With a Little Help from My Friends – A Comparative Study of Decentralized Deep Learning Strategies

Tom Hagander, Eric Ihre-Thomason

Master's thesis
2024:E28

LUND UNIVERSITY

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

With a Little Help from My Friends

– A Comparative Study of Decentralized Deep Learning Strategies

Eric Ihre-Thomason & Tom Hagander

Abstract

This thesis investigates various communication strategies and similarity metrics within decentralized deep learning (DL). Decentralized learning allows organizations or users to collaborate on improving personalized deep neural networks while maintaining the privacy of their datasets. When the distribution of data varies across participating users, this task becomes more challenging, as not all collaboration is beneficial. This underscores the need for effective algorithms and similarity metrics that can identify good collaborators without sharing private data.

Specifically, this study considers two main communication strategies: Decentralized Adaptive Clustering (DAC) and Personalized Adaptive Neighbor Matching (PANM). It utilizes diverse similarity metrics such as inverse training loss, cosine similarity of weights and gradients, and the inverse L_2 distance between weights. Different model merging protocols are also examined to provide a comprehensive analysis of DL strategies. Our research provides insights into the performance of these metrics and communication strategies, highlighting their potential for effective collaboration in DL and contributing to the development of robust DL methods.

Keywords— Decentralized Deep Learning, Privacy-Preserving Machine Learning, Collaborative Learning, Federated Learning, Similarity Metrics, Non-IID Data, Model Averaging

Acknowledgments

We would like to sincerely thank our supervisors Edvin Listo Zec, Sarunas Girdzijauskas, Erik Tegler and Niels Christian Overgaard for their mentorship. We greatly appreciate their valuable insights that kept this project on track, and how they valued and listened to what we had to say.

We would also like to thank everyone in the Deep Learning Research Group at RISE for a fun and stimulating work environment, and of course for the table tennis matches.

Many thanks also go out to Karro and Olle for making us feel at home in Göteborg, all the other master's thesis students at RISE for the lessons in geography and Per Hagander for being an inspiration throughout our studies.

"Oh, I get by with a little help from my friends"
— Ringo Starr

Contents

Chapter 1

Introduction

In recent years, machine learning has emerged as a leading research frontier in advancing technology and solving complex problems. Machine learning models have significantly enhanced efficiency and predictive accuracy in industries ranging from healthcare to finance, but are also impacting the daily life of most people through tools like chatbots and more personalized user experiences. Training these models requires vast amounts of data, which are collected from different sources, including mobile devices, business systems, and the internet. As the development of machine learning technologies progresses and the demand for higher quality and larger datasets increases, concerns about user and data privacy intensify. This highlights the necessity for machine learning methodologies that preserve data privacy.

Today national and international regulations such as the Data Protection Act [9] and GDPR [1], along with increasing user awareness, also push the issue of preserving user data privacy. Additionally, corporations and organizations often possess proprietary data that cannot be shared externally, further motivating the development of privacy-preserving frameworks. In healthcare, the ability to ensure data privacy is especially critical. Hospitals handle sensitive patient information that requires the highest levels of confidentiality and security. Implementing machine learning methods that preserve data privacy in such settings would circumvent concerns about data security while still tapping into the potential of machine learning to improve healthcare delivery and treatment outcomes.

Federated learning [6] has emerged as the most widely adopted framework for distributed deep learning, facilitating model training across multiple clients with private datasets. Federated learning consists of a global model that is trained locally on private datasets, before consolidating that information on a central server. This creates a model that is more capable than a model trained on any of the private datasets. While effective, federated learning's reliance on a central server is a point of vulnerability and can become a bottleneck as the client base expands. These limitations have sparked interest in fully decentralized systems, known as decentralized learning, where clients interact in a peer-to-peer network, eliminating the need for a central server. Decentralized learning offers improved scalability and inherent robustness, as it is less vulnerable to single-point failures or malicious attacks.

Decentralized learning is similar to federated learning in that it enables model training across multiple clients with private datasets. The key difference is that instead of using a central server that creates a global model the clients themselves create local models that communicate with each other without sharing their data. These models are specialized to their local datasets while still gaining insights from other devices in the network. The computational cost of training is distributed, facilitating improved learning by clients that do not necessarily have access to vast computational resources. By enabling collaborative model training without a centralized data pool, decentralized learning frameworks ensure that no single client (entity) holds extensive datasets. Instead, multiple stakeholders, such as businesses and healthcare providers, can securely contribute to and benefit from shared insights while maintaining strict compliance with privacy laws. This method protects sensitive data and fosters a more robust and inclusive data ecosystem.

Despite the many upsides of decentralized learning, it is a fairly new area of research and is not yet implemented in any real-world applications. It is unclear which communication strategies are most suitable for various problem settings, and which novel challenges and problems can arise when a machine learning problem is solved in a decentralized setting. To pave the way for decentralized learning to become a valuable tool in applications of machine learning, we investigate the following research questions.

