Université de Montréal

Re-Weighted Softmax Cross-Entropy to Control Forgetting in Federated Learning

par

Gwendolyne Legate

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté en vue de l'obtention du grade de
Maître ès sciences (M.Sc.)
en Discipline

December 31, 2022

Université de Montréal

Faculté des arts et des sciences

Ce mémoire intitulé

Re-Weighted Softmax Cross-Entropy to Control Forgetting in Federated Learning

présenté par

Gwendolyne Legate

a été évalué par un jury composé des personnes suivantes :

Irina Rish
(président-rapporteur)

Eugene Belilovsky
(directeur de recherche)

Guy Wolf
(membre du jury)

Résumé

Dans l'apprentissage fédéré, un modèle global est appris en agrégeant les mises à jour du modèle calculées à partir d'un ensemble de nœuds clients, un défi clé dans ce domaine est l'hétérogénéité des données entre les clients qui dégrade les performances du modèle. Les algorithmes d'apprentissage fédéré standard effectuent plusieurs étapes de gradient avant de synchroniser le modèle, ce qui peut amener les clients à minimiser exagérément leur propre objectif local et à s'écarter de la solution globale. Nous démontrons que dans un tel contexte, les modèles de clients individuels subissent un oubli catastrophique par rapport aux données d'autres clients et nous proposons une approche simple mais efficace qui modifie l'objectif d'entropie croisée sur une base par client en repondérant le softmax de les logits avant de calculer la perte. Cette approche protège les classes en dehors de l'ensemble d'étiquettes d'un client d'un changement de représentation brutal. Grâce à une évaluation empirique approfondie, nous démontrons que notre approche peut atténuer ce problème, en apportant une amélioration continue aux algorithmes d'apprentissage fédéré standard. Cette approche est particulièrement avantageux dans les contextes d'apprentissage fédéré difficiles les plus étroitement alignés sur les scénarios du monde réel où l'hétérogénéité des données est élevée et la participation des clients à chaque cycle est faible. Nous étudions également les effets de l'utilisation de la normalisation par lots et de la normalisation de groupe avec notre méthode et constatons que la normalisation par lots, qui était auparavant considérée comme préjudiciable à l'apprentissage fédéré, fonctionne exceptionnellement bien avec notre softmax repondéré, remettant en question certaines hypothèses antérieures sur la normalisation dans un système fédéré.

Mots clés : Apprentissage fédéré, dérive du client, généralisation hors distribution, oubli catastrophique

Abstract

In Federated Learning, a global model is learned by aggregating model updates computed from a set of client nodes, a key challenge in this domain is data heterogeneity across clients which degrades model performance. Standard federated learning algorithms perform multiple gradient steps before synchronizing the model which can lead to clients overly minimizing their own local objective and diverging from the global solution. We demonstrate that in such a setting, individual client models experience a catastrophic forgetting with respect to data from other clients and we propose a simple yet efficient approach that modifies the cross-entropy objective on a per-client basis by re-weighting the softmax of the logits prior to computing the loss. This approach shields classes outside a client's label set from abrupt representation change. Through extensive empirical evaluation, we demonstrate our approach can alleviate this problem, providing consistent improvement to standard federated learning algorithms. It is particularly beneficial under the challenging federated learning settings most closely aligned with real world scenarios where data heterogeneity is high and client participation in each round is low. We also investigate the effects of using batch normalization and group normalization with our method and find that batch normalization which has previously been considered detrimental to federated learning performs particularly well with our re-weighted softmax, calling into question some prior assumptions about normalization in a federated setting.

Keywords: Federated Learning, Client Drift, Out of Distribution Generalization, Catastrophic Forgetting

Contents

List of tables

List of figures

List of acronyms and abbreviations

BN	*Batch normalization*
CE	*Cross entropy*
CIFAR	*Canadian Institute for Advanced Research*
CL	*Continual learning*
EWC	*Elastic weight consolidation*
FL	*Federated learning*
GD	*Gradient descent*
GN	*Group normalization*
i.i.d.	*Independent and identically distributed*
IoT	*Internet of things*
ML	*Machine learning*

OOD	*Out of distribution*
SCG	*Stochastic gradient descent*
WSM	*Re-weighted softmax*

Acknowledgements

During my masters, I have received a lot of valuable support and guidance from a number of people that I would like to take a moment to thank here. First of all, I would like to thank my supervisor, Dr. Eugene Belilovky for agreeing to supervise me and always being so accessible whenever I had questions or wanted to discuss my results. I look forward to continuing to grow as a researcher wit his help as I continue in my studies.

I would also like to thank my collaborator Lucas Caccia, who devoted his time to understanding aspects of the research outside of his domain and offered many insightful comments, suggestions and valuable coding skills. It was wonderful to meet and work with you on this project.

Finally I would like to thank my family and friends for supporting my decision to pursue a masters degree, you have all provided encouragement and much needed diversions during this process. In particular my brother (Sebastien) who always makes an effort to link our academic interests so we can happily discuss on common ground and my boyfriend (Marc) who takes walks with me when I need a break and always brings me food or coffee if I haven't surfaced in several hours.

Introduction

Machine learning (ML), is a branch of artificial intelligence that uses data and specialized algorithms to iteratively learn how to perform tasks of interest. Data collection and availability has increased rapidly in the recent decade and often a single computation node is infeasible to store all the required data and optimize the machine learning objective functions necessary to derive insight from it and as we continue to scale ML operations, the use of a distributed computational frameworks becomes increasingly necessary. In distributed learning, training data is distributed across a number of interconnected nodes and the optimization problem is solved collectively by the cluster of nodes. Federated Learning (FL) is distributed machine learning paradigm introduced by Google [**42**] that has become a popular choice for distributed learning due to its communication efficiency and concern for user privacy.

In FL a shared global model is learned from decentralized data located at a number of independent client nodes [**42, 28, 36, 25**]. Motivated by communication constraints, FL algorithms typically perform a number local gradient update steps before synchronizing with the global model [**42**]. This reduced communication strategy is very effective under independent and identically distributed (i.i.d.) settings but data inhomogeneity across clients has direct implications on the convergence of FL algorithms and its performance degrades on clients without i.i.d. data distributions [**71**]. [**26**] show that non i.i.d. conditions frequently induce client drift, a phenomenon in which clients progress too far towards optimizing their own local objective, leading to a solution that has severely "drifted" from an optimal global solution. Under realistic settings, client data will often have non-i.i.d. distributions so federated learning algorithms that are able to address the challenges of convergence and client drift are an important research direction. Prominent examples of approaches tackling heterogeneous data in a federated learning setting include but are not limited to [**37, 26, 36, 39, 59**].

Continual learning is another emerging paradigm in which a model is trained on a number of tasks sequentially. The learner needs to learn each new task without forgetting

knowledge obtained from the preceding tasks. When learning new information a continual learning model has a tendency to forget previously learned information. This phenomenon, termed catastrophic forgetting [**41**] is typically a focus of study in continual learning literature. Similar to FL, data heterogeneity in continual learning presents a challenge since different tasks typically contain data drawn from different underlying distributions. We can draw a connection between the continual learning problem of catastrophic forgetting and the client drift problem in federated learning. We consider one round of federated learning in which C random clients are selected and initialized with a copy of the current global model. Each client performs a pre-determined number of local update steps to optimize the objective on their local data. A round ends with an update to the global model achieved by aggregating the updates from each client. At the beginning of a round, the clients receive a model previously derived from training on other clients data. As local training proceeds, the model becomes increasingly biased towards a given client and as discussed in [**31, 18**], this can cause client models to rely on spurious correlations to improve their in-distribution performances creating a situation in which local models to experience a catastrophic forgetting with respect to data (drawn from distinctly different distributions) of the other clients. Naturally, aggregating models that have deviated from a joint solution will lead to degraded results with respect to the global objective. We denote this problem as *local client forgetting* and it is covered in greater depth in section 2.1. The ability to achieve a reduction in local client forgetting would moderate the increase in loss with respect to other clients data at individual client models. This would increase the ability of local models to generalize to out of distribution (OOD) data and improve the loss of individual models over the combined data. Therefore we propose to reduce client drift by tackling local client forgetting.

There are numerous approaches to tackle catastrophic forgetting in the continual learning literature [**27, 38, 12, 53, 14**] but we find them to be largely impractical in the FL setting. Experience Replay methods [**12**] require access to other clients data, violating the primordial data communication constraints of FL. Similar concerns exist for many regularization methods such as elastic weight consolidation (EWC) [**27**] which require communicating additional information. Regularization methods can additionally require many steps to converge due to the additional conflicting objectives [**5**] and this computational constraint can hurt convergence of the FL algorithm, a key desiderata. For the supervised continual learning setting [**10, 3**] proposed a modification of the standard cross entropy objective function that truncates the softmax denominator, removing terms corresponding to classes from old tasks. This simple approach mitigates catastrophic forgetting by reducing the bias on the model to avoid predicting old classes.

Inspired by the parallels between client drift in FL and catastrophic forgetting in the

continual learning case, we propose an adaptation of the CL method from [**10, 3**] to modify the loss function of each client based on its class distribution using a re-weighted softmax. We will empirically demonstrate this approach can drastically reduce client level forgetting in the heterogeneous setting leading to substantially improved overall global model convergence and final performance. We apply the re-weighted softmax to baseline methods FedAvg [**42**], SCAFFOLD [**26**] and FedProx [**36**] demonstrating performance improvements in all cases.

Main Contributions

This thesis addresses the problem of local client forgetting for clients with heterogeneous data distributions in a federated learning setting. Our main findings and contributions are summarized as follows:

- We link the concepts of client drift in federated learning with catastrophic forgetting in continual learning and contextualize the challenge of federated learning in an i.i.d. setting as local client forgetting.
- We develop the re-weighted softmax (WSM) as a means to tackle local client forgetting. It directly tackles the problem of distribution shifts between clients by weighting class labels present in the local distributions according to the proportions of labels present in the local dataset.
- We show empirically that WSM is effective in significantly improving performance outcomes for federated learning models across a broad range of hyper parameters and under different federated learning conditions such as different fractions of clients participating in each round or for different scales of data heterogeneity.
- We investigate the role of different normalization methods on FedAvg and FedAvg+WSM by performing experiments in which we use only batch normalization (BN) and only group normalization (GN).
- We demonstrate that WSM can easily be added to different federated learning algorithms from the literature and that it provides consistent improvement to these algorithms.

Outline

The thesis is organized as follows. Chapter 1 provides the background information required to understand the fundamentals of topics related to this work. It covers distributed learning, federated learning, continual learning and out of distribution data, providing the motivation behind the development of each field and open research areas within them. Chapter 2 addresses the formulation of WSM as well as the concept of local client forgetting that

motivates it. We describe a method of evaluating local client forgetting in a federated setting and demonstrate the success of WSM in addressing it. Chapter 3 contains the results of our experiments using WSM. There are four subsets of experiments presented:

- a direct comparison of WSM with FedAvg in which we show that WSM has a positive effect across a variety of hyperparameters
- an ablation study in which we ablate one parameter at a time to evaluate it's effect with and without WSM
- we perform a series of experiments investigating the impact of using batch normalization and group normalization in the federated setting with and without WSM.
- we apply WSM to different federated learning algorithms from the literature demonstrating its widespread effectiveness and ease of application.

Working Paper

Chapters 2 and 3 of this thesis are based on the working paper "Re-Weighted Softmax Cross-Entropy to Control Forgetting in Federated Learning", which is currently under review for the Conference on Computer Vision and Pattern Recognition (CVPR) 2023. As the first author of the paper, I contributed to literature review, implementation, experiments, and paper writing. An earlier version of this work was presented at the Neurips 2022 Dist-shift workshop [**30**].

Chapter 1

Background and Related Work

Machine learning is a branch of artificial intelligence that enables systems to learn and improve from information/experience without being explicitly programmed. Typical subsets of machine learning include the following:

- Supervised Learning: the ML system is provided with a set of labeled data and since the desired outcome is know, the model recieves feedback on it's predictions and adjusts its weights appropriately until it has converged to a solution
- Unsupervised Learning: The ML system is given a set of unlabeled data with no predefined output expectations. The algorithm uses the data to derive patterns and underlying similarities that lead to insights about the provided data.
- Reinforcement Learning: The ML system operates in a virtual environment where a reward is provided for actions taken to achieve a goal.

At the core of ML is the use of large sets of training data which allows ML models to search for and model patterns within the data. Given a set of n input/output pairs $x \in \mathbb{R}^d$ and $y \in \mathbb{R}$, respectively and a loss function $f(w) = \ell(x_i, y_i, w)$ where w are the parameters of the model, a finite sum objective function is defined as follows:

$$\min_{w \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} f_i(w) \tag{1}$$

Equation 1 is a general formulation that can be applied to a range of machine learning problems such as linear or logistic regressions, support vector machines or more complicated, non-convex models including neural networks [**28**]. Model parameters are learned by optimizing the objective function by minimizing the error function. Because the error function, $\ell(x_i, y_i, w)$ is a continuous function of w, its smallest value will occur at a point in weight space such that norm of the gradient of the loss function vanishes $\rightarrow ||\nabla \ell(x_i, y_i, w)|| = 0$

[**8**]. Gradient descent(GD) [**44**] is an optimization algorithm used to train machine learning models by iteratively finding a solution to equation 1 (sub-gradient descent is used for non smooth functions). This is done using equation 2 where the parameter η is the learning rate that governs the step size GD will take in the optimal direction for optimization and t is a given update round.

$$w^{t+1} = w^t - \eta \nabla f(w_t) \tag{2}$$

Since the number of samples n is typically very large, GD is an impractical optimization method and stochastic gradient descent(SGD) [**51**] is used instead. In SGD, a sample or a subset of samples $i \in \{1, ..., n\}$ is chosen uniformly at random and the gradient is calculated using equation 3.

$$w^{t+1} = w^t - \eta \nabla f_i(w_t) \tag{3}$$

This chapter will provide the fundamentals required to understand the core concepts covered in this thesis including distributed learning and federated learning (a subset of distributed learning central to this work). Within federated learning, a particular area of focus will be data heterogeneity between clients along with the related topics of catastrophic forgetting from continual learning, distribution shifts and out of distribution generalization. Some specific FL algorithms developed to address client heterogeneity will be given more in depth coverage and the concepts of batch normalization and group normalization which are relevant to a subset of our experiments will also have their fundamentals covered.

1.1. Distributed Learning

With the increased availability of data in the last decade, new challenges have arisen in machine learning regarding the scalability and efficiency of optimizing models with respect to available computational and memory resources [**46**]. One of the most promising avenues for scaling up storage and compute resources for large-scale learning is distributed learning in which the training data is stored in a distributed fashion across a number of interconnected nodes and we solve the objective function in equation 1 for distributed learning. This can include a range of scenarios from just two different RAM/computer/nodes, to a situation in which data are distributed across multiple data centers around the world, and across many nodes in those data centers. In the system described above, no single processing unit will have direct access to all the data and the optimization problem will then be solved collectively by the cluster of nodes [**28**].

New computational challenges arise in a distributed learning situation, including difficulty with optimization procedures and the added practical consideration of the high cost of sharing and communicating parameters across multiple nodes. To illustrate the additional complexity inherent in distributed learning, section 1.1.1 outlines the structure of a distributed learning system and illustrates how gradient descent would proceed under such conditions. Additionally, in much of the distributed learning literature data is assumed to be evenly distributed, with an assumption that each node possesses an i.i.d. sample from the underlying distribution. [**28, 42**] indicate these assumptions are often too strong and bring attention to a sub domain of distributed learning which they call federated learning (described in section 1.2) in which none of these assumptions hold.

1.1.1. Distributed Stochastic Gradient Descent

As outlined at the beginning of section 1, a classical supervised ML problem consists of optimizing an objective function such as equation 1. This function is optimized incrementally over many epochs to minimize the error of prediction using SGD (equation 3). Distributed SGD has an added layer of complexity due to the distributed nature of the computations, the system consists of server nodes and worker nodes with different responsibilities in the system [**32**]. The training data is partitioned among all of the worker nodes and at each training iteration the workers independently compute the the gradient using their own training data. Since an individual worker's update reflects only its own training on a subset of the data, the subgradients calculated at each client are sent to the server nodes which aggregate all of them before applying an update to the model parameters. Algorithm 1 outlines the steps taken at each node involved in the computation where $\Omega(\cdot)$ is a regularizer that penalizes high model complexity. Figure 1.1 demonstrates the algorithm pictorially for additional clarity.

In algorithm 1, the most computationally expensive task is the calculation of the gradient over all the data so it is distributed between the worker nodes to lighten the load. Workers compute $w^T x_{ik}$ so for very high dimensional models this computation would be infeasible on any single node. However, one of the benefits of distributed learning is that each node works with only a subset of the data and that worker only needs the weights for which features of their training set are relevant when making a prediction [**32**]

1.1.2. Parameter Server

To address some of the challenges inherent in distributed learning, we introduce the concept of the parameter server. The original ideas behind the parameter server were proposed by

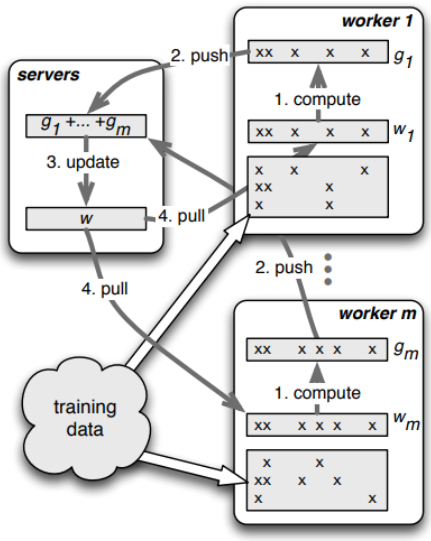

Fig. 1.1. Steps required to perform distributed subgradient descent [**32**]

[**57**] followed by a complete description of the system and an API by [**32**]. Parameter server has the following requirements

- Efficient communication: An asynchronous task model and API that can reduce the overall network bandwidth for ML algorithms
- Flexible consistency models: Relaxed consistency helps reduce the cost of synchronization and offers developers the choice between algorithmic convergence and system performance.
- Elasticity for adding resources: Can add more capacity without restarting a computation
- Efficient Fault tolerance: Given high rate of failures and large amounts of data, recovery of tasks is possible if failures are not catastrophic
- Ease of use: API to supports ML constructs such as sparse vectors, matrices or tensors.

Parameter Server [**32**], whose architecture is shown in figure 1.2, has nodes that are grouped into a server group and several worker groups. Server nodes facilitate the running of multiple algorithms and each node in the server group is responsible for a partition of the globally shared parameters. Servers can communicate with each other to migrate or replicate parameters for scalability and availability. A server manager node is responsible for maintaining the consistent view of the servers, it monitors node liveness and assigns parameter partitions to each server node. Each worker group is typically assigned to an application and will

Algorithm 1 Distributed Subgradient Descent [**32**]

Task Scheduler:

issue LoadData() to all workers
for iteration, t = 0, 1,..., T **do**
  issue WorkerIterate(t) to all workers
end for

Worker r = {1, ...m}:

function LoadData()
  load a part of training data $\{y_{ik}, x_{ik}\}_{k=1}^{n_r}$
  pull the working set $w_r^{(0)}$ from servers
end function

function WorkerIterate(t)
  gradient $g_r^{(t)} \leftarrow \sum_{k=1}^{n_r} \partial \ell(y_{ik}, x_{ik}, w_r^{(t)})$
  push $g_r^{(t)}$ to servers
  pull $w_r^{(t+1)}$ from servers
end function

Servers

function ServerIterate(t)
  aggregate $g^{(t)} \leftarrow \sum_{r=1}^{m} g_r^{(t)}$
  $w^{t+1} \leftarrow w^{(t)} - \eta(g^{(t)} + \partial\Omega(w^{(t)})$
end function

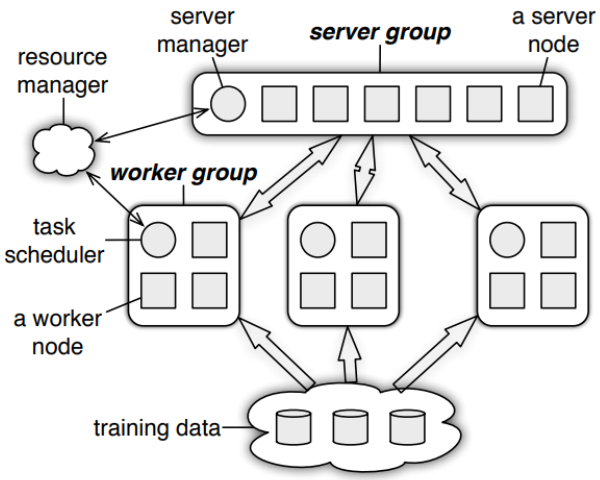

Fig. 1.2. Architecture of a parameter server communicating with several groups of workers.

communicate with the sever groups for pulling of parameters and pushing of gradients as mentioned in section 1.1.1 but they do not communicate amongst themselves. A scheduler

node exists for each worker group, it assigns tasks, monitors progress and reassigns tasks if workers are added or removed from the group. Generally the same worker node leverages data stored locally by running iterative algorithms on the same dataset. Parameter namespaces can be used for parallelizing work further among multiple worker groups. In addition, same parameter namespace can be shared among multiple groups if additional resources are required. The objective of the parameter server is to simplify distributed machine learning. The shared parameters are stored as key, value pairs and distributed across a group of server nodes. Any node has the capability to push its local parameters and pull parameters from remote nodes. Tasks are designed to be asynchronous and run in parallel.

1.1.3. Distributed Learning vs. Federated Learning

Federated learning is a subset of distributed learning and appears very similar at first glance. The parameter server framework detailed in section 1.1.2, is a typical feature in distributed machine learning, it stores data on distributed worker nodes that perform the bulk of the computations and allocates data and computing resources through a central scheduling node that controls all of these decisions. In federated learning the client node has complete autonomy over its local data and decision to share its model updates or not and with what frequency [**42, 28**]. This additional autonomy of the client nodes makes the federated learning setting more complex [**66**], federated learning also places a strong emphasis on data privacy at the client nodes creating additional constraints not required for distributed learning in general.

Another feature of FL is that data is massively distributed, so the data is stored across a much larger number of nodes than it typically the case with distributed learning. With FL, the number of nodes participating in an update round can be much larger than than the average amount of data stored at each node [**28**] whereas with distributed learning data is partitioned by the scheduling node to take maximum advantage of available resources so fewer nodes handle more data [**32**]. Logistically, in distributed learning, nodes will be able to communicate relatively quickly and reliably unlike in the case of FL in which some clients may suffer from weak connections, slow networks or dead batteries [**42**].

1.2. Federated Learning

Federated learning (FL) is a distributed learning paradigm that aims to preserve privacy by leaving training data distributed on individual nodes, and learning a shared model by aggregating updates computed locally. Different FL architectures exist for data structures across nodes [**66**] with the most typically used architecture being the one described above, for a horizontal FL system. A typical assumption is that FL clients are honest and the server

is honest-but-curious so no information leakage from participants to the server is permitted [**66**]. Optimization in a federated learning setting has some key properties, outlined below, that distinguish it from the typical distributed optimization problem [**28, 42**]

- Data is massively distributed, meaning the number of nodes can be very large and the average number of training examples stored per node can be comparatively small
- Data at each node may be drawn from a different distribution than the distribution that exists over the complete dataset from all nodes K and the number of samples at each node may also vary considerably between nodes
- Communication may be limited or sporadic since mobile devices are sometimes offline or on slow or expensive connections.

For federated learning, training data is distributed and optimization occurs over K clients with each client $k \in 1, ..., K$ having data $\mathbf{X}_k$ with number of samples n_k, drawn from distribution D_k. We define the total number of samples across all clients as $n = \sum_{k=1}^{K} n_k$. The data $\mathbf{X}_k$ at each node may be drawn from different distributions and/or may be unbalanced with some clients possessing more training samples than others. We modify the finite sum objective given in equation 1 to give the typical objective function for federated optimization in equation 4.

$$\min_{\mathbf{w} \in \mathbb{R}^d} \sum_{k=1}^{K} \frac{n_k}{n} \mathcal{L}(\mathbf{w}, \mathbf{X}_k), \tag{4}$$

with $\mathcal{L}(\mathbf{w}, \mathbf{X}_k)$ measuring client k's local objective, and $\mathbf{w}$ representing the global model parameters.

In local SGD [**58**] each participating device performs a single local SGD step and then communicates their updated local models back to the server. Since the communication cost between two nodes is orders of magnitude larger than the cost between processor and memory on the same node, communication efficiency in federated learning of upmost importance [**28**]. The most commonly used baseline in federated learning is the FedAvg algorithm proposed by [**42**] which is a generalization of local SGD. FedAvg reduces communication costs by allowing clients to train multiple iterations successively by performing a number of epochs of local SGD on each client before sending the locally updated model weights to a central server for averaging. There are many possible variations of FL algorithms but in general, they follow a similar structure to FedAvg which proceeds as follows:

- **client selection:** for a set of K clients, $K * C$ are selected at each round $\{t_i\}_{i=1}^{T}$, where $0 < C \leq 1$ is a pre-determined fraction.

- **client updates:** At the beginning of round, client models are initialized with the current weights of the server model. Each client selected for the round performs E local iterations of SGD.
- **server update:** The weights of the individual client models are aggregated to form an update to the shared global model.

The steps in the FedAvg algorithm are outlined explicitly in algorithm 2.

Algorithm 2 FedAVG [**42**]

Server Executes:
Initialize w_0
for each server epoch, t = 1,2,3,... **do**
$m \leftarrow \max(C \cdot K, 1)$
$\mathcal{S}_t \leftarrow$ random set of m clients
for $k \in \mathcal{S}_t$ in parallel **do**
$w_{t+1}^k = \text{ClientUpdate}(w_t)$
end for
$w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$
end for

ClientUpdate(w):
Initialize $w_0 = w$
$\mathcal{B} \leftarrow$ split client data into batches of size B
for each local client iteration, $i \in \{1, 2, .., E\}$ **do**
for $b \in \mathcal{B}$ **do**
$w \leftarrow \eta \nabla \ell(w, b)$
end for
end for
return w to server

1.2.1. Applications and Challenges in Federated Learning

Federated learning has broad applications including learning from personal devices such as smart phones which possess a wealth of data that can assist in training text prediction models or facial recognition software [**42, 35**]. Organizations or institutions can also be viewed as clients in a FL system, hospitals are examples of organizations that contain a large amount of patient data which has major potential benefits for applications such as finding tumors in medical imaging or genome sequencing [**50**]. Modern IoT networks, such as wearable devices, autonomous vehicles, or smart homes contain numerous sensors that allow them to collect, react, and adapt to incoming data in real-time. In this case data like traffic conditions from autonomous vehicles or information from home monitoring devices can

help to train models that will benefit users in everyday tasks [**35**]. In all of the previously mentioned scenarios, privacy is an important consideration. Hospitals operate under strict privacy practices, and can face strong legal, administrative, or ethical constraints that require data to remain local. While privacy is more relaxed in the other two contexts, smartphone and IoT users may also be sensitive about sharing their data in order to protect their personal privacy or to save limited bandwidth/battery power of their phones. A key focus of FL are its ability to train on massively distributed data and privacy advantages compared to distributed learning which motivate the need to keep each clients raw data private. As such FL offers a promising solution for these situations since it can train models or greater volumes of data and facilitate private learning between devices or organizations. However, recent work has indicated that sharing information such as model updates which are required in the training process can also leak sensitive user information [**7, 43**]. Privacy in a federated setting is challenging for existing privacy preserving algorithms such as differential privacy [**16**]. FL requires rigorous privacy guarantees, computationally cheap methods that are also communication-efficient, and tolerant to dropped devices. All of this must be achieved without compromising the accuracy of the training method [**35**]

Convergence of FedAvg has been widely studied for both i.i.d [**58, 61, 49**] and non i.i.d settings [**36, 17, 48, 62, 69**]. Convergence guarantees can be difficult to achieve for FL since a number of assumptions are made that do not necessarily hold in a federated setting. In the i.i.d. case results often rely on the premise that each client node is a copy of the same stochastic process, which is not the case in typical federated settings [**35**]. For the non i.i.d. case, simplifying assumptions such as convexity for [**62**] or uniformly bounded gradients for [**69**] are made in order to achieve convergence guarantees. Heterogeneous data settings offer additional challenges, convergence deteriorates as a function of increasing heterogeneity [**21, 71**] and [**36**] show that when data is not distributed i.i.d. across clients networks can even begin to diverge.

1.2.2. Distribution Shifts in Federated Learning

The goal of an ML model is to learn the underlying distribution of the data it is trained on with the goal of leveraging this learned distribution to generate accurate predictions for unseen data. Traditionally the test data used to evaluate a model during development comes from a stationary distribution with the same underlying distribution as the training data.[**22**] define three types of distribution shift: covariate shift, label shift and concept drift. Given a traditional, supervised ML model, a set of inputs X and a set of outputs Y, the joint distribution from which we want the model to learn can be defined as $\mathcal{P}(X, Y)$. The joint

distribution can be decomposed in two ways:

$$\mathcal{P}(X, Y) = \mathcal{P}(X|Y)\mathcal{P}(Y) \tag{5}$$

$$\mathcal{P}(X, Y) = \mathcal{P}(Y|X)\mathcal{P}(X) \tag{6}$$

Formally, covariate shift is when $\mathcal{P}(X)$ changes but $\mathcal{P}(Y|X)$ does not, this occurs when feature distributions differ across parties. Concept drift occurrs when $\mathcal{P}(Y|X)$ changes but $\mathcal{P}(X)$ stays the same, in such a situation the distribution of features remains the same but the relation of those features to their labels has shifted. Both of these shifts refer to decomposition 6. Label shift which refers to 5 is when $\mathcal{P}(Y)$ changes but $\mathcal{P}(X|Y)$ does not, this occurs when label distributions differ across parties [**33, 22**].

In a federated learning setting, one significant challenge encountered when training on decentralized data is heterogeneity of samples across clients [**25, 34**]. Partitions contain data generated under different conditions which can reasonably be expected to create different local distributions at each client. For the case of supervised multi-class classification, which is the focus of this thesis, users may frequently be missing data from an entire class or multiple classes of the global underlying distribution. For example, smartphones containing images of sailboats will be more concentrated in coastal regions than in desert regions. This type of distribution shift corresponds to a label shift between clients which is our focus in this work. During FL training, data at each client are sampled from these local distributions, creating different local objectives. When clients progress too far towards minimizing their own objective, local models drift from one another, degrading the performance of the shared global model and slowing down convergence [**67, 37, 26**]. Performing several local client optimization steps at a time saves on communication costs which is crucial for large scale federated learning systems [**28, 42**] therefore creating an environment in which each client is able to generalize well to the unseen distributions of the others is a desirable goal. Unfortunately many machine learning models do not generalize well to data that is not drawn i.i.d. from the training set distribution. This problem is known as out of distribution generalization [**54**]. A clients local data will be biased and the local models trained on this data may be relying on spurious correlations to improve their performance [**18, 4**]. In order to achieve good generalization on unseen OOD data, the classifier must model invariant mechanisms, shared across environments [**45**] these features are difficult to identify and model while spurious correlations are much easier to spot and model but are detrimental to generalization. In the FL environment, an algorithm capable of OOD generalization will learn these invariant mechanisms while ignoring spurious correlations. Local models that achieve better generalization will also reduce client drift.

Several attempts have been made to alleviate the problem of client drift through various methods. One approach centers around knowledge distillation to regulate local training, [**72**] and [**39**] ensemble information about the global data distribution and disseminate it to clients via additional models trained at the server. These methods possess the added risk of privacy attacks and while [**72**] take steps to mitigate this risk, their method requires the existence of an unlabeled dataset, which may not be available in all settings. Other approaches attempt to constrain gradient updates from the clients to reduce the impact of client drift. FedProx [**36**] adds a proximal term to the local objective to limit the impact of variation in local updates. This term is weighted with an additional hyperparameter, μ which must be tuned appropriately. [**26**] propose SCAFFOLD, an algorithm to control client drift using control norms to modify client gradients. The control norms estimate the drift at each client use that estimate to correct the local updates. [**1**] also estimate client drift for each client but they do this at the server and corrects the server updates thus avoiding SCAFFOLD's use control norms and saving on the inherent communication burden. The general strategy for each of these methods is similar. They attempt to estimate client drift using gradient updates and then constrain the updates to reduce the drift. SCAFFOLD has additionally been shown to have unstable accuracy and has twice the communication burden of the other algorithms due to its use of control norms required for each client as well as the global model [**33**]. Our proposed method modifies the objective functions locally, creating no additional communication burdens and introducing no additional hyperparameters. It also directly tackles the problem of the underlying distribution shift unlike the other methods mentioned which attempt to address client heterogeneity through constrained gradient optimization and/or knowledge distillation. [**59**] propose a gradient masking technique that modifies the aggregation of updates on the server side. This method does directly tackle the problem of distribution shift and while it has been shown to be effective at achieving better generalization on non i.i.d. data, our re-weighted softmax is able to take advantage of the structure of the commonly used cross-entropy loss making it simpler to implement. Additionally, since we modify the objective functions locally, the re-weighted softmax method is compatible with any federated optimization method in the literature.

[**15**] propose HeteroFL which challenges the notion that FL models at the server and all of the client nodes need to be the same. As part of this work they propose a method very similar to ours called masked cross entropy. In masked cross entropy the final layer outputs not associated with labels existing the in the client distribution are masked for each client. Masked cross entropy is also applied by [**19**] in their work building off of the previous work done in HeteroFL. While the methods are very similar, WSM proposes a masking of the CE loss which offers greater flexibility in label customization at the clients. Additionally,

masked cross entropy was not the focus of HeteroFL and therefore lacks the focus on studying its behavior under different FL conditions and applied to different algorithms that our work on WSM offers.

1.2.3. Heterogeneous Federated Learning Algorithms

Since both SCAFFOLD and FedProx are used as comparison baselines in section 3.5, they are covered here in greater depth than other algorithms mentioned previously.

FedProx

FedProx [**36**] improves model performance with i.i.d. data over FedAvg. This is achieved by limiting the size of local updates by introducing an additional L2 regularization term weighted by hyperparameter μ to its local objective functions. This has the effect of limiting the distance between the local model and the global model. FedProx must be tuned carefully to achieve good model performance, the choice of μ can have a significant effect on training outcomes since a small μ drastically reduces the effect of the regularization term and a large μ causes very small local updates which can greatly slow down convergence. FedProx does require additional computation when compared with FedAvg but its communication requirements remain the same.

SCAFFOLD

SCAFFOLD [**26**] considers that non i.i.d. data introduces variance to the learning situation and thus introduces control norms for both the global and the local models to control the induction of variance. The control norms are used to estimate the update directions of the global model and each client and client drift is estimated as the difference between the global model and each client. There are two approaches proposed to update the control norms. Either the gradient of the local data is re-computed at the server or by the previously computed gradients at the client can be reused. The second approach has a lower computational overhead but the first is more stable. Compared to FedAvg, SCAFFOLD doubles the communication per round due to the control norms which must be communicated at every round and it requires greater computational overhead due to the control norm updates required at every round.

1.2.4. Normalization Methods in Federated Learning

The goal of normalization is to transform features so they exist on a similar scale this results in improved performance, faster convergence and better overall stability when training [**56**]. When training a neural network, weight parameters are adjusted iteratively in an attempt to converge to an optimal solution. The modification of parameters in previous

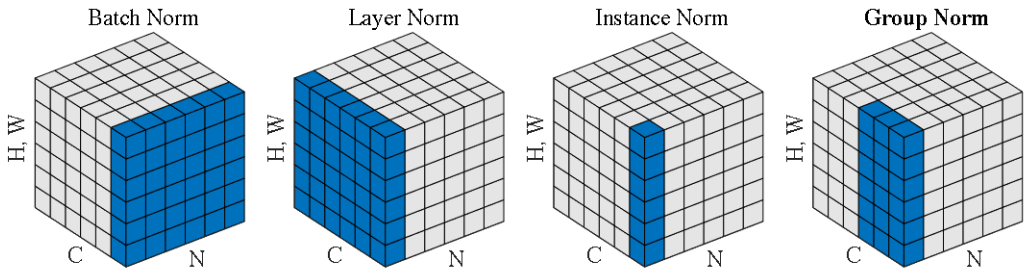

Fig. 1.3. This figure from [**63**] shows feature map tensors, with N as the batch axis and C as the channel axis and W and H are the width and height of your input. For each subfigure, the blue sections are normalized by the same mean and variance showing the differences between the normalization techniques.

layers has the effect of changing the distribution of the following layer inputs, [**23**] refer to this phenomenon as internal covariate shift. Internal covariate shift slows down training by requiring lower learning rates and careful parameter initialization it also makes some models difficult to train. The most commonly used normalization methods for neural networks are batch normalization (BN) [**23**] and group normalization (GN) [**63**]. Other options include layer normalization [**6**] which estimates normalization statistics directly from the summed inputs to the neurons in a hidden layer and instance normalization [**60**] which is similar to layer norm except that it normalizes across each channel in each training example instead of summed inputs to the hidden layer. Both layer norm and instance norm have limited success in visual recognition tasks [**63**]. Figure 1.3 shows representations of each normalization method that illustrate how they differ from one another.

Batch Normalization

BN tries to reduce internal covariate shift by applying a normalization step to modify the means and variances of layer inputs. For an input x, normalization is done for each feature independently giving it a mean of 0 and a variance of 1. Ideally, normalization would take place over the entire training set, but with the use of stochastic optimization methods in ML this is impractical and normalization statistics are calculated with respect to each mini-batch instead. For a mini batch $\mathcal{B}$ of size m, we define mini batch mean and variance in equations 7 and 8, respectively.

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i \tag{7}$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2 \tag{8}$$

For an individual input $x_i \in \mathbb{R}^d$, normalization is applied feature wise, prior to the non linearity of each layer according to equation 9 where ε is a small constant added for numerical stability and the k superscript indicates feature k of input i.

$$\hat{x}_i^{(k)} = \frac{x_i^{(k)} - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}} \tag{9}$$

To restore the representation power of the network a transformation step shown in equation 10 will follow, where parameters $\gamma^{(k)}$ and $\beta^{(k)}$ are learned during optimization.

$$y_i^{(k)} = \gamma^{(k)} x_i^{(k)} + \beta^{(k)} \tag{10}$$

The batch normalization transform $BN_{\gamma^{(k)},\beta^{(k)}} : x_i^{(k)} \rightarrow y_i^{(k)}$ combines these operations to achieve batch normalization during training. During the inference stage a dependence on mini batches is no longer useful so normalization is achieved using population statistics instead.

Batch norm has strong empirical performance and despite its limitation of requiring sufficiently large batch sizes to be effective [**63**], it continues to be widely used. [**23**] attribute the success of BN to its ability to reduce internal covariate shift but this conclusion is challenged by [**52**] who find the difference in the mean and variance in networks trained with and without BN is marginal. They instead suggest BN works by smoothing the parameter space and gradients which increases the "Lipschitznes" of the loss function and of the gradients. For reference, we define a function f to be L-Lipschitz if $|f(x_1) - f(x_2)| \leq L||x_1 - x_2||, \forall x_1, x_2$. The key of this is to make the gradients more reliable and predictive. Improved Lipschitzness offers increased confidence that a step in a direction of a computed gradient is a fairly accurate estimate of the actual gradient direction thus allowing us to take larger steps with greater confidence.

Group Normalization

GN seeks to improve upon some of the inherent weaknesses in BN, in particular, GN is computed independent of batch size, and it is stable in a wide range of batch sizes. If we define a 2D image as $i = (i_N, i_C, i_H, i_W)$ where N is the batch axis, C is the channel axis and H and W are the height and width axes, we can define the set, $\mathcal{S}_i = \{k | k_N = i_N, \lfloor \frac{k_C}{C/G} \rfloor = \lfloor \frac{i_C}{C/G} \rfloor\}$ where G, the number of groups is a predefined hyperparameter and C/G is the number of channels per group which means that the indexes i and k are in the same group of channels, assuming each group of channels are stored sequentially along the C axis. Group norm computes normalization statistics along the (H, W) and the group along the C axis.

Normalization is then carried out in the same manner as for BN.

Within the context of FL, work by [**20**] claim that distributed training using BN under non i.i.d. settings is particularly vulnerable to performance issues due to its dependence on mini batch statistics. Since BN uses μ_B and σ_B for training, but an estimated global mean and variance for validation, if there is a major mismatch between these statistics, as in the case of non i.i.d. partitions, validation accuracy will be low. They show that batch statistics across clients vary significantly more in the non i.i.d. case than in the i.i.d. case and seek to mitigate this effect by using GN in lieu of BN since GN is not dependent on batch statistics. Additionally, they show that for experiments using BN-LeNet and CIFAR-10 FedAvg with non i.i.d data partitions, GN performs significantly better than BN.

1.3. Continual Learning

In the quest to develop an artificial general intelligence, we require agents capable of learning and retaining that knowledge on a variety of unrelated tasks. In continual learning, tasks are learned sequentially over a period of time and knowledge of previous tasks is retained and leveraged to learn new tasks [**13**]. It is therefore critical for intelligent agents to demonstrate a capacity for continual learning: that is, the ability to learn consecutive tasks without forgetting how to perform previously learned tasks [**27**]. CL in real-world settings has proven particularly difficult since:

- a sequence of tasks may not be explicitly labelled
- tasks may switch unpredictably
- individual tasks may not recur for long time intervals

Continual learning is made difficult by the fact that neural networks suffer from catastrophic forgetting [**41**]. In catastrophic forgetting learning a new task causes weights in the network that are important for previously learned tasks to change in order to meet the objectives of the new task, thus degrading model performance on previously learned tasks. Several families of methods have been developed to mitigate catastrophic forgetting. The first class of methods are architecture based approaches [**53**] that attempt to grow or modify an architecture over time to expand its knowledge. In the second class of methods, approaches which store some subset of old data for rehearsal are applied [**40, 12, 47**]. Finally, a third class of methods regularizes the learned parameters to limit drastic weight changes when learning a new task [**27, 70**]. None of these solutions are widely applicable in a FL setting since they typically require some sort of information sharing across client nodes which is prohibited in the FL framework. A federated continual learning setting has been considered in the literature [**68**]. Here each client in the federated network continuously collects data. Our

work on the other hand considers the standard FL setting where each client maintains a fixed set of data and draws connections to a notion of forgetting across clients to motivate a modification of the loss function. [**55**] have used ideas from continual learning to propose FedCurv, based on the EWC algorithm [**27**] from continual learning. FedCurv requires sending additional information and is not compatible with all FL methods. Along this line, [**64**] also proposed an approach inspired from rehearsal methods, generating pseudo data and adding an additional regularization term. This requires an expensive pseudo data generating procedure and increases the local training time.

Chapter 2

Reducing Local Client Forgetting with a Re-Weighted Softmax Cross-Entropy

2.1. Local client forgetting

The notion of local client forgetting mentioned in the introduction will now be formalized for a multi-class classification problem. Local client forgetting is a feature of training a FL model with heterogeneous data. It combines the concept of catastrophic forgetting from continual learning with the concept of local client drift from federated learning to frame client drift as catastrophic forgetting on the part of the client model with respect to the distributions of other clients and the global distribution as the rounds of local SGD progress.

We denote $Acc_k(\mathbf{w})$, as the accuracy on client k's local test data, where $\mathbf{w}$ are the model parameters of the model being evaluated on the test data of client k. Local client forgetting F_{ki} is defined according to equation 11 where $\mathbf{w}_t$ refers to the global model parameters after the aggregation step of round t. These $\mathbf{w_t}$ are used to initialize the local models at the beginning of round $t+1$. $\mathbf{w}^i_{t-1}$ are the model parameters of client i after local training during round $t+1$ (prior to aggregation).

$$F_{ki} = Acc_k(\mathbf{w}_t) - Acc_k(\mathbf{w}^i_{t-1}) \tag{11}$$

.

For client i where $i \neq k$, we observe a performance degradation of the model of client i when it is evaluated on the data of client k since by optimizing for its own local objective, it has "forgotten" how to generalize on out of distribution data such as the dataset of a different client. Figure 2.1 illustrates local client forgetting within a round of federated learning. In equation 12 we define an average forgetting for a client k's model evaluated on all of the $K-1$ datasets of other clients.

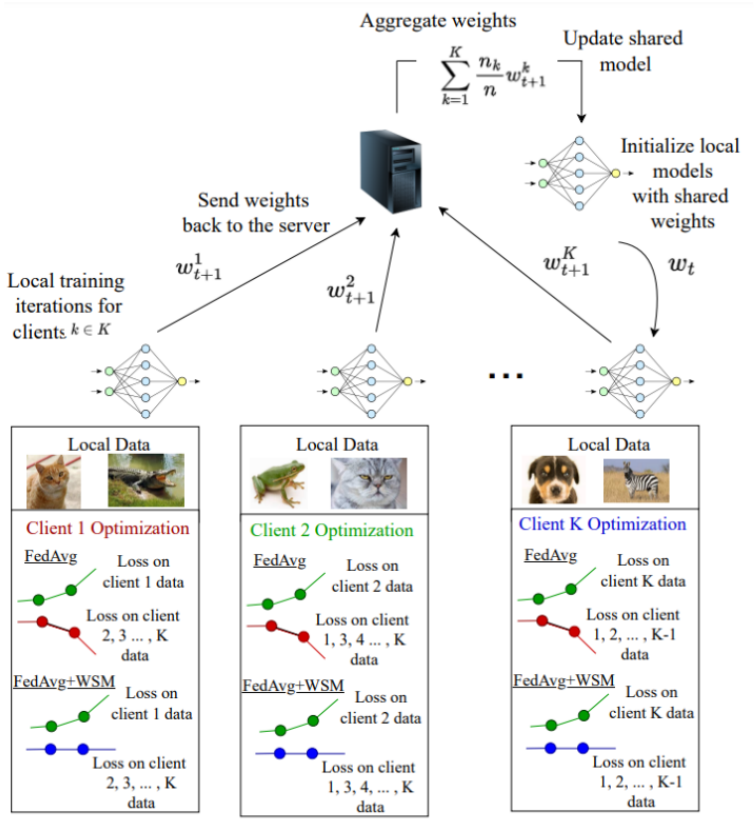

Fig. 2.1. A global model with knowledge of all classes is sent to all clients participating in a given FL round. Local training decreases the loss for a clients local data distribution but tends to simultaneously increase the loss with respect other clients distributions leading to poor aggregation and overall model performance. Mitigating the effects of local client forgetting, which we propose to do with our re-weighted softmax, is capable of greatly improving model performance under these conditions.

$$F_k = \frac{1}{K-1} \sum_{i \neq k} F_{ki} \tag{12}$$

In order to demonstrate the impact of local client forgetting and evaluate the effectiveness of WSM in alleviating this effect, for one round of FL, each clients model is evaluated on its own dataset and the dataset of every other client selected for the round, both before and after local training. It is important to note that for FL in general sharing of data between clients is prohibited. Our evaluation of local client forgetting shares data between nodes for evaluation purposes only and never as a part of model training. For comparison we do FedAvg using typical cross entropy (equation 14) and FedAvg with cross entropy using our re-weighted softmax (equation 15). Figure 2.2 shows client forgetting for ten clients selected for round 2800 training. On the LHS of figure 2.2 we have the difference heatmap, the values shown on the heatmap are F_{ik} from equation 11, indicating the forgetting of that

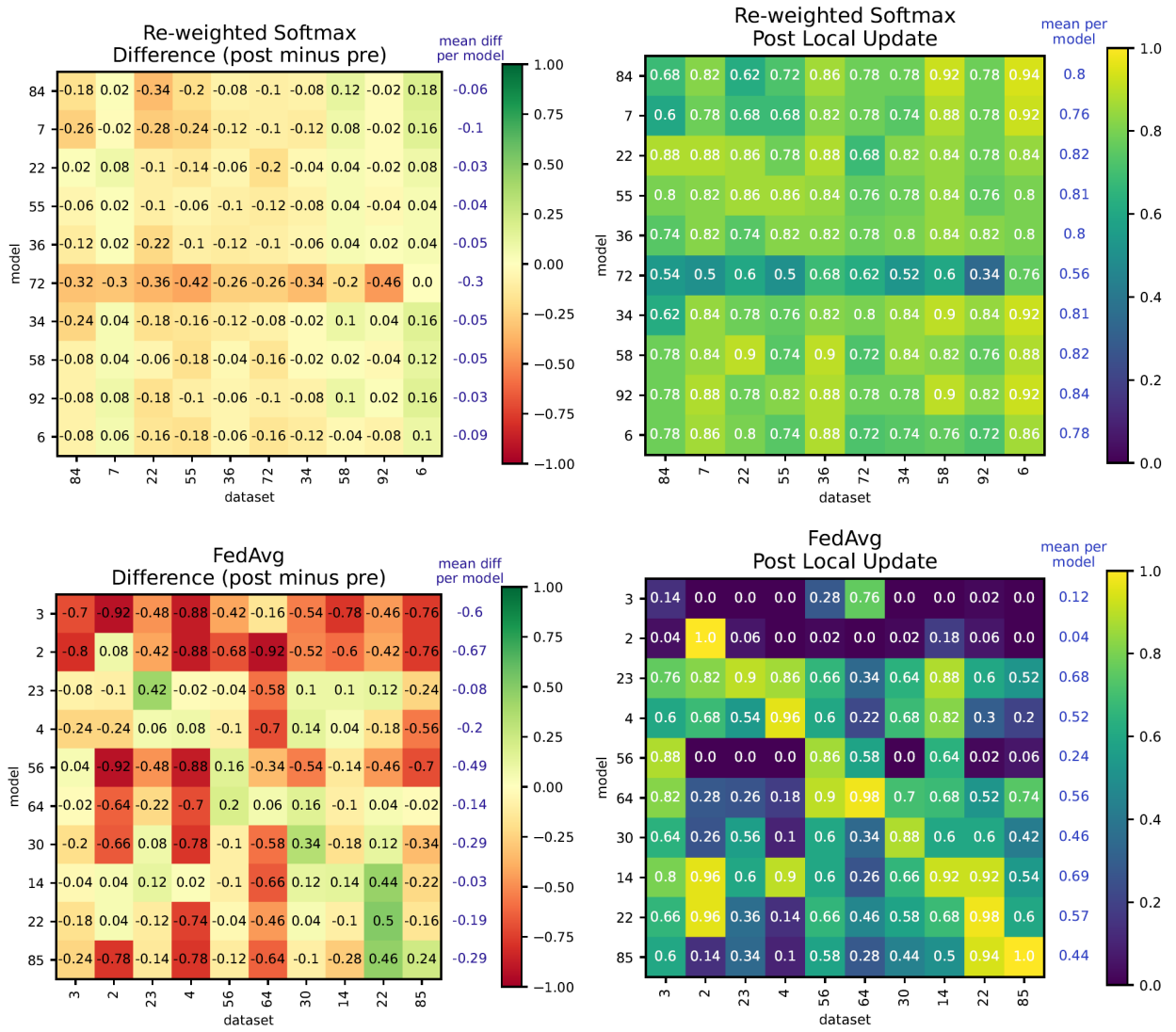

Fig. 2.2. We show local client forgetting for a given round with and without WSM. The x axes contain the indices of clients selected for the round where each of the 100 clients have labels in the set $\{0, ..., 99\}$. The Difference heatmaps (LHS) show the difference in accuracy before and after local training when the k^{th} client's model is evaluated on the i^{th} client's dataset. The final column gives F_k, the average forgetting over all clients. The Post Local Update heatmaps (RHS) show the accuracy of each client's model on the other client's data.

particular client model. Smaller absolute values of F_{ik} indicate less forgetting since there is less difference in accuracy between the model before and after local training. The final column gives the average forgetting F_k (equation 12) which is the average of each row of the heatmap. The "post local update" heatmaps on the RHS of figure 2.2 shows accuracy of each client model on each other clients dataset. We note that in some cases the accuracy can completely collapse on out of distribution client data, we observe this particularly when we evaluate the model of client k on the data of client i and the datasets of client k and

client i do not share any common class labels.

The difference in forgetting observed when using FedAvg+WSM (top row of figure 2.2) and vanilla FedAvg (bottom row of figure 2.2) is significant. In the difference heatmaps of F_{ik} we notice that when adding WSM to FedAvg (top left of figure 2.2), the forgetting values are all relatively small in magnitude *i.e.* the difference between $Acc_k(\mathbf{w}_t)$ and $Acc_k(\mathbf{w}_{t-1}^i)$ is small indicating less forgetting. FedAvg without WSM (bottom left of figure 2.2) has several values farther towards the red end of the heatmap indicating forgetting by that client model after local training. On the RHS when we examine the accuracy of each model on each other clients local dataset we observe FedAvg + CE (bottom right of figure 2.2) has a string preference for it's own local dataset which is indicated by the markedly better accuracies observed along the diagonal of the heatmap. FedAvg+WSM however, (top right of figure 2.2) shows no such preference along the diagonal. In this case we observe similar accuracies clustered along rows indicating the row representing a particular model has approximately equal difficulty with all datasets which indicates that WSM has the ability to greatly limit the effects of local client forgetting by de-prioritizing classes outside of its data distribution and focusing learning on the classes present. This narrower focus also leads to better overall performance of the global model after aggregation. The observations made for round 2800 are not unique and are further confirmed for other rounds as shown in appendix A.

2.2. Re-weighted Softmax Cross Entropy

Based on the work of [**10**] we know catastrophic forgetting in continual learning can be reduced by removing terms corresponding to old tasks from the cross entropy loss term, this reduces the model bias to avoid predicting old classes. Since some federated learning clients may not possess a complete set of class labels in their distributions and/or will have class labels present in very low proportions, this same bias to avoid predicting old classes is a factor in local client forgetting. In our approach, which we call the re-weighted softmax (WSM), we adapt the approach in [**10**] for federated learning and scale the terms in the denominator of the cross entropy loss so they match the clients local distribution. While we have only attempted this method for stationary distributions, it is extremely flexible and could easily be applied to a situation in which the label distributions at the client nodes are evolving over time.

Consider a neural network $f : \mathcal{R}^D \rightarrow \mathcal{R}^C$ where C is the total number of distinct labels in the complete data distribution across all clients. The standard cross entropy is

given by equation 13 where $y(\mathbf{x})$ is the label of a sample $\mathbf{x}$ and $\mathcal{C}$ is the set of all labels available to the clients.

$$\mathcal{L}_{CE}(\mathbf{X}_k, \mathbf{w}) = -\sum_{\mathbf{x}\in\mathbf{X}} \log \frac{\exp(f_{\mathbf{w}}(\mathbf{x})_{y(\mathbf{x})})}{\sum_{c\in\mathcal{C}} \exp(f_{\mathbf{w}}(\mathbf{x})_c)} \tag{13}$$

$$= -\sum_{\mathbf{x}\in\mathbf{X}} \Big[f_{\mathbf{w}}(\mathbf{x})_{y(\mathbf{x})} - \log\Big(\sum_{c\in\mathcal{C}} \exp(f_{\mathbf{w}}(\mathbf{x})_c\Big)\Big] \tag{14}$$

One interpretation of this classical loss function considers the two terms in equation 14 as a tightness term (the first term) which brings samples close to their representative classes and a contrast term (the second term) which pushes them away from other classes [**9**].

We propose to modify the standard cross entropy using a re-weighted softmax (WSM) which leads to our per-client objective function in Equation 15 where $\alpha_k \in \mathbf{R}^C$ is a weight vector for each client k containing the fraction of each class that makes up the dataset X_k for that client.

$$\mathcal{L}_{WSM}(\mathbf{X}_k, \mathbf{w}) = -\sum_{\mathbf{x}\in\mathbf{X}} \Big[f_{\mathbf{w}}(\mathbf{x})_{y(\mathbf{x})} - \log\Big(\sum_{c\in\mathcal{C}} \alpha_c \exp(f_{\mathbf{w}}(\mathbf{x})_c)\Big)\Big] \tag{15}$$

We note that in Equation 15 the weighting α_k introduced in the second term is a function only of the labels present in the client data $\mathbf{Y}_k$. In a highly imbalanced class scenario as is often studied for FL, for a class fraction where $\alpha_c \in \alpha_k$, many α_c will be zero or very small values. This has the effect of removing or substantially minimizing the contribution of that class to the contrast term. The intuition behind this choice is that aggressive optimization of $\mathcal{L}_{CE}$ for multiple gradient steps during a client update round can lead to a drastic increase in $\mathbb{E}_{x,y\sim D_{j\neq k}}[l_{CE}(\mathbf{x}, \mathbf{y})]$ where D_j are the distributions of clients other than client k. This increase is caused by the contrast term in equation 14 which discourages the local model from ever predicting class labels not present in that client's local dataset.

Our re-weighted softmax approach modifies the second term of the original cross entropy loss to avoid the excessive pressure that will cause the loss to increase when evaluated on the datasets of other clients. This re-weighting causes class labels not present in the dataset of client k to be ignored by the local optimization for that client and encourages the local model to learn by adapting its internal representation of only the classes present in its training data. It discourages abruptly shifting representations of classes outside its local distribution [**10**]. This thesis empirically demonstrates that using WSM instead of

traditional cross entropy leads to a reduction in local client forgetting which is defined in section 2.1.

Chapter 3

Experiments

3.1. Datasets and Data Partitioning

Datasets used in these experiments are CIFAR-10, CIFAR-100 [**29**] and FEMNIST [**11**]. Since local client forgetting occurs when the underlying distributions between clients differ we require a method of dividing the data between clients such that their datasets are not i.i.d. Furthermore, we require a means of quantifying how i.i.d. the distributions between clients so we can investigate the effectiveness of WSM under different conditions. The method most commonly employed in the federated learning literature was popularized by [**21**] where they achieve this by dividing the complete dataset between clients using the Dirichlet distribution. The Dirichlet distribution is parameterized by α where as $\alpha \leftarrow \infty$ the distribution at each client that is approaches i.i.d. and as $\alpha \leftarrow 0$ the distribution at each client approaches a situation where each client possesses only a single class label. To provide some intuition into what client partitions parameterized by α look like in practice, Fig. 3.1 shows the data distributions of ten randomly selected clients for $alpha = \{0.01, 0.1, 0.5, 100\}$. Dirichlet distributions parameterized by $\alpha = 0.1$ and $\alpha = 0.5$ are commonly used in federated learning literature [**21, 48, 33**].

In our baseline experiments we use $\alpha = 0.1$ where the entire training set is separated into K equally sized non-i.i.d. partitions where K is the number of clients. Referring to figure 3.1 we observe that the for $\alpha = 0.1$ distribution, some clients such as client 6 have heavily skewed partitions and no client possesses each of the ten class labels in their distributions. Since each client requires its own training and validation sets according to their own unique distributions the client partitions are further separated into training (90%) and validation (10%) sets for each client. For example, 100 clients trained using CIFAR-10 which contains 50 000 training samples would each have 500 of these training samples. Of those 500 samples, 450 would be used for local model updates and 50 would be used exclusively for validation. The testing sets provided with each dataset are accurate

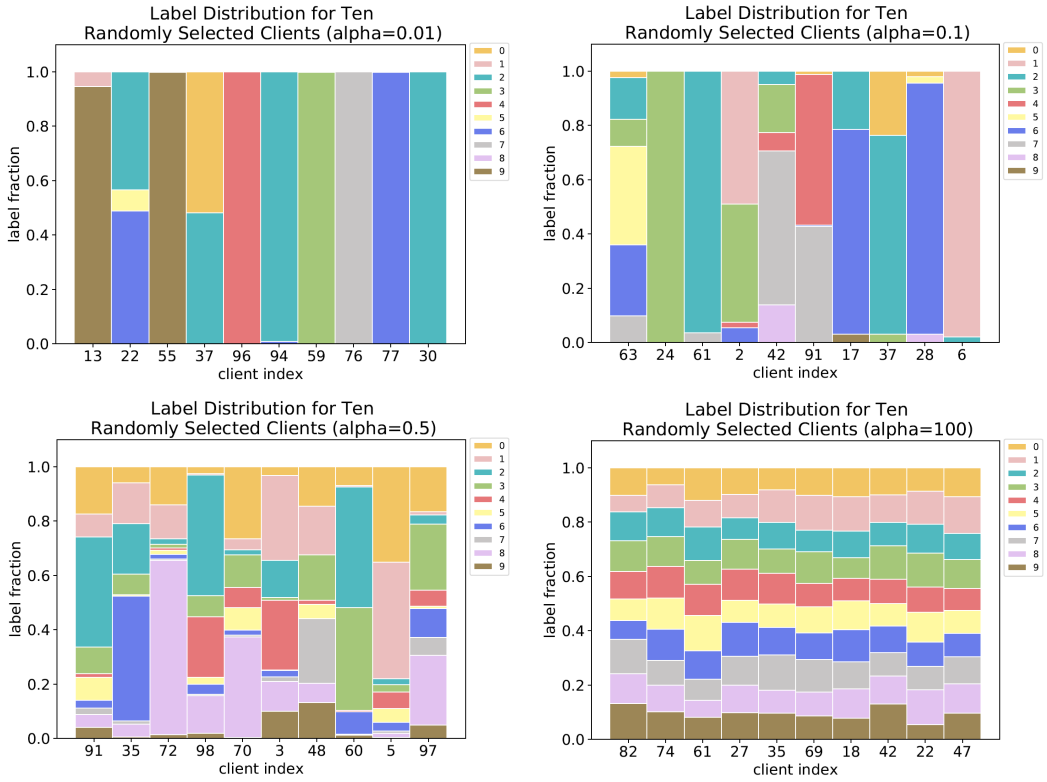

Fig. 3.1. The data proportions of ten randomly selected clients whose data was partitioned according to a Dirichlet distribution with $\alpha = 0.01$ (top left), 0.1 (top right), 0.5 (bottom left), 100 (bottom right). The ten cifar-10 classes are represented by different colors and the y-axis indicates the percentage occurrence of each label for each client listed along the x-axis.

representations of the global distributions of the combined data of each client. The testing sets are used only at then end of training to evaluate the final accuracy of the global model, no part of the test sets are partitioned or distributed to the clients.

3.2. Experimental Settings

In our baseline experiments, for each round of federated learning a fraction, 0.1, of the total 100 clients are selected randomly and initialized with a copy of the current global model. Clients are sampled without replacement for each round but can be selected again in subsequent rounds. The fraction of clients sampled is 10% for CIFAR-10 and FEMNIST datasets and 2% for CIFAR-100.

We train a ResNet-18 model over 4000 communication rounds and each client performs 3 local update steps, using a mini-batch of size 64. and a learning rate of 0.1 for CIFAR-10 and CIFAR-100. Femnist, which converges faster due to its large size, is trained for 2000 rounds with all other settings the same as for the CIFAR datasets. We use SGD as

our optimizer, with weight decay of 1×10^{-4} following [**67, 21**]. In these experiments, we use a combination of GN and BN, the exact configuration of which is specified in appendix B.

3.3. Validation Methods

Each client receives a sample of the training data corresponding to their unique underlying distribution. In order to have test sets for each client that also correspond to this distribution, the training sample at each client is split 90 : 10 into training and validation sets. Throughout the training process the global model is periodically evaluated on the aggregation of the client validation sets to gauge overall training progress. The test data provided as part of the CIFAR and FEMNIST datasets, on the other hand is only used at the end of the training process to evaluate the final global model accuracy.

It is a known feature of federated learning training that as client distributions become more skewed, in our case when we use lower values of α, there can be significant changes in accuracy between training runs [**21**]. Since we focus our analysis on the highly heterogeneous case in which the Dirichlet distributions at each client are parameterized by $\alpha = 0.1$, we observe high variance in our results. Variance is particularly pronounced for smaller datasets such as CIFAR-10 and CIFAR-100. To mitigate these effects on the validation statistics, we follow the lead of [**48**] and report our final accuracy as the average of the test accuracies taken over the last 100 rounds of training, it is this value that is reported in table 3.1.

3.4. Evaluation of WSM

In this section we demonstrate how WSM used in combination with FedAvg can greatly improve model performance. Figure 3.2 shows the best performing models among FedAvg and FedAvg+WSM for CIFAR-10, CIFAR-100 and FEMNIST. We observe that combining FedAvg with WSM generally provides a stronger performance from the very beginning of training since for CIFAR-10 and CIFAR-100 the green curve showing the WSM training progression starts off with higher reported accuracy than FedAvg and this improvement in performance persists for the duration of training. Work on critical learning periods, where critical learning periods are defined as the early epochs of a training regime, have shown they can determine the final quality of a deep neural network for traditional ML methods [**24, 2**]. [**65**] Investigate critical learning periods in the FL setting and discover they do indeed exist consistently in FL. Based on this work we can theorize that the early training advantage we see when applying WSM may be having a positive impact on it's consistent ability to outperform other FL algorithms it is applied to.
Table 3.1 shows model performance across a range of learning rates. For all three datasets

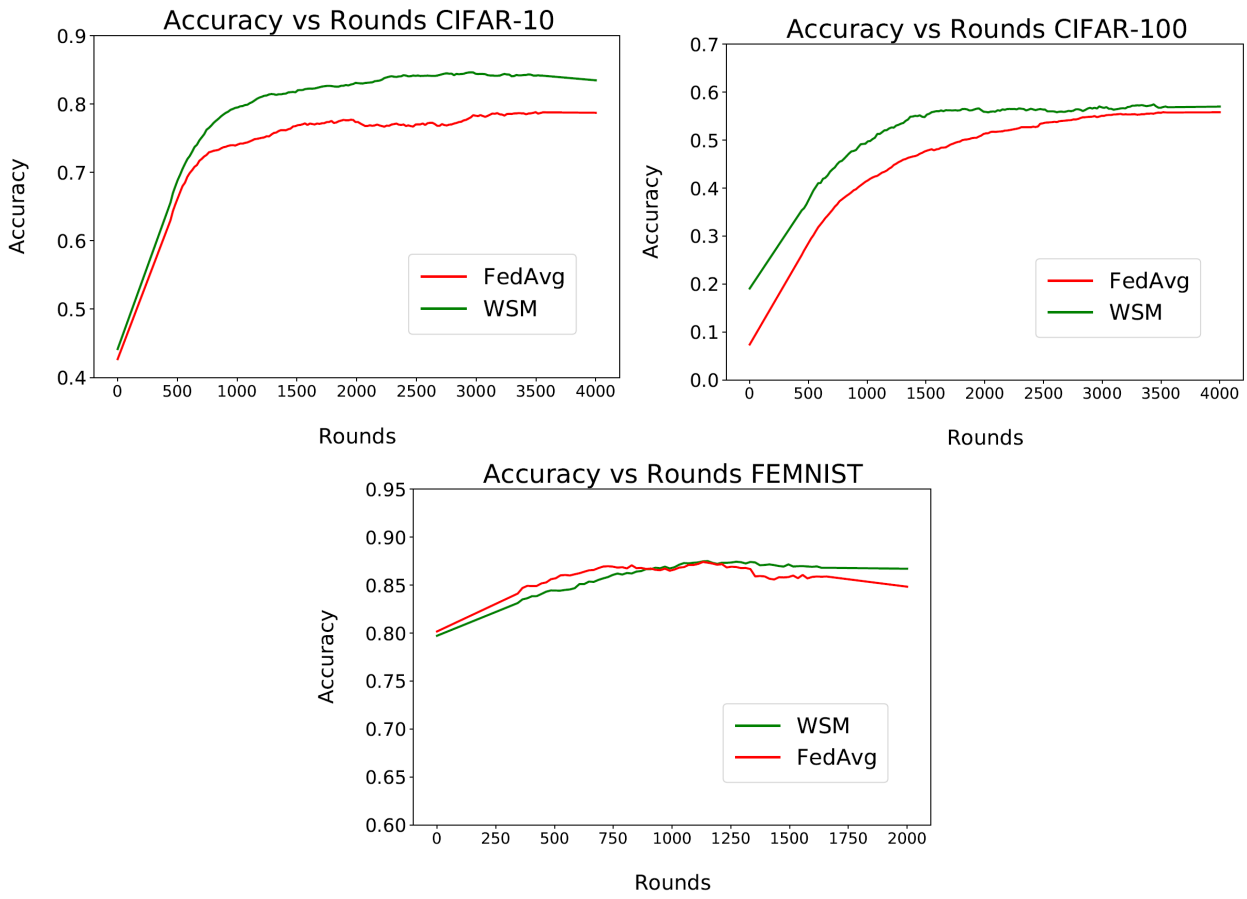

Fig. 3.2. We show the performance on CIFAR-10 and CIFAR-100 over 4000 rounds in a highly heterogenous setting with FedAvg, FedAvg+WSM (top row) and over 2000 rounds of training on FEMNIST (bottom). We observe that WSM consistently gives the best performance, out performing FedAvg on a per round basis For both CIFAR-10 and CIFAR-100. Results for the FEMNIST dataset are more subtle due to the size of the dataset. FEMNIST has many times the sample data of the CIFAR datasets so the amount of training per round is much greater. However, even with a very large datasets, WSM does provide obvious improvement with FedAvg+WSM clearly overtaking FedAvg in accuracy results at $\approx$ round 1200 and remining the better performer thereafter.

used in training, FedAvg+WSM offers a substantial improvement in performance over the top vanilla FedAvg accuracy. We observe improvements in model accuracy of $\approx$ 3% for CIFAR-10 and FEMNIST datasets and $\approx$ 1% for CIFAR-100. Additionally, we note that improvement using WSM is fairly consistent across learning rates, in table 3.1 we notice that for each dataset, FedAvg+WSM generally has a higher accuracy than for FedAvg.

3.5. WSM for Heterogeneous FL methods

In section 3.4 we demonstrated the ability of WSM to improve model performance for the FedAvg algorithm but we wish to demonstrate its versatility by showing WSM is effective

	lr	*Dataset* CIFAR-10	CIFAR-100	FEMNIST
FedAvg		0.771	0.486	0.803
FedAvg+WSM (ours)	0.7	**0.777**	**0.539**	**0.822**
FedAvg		0.740	0.522	0.806
FedAvg+WSM (ours)	0.5	**0.803**	**0.546**	**0.838**
FedAvg		0.775	0.533	0.804
FedAvg+WSM (ours)	0.3	**0.815**	**0.560**	**0.832**
FedAvg		0.801	0.573	0.832
FedAvg+WSM (ours)	0.1	**0.846**	**0.591**	**0.855**
FedAvg		0.811	0.569	0.787
FedAvg+WSM (ours)	0.07	**0.838**	**0.596**	**0.857**
FedAvg		0.791	0.577	0.817
FedAvg+WSM (ours)	0.05	**0.836**	**0.595**	**0.851**
FedAvg		0.796	**0.589**	0.740
FedAvg+WSM (ours)	0.03	**0.809**	0.586	**0.861**
FedAvg		0.753	0.564	**0.833**
FedAvg+WSM (ours)	0.01	**0.767**	**0.580**	0.830
FedAvg		0.740	0.552	**0.829**
FedAvg+WSM (ours)	0.007	**0.758**	**0.569**	0.828

Table 3.1. Accuracy results of FedAvg with and without WSM for different settings of client learning rates. We observe that for many learning rate settings WSM consistently improves performance, as well as having the highest overall accuracy by a large margin.

over a range of FL algorithms. In Table 3.2 we apply WSM to SCAFFOLD [**26**] and FedProx [**36**]. These are two constrained optimization methods specifically designed to address the challenges of heterogeneous data in federated learning. Since both SCAFFOLD and FedProx perform better with smaller learning rates we decrease the learning rate for these algorithms to 0.01 while we keeping 0.1 for FedAvg. The best value for the additional FedProx hyperparameter $\lambda = 1.1$ was empirically determined by a grid search of $\lambda = \{0.8, 0.9, ..., 1.3\}$. In these experiments we use 20 clients and select the fraction 0.1 of all clients at each round. As in the previous section, the data is distributed between clients using a Dirichlet distribution parameterized by $\alpha = 0.1$ and we train the models over 1500 rounds, keeping the same local batch size of 64 and 3 local iterations from the base case. Since results are more stable with 20 clients than with 100 clients, accuracies in this case are evaluated one time after the completion of training.

From table 3.2 we see that both FedProx and SCAFFOLD outperform FedAvg without WSM. This is expected since both the FedProx and SCAFFOLD algorithms were developed to tackle the problem of heterogeneous data distributions and they have been

Method	CIFAR-10	CIFAR-100
FedAvg	0.684	0.519
FedAvg+WSM (ours)	**0.833**	**0.603**
Scaffold	0.816	0.532
Scaffold+WSM (ours)	**0.825**	**0.598**
FedProx	0.752	0.498
FedProx+TCE (ours)	**0.822**	**0.554**

Table 3.2. Accuracy results of FedAvg [**42**], Scaffold [**26**] and FedProx [**36**] with and without WSM. We observe that even combined with FL optimization based methods designed to address heterogeneity, WSM provides consistent performance gains. Furthermore, the best performance overall is highlighted in red. We find that although SCAFFOLD and FedPROX can handle heterogeneity better than FedAvg, when combined with WSM FedAvg can obtain the best performance amongst all methods.

previously shown to outperfrom FedAvg [**36, 26, 33**]. Combining each of FedAvg, FedProx and SCAFFOLD with WSM shows improved performance for each algorithm on both CIFAR-10 and CIFAR-100. These results validate our hypothesis that using WSM in combination with and FL algorithm will help to mitigate local client forgetting therefore providing improvement over base cases for each method. Additionally, the implementation of SCAFFOLD and FedProx was taken from the repository for [**33**] and the substitution of WSM for CE was very simple to make since it is entirely self contained. The ease of application of WSM is a significant advantage since this feature makes it an easy way to achieve a substantial improvement over a baseline algorithm. Finally, we observe that combining WSM with FedAvg has the largest overall effect since the best overall method in table 3.2 is FedAvg+WSM. While WSM does offer improvements over all baseline algorithms, this result is somewhat surprising since our intuition was that WSM would work independently alleviate local client forgetting and would augment the value of the modifications offered by the different federated learning algorithms. We theorize that the superior performance of FedAvg+WSM over SCAFFOLD+WSM or FedProx+WSM is because while SCAFFOLD, FedProx and WSM all individually improve the baseline FedAvg, they may need to be further optimized to work effectively together.

3.6. Ablations

We now further study the behavior of WSM in combination with FedAvg under different data distributions, client participation settings and different numbers of local iterations. In Figure 3.3 ablations are shown using the CIFAR-10 dataset where we ablate one setting at a time. Except for where we specify the value of the parameter being ablated, the hyperparameters for the ablation studies are the same as for our base case described in

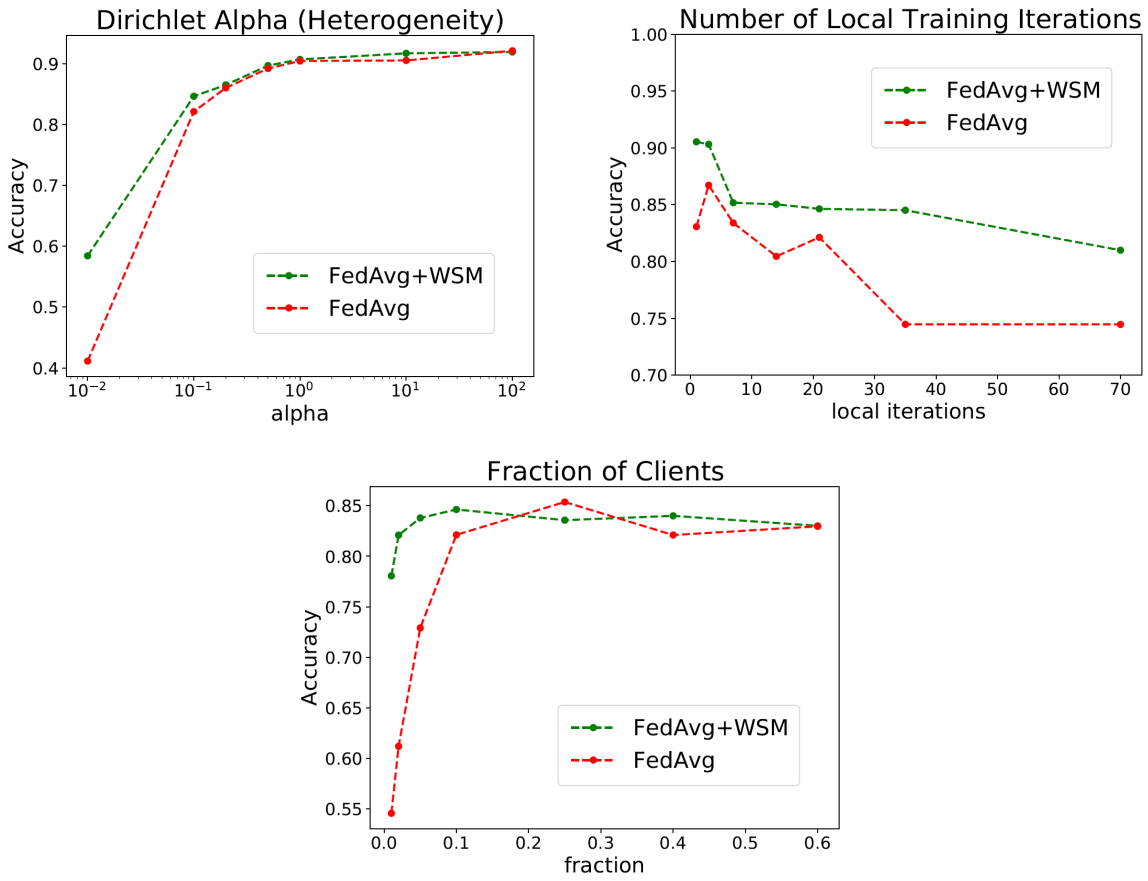

Fig. 3.3. Plots of ablations of dataset heterogenaity, number of local iterations and the fraction of clients selected at each round studied with WSM. We observe WSM provides performance increases under most of the conditions studied but it's most significant advantages are in scenarios with very heterogeneous client data and smaller fractions of clients are selected for each training round.

section 3.4. We observe that under conditions of low α (high heterogeneity) and low fractions of clients selected at each round, the use of WSM is particularly advantageous.

3.6.1. Parameter α of the Dirichlet Distribution

The Dirichlet distribution is parameterized by α as $\alpha \to 0$ the client distributions will become increasingly heterogeneous and as $\alpha \to \infty$, each client data edges closer towards the same i.i.d. distribution. In our base case we use $\alpha = 0.1$ and we now investigate how model performance is affected by changes in the heterogeneity of the client data distributions. Similar to the work of [**21**], we investigate $\alpha = \{0.01, 0.1, 0.2, 0.5, 1, 10, 100\}$. Figure 3.1, showing the label distributions for ten randomly selected clients, offers a practical illustration of how local data partitions change as a function of α. From Figure 3.3 we observe WSM has a more significant effect as α decreases with the biggest margin of improvement over vanilla

FedAvg occurring when $\alpha = 0.01$. As the data becomes increasingly homogeneous, the α vector in equation 15 representing proportions of labels present at each client, converges to equal proportions at each client (i.i.d. data). The gap between cross entropy and WSM then shrinks until the two methods are equivalent. From Figure 3.1 (bottom left) we see that by the time $\alpha = 0.5$ most of the clients possess all of the classes, albeit in very skewed proportions. This is also the point at which the performance of FedAvg and FedAvg+WSM become very close. While WSM continues to offer a performance increase over the entire range of α except for $\alpha = 100$, we conclude WSM is most advantageous when clients do not possess the entire set possible class labels.

3.6.2. Fraction of Participating Clients

One of the characteristics of FL is that data is massively distributed and nodes may have limited communication with the central server as nodes may be offline or have slow connections limiting their communication [**42, 36**]. The direct result of the limited capacity for connection is that only a small number of client nodes may participate in updates at any given time. These ablations focus on the fraction of clients participating in an update round, for each experiment the fraction of participating clients is constant for all rounds of training and we explore fractions from 1%, where only one client participates in the update, up to 60%.

For the fraction C of participating clients, we observe the largest performance gap between FedAvg+WSM and FedAvg when the number of participating clients is low. This feature is significant since as explained above, low client participation is a known feature of real world federated learning settings. We hypothesize the performance gap as a function of client fraction between the FedAvg and FedAvg+WSM is due to the larger impact of local client forgetting when we limit communication capability. Unless we actively take steps to control forgetting, non-participating clients will have their data distributions forgotten because unlike participating clients, they will be unable to contribute their updates to the global model. As the fraction of clients selected at each round increases, we observe the performance gap between the two methods narrow and ultimately converge since more clients will have the opportunity to be selected at each round and "remind" the model of their data distributions.

3.6.3. Local Iterations

Altering the number of local training iterations results in the least accuracy variation of the three parameters investigated in the ablation study. We observe model performance increase as local training iterations decreases, and find that WSM prevents the steep decline in accuracy observed in FedAvg as the number of training iterations increases. Since WSM

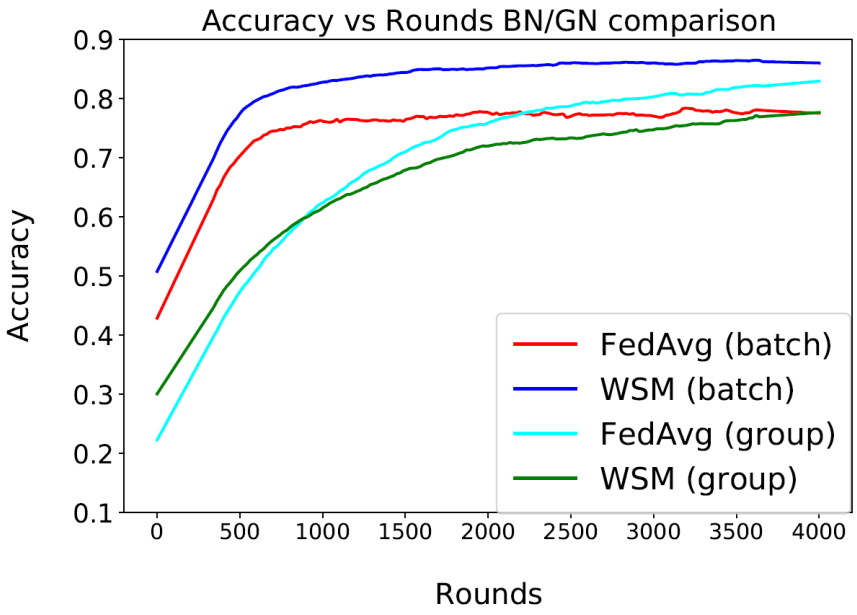

Fig. 3.4. Accuracy over 4000 rounds for FedAvg and FedAvg+WSM with batch norm and with group norm. Experiments are performed using the CIFAR-10 dataset.

helps to alleviate the effects of local client forgetting, this explains the more pronounced positive effect we see as the number of local iterations increases causing the effects of local client forgetting become more serious.

3.7. Effects of Batch and Group Norm on WSM

Our primary experiments used a combination of BN and GN, in this section we perform a more rigorous evaluation of the effects of the choice of normalization on model performance. These experiments are performed for the CIFAR-10 dataset only under the same conditions as specified in section 3.4 except for the choice of normalization scheme and the number of rounds. In section 3.4 we use 4000 rounds of training for each experiment but referring to figure 3.4 we observe a much slower learning progression when group norm is used exclusively and it appears the training using group norm may not have fully converged by 4000 rounds. In order to be certain the group norm experiments had indeed converged, 8000 rounds of training were performed for each result presented in table 3.3. Figure 3.5 shows the accuracies over 8000 rounds of training.

As mentioned previously, one of the main differences we observe in figure 3.5 is model accuracies improve much faster when using BN than with GN both FedAvg and FedAvg+WSM. The speed of improvement is unsurprising since BN is known to accelerate convergence [**23, 63**]. However, BN is also thought to have a negative effect on training in a federated setting [**20, 63**] so while it is also unsurprising to see FedAvg with BN plateau and begin to diverge the surprising result is that FedAvg+WSM trained with BN does not suffer this

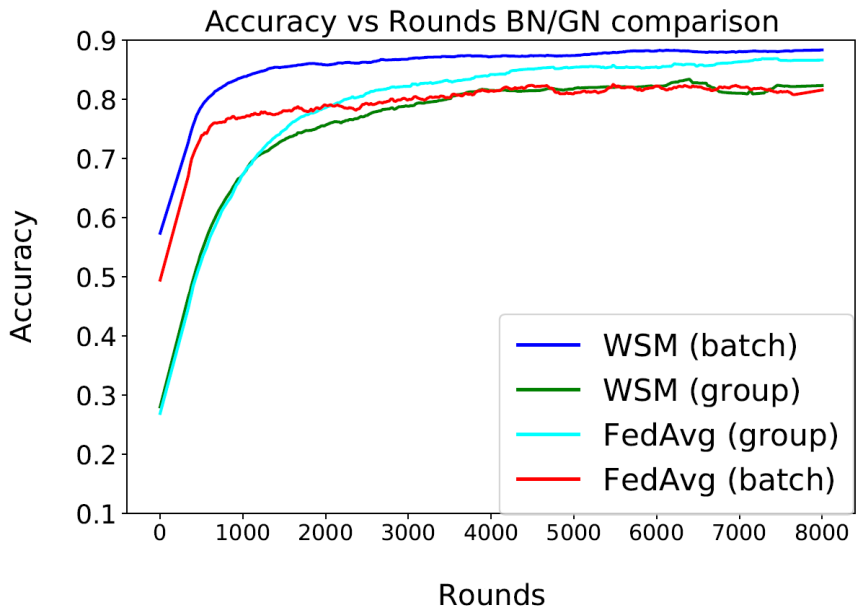

Fig. 3.5. Accuracy over 8000 rounds for FedAvg and FedAvg+WSM with BN and with GN. Experiments are performed using the CIFAR-10 dataset.

fate and is able to obtain the best performance at any given round compared to both versions of FedAVG. Figure 3.5 shows that FedAvg+WSM trained with BN outperforms all other combinations. Not only does it preform significantly better than the other combinations from the very beginning of training, it requires much fewer rounds of training than the next best performer, FedAvg with GN to achieve that result. Indeed, referring to table 3.3 which shows a summary of model accuracies over a range of learning rates we observe that FedAvg+WSM with BN consistently performs the best over the range of learning rates, it also has the overall best performance with an accuracy of 0.882. These results suggest the conclusion that GN outperforms BN in a federated setting due to BN's dependence on batch statistics [**20**] may not provide the complete story. [**20**] do show that GN outperform BN in their experiments but the conditions under which these experiments were performed are very limited. Further, while our results support their conclusions in the case of our FedAvg baseline, we find that FedAvg+WSM performs much better with BN than with GN and WSM does not in any way modify the divergence of batch statistics across clients which is central to the argument that GN will perform better in a federated setting. One potential explanation for the success of FedAvg+WSM with BN is based on the work of [**52**], in their work, they suggest that the success of BN is due to a smoothing of the loss function and a resultant smoothing of the gradients. Since WSM is a modification of the loss function that reduces local client forgetting, we theorize this improvement in generalization makes the gradient smoothing induced by BN more broadly applicable than it would be when using

	lr	norm	accuarcy
FedAvg			0.824
FedAvg+WSM (ours)	0.3	group	0.761
FedAvg			**0.828**
FedAvg+WSM (ours)	0.3	batch	0.767
FedAvg			0.856
FedAvg+WSM (ours)	0.1	group	0.800
FedAvg			0.779
FedAvg+WSM (ours)	0.1	batch	**0.858**
FedAvg			0.853
FedAvg+WSM (ours)	0.07	group	0.796
FedAvg			0.811
FedAvg+WSM (ours)	0.07	batch	**0.854**
FedAvg			0.865
FedAvg+WSM (ours)	0.05	group	0.814
FedAvg			0.822
FedAvg+WSM (ours)	0.05	batch	**0.882**
FedAvg			0.864
FedAvg+WSM (ours)	0.03	group	0.808
FedAvg			0.827
FedAvg+WSM (ours)	0.03	batch	**0.873**
FedAvg			**0.856**
FedAvg+WSM (ours)	0.01	group	0.808
FedAvg			0.810
FedAvg+WSM (ours)	0.01	batch	0.847
FedAvg			0.808
FedAvg+WSM (ours)	0.007	group	0.853
FedAvg			0.824
FedAvg+WSM (ours)	0.007	batch	**0.860**
FedAvg			0.824
FedAvg+WSM (ours)	0.005	group	0.811
FedAvg			0.812
FedAvg+WSM (ours)	0.005	batch	**0.841**

Table 3.3. Accuracy results of FedAvg with and without WSM for ResNet18 with BN and with GN. Experiments are performed over a range of learning rates between 0.3 and 0.005. We observe that FedAvg+WSM with BN consistently performs the best with the overall highest accuracy of 0.882. The colored boxes indicate the best accuracy obtained for each of FedAvg (group), FedAvg+WSM (group), FedAvg (batch), FedAvg+WSM (batch)

FedAvg alone where the forgetting induced by local optimization would be amplified by the gradient smoothing effects of BN.

3.8. Conclusions

In this paper we took a deeper look at the *local client forgetting* problem. We show that when a client performs local updates during FL, it risks overly optimizing its local objective leading to forgetting on other subsets of data. We make a connection between the catastrophic forgetting problem in continual learning and the client drift problem in FL and propose a client level modification of the objective function which we call the re-weighted softmax. Our experiments on local client forgetting show that it degrades performance of the global model, especially in cases where there is a significant distribution mismatch across clients. We observe that WSM has a significant and consistent positive effect on reducing client level forgetting which allows us to mitigate local client forgetting across a variety of conditions commonly studied in FL. We also observe a strong performance of WSM in early rounds of training, termed the critical learning period, which may be additional feature of its success.

Practically, we show that WSM is very easy to apply to any FL algorithm and while we use FedAvg heavily as a baseline for our comparisons, we also see performance improvements for SCAFFOLD and FedProx when we apply WSM to them. An ablation study demonstrated that WSM is particularly effective in the regime of highly heterogeneous client datasets and/or when a small percentage of clients are selected at each round. These settings are important for the federated setting since the nature of real-world massively distributed data means a relatively small percentage of clients will be participating in each federated round and client nodes often possess only a very small subset of this massively distributed underlying dataset. An additional study into the effects of BN and GN on FedAvg and FedAvg+WSM provide the unexpected result that WSM with BN is the most beneficial combination in terms of accuracy. Prior work has shown that BN performs poorly in a federated setting and suggests its dependence on batch statistics as the reason for this but our work contradicts this evaluation since batch statistics do not change between the FedAvg and FedAvg+WSM settings.

Our current work shows that WSM is broadly effective and widely applicable to a variety of settings. Our results indicate that addressing local client forgetting in general is an important consideration for federated learning optimization. Our experiments with BN and GN also raise interesting questions about normalization in FL and how it may impact different algorithms

Références bibliographiques

[1] Durmus Alp Emre Acar, Yue Zhao, Ramon Matas Navarro, Matthew Mattina, Paul N Whatmough et Venkatesh Saligrama : Federated learning based on dynamic regularization. *arXiv preprint arXiv:2111.04263*, 2021.

[2] Alessandro Achille, Matteo Rovere et Stefano Soatto : Critical learning periods in deep neural networks. *arXiv preprint arXiv:1711.08856*, 2017.

[3] Hongjoon Ahn, Jihwan Kwak, Subin Lim, Hyeonsu Bang, Hyojun Kim et Taesup Moon : Ss-il: Separated softmax for incremental learning. *In Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 844–853, 2021.

[4] Kartik Ahuja, Karthikeyan Shanmugam, Kush Varshney et Amit Dhurandhar : Invariant risk minimization games. *In International Conference on Machine Learning*, pages 145–155. PMLR, 2020.

[5] Rahaf Aljundi, Eugene Belilovsky, Tinne Tuytelaars, Laurent Charlin, Massimo Caccia, Min Lin et Lucas Page-Caccia : Online continual learning with maximal interfered retrieval. *In* H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox et R. Garnett, éditeurs : *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[6] Jimmy Lei Ba, Jamie Ryan Kiros et Geoffrey E Hinton : Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[7] Abhishek Bhowmick, John Duchi, Julien Freudiger, Gaurav Kapoor et Ryan Rogers : Protection against reconstruction and its applications in private federated learning. *arXiv preprint arXiv:1812.00984*, 2018.

[8] Christopher M. Bishop : *Pattern Recognition and Machine Learning*. Springer, 2006.

[9] Malik Boudiaf, Jérôme Rony, Imtiaz Masud Ziko, Eric Granger, Marco Pedersoli, Pablo Piantanida et Ismail Ben Ayed : A unifying mutual information view of metric learning: cross-entropy vs. pairwise losses. *In European conference on computer vision*, pages 548–564. Springer, 2020.

[10] Lucas Caccia, Rahaf Aljundi, Nader Asadi, Tinne Tuytelaars, Joelle Pineau et Eugene Belilovsky : New insights on reducing abrupt representation change in online continual learning. *In International Conference on Learning Representations*, 2022.

[11] Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konečnỳ, H Brendan McMahan, Virginia Smith et Ameet Talwalkar : Leaf: A benchmark for federated settings. *arXiv preprint arXiv:1812.01097*, 2018.

[12] Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K Dokania, Philip HS Torr et Marc'Aurelio Ranzato : On tiny episodic memories in continual learning. *arXiv preprint arXiv:1902.10486*, 2019.

[13] Zhiyuan Chen et Bing Liu : Lifelong supervised learning. *In Lifelong Machine Learning*, pages 33–74. Springer, 2018.

[14] MohammadReza Davari, Nader Asadi, Sudhir Mudur, Rahaf Aljundi et Eugene Belilovsky : Probing representation forgetting in supervised and unsupervised continual learning. *In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16712–16721, 2022.

[15] Enmao Diao, Jie Ding et Vahid Tarokh : Heterofl: Computation and communication efficient federated learning for heterogeneous clients. *arXiv preprint arXiv:2010.01264*, 2020.

[16] Cynthia Dwork et Aaron Roth : The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3–4):211–407, aug 2014.

[17] Alireza Fallah, Aryan Mokhtari et Asuman Ozdaglar : Personalized federated learning: A meta-learning approach. *arXiv preprint arXiv:2002.07948*, 2020.

[18] Sharut Gupta, Kartik Ahuja, Mohammad Havaei, Niladri Chatterjee et Yoshua Bengio : Fl games: A federated learning framework for distribution shifts. *arXiv preprint arXiv:2205.11101*, 2022.

[19] Junyuan Hong, Haotao Wang, Zhangyang Wang et Jiayu Zhou : Efficient split-mix federated learning for on-demand and in-situ customization. *arXiv preprint arXiv:2203.09747*, 2022.

[20] Kevin Hsieh, Amar Phanishayee, Onur Mutlu et Phillip Gibbons : The non-iid data quagmire of decentralized machine learning. *In International Conference on Machine Learning*, pages 4387–4398. PMLR, 2020.

[21] Tzu-Ming Harry Hsu, Hang Qi et Matthew Brown : Measuring the effects of non-identical data distribution for federated visual classification. *arXiv preprint arXiv:1909.06335*, 2019.

[22] Chip Huyen : *Designing Machine Learning Systems*. O'Reilly Media, USA, 2022.

[23] Sergey Ioffe et Christian Szegedy : Batch normalization: Accelerating deep network training by reducing internal covariate shift. *In International conference on machine learning*, pages 448–456. PMLR, 2015.

[24] Stanisław Jastrzębski, Zachary Kenton, Nicolas Ballas, Asja Fischer, Yoshua Bengio et Amos Storkey : On the relation between the sharpest directions of dnn loss and the sgd step length. *arXiv preprint arXiv:1807.05031*, 2018.

[25] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings *et al.* : Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210, 2021.

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

[27] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska *et al.* : Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[28] Jakub Konečnỳ, H Brendan McMahan, Daniel Ramage et Peter Richtárik : Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527*, 2016.

[29] Alex Krizhevsky et Geoffrey Hinton : Learning multiple layers of features from tiny images. Rapport technique 0, University of Toronto, Toronto, Ontario, 2009.

[30] Gwen Legate, Lucas Caccia et Eugene Belilovsky : Reducing forgetting in federated learning with truncated cross-entropy. *In NeurIPS 2022 Workshop on Distribution Shifts: Connecting Methods and Applications*, 2022.

[31] Timothée Lesort : Continual feature selection: Spurious features in continual learning. *arXiv preprint arXiv:2203.01012*, 2022.

[32] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita et Bor-Yiing Su : Scaling distributed machine learning with the parameter server. *In Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, OSDI'14, page 583–598, USA, 2014. USENIX Association.

[33] Qinbin Li, Yiqun Diao, Quan Chen et Bingsheng He : Federated learning on non-iid data silos: An experimental study. *In 2022 IEEE 38th International Conference on Data Engineering (ICDE)*, pages 965–978. IEEE, 2022.

[34] Qinbin Li, Zeyi Wen, Zhaomin Wu, Sixu Hu, Naibo Wang, Yuan Li, Xu Liu et Bingsheng He : A survey on federated learning systems: vision, hype and reality for data privacy and protection. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

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

[36] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar et Virginia Smith : Federated optimization in heterogeneous networks. *Proceedings of Machine Learning and Systems*, 2:429–450, 2020.

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

[38] Zhizhong Li et Derek Hoiem : Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

[39] Tao Lin, Lingjing Kong, Sebastian U Stich et Martin Jaggi : Ensemble distillation for robust model fusion in federated learning. *Advances in Neural Information Processing Systems*, 33:2351–2363, 2020.

[40] David Lopez-Paz et Marc'Aurelio Ranzato : Gradient episodic memory for continual learning. *Advances in neural information processing systems*, 30, 2017.

[41] Michael McCloskey et Neal J Cohen : Catastrophic interference in connectionist networks: The sequential learning problem. *In Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989.

[42] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson et Blaise Aguera y Arcas : Communication-efficient learning of deep networks from decentralized data. *In Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

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

[44] Yurii Nesterov *et al.* : *Lectures on convex optimization*, volume 137. Springer, 2018.

[45] Giambattista Parascandolo, Alexander Neitz, Antonio Orvieto, Luigi Gresele et Bernhard Schölkopf : Learning explanations that are hard to vary. *arXiv preprint arXiv:2009.00329*, 2020.

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

[47] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl et Christoph H. Lampert : icarl: Incremental classifier and representation learning. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[48] Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečnỳ, Sanjiv Kumar et H Brendan McMahan : Adaptive federated optimization. *arXiv preprint arXiv:2003.00295*, 2020.

[49] Sashank J Reddi, Jakub Konečnỳ, Peter Richtárik, Barnabás Póczós et Alex Smola : Aide: Fast and communication efficient distributed optimization. *arXiv preprint arXiv:1608.06879*, 2016.

[50] Nicola Rieke, Jonny Hancox, Wenqi Li, Fausto Milletari, Holger R Roth, Shadi Albarqouni, Spyridon Bakas, Mathieu N Galtier, Bennett A Landman, Klaus Maier-Hein *et al.* : The future of digital health with federated learning. *NPJ digital medicine*, 3(1):1–7, 2020.

[51] Herbert Robbins et Sutton Monro : A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, 22(3):400 – 407, 1951.

[52] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas et Aleksander Madry : How does batch normalization help optimization? *Advances in neural information processing systems*, 31, 2018.

[53] Jonathan Schwarz, Wojciech Czarnecki, Jelena Luketina, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu et Raia Hadsell : Progress & compress: A scalable framework for continual learning. *In International Conference on Machine Learning*, pages 4528–4537. PMLR, 2018.

[54] Zheyan Shen, Jiashuo Liu, Yue He, Xingxuan Zhang, Renzhe Xu, Han Yu et Peng Cui : Towards out-of-distribution generalization: A survey. *arXiv preprint arXiv:2108.13624*, 2021.

[55] Neta Shoham, Tomer Avidor, Aviv Keren, Nadav Israel, Daniel Benditkis, Liron Mor-Yosef et Itai Zeitak : Overcoming forgetting in federated learning on non-iid data. *arXiv preprint arXiv:1910.07796*, 2019.

[56] Dalwinder Singh et Birmohan Singh : Investigating the impact of data normalization on classification performance. *Applied Soft Computing*, 97:105524, 2020.

[57] Alexander Smola et Shravan Narayanamurthy : An architecture for parallel topic models. *Proc. VLDB Endow.*, 3(1–2):703–710, sep 2010.

[58] Sebastian U Stich : Local sgd converges fast and communicates little. *arXiv preprint arXiv:1805.09767*, 2018.

[59] Irene Tenison, Sai Aravind Sreeramadas, Vaikkunth Mugunthan, Edouard Oyallon, Eugene Belilovsky et Irina Rish : Gradient masked averaging for federated learning. *arXiv preprint arXiv:2201.11986*, 2022.

[60] Dmitry Ulyanov, Andrea Vedaldi et Victor Lempitsky : Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

[61] Jianyu Wang et Gauri Joshi : Cooperative sgd: A unified framework for the design and analysis of communication-efficient sgd algorithms. *arXiv preprint arXiv:1808.07576*, 2018.

[62] Shiqiang Wang, Tiffany Tuor, Theodoros Salonidis, Kin K Leung, Christian Makaya, Ting He et Kevin Chan : Adaptive federated learning in resource constrained edge computing systems. *IEEE Journal on Selected Areas in Communications*, 37(6):1205–1221, 2019.

[63] Yuxin Wu et Kaiming He : Group normalization. *In Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

[64] Chencheng Xu, Zhiwei Hong, Minlie Huang et Tao Jiang : Acceleration of federated learning with alleviated forgetting in local training. *arXiv preprint arXiv:2203.02645*, 2022.

[65] Gang Yan, Hao Wang et Jian Li : Critical learning periods in federated learning. *arXiv preprint arXiv:2109.05613*, 2021.

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

[67] Dezhong Yao, Wanning Pan, Yutong Dai, Yao Wan, Xiaofeng Ding, Hai Jin, Zheng Xu et Lichao Sun : Local-global knowledge distillation in heterogeneous federated learning with non-iid data. *arXiv preprint arXiv:2107.00051*, 2021.

[68] Jaehong Yoon, Wonyong Jeong, Giwoong Lee, Eunho Yang et Sung Ju Hwang : Federated continual learning with weighted inter-client transfer. *In International Conference on Machine Learning*, pages 12073–12086. PMLR, 2021.

[69] Hao Yu, Sen Yang et Shenghuo Zhu : Parallel restarted sgd with faster convergence and less communication: Demystifying why model averaging works for deep learning. *In Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 5693–5700, 2019.

[70] Friedemann Zenke, Ben Poole et Surya Ganguli : Continual learning through synaptic intelligence. *In International Conference on Machine Learning*, pages 3987–3995. PMLR, 2017.

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

[72] Zhuangdi Zhu, Junyuan Hong et Jiayu Zhou : Data-free knowledge distillation for heterogeneous federated learning. *In International Conference on Machine Learning*, pages 12878–12889. PMLR, 2021.

Appendix A

Additional heatmaps for local client forgetting

Appendix A shows additional forgetting heatmaps evaluated at different rounds

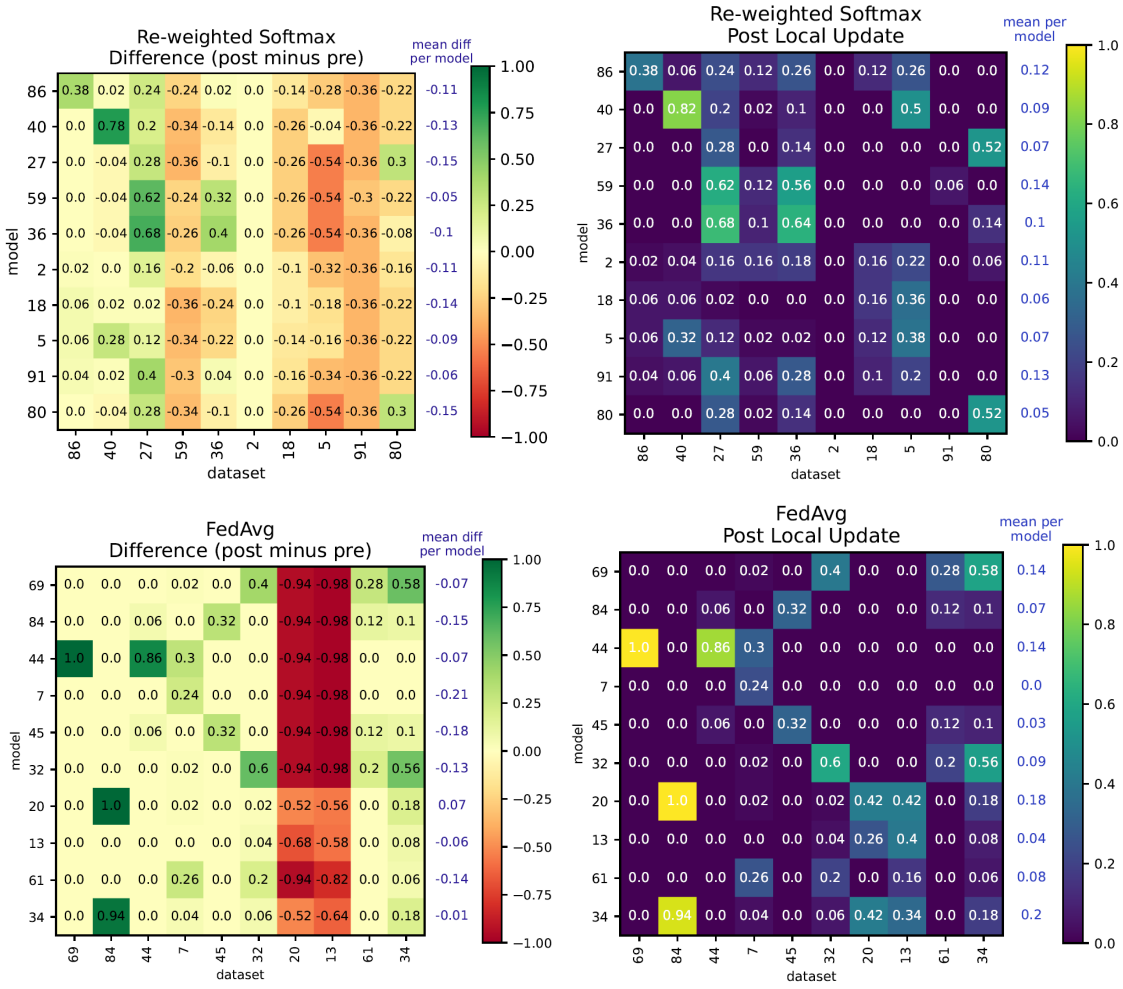

Fig. A.1. Local client forgetting for round 1 with and without WSM. The x axes contain the indices of clients selected for the round where each of the 100 clients have labels in the set $\{0, ..., 99\}$. The Difference heatmaps (LHS) show the difference in accuracy before and after local training when the k^{th} client's model is evaluated on the i^{th} client's dataset. The final column gives F_k, the average forgetting over all clients. The Post Local Update heatmaps (RHS) show the accuracy of each client's model on the other client's data.

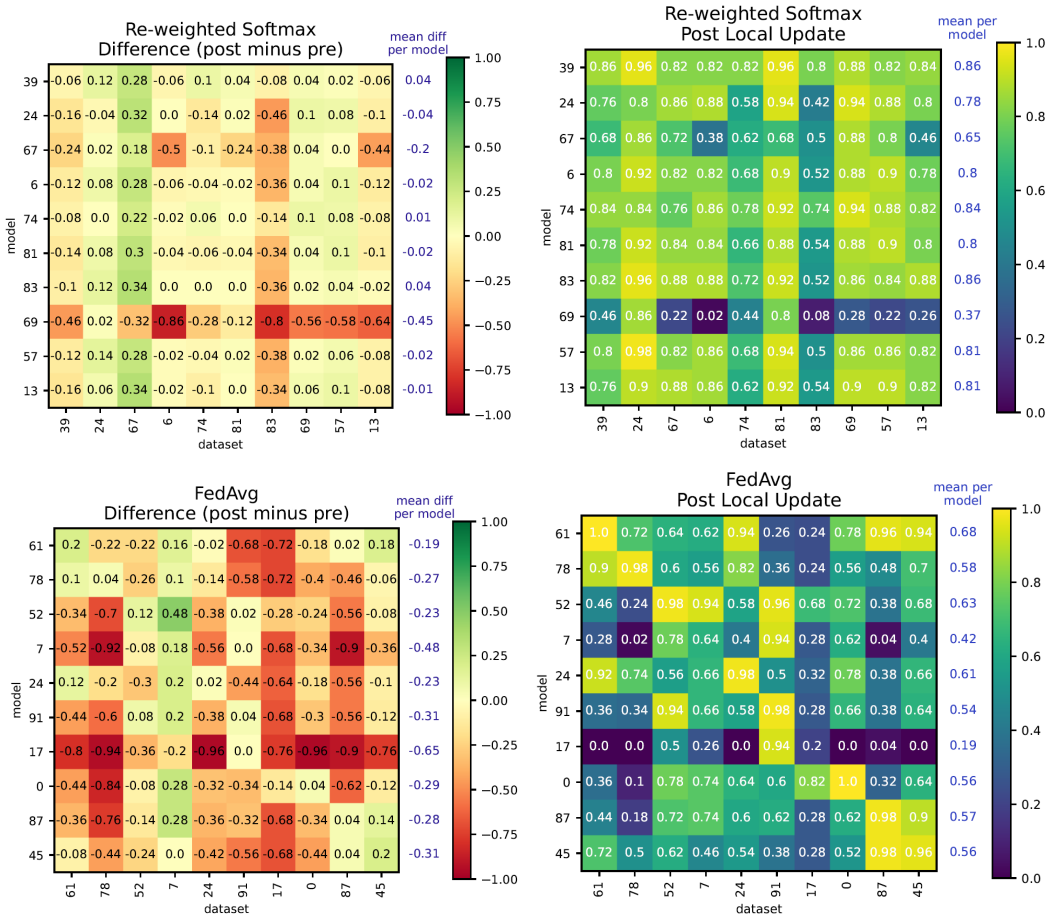

Fig. A.2. Local client forgetting for round 4000 with and without WSM. The x axes contain the indices of clients selected for the round where each of the 100 clients have labels in the set $\{0, ..., 99\}$. The Difference heatmaps (LHS) show the difference in accuracy before and after local training when the k^{th} client's model is evaluated on the i^{th} client's dataset. The final column gives F_k, the average forgetting over all clients. The Post Local Update heatmaps (RHS) show the accuracy of each client's model on the other client's data.

A.1. Round 4000

The heatmaps shown in figure A.2 are evaluated after the last round of training, round 4000. For the last round of training we again note higher accuarcy observed along the diagonal of the post local update for the FedAvg model not using WSM indicating a preference for their own local datasets and at times demonstrating significant forgetting on the datasets of other clients. With WSM we again observe tends in the accuracies such as one column with higher accuracies indicating a dataset that all local models were able to do well on or a row of lower accuracies indicating a model that did badly on all distributions including its own local distribution. These results are encouraging since they are present across local models and indicating forgetting is not the cause of the different performances..

Appendix B

ResNet-18 model used in experiments

The options for the ResNet18 model used in the experiments are shown explicitly in the model structure presented below:

```
ResNet(
    (conv1): Sequential(
        (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): ReLU(inplace=True)
    )
    (layer_1): Sequential(
        (0): ResidualBlock(
            (left): Sequential(
                (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
                (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,track_running_stats=True)
                (2): ReLU(inplace=True)
                (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
                (4): GroupNorm(2, 64, eps=1e-05, affine=True)                )
            (shortcut): Sequential()
        )
        (1): ResidualBlock(
            (left): Sequential(
                (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
                (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,track_running_stats=True)
                (2): ReLU(inplace=True)
                (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
                (4): GroupNorm(2, 64, eps=1e-05, affine=True)
            )
            (shortcut): Sequential()
```

```
        )
    )
    (layer_2): Sequential(
        (0): ResidualBlock(
            (left): Sequential(
                (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
                (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,track_running_stats=True)
                (2): ReLU(inplace=True)
                (3): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
                (4): GroupNorm(2, 128, eps=1e-05, affine=True)
            )
            (shortcut): Sequential(
                (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2))
                (1): GroupNorm(2, 128, eps=1e-05, affine=True)
            )
        )
        (1): ResidualBlock(
            (left): Sequential(
                (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
                (1):BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
                (2): ReLU(inplace=True)
                (3): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
                (4): GroupNorm(2, 128, eps=1e-05, affine=True)
            )
            (shortcut): Sequential()
        )
    )
    (layer_3): Sequential(
        (0): ResidualBlock(
            (left): Sequential(
                (0): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
                (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,track_running_stats=True)
                (2): ReLU(inplace=True)
                (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
                (4): GroupNorm(2, 256, eps=1e-05, affine=True)
            )
            (shortcut): Sequential(
```

```
            (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2))
            (1): GroupNorm(2, 256, eps=1e-05, affine=True)
        )
    )
    (1): ResidualBlock(
        (left): Sequential(
            (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,track_running_stats=True)
            (2): ReLU(inplace=True)
            (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (4): GroupNorm(2, 256, eps=1e-05, affine=True)
        )
        (shortcut): Sequential()
    )
)
(layer_4): Sequential(
    (0): ResidualBlock(
        (left): Sequential(
            (0): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
            (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,track_running_stats=True)
            (2): ReLU(inplace=True)
            (3): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (4): GroupNorm(2, 512, eps=1e-05, affine=True)
        )
        (shortcut): Sequential(
(0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2)) (1): GroupNorm(2, 512, eps=1e-05,
affine=True)            )
    )        (1): ResidualBlock(
        (left): Sequential(
            (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,track_running_stats=True)
            (2): ReLU(inplace=True)
            (3): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (4): GroupNorm(2, 512, eps=1e-05, affine=True)
        )
        (shortcut): Sequential()
    )
)
```

```
    (avgpool): AvgPool2d(kernel_size=(3, 3), stride=2, padding=0)
    (fc): Linear(in_features=512, out_features=10, bias=True)
)
```