* $\mathbf{W_{hh}} \in \mathbb{R}^{H \times H} \sim \mathcal{N}(0, \epsilon)$
- *Initialize* $\mathbf{W_{he}} \in \mathbb{R}^{H \times E} \sim \mathcal{N}(0, \epsilon)$
- *Initialize* $\mathbf{W_{oh}} \in \mathbb{R}^{vocab\_size \times H} \sim \mathcal{N}(0, \epsilon)$
- $t_0 \leftarrow$ *Sample* $[1, ..., K]$ *with uniform probability*
- **for** $seq \in [1, ..., n\_seq]$ **do**
  - $hidden_{seq,1} \leftarrow 0^T$
  - $t_{seq,1} \leftarrow t_0$
  - **for** $t \in [1, ..., seq\_len]$ **do**
    - $input_{seq,t} \leftarrow onehot(t_{seq,t})$
    - $embed_{seq,t} \leftarrow \mathbf{W_{ei}} \cdot input_{seq,t}$
    - $hidden_{seq,t+1} \leftarrow tanh(\mathbf{W_{hh}} \cdot hidden_{seq,t} + \mathbf{W_{he}} \cdot embed_{seq,t})$
    - $output_{seq,t} \leftarrow \mathbf{W_{oh}} \cdot hidden_{seq,t+1}$
    - $t_{seq,t+1} \leftarrow$ *Sample* V *with probability* $softmax(output_{seq,t})$
  - **end**
- **end**
- ***return*** **T**

end

the output probability for token t. This ensures some variation within clusters. To increase the variation between clusters, a larger sequence length can be chosen.

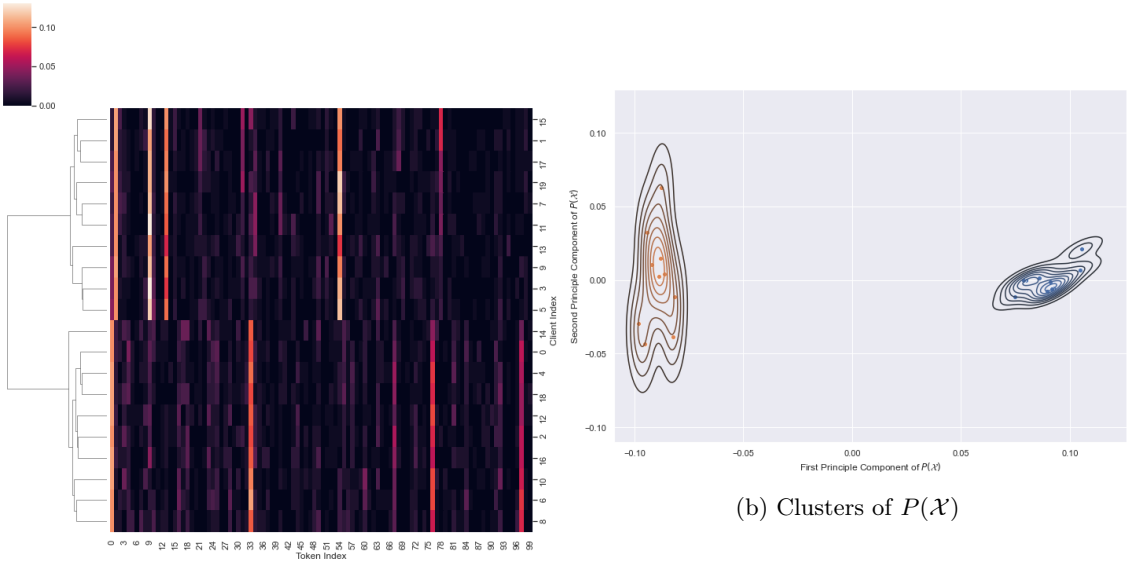


(a) Token frequency, $P(\mathcal{X})$, across clients



(b) Clusters of $P(\mathcal{X})$

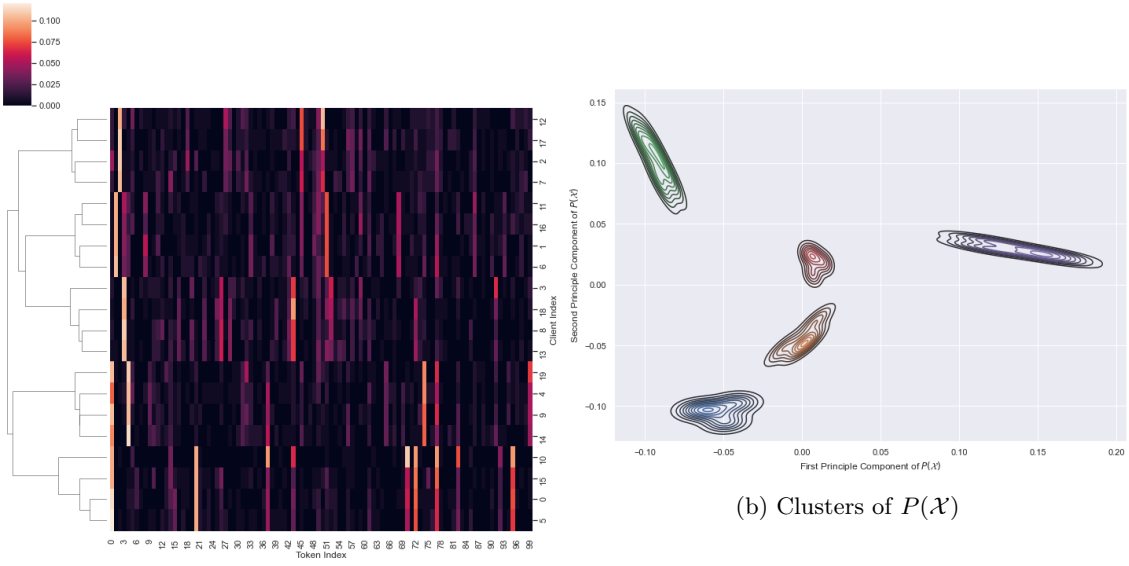
Figure 4.1: `SyntheticSeq`$(1, 0)$ with 20 Clients



(a) Token frequency, $P(\mathcal{X})$, across clients

(b) Clusters of $P(\mathcal{X})$

Figure 4.2: `SyntheticSeq`$(2, 0)$ with 20 Clients



(a) Token frequency, $P(\mathcal{X})$, across clients

(b) Clusters of $P(\mathcal{X})$

Figure 4.3: `SyntheticSeq`$(5, 0)$ with 20 Clients

Varying concept-drift

To vary the concept-drift, `Synthetic`, controls the variance of the weights and biases of the single-layer perception used to perform the forward pass. This has the consequence of varying individuals client's optimal model. Using the same technique, `SyntheticSeq` varies the weights of the recurrent network. `SyntheticSeq`, however, utilizes four weight matrices that represent different aspects of how a client might generate their next word. These four weights are:

- $\mathbf{W_{ei}}$ represents how a client embeds tokens into their embedding space. Clients with similar embedding matrices can be thought to share semantic meaning of the tokens.
- $\mathbf{W_{hh}}$ represents how a client's hidden states evolves. Clients with similar hidden-to-hidden matrices can be thought to share sentence construction preferences.
- $\mathbf{W_{he}}$ represents how the previous word affects the client's hidden state. If this matrix is

similar between clients, it means that clients share how the current context is affected by the previous token.

- $\mathbf{W_{oh}}$ represents how the hidden state of a client maps to the prediction of the next word. If this matrix is similar between clients, it means that clients share how context is mapped to the next token.

To control concept-drift, the variance of these matrices is varied. As discussed in section 2.3, $P(\mathcal{Y}|\mathcal{X})$ in next-word prediction can be thought of as a $vocab\_size \times vocab\_size$ sized matrix. Under no feature-shift, $K = 1$, figure 4.4 shows the increase in variance of $P(\mathcal{Y}|\mathcal{X})$ as ϵ increases.

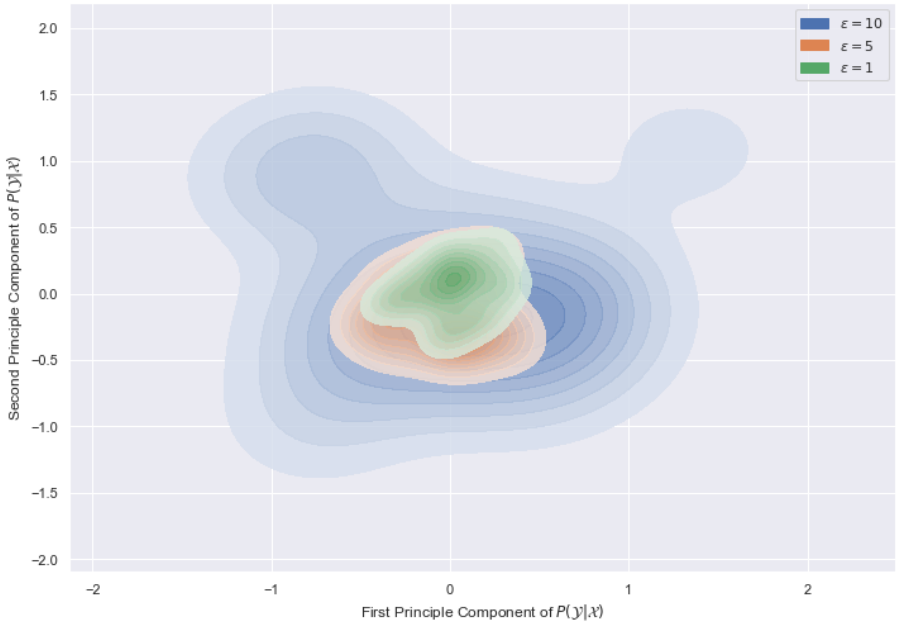


Figure 4.4: Effect of `SyntheticSeq`$(1, \epsilon), \epsilon \in [1, 5, 10]$ on concept-drift

4.2.2 `FedAvg` Parameters

Model

The model used in this experiment is symmetric to the generative process `SytheticSeq`. As the data is produced through the forward pass of a single-layer recurrent neural network, the model used is also a single-layer recurrent neural network. This is implemented using PyTorch shown in listing 4.1.[29]

```
class RNNLM(nn.Module):
    def __init__(self, n_tokens, embedding_dim, hidden_dim):
        super(RNNLM, self).__init__()
        self.W_ei = nn.Embedding(n_tokens, embedding_dim)
        self.rnn = nn.RNN(input_size=embedding_dim, hidden_size=hidden_dim,
                          nonlinearity='tanh', bias=False)
        self.W_ho = nn.Linear(hidden_dim, n_tokens, bias=False)
        self.softmax = nn.Softmax(dim=1)

    def forward(self, x):
        e = self.W_ei(x)
        hidden, _ = self.rnn(e)
        output = self.softmax(self.W_ho(hidden))
        return output
```

Listing 4.1: Recurrent neural language model in PyTorch

Hyperparamters

To observe the behavior of `FedAvg`, each experiment consisted of 20 clients with the same amount of data, and was run for $1,000$ rounds. In line with the assumptions in chapter 3, a full batch $B = \infty = 40$ was used, and all clients participated in each round $q = |\mathcal{C}| = 20$. The following hyperparamters were varied along the domain:

- **Learning Rate**: $lr = [0.1, 0.01, 0.001]$
- **Number of Local Steps**: $N = [1, 2, 5, 10, 20]$
  this given $B = \infty$, translates to $E = [1, 2, 5, 10, 20]$

4.2.3 Metrics

Similar to section 2.4 and chapter 3, the interest is in measuring how far `FedAvg` models drift from their centrally trained counterparts. However given the non-convexity of the problem, a valid `FedAvg` training might take the model to a minimum that is not the minimum parameterized by generative process. This is because where the model ends up is conditioned on the initialization of the model parameters. [11] Direct comparison between the optimal parameters and the `FedAvg` parameters are only made in section 2.6 because:

1. The problem is convex problem, i.e. there is only one valid minimum
2. The minimum can be calculated in closed form

Given these challenges, instead of calculating the divergence in the parameter space, the divergence can be calculated in the loss space. That is:

$$\Omega = \mathcal{L}_{centralized} - \mathcal{L}_{FedAvg}$$
$$\text{where } \mathcal{L}_{centralized} \text{ is the loss the centralized gradient descent reaches} \tag{4.1}$$
$$\text{and where } \mathcal{L}_{FedAvg} \text{ is the average loss of the } \texttt{FedAvg} \text{ model evaluated on all clients}$$

Measuring the divergence in loss space is an approximation to the divergence in parameter space and hence can be used to measure the divergence in a non-convex setting where the optimal parameters cannot be calculated in closed form.

4.3 Impact of Feature-Shift

To measure the impact of feature-shift on the behavior of `FedAvg`, `FedAvg` is performed on `SyntheticSeq`$(K, 0)$ where $K \in [1, 2, 5, 10]$. This is shown in figure 4.5. The x-axis of each plot represents the round and the y-axis represents the number of client steps performed per round. Across the sub-figures, the learning rate is varied across the x-axis, anf the feature-shift is varied across the y-axis. The experiments pertaining to each (K, lr) pair are grouped into the same plot where the contour represents Ω. Even though only $E = [1, 2, 5, 10, 20]$ experiments are performed, the contour plot interpolates the behavior of `FedAvg` for other values of E.

Figure 4.5 highlights that generally, as feature-shift increases, Ω increases. Further as the learning rate decreases, Ω also generally increases. This is in line with the experimental results in section 2.4 and section 2.6 as well as the theory suggested in chapter 3.

It is interesting to note however, that in the IID case of `SyntheticSeq`$(1, 0)$ for learning rate $lr = 0.1$ (figure 4.5a), as the number of client steps increases after $E = 5$, Ω begins to increase. For example, `FedAvg` for $E = 1$ and $E = 20$ have similar performances. This means that for the $lr = 0.01$ experiment (figure 4.5b), for a small region of E where $15 \leq E \leq 20$, `FedAvg` has better performance for rounds $T \approx 400$ compared to $lr = 0.1$. This behavior, however, is not present in the case of `SyntheticSeq`$(2, 0)$ for learning rate $lr = 0.1$ (figure 4.5d).

This behavior in figure 4.5a contrasts the behavior that would be expected if the data was convex. This could be explained due to fact that perhaps in this example for the average loss, there are actually a large number of minima in close proximity, and the average of the minima is not a minimum on the loss-surface. When the number of clusters increases to two, the variance in loss-surfaces increases between clients. If the increased variance is just small enough, it may add noise to the minima and prevent the model from reaching a bad average. In other word, the

increased variance helps the average of the model to better generalize. As the feature-shift further increases however, Ω increases. This can be attributed to the fact that each client's loss-surface is too different for any meaningful averaging.

4.4 Impact of Concept-Drift

To measure the impact of concept-drift on the behavior of `FedAvg`, `FedAvg` is performed on $\texttt{SyntheticSeq}(1, \epsilon)$ where $\epsilon \in [1, 2, 5, 10]$. This is shown in figure 4.6. Just like in figure 4.5, the x-axis of each plot represents the round and the y-axis represents the number of client steps performed per round. The experiments pertaining to each (ϵ, lr) pair are grouped into the same plot where the contour represents Ω.

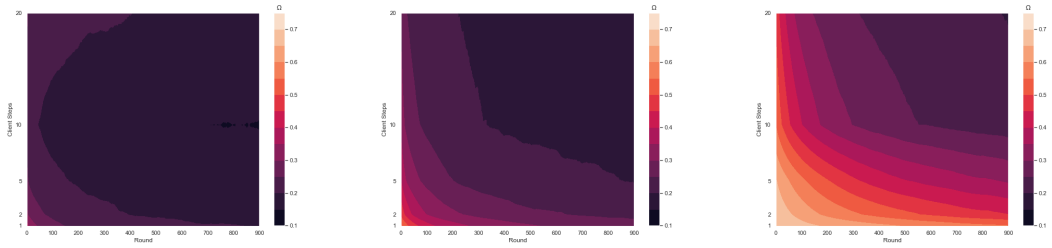
Figure 4.6 highlights that generally, as the learning rate decreases, Ω increases. The main distinction in these sets of experiments is that unlike figure 4.5, as the amount of concept-drift increases, the models actually perform better; indicated by an, on-average, decrease in Ω in figure 4.6.

This at first glance seems counter-intuitive. In one-shot averaging, increasing concept-drift increased Ω (more so than even feature-shift). Also as discussed in chapter 3, as the concept-drift increases, the variance between loss-surfaces should also increase resulting in greater divergence. What we observe, however, is the complete opposite.

This can be attributed to how varying K and ϵ effects the mean of $P_i(\mathcal{X})$ and $P_i(\mathcal{Y}|\mathcal{X})$ differently. Figure 4.4, shows that even though the variance in $P(\mathcal{Y}|\mathcal{X})$ increases as ϵ increases, the mean of the distribution does not change. This is different from figure 4.1, 4.2, and 4.3, where as the number of clusters increase the mean changes arbitrarily.

This would mean that as ϵ increases in $\texttt{SyntheticSeq}(1, \epsilon)$, the training is exposed to a greater variety of data even though the "average" data point stays the same. Hence the model is able to better generalize. This effect can explain why in figure 2.6a, even though σ_1 increases at $\mu_1 = 0$, the model performance does not deteriorate.

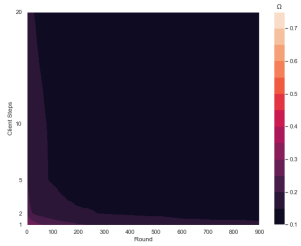
It is also observed there is no upward curving behavior for $\texttt{SyntheticSeq}(1, \epsilon), \epsilon \in [2, 5, 10]$, for $lr = 0.1$ (figure 4.6d, 4.6g and 4.6j) This means that increasing the number of rounds no longer increases the performance of the model. This could be due to the fact that client models have all reached their own minimum, and the number of local steps performed continually bring the average model to the minimum of each client. As such training progressive rounds does not improve the model.



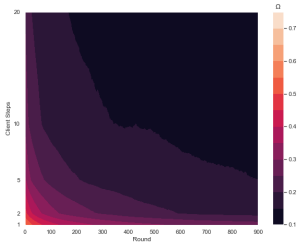
(a) `SyntheticSeq`$(1, 0)$ lr=0.1

(b) `SyntheticSeq`$(1, 0)$ lr=0.01

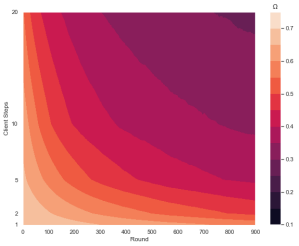
(c) `SyntheticSeq`$(1, 0)$ lr=0.001



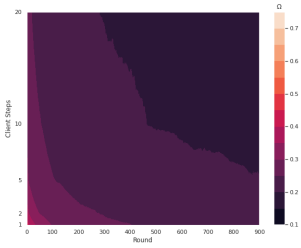
(d) `SyntheticSeq`$(2, 0)$ lr=0.1



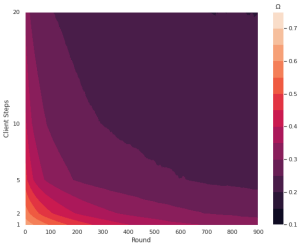
(e) `SyntheticSeq`$(2, 0)$ lr=0.01



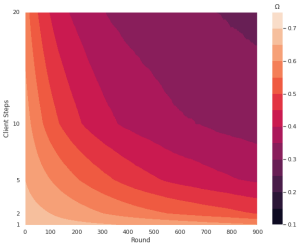
(f) `SyntheticSeq`$(2, 0)$ lr=0.001



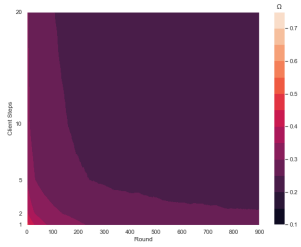
(g) `SyntheticSeq`$(5, 0)$ lr=0.1



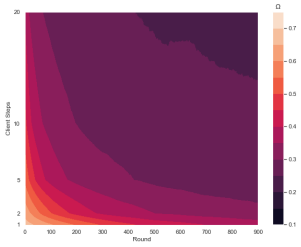
(h) `SyntheticSeq`$(5, 0)$ lr=0.01



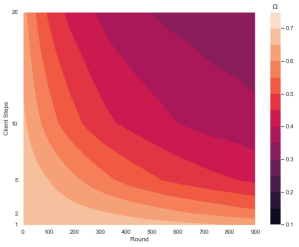
(i) `SyntheticSeq`$(5, 0)$ lr=0.001



(j) `SyntheticSeq`$(10, 0)$ lr=0.1

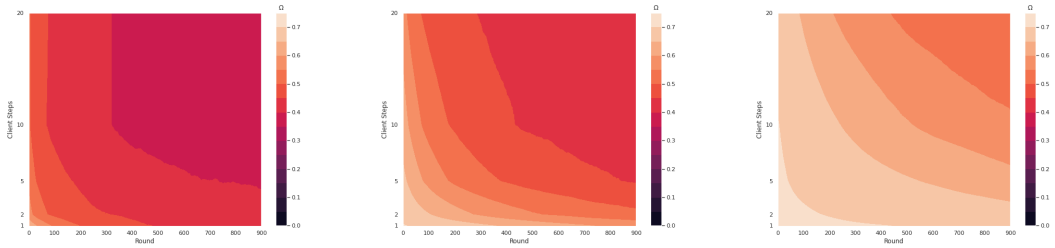


(k) `SyntheticSeq`$(10, 0)$ lr=0.01



(l) `SyntheticSeq`$(10, 0)$ lr=0.001

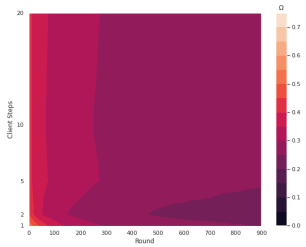
Figure 4.5: Behavior of `FedAvg` on data generated by `SyntheticSeq`$(K, 0)$ for various K



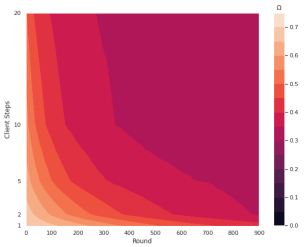
(a) $\texttt{SyntheticSeq}(1, 1)$ lr=0.1

(b) $\texttt{SyntheticSeq}(1, 1)$ lr=0.01

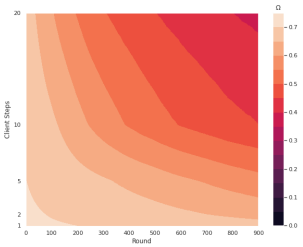
(c) $\texttt{SyntheticSeq}(1, 1)$ lr=0.001



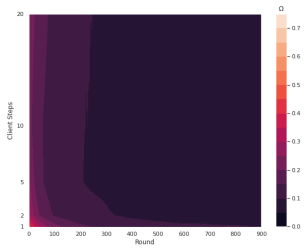
(d) $\texttt{SyntheticSeq}(1, 2)$ lr=0.1



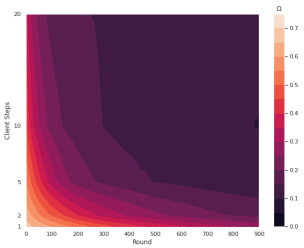
(e) $\texttt{SyntheticSeq}(1, 2)$ lr=0.01



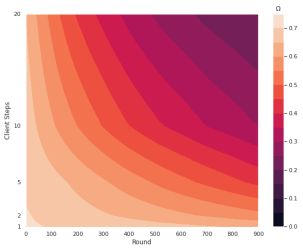
(f) $\texttt{SyntheticSeq}(1, 2)$ lr=0.001



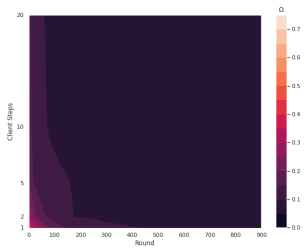
(g) $\texttt{SyntheticSeq}(1, 5)$ lr=0.1



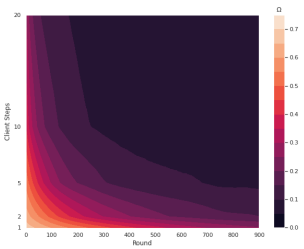
(h) $\texttt{SyntheticSeq}(1, 5)$ lr=0.01



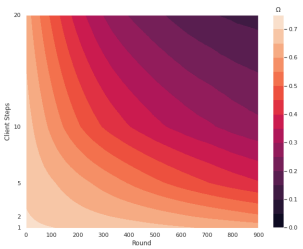
(i) $\texttt{SyntheticSeq}(1, 5)$ lr=0.001



(j) $\texttt{SyntheticSeq}(1, 10)$ lr=0.1



(k) $\texttt{SyntheticSeq}(1, 10)$ lr=0.01



(l) $\texttt{SyntheticSeq}(1, 10)$ lr=0.001

Figure 4.6: Behavior of `FedAvg` on data generated by $\texttt{SyntheticSeq}(1, \epsilon)$ for various ϵ

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In conclusion, this report attempted to build the intuition behind why `FedAvg` observes performance degradation when trained on increasingly NIID data.

To do this NIID data was first defined in section 2.2, and the causes of NIID data were explored in section 2.2.3 and 2.2.4. To then explain why naively averaging models does not work, the impact of NIID data in one-shot averaging was explored in section 2.4. Section 2.4 also demonstrated empirically that in the case of linear regression, increasing feature-shift, and concept-drift alters the loss-surface between clients. This causes a miss-match between the average of the minimum of the client's loss-surfaces and the minimum of the average loss-surface.

Following this section, `FedAvg` is described in section 2.5, and previous experiments in the literature that explore `FedAvg`'s robustness to NIID data are also described. Section 2.6 highlights under what hyperparamters `FedAvg` behaves like one-shot averaging and under what hyperparamters `FedAvg` behaves like centralized gradient descent. However, given the communication assumptions made in section 2.1, section 2.6 explain that these hyperparamters must be tuned carefully.

Chapter 3, gives mathematical rigour to the problem. When the loss-surface is assumed to be L-smooth and when the client gradient variance is bounded by σ, **Theorem** 1 shows that in full-batch `FedAvg` when all client's participates, the divergence caused by one round of `FedAvg` when each client takes two local steps is bounded by $\eta L\sigma$. **Theorem** 2 states that if three steps are taken instead, for one round, this divergence increases to $\eta L\sigma + \eta 2L\sigma(L+2)$. **Theorem** 3 states that if two rounds are taken for two-step `FedAvg`, this bounds increases to $\eta L\sigma + \eta 2L\sigma(\eta L2 + 2L + 1)$. It is hypothesized that divergence under any hyperparamters is proportional to σ.

Chapter 4 presents the behavior of `FedAvg` under `SyntheticSeq`. The results show that when feature-shift is increased by increasing the number of client feature clusters, `FedAvg` performance deteriorates when the amount of NIID-ness increases. However when the concept-drift is increased by increasing the variance of the distribution over $P_i(\mathcal{Y}|\mathcal{X})$ for an unchanging mean, increasing the concept-shift improves `FedAvg` performance. Chapter 4 hence highlights that the specifics of the shape of the distributions of feature-shift and concept-drift might have an effect on `FedAvg` performance.

5.2 Future Work

5.2.1 Formalizing Divergence Generalizations

In the future, I would like extend the domain for which the theorems presented in chapter 3 can be applied to. Specifically following the arguments in section 3.6.1, I would like to derive the divergence bounds for the general case of B batch, E epoch , and T round `FedAvg`.

In addition to this, I would like to add theoretically underpin the empirical exploration in section 2.4. I would like to do this, if possible, by deriving a closed form expression that given the parameters of process 2.12, gives the difference between $\bar{\alpha}$, $\bar{\beta}$ and α^*, β^*.

5.2.2 `FedAvg` on a Larger Space of `SyntheticSeq`(K, ϵ)

Chapter 4, showed that the theory presented in chapters 2 and 3, partially holds when sequential data is generated through a synthetic process. Given how figure 4.5 and 4.6 each summarize 60 experiments each, with some individual experiments taking up to 30 minutes to run, the time-constraint of this project limited the combination of K and ϵ that could be explored. In the future, I would like to see how `FedAvg` performs on `SytheticSeq` when there is a combination of feature-shift and concept-drift.

Additionally, I unexpectedly discovered that perhaps `FedAvg` behaves differently when NIID data is due to client clustering. I want to test this hypothesis by producing synthetic datasets where:

1. $P_i(\mathcal{X})$ and $P_i(\mathcal{Y}|\mathcal{X})$ form clusters
2. Only $P_i(\mathcal{Y}|\mathcal{X})$ forms clusters
3. Neither $P_i(\mathcal{X})$ nor $P_i(\mathcal{Y}|\mathcal{X})$ form clusters

and observing how `FedAvg` behaves on these classes of synthetic datasets.

5.2.3 Reddit Dataset

This report has mainly been concerned with the theoretical implications of NIID data in next-word prediction. In the future I would like to see whether the theory presented in this report stands-up up to real-life datasets. For example I would like to use the methods defined in 2.3, to see what the distributions of $P_i(\mathcal{X})$ and $P_i(\mathcal{Y}|\mathcal{X})$ look for the Reddit comment dataset.[18] I would like to see what feature-shifts and concept-drifts look like in real-life.

I would like to use the properties of these distributions to design experiments that consist of clients with varying amounts of feature-shift and concept-drifts. I would like to observe the behavior of `FedAvg` under these experiments.

Bibliography

[1] Yoshua Bengio et al. "A neural probabilistic language model". In: *Journal of machine learning research* 3.Feb (2003), pp. 1137–1155.

[2] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[3] Ueli Maurer. "Secure multi-party computation made simple". In: *Discrete Applied Mathematics* 154.2 (2006), pp. 370–381.

[4] Craig Gentry. "Fully homomorphic encryption using ideal lattices". In: *Proceedings of the forty-first annual ACM symposium on Theory of computing*. 2009, pp. 169–178.

[5] Jose G Moreno-Torres et al. "A unifying view on dataset shift in classification". In: *Pattern recognition* 45.1 (2012), pp. 521–530.

[6] Yuchen Zhang, John C Duchi, and Martin J Wainwright. "Communication-efficient algorithms for statistical optimization". In: *The Journal of Machine Learning Research* 14.1 (2013), pp. 3321–3363.

[7] Cynthia Dwork, Aaron Roth, et al. "The algorithmic foundations of differential privacy." In: *Foundations and Trends in Theoretical Computer Science* 9.3-4 (2014), pp. 211–407.

[8] Ian J Goodfellow, Oriol Vinyals, and Andrew M Saxe. "Qualitatively characterizing neural network optimization problems". In: *arXiv preprint arXiv:1412.6544* (2014).

[9] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[10] Yossi Arjevani and Ohad Shamir. "Communication complexity of distributed convex learning and optimization". In: *Advances in neural information processing systems*. 2015, pp. 1756–1764.

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

[12] *MATTER OF WARRANT SEARCH CERTAIN E-MAIL*. 2016.

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

[14] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747* (2016).

[15] Apple Differential Privacy Team. *Learning with Privacy at Scale*. https://machinelearning.apple.com/docs/learning-with-privacy-at-scale/appledifferentialprivacysystem.pdf. 2017.

[16] Brendan McMahan and Daniel Ramage. *Federated Learning: Collaborative Machine Learning without Centralized Training Data*. https://ai.googleblog.com/2017/04/federated-learning-collaborative.html. 2017.

[17] Virginia Smith et al. "Federated multi-task learning". In: *Advances in Neural Information Processing Systems*. 2017, pp. 4424–4434.

[18] Sebastian Caldas et al. "Leaf: A benchmark for federated settings". In: *arXiv preprint arXiv:1812.01097* (2018).

[19] *Carpenter v. US*. 2018.

[20] Andrew Hard et al. "Federated learning for mobile keyboard prediction". In: *arXiv preprint arXiv:1811.03604* (2018).

[21] Tian Li et al. "Federated optimization in heterogeneous networks". In: *arXiv preprint arXiv:1812.06127* (2018).

[22] Alex Sherstinsky. "Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network". In: *arXiv preprint arXiv:1808.03314* (2018).

[23] Keith Bonawitz et al. "Towards federated learning at scale: System design". In: *arXiv preprint arXiv:1902.01046* (2019).

[24] Hubert Eichner et al. "Semi-Cyclic Stochastic Gradient Descent". In: *CoRR* abs/1904.10120 (2019). arXiv: 1904.10120. URL: http://arxiv.org/abs/1904.10120.

[25] Yihan Jiang et al. "Improving federated learning personalization via model agnostic meta learning". In: *arXiv preprint arXiv:1909.12488* (2019).

[26] Peter Kairouz et al. *Advances and Open Problems in Federated Learning*. 2019. arXiv: 1912.04977 [cs.LG].

[27] He Li, Lu Yu, and Wu He. *The impact of GDPR on global technology development*. 2019.

[28] Xiang Li et al. "On the convergence of fedavg on non-iid data". In: *arXiv preprint arXiv:1907.02189* (2019).

[29] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[30] Sebastian Caldas. *Leaf: A Benchmark for Federated Settings data/reddit*. 2020. URL: https://github.com/TalwalkarLab/leaf/tree/master/data/reddit.

[31] Marc Peter Deisenroth, A. Aldo Faisal, and Cheng Soon Ong. *Mathematics for Machine Learning*. Cambridge University Press, 2020.

[32] Domo. *DATA NEVER SLEEPS 6.0*. https://www.domo.com/assets/downloads/18_domo_data-never-sleeps-6+verticals.pdf. 2020.

[33] Alireza Fallah, Aryan Mokhtari, and Asuman Ozdaglar. *Personalized Federated Learning: A Meta-Learning Approach*. 2020. arXiv: 2002.07948 [cs.LG].

[34] Tao Yu, Eugene Bagdasaryan, and Vitaly Shmatikov. *Salvaging Federated Learning by Local Adaptation*. 2020. arXiv: 2002.04758 [cs.LG].