1.1 Research Questions

- How well do different communication strategies solve the local objectives of each client?
- How well do different similarity metrics perform compared to each other, and how does the problem setting influence their performance?
- What problems can arise when solving a set of local objectives in a decentralized setting, and how can these be mitigated?

1.2 Aims

In this thesis, various decentralized learning methodologies are applied across different problems and datasets. The aim is to provide a nuanced and scientific evaluation of decentralized learning's performance on diverse tasks, while also offering insights into the inner workings of these methods. To achieve this, we implement state-of-the-art methods and algorithms along with novel variations and relevant extensions of these. We characterize performance of these methodologies by considering performance on each local task, as well as the volume of data that has to be transmitted in the network of clients and the computational complexity of the communication strategies implementing them.

1.3 Problem Formulation

We investigate the setting where data is distributed in relatively small volumes over several devices or servers, each of which has the goal of training some model on that data. To simulate the skewed and unbalanced nature of data distributed across a multitude of clients in real-world scenarios, we implement several different datasets and distribute these unevenly and in low volumes between clients. We refer to the devices or servers as *clients*, the data they have locally as their *local dataset*, and their model as their *local model.* Each client aims to produce a model which performs a given task well on the local dataset. In this work, we focus on image classification and regression as the given tasks. The local dataset of each client is too small to solve the given task well, creating the need to exchange information with other clients. Each client wishes to keep their local datasets private, meaning only model parameters and data such as model performance may be communicated between clients.

Previous works have employed *federated learning* to solve this problem, which creates a global model that aims to solve the tasks of each client [6]. Broadly, federated learning consists of training the model locally on a set of clients in parallel, which yields a set of new models. These models are then sent back to a central server, where the weights of each model are averaged (weighted by the size of the local dataset they were trained on) to produce a new global model. This process is then repeated several times, which finally yields a model trained on the local data of each client (an overview of a federated learning approach is shown in Figure 1.1b). This algorithm is described in greater detail in Section 2.2. This approach works well to increase the performance on the given tasks, compared to training a model on solely the local dataset, if the data is relatively similar between clients, but not when the data distribution differs too much between clients [11]. If the local datasets of clients are generated by very different underlying probability distributions, the weights of the models diverge too much during the local training step and averaging the weights of those models yields a global model that can perform worse on each local dataset than if the models were trained only on the local data.

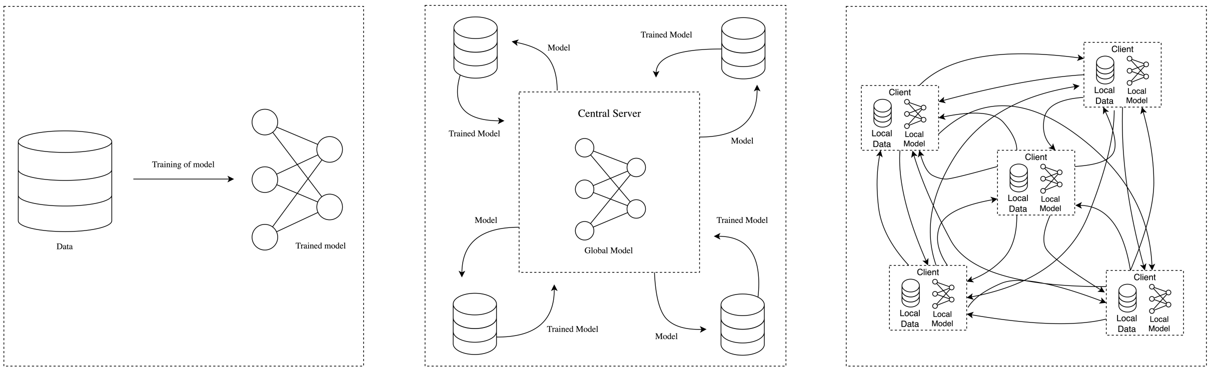
To address this, a decentralized approach can be used, which is referred to as *decentralized learning*. With decentralized learning, clients communicate their local models directly with each other, which allows them to choose which other clients are worth averaging models with (an overview of a decentralized learning approach is shown in Figure 1.1c). This also has several added benefits compared to federated learning. Each client now has a local model dedicated to solving their specific task, as opposed to a global model which needs to generalize to all tasks. Also, the need for a central server overseeing communication between clients is eliminated, further improving the privacy of a client's local data. However, the problem becomes more complex, as each client needs a *communication strategy* to determine which other clients they should communicate with. A client therefore needs a way to compare its dataset to that of other clients without having access to other datasets, which we refer to as a *similarity metric*.

In this study, we explore two different communication strategies from previous works called *Decentralized Adaptive Clustering* (`DAC`) [10] and *Personalized Adaptive Neighbor Matching* (`PANM`) [5]. For each client, `DAC` operates by measuring and estimating a client's similarity to other clients during communication, thereby creating a distribution of similarities among other clients. This distribution of similarities is then transformed into a probability distribution over all other clients, from which a client samples which clients to merge weights with that round. Since the client receives other clients' models in this step, that information is used to update the similarities of those clients. `PANM` operates similarly but divides the learning process into two stages. The first stage consists of random communication to collect information on other clients in the network. The second stage uses the information gained in the first stage to classify other clients into either "beneficial to communicate with" or "not beneficial to communicate with" while continuing to collect information. Both these communication strategies are described in greater detail in Section 2.4.

Each of these communication strategies uses a similarity metric to determine the similarity of local datasets between clients. Since the data is private, direct comparisons between the datasets cannot be employed. Instead, the comparisons researched in this study are based on the *models* of other clients, which are a representation of a client's local dataset but are still considered to be privacy-preserving enough to share. The four similarity metrics explored in this work are:

1. *Inverse training loss*
2. *Inverse L_2 distance of weights*
3. *Cosine similarity of gradients*
4. *Cosine similarity of weights*

The inverse training loss metric determines the similarity to another client by taking the inverse of the loss produced by the other client's local model on a client's local data. The inverse L_2 distance and cosine similarity of weights metrics determine similarity by taking the L_2 distance between the weights of two models and the cosine of the angle between the weights of two models, respectively. The cosine similarity of gradients metric is based on the similarity score of the cosine of the angle between the most recent training gradient of the two models. Again, these similarity metrics will be covered in greater detail in Section 2.5. In Figure 1.1 a summary of how traditional machine learning, federated learning and decentralized learning works is shown.



(a) Standard training of ML models

(b) Federated approach to training.

(c) Decentralized approach to training.

Figure 1.1: Visualizations of how ML models are trained traditionally, in federated learning, and in decentralized learning.

Chapter 2

Mathematical Formulation

2.1 Supervised Learning Using Neural Networks

Supervised learning is a type of machine learning where the goal is to learn a function $\hat{y} = F(x)$ from input-output pairs (x, y) in order to predict the output y as accurately as possible. Ideally, we aim to do this by minimizing the *loss*, $L(w)$, over the distribution $\mathcal{P}(x, y)$ which generates the input-output pairs (x, y). However, this distribution is usually unobservable, and we instead estimate F by minimizing the loss on a training set $\mathcal{D}_{train} = \{(x_1, y_1); (x_2, y_2); \dots\}$, which is assumed to be sampled from $\mathcal{P}(x, y)$. An effective way of achieving this for nonlinear functions is through neural networks.

A neural network is a parameterized function composed of parameters, or weights, organized in layers, where each layer progressively refines input data into meaningful output. Neural networks are versatile and can be adapted to many types of inputs x and outputs y. For example, the input data x could be an image, for which the output y could be a classification of that image. A neural network with d weights, with weights denoted w, produces output $\hat{y}$ given an input x, which can be written as

$$\hat{y} = F(x; w) \tag{2.1}$$

This output $\hat{y}$ can be compared to the true *label* y of the input x, to produce a loss. The objective is to find weights that minimize the loss of the prediction $\hat{y}$ on all examples (x, y). This can be formulated as the optimization problem

$$\min_{w \in \mathbb{R}^d} L(w) \quad \text{where} \quad L(w) \overset{\text{def}}{=} \frac{1}{n} \sum_{i=1}^{n} \ell(x_i, y_i; w), \tag{2.2}$$

where $\ell(x_i, y_i; w)$ is the loss of the prediction on example (x_i, y_i). In this study, we mostly have multiclass classification problems, for which the Cross-Entropy loss,

$$\ell(x_i, y_i; w) = -\sum_{c=1}^{C} \delta(y_i, c) \log(\hat{y}_{i,c}) \tag{2.3}$$

is suitable. Here, C is the number of classes to classify between, $\delta(y_i, c)$ is equal to one if (x_i, y_i) belongs to class c and zero otherwise, and $\hat{y}_{i,c}$ is the predicted probability that x_i belongs to class c. This loss is minimized if the model with complete certainty correctly predicts the class of each example in the dataset.

We also use the Mean Squared Error (MSE) Loss for regression tasks. In such tasks, y_i can take on values in a continuous interval, which means a different approach than Cross-Entropy is necessary. The MSE loss can be formulated as

$$\ell(x_i, y_i; w) = (y_i - \hat{y}_i)^2, \tag{2.4}$$

where similarly $\hat{y}_i$ is the predicted label of the model.

To solve the optimization problem in (2.2), most successful applications of deep learning use various variants of Stochastic Gradient Descent (SGD). SGD typically involves iterating over random subsets of data called *batches* to estimate the gradient of the objective function with respect to the weights, denoted by $\nabla_w f(w)$. One iteration over the complete dataset is called an *epoch*. The estimated gradient is used to update the weights in the direction that potentially lowers the loss. The update formula can be expressed as

$$w^{t+1} = w^t - \eta \nabla_w f(w^t), \tag{2.5}$$

with η being the *learning rate*, which can be tuned for each specific problem. This update rule has many more advanced variations, but this serves as a simple example of the concept of SGD.

In the setting of this work, the data is distributed over a set of clients in a network. The set of all clients is denoted as $\mathcal{S}$, and the local dataset of a client i is denoted as $\mathcal{D}_i = \{(x_n, y_n)\}_{n=1}^{N_i}$, where N_i is the number of data points in $\mathcal{D}_i$. $\mathcal{D}_i$ is sampled from a generating probability distribution $\mathcal{P}_i(x, y)$, which may differ between clients. Because all data is not available to any model, the traditional approach of repeatedly iterating over subsets of the data to optimize the weights using SGD is impossible. One could get close to the traditional approach by training a model on a client, sending that model to another client and training it there, and repeating this for many rounds for all clients in $\mathcal{S}$. However, this becomes prohibitively expensive in terms of communication as the number of clients in the network becomes large, and does not allow for each client to have a model that is specialized to their specific optimization problem.

2.2 Federated Learning

A method that has previously been used to preserve data privacy is federated learning which aims to make use of all the data of all the clients while keeping the communication costs low [6]. Federated learning consists of a central server that has a model w, in addition to the network of clients. In each round of training t, the server sends out the model w^t to a random subset of clients $\mathcal{S}_t$ which are indexed by k, which train the model locally to produce a set of new models w_k^{t+1}. The local training for a client i consists of performing SGD to optimize the local objective

$$\min_{w \in \mathbb{R}^d} f_i(w) \quad \text{where} \quad f_i(w) \overset{\text{def}}{=} \frac{1}{n} \sum_{(x,y) \sim \mathcal{P}_i} \ell(x, y; w). \tag{2.6}$$

The models produced from this are then sent back to the central server, where they are averaged, weighted by the sizes of their local datasets, producing a new global model. We call this averaging `FedAvg`, and it can be written as

$$w^{t+1} = \sum_{k \in S_t} \alpha_k w_k^{t+1} \quad \text{where} \quad \alpha_k = \frac{N_k}{N} \quad \text{and} \quad N = \sum_{k \in S_t} N_k, \tag{2.7}$$

where N_k is the number of data points of client k. This process is repeated until training is completed. There are other ways of defining how the averaging is weighted through different choices α_k, another way is described in Equation (2.12). The federated learning algorithm is described in Algorithm 1.

Algorithm 1 `Federated Learning` m is the number of clients sampled in each round, B is the local minibatch size, E is the number of local epochs, and η is the learning rate.

```
Server executes:
initialize $w^0$
for each round $t = 1, 2, \dots$ do
    $\mathcal{S}_t \leftarrow$ (random set of $m$ clients)
    for each client $k \in \mathcal{S}_t$ in parallel do
        $w_k^{t+1} \leftarrow$ ClientUpdate$(k, w^t)$
    $N \leftarrow \sum_{k \in S_t} N_k$
    $w^{t+1} \leftarrow \sum_{k \in S_t} \frac{N_k}{N} w_k^{t+1}$                ▷ Model averaging by FedAvg


procedure ClientUpdate$(k, w)$:                                       ▷ Run on client $k$
    $B \leftarrow$ (split $\mathcal{D}_k$ into batches of size $B$)
    for each local epoch $i$ from 1 to $E$ do
        for batch $b \in \mathcal{B}$ do
            $w \leftarrow w - \eta \nabla l(w; b)$                          ▷ $\eta$ is the learning rate
    return $w$ to server
```

The local optimization problem can be formulated as minimizing the loss based on the generating distribution $\mathcal{P}_i(x, y)$ for each client i. If the data distributions $\mathcal{P}_i(x, y)$ differ significantly among clients, the weights sent to the server will vary correspondingly, and `FedAvg` may yield a global model that performs poorly on many if not all local optimization tasks. This is illustrated in Figure 2.1 This implies that the effectiveness of federated learning might decrease when data is not independent and identically distributed (IID) across clients. One study indicates that the accuracy of a model trained with federated learning could decrease by up to 55% when clients' data is non-IID [11]. Conversely, if the data is IID across clients, then $\mathbb{E}_{\mathcal{P}_i}[f_i(w)] = \mathbb{E}_{\mathcal{P}_j}[f_j(w)]$ for all client pairs (i, j), indicating that each local objective would be effectively the same.

2.3 Decentralized Learning

The concept of federated learning can be expanded into a decentralized setting, which eliminates the need for a central server to coordinate the learning process. In this scenario, each client i maintains a local model w_i, aimed at solving their local optimization problem:

$$\min_{w_i \in \mathbb{R}^d} f_i(w_i) \quad \text{where} \quad f_i(w_i) \overset{\text{def}}{=} \frac{1}{n} \sum_{(x,y) \sim \mathcal{P}_i} \ell(x, y; w_i). \tag{2.8}$$

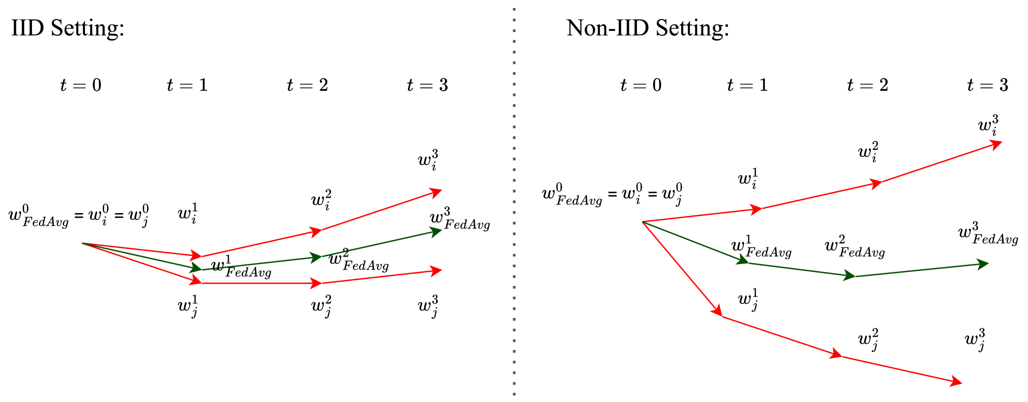


Figure 2.1: An example of two clients i and j training on local data with the model that is obtained through `FedAvg` of the clients in IID and non-IID settings. In the non-IID setting, weights diverge more significantly, meaning the averaged model is further from the optimum for the local objectives.

Clients independently train their models on their respective local datasets. Subsequently, they receive local models from clients from a subset of $\mathcal{S}$, averaging these models with their own using the `FedAvg` algorithm. This process is repeated for x *communication rounds* until training concludes. Each client autonomously determines which other clients to interact with through a defined *communication strategy*. We describe the process of obtaining a model from another client as *communicating* with that client. By facilitating communication between clients with similar data distributions, the challenges associated with non-IID local datasets can be mitigated. This allows a client to tailor its model more closely to its local dataset while still benefiting from the broader data diversity within the network. A key difference between decentralized learning and federated learning is that for federated learning there is one global model and for decentralized learning each client has their own local model.

2.4 Communication Strategies

The goal of a communication strategy is to find clients with similar local data distributions and communicate with these clients more frequently. Each communication strategy has a hyperparameter $n_{sampled}$ which is how many other clients a client communicates with each communication round.

2.4.1 Baselines: Random, Oracle, and No Communication

In this study, we create non-IID data by dividing clients into several *clusters* of clients, where all clients in a cluster have data from the same generating distribution $\mathcal{P}_i(x, y)$ which differs from the generating distributions of other clusters. To analyze how well a decentralized communication strategy works we use three different baselines strategies to compare with.

The first baseline is the simplest communication strategy which is to pick clients to communicate with at random. This serves as a floor for all of our experiments; if a communication strategy does not outperform random communication it is not an effective strategy. Another baseline we use to determine how well communication strategies perform is oracle communication where each client is aware of which cluster it belongs to, and therefore only communicates within that cluster. Theoretically, it is possible for a client using another communication strategy to outperform oracle if it is beneficial to communicate outside one's cluster, but it serves as a baseline for what a good communication strategy is. A client could benefit from communicating outside the cluster it belongs to if other clusters are very similar or if its cluster is too small. Our third baseline is no communication, which is when a client only trains on its local dataset without interacting with the network.

2.4.2 Decentralized Adaptive Clustering (DAC)

`DAC` is a communication strategy that takes a similarity metric and forms a set of measured and estimated similarities to other clients [10]. These similarities are used as a basis for choosing which other clients to communicate with. As more and more communication rounds passes, new measurements are collected and better estimates are formed, allowing clients to make more informed decisions on which other clients to communicate with.

For client i, `DAC` starts the first round of communication by communicating randomly in the network (performing `FedAvg` with these randomly selected clients). Similarities to these clients are measured and stored locally in client i in a key-value map we call s_i. Each client k that was communicated with, in addition to sending their local models to client i, also shares their own similarities s_k, and client i forms the temporary set of tuples (client, similarity) $s_{twostep}$, which is a concatenation of all s_k. If $s_{twostep}$ contains information on clients not in s_i, s_i is updated with the sampled clients' information on those clients. If there are multiple two-step connections to a client, say client m, meaning there multiple tuples in $s_{twostep}$ that contain client m, s_i is updated with the estimate of the client k that has the highest similarity to client i. An example of this is illustrated in Figure 2.2.

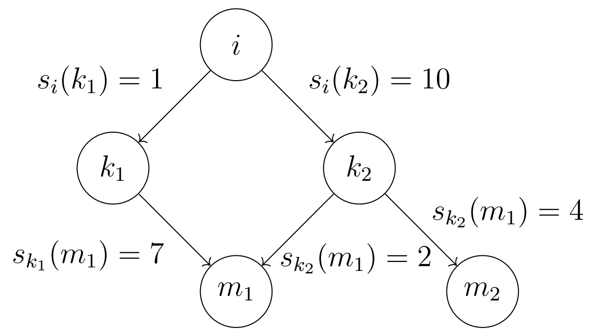


Figure 2.2: An example of two-step similarity estimation in `DAC`. Here, clients k_1 and k_2 are sampled by client i. Client i measures its similarity to these, but does not have any information on m_1 and m_2, while k_1 and k_2 do. Since the path $i \to k_2 \to m_2$ is the only path from i to m_2, client i will set $s_i(m_2) = s_{k_2}(m_2) = 4$. From i to m_1 there are two paths, and since $s_i(k_2) = 10 > 1 = s_i(k_1)$, client i will set $s_i(m_2) = s_{k_2}(m_2) = 2$. This estimation of similarities allows a client to quickly gain an understanding of its similarities to other clients in the network.

Now that client i has an updated set of similarities s_i, these are used to create *priors* p_i for all clients in $\mathcal{S}$. If $\mathcal{S}$ is indexed by j, $p_i(j)$ is assigned as follows:

$$\hat{p}_i(j) = \begin{cases} \exp(s_i(j) \cdot \tau) & \text{if } j \in s_i, \\ 0 & \text{otherwise.} \end{cases} \tag{2.9}$$

Here, τ is a temperature hyperparameter, defined as the inverse of what a temperature is in typical other literature, but referred to as 'temperature' for consistency in this study. p_i is then created as a normalized version of $\hat{p}_i$ to reflect that it is a probability distribution with

$$p_i = \frac{\hat{p}_i}{\sum_{j=1}^{|\mathcal{S}|} \hat{p}_i(j)}. \tag{2.10}$$

This is essentially a softmax function on all existing values in s_i where $p_i(i)$ is always zero because a client cannot sample itself. Our implementation of this softmax function differs slightly from this definition. To avoid numerical instability, we subtract the lowest value of $s_i(j) \cdot \tau$ from all such values before performing the exponential. Since the softmax function is invariant to constant shifts ($\text{softmax}(x) = \text{softmax}(x + c)$), this is equivalent to the definition above but avoids $\hat{p}_i$ taking values that are too large for the computer to handle. Also, we have encountered the situation where p_i becomes a vector of zeros with a single one, which can occur for large values of τ. This becomes a problem when p_i is used to sample more than one new client. To mitigate this, we add a small constant $\epsilon = 10^{-6}$ to each element in p_i and re-normalize. The priors p_i are used in the next round of communication to sample new clients to communicate with. Before each round of communication and after the last one, each client trains its model locally.

In this study, we also introduce a variation called `DAC with Min-Max scaling`, which is the same communication strategy as `DAC`, except for re-scaling the similarities s_i of a client i to the interval $[0, 1]$ before the softmax function is applied to produce p_i. This adjustment causes the optimal value of the hyperparameter τ to be more consistent across different similarity metrics, which can give similarities in different ranges of $\mathbb{R}$. Also, we have seen that the distribution of similarities $s_i(j)$ can differ significantly between early and later rounds of communication. `DAC with Min-Max scaling` can allow for a single value of τ to be more optimal during all rounds of communication. Other work has addressed this with other variations of `DAC`, such as `DAC-var` [10].

Averaging model parameters based on similarity (`FedSim`)

We also introduce `FedSim`, a variation of `FedAvg`. While `FedAvg` averages the weight of models based on the size of the local datasets they were trained on, `FedSim` instead weights the averaging by the similarity of the local model to the other models in the average. The local model is given a weighting equal to the maximum of the weightings of the other sampled models. `FedSim` of client i can be expressed as

$$w_i^{t+1} = \alpha_0 w_i^t + \sum_{k \in S_t} \alpha_k w_k^t \tag{2.11}$$

$$\text{where} \quad \alpha_k = \begin{cases} N^{-1} \cdot \max_{l \in S_t} p_i(l) & \text{if} \quad k = 0, \\ N^{-1} \cdot p_i(k) & \text{otherwise} \end{cases} \tag{2.12}$$

$$\text{and} \quad N = \max_{k \in S_t} p_i(k) + \sum_{k \in S_t} p_i(k). \tag{2.13}$$

Here S_t is the set of sampled clients, indexed by k (and $p_i(k)$ is the probability that client i samples client k). The idea of `FedSim` is to be more effective in adversarial settings, where merging model weights with the wrong model can be very disadvantageous. If a client samples another client that it has a low similarity with, which can happen in the early rounds of communication when a client has little information about the network, `FedSim` ensures that the sampled model has a low influence on the weights of the local model. Since the baseline communication methods do not have similarities or priors, `FedSim` cannot be applied to these. A communication strategy using `FedSim` can still be compared to the baselines.

2.4.3 Personalized Adaptive Neighbor Matching (PANM)

Another communication strategy is `PANM`, which stands for *Personalized Adaptive Neighbor Matching* [5]. `PANM` is divided into two phases and the first phases is the same as the algorithm as the `PENS` (*Performance-Based Neighbor Selection*) algorithm [7]. Then after T_1 communication rounds (T_1 is a hyperparameter) it changes into another algorithm, where it uses the clients identified in the first phase to make estimates on which clients are in its own cluster. `PANM` just like `DAC` uses a similarity metric to estiamte how similar it is to other clients. The details of how `PANM` works is described in appendix B.

2.5 Similarity Metrics

Both `DAC` and `PANM` use similarity metrics to determine similarity between clients. As previously mentioned, because of the private nature of the local datasets of other clients, a similarity metric can only be based on a client's local data, the models of other clients, and the model of the client itself. In this study, four different similarity metrics are explored and tested.

2.5.1 Inverse Loss

A proven way [10] to measure similarity between clients is to take the inverse of the loss produced when a model from client j predicts on the local dataset of client i, and can be expressed as

$$s_i(j) = \left(\sum_{(x,y) \in \mathcal{D}_i} \ell(x, y; w_j) \right)^{-1}. \tag{2.14}$$

Since only client i has access to that dataset, this has to be performed on client i. The idea is that a model w_j is similar to a model w_i if it performs well on the data on which w_i was trained. If it performs well, the loss is low, meaning the similarity is high. In figures and tables inverse loss is referred to as *Inv loss.*

2.5.2 Cosine Similarity of Gradients

To estimate a similarity between two clients i and j the similarity of the gradients of their model weights is of interest. If both models evolve in the same direction in parameter space, it could be an indication that they are trained on similar data and that merging could be beneficial. One way to do this is by looking at the cosine similarity between these high-dimensional gradients. Several gradients can be considered, and in this study, we explore the most recent gradient of the local training and the change in weights since the initialization of the models.

The most recent gradient of the local training can be expressed as $g^t_{1,i} = w^t_i - w^{t-1}_i$ and $g^t_{1,j} = w^t_j - w^{t-1}_j$ for clients i and j, where w^t_i are the current weights of client i:s model and w^{t-1}_i are the model weights before the last local training. w^t_j and w^{t-1}_j are the same but for client j. We can now formulate the cosine similarity of clients i and j based on the most recent local training, defined as $\cos\theta^1_{i,j}$, as

$$\cos\theta^1_{i,j} = \frac{\langle\, g^t_{1,i}, g^t_{1,j}\rangle}{||g^t_{1,i}|| \cdot ||g^t_{1,j}||}. \tag{2.15}$$

Similarly, the change in weights since the initialization of the models can be expressed as $g^t_{2,i} = w^t_i - w^0_i$ and $g^t_{2,j} = w^t_j - w^0_j$ for clients i and j, respectively. The cosine similarity based on the change in weights since the initialization of the models $\cos\theta^2_{i,j}$ can be expressed as

$$\cos\theta^2_{i,j} = \frac{\langle\, g^t_{2,i}, g^t_{2,j}\rangle}{||g^t_{2,i}|| \cdot ||g^t_{2,j}||}. \tag{2.16}$$

The similarity metric is can now be defined as

$$s_i(j) = s_j(i) = \alpha\cos(\theta^1_{i,j}) + (1-\alpha)\cos(\theta^2_{i,j}) \tag{2.17}$$

where α is a hyper-parameter, weighting the two cosine similarities. Notice that this similarity metric is symmetric, unlike inverse training loss. Cosine similarity of gradients with $\alpha = 1$ is referred to as *Cos grad* in tables and figures.

2.5.3 Cosine Similarity of Weights

This similarity metric is similar to the cosine similarity of gradients and can be expressed as

$$s_i(j) = s_j(i) = \frac{\langle\, w^t_i, w^t_j\rangle}{||w^t_i|| \cdot ||w^t_j||}. \tag{2.18}$$

The idea behind this similarity metric is similar to the one behind the cosine similarity of weights. In tables and figures, the cosine similarity of weights is referred to as *Cos weight.*

2.5.4 Inverse L_2-Distance Between Weights

This similarity takes the inverse of the L_2 distance between the weights of two models. The similarity between clients i and j can be written as

$$s_i(j) = s_j(i) = \left(||w_i - w_j||_2\right)^{-1}. \tag{2.19}$$

If two models are similar, the distance between them in the parameter space will be low, which leads to the similarity being high. The inverse L_2-distance between weights is referred to as *L2* in tables and figures.

2.6 Datasets, Shifts and Generating Distributions

In this work, we have used several different datasets and model architectures to simulate settings where data is distributed across several clients. To simulate the non-IID nature of the data in real-world applications of decentralized learning, we have created several different dataset *shifts*, which allow clients to have different generating distributions of their data, $\mathcal{P}_i(x, y)$. We have employed *label shifts*, *covariate shifts*, and *concept shifts*. With a label shift, the distribution of labels y differs between clusters, meaning different clients only have data with different subsets of the set of all available labels for a classification problem. For example, if the dataset has 10 labels to classify between, a client may only have examples from that dataset that contain the first four labels. The local optimization problem then becomes minimizing the loss on data with only those four labels. It is important to note that the client is not aware of this, and its local model will still have 10 outputs, one for each label in the dataset. The probability that a client has a data point (x, y) can be written as

$$\Pr(x, y) = \Pr(x|y)\Pr(y), \tag{2.20}$$

according to the chain rule of probability. With a label shift, $\Pr(y)$ differs between clients, while $\Pr(x|y)$ does not. In a real-world setting, an example of label shift is that some smartphone users have more pictures of, say, people, while others have more pictures of, say, cats. With a covariate shift, each client has examples (x, y) from the dataset with all available labels, but the covariates x (in our case images x) are transformed in ways that differ from those of other clients. Again, with a slightly varied use of the chain rule of probability, the probability that a client has a data point (x, y) can be written as

$$\Pr(x, y) = \Pr(y|x)\Pr(x). \tag{2.21}$$

With a covariate shift, $\Pr(x)$ differs between clients and $\Pr(y|x)$ does not. Again, the local optimization problem becomes correctly identifying labels for covariates with these transformations. In this work, we simulate covariate shifts by letting different clients have training data with images rotated by different amounts. In a real-world setting, an example of covariate shift could be patient data from different parts of the world, where the characteristics of patients may differ but the same diseases are prevalent. With a concept shift, $\Pr(y|x)$ differs between clients while $\Pr(x)$ does not. In the next section, we describe the shifts employed for each of our datasets in greater detail.

Chapter 3

Experimental Setup

To run decentralized learning architectures we simulate a distributed decentralized system by running our clients' training and information exchange sequentially. This was implemented using Python [2], with Pytorch [8] to create and train the neural networks. All code is available on GitHub at `github.com/tomhagander/decentralized_deep_learning`. The training of the models was done on two GPUs, one NVIDIA GeForce RTX 3090 and one NVIDIA GeForce RTX 2080 Ti.

All client models are initialized with the same weights and are given a local training dataset and a local validation dataset. The training dataset is used to train models locally, and the validation dataset is used for validation after each local training and each merging of models. A round of local training starts off the training. This is followed by many *communication rounds* until training is completed. We define a communication round as a round of *information exchange* with $n_{sampled}$ clients (the number of clients a client merges with is a hyperparameter) repeated for each client, followed by a round of local training for each client. Each client stops local training and `FedAvg`/`FedSim` if it has not improved its validation loss in 50 communication rounds. When this happens, that client continues to share its local model that achieved the highest validation accuracy with other clients that request it.

When training is concluded, each client tests its model that achieved the highest validation accuracy on an unseen test dataset, taken from the same generating distribution as the training and validation datasets. The average validation accuracy of clients is used to tune hyperparameters, while the average testing accuracy is used as a final measure of how well the clients were trained to solve their local objectives.

3.1 Experiment Settings - Datasets, Shifts, and Network Architectures

All our different experiment settings are summarized in Table 3.1. A general description of each setting and the neural networks used is given in this section. For all experiment settings, each client has a local dataset where each example is unique and not present in the local dataset of any other client. However, the generating distribution of a client is the same for all other clients in its cluster. A more detailed overview of all the settings and hyperparameters used for each experiment, as well as a complete description of the neural networks that were used is given in Appendix A.

Dataset	Shift	Local model	No. of clusters	No. of clients
CIFAR-10	Label	Custom CNN 1	2	100
CIFAR-10	Label	Custom CNN 1	5	100
Synthetic data	Concept	Linear regression	3	99
Fashion-MNIST	Covariate	Custom CNN 2	4	100
Fashion-MNIST + MNIST	Concept	Custom CNN 2	2	100
CIFAR-100	Label	ResNet-18	3	52

Table 3.1: An overview of all different experiments, each row corresponds to one experiment setting.

3.1.1 CIFAR-10 Animal/Vehicle Label Shift

For the CIFAR-10 label shift experiments, we use the CIFAR-10 dataset which consists of 60000 32x32 color images in 10 classes. The classes can be split into two groups, vehicles, and animals, with 4 and 6 corresponding labels respectively. We create 100 clients that train locally and communicate over 300 rounds. The 100 clients are split into two clusters with 40 and 60 clients each. The smaller cluster only has data points that are labeled as vehicles and the larger cluster only has data points labeled as animals. Each client has 400 data points for training and 100 for validation. This is the same cluster arrangement as in [10]. We do this to be able to compare our new methodologies to existing experiments in the literature (specifically `DAC` with `FedAvg` and using inverse loss as the similarity metric). Also, the architecture of the local model of each client is the same as in [10], which is a small convolutional neural network with two convolutional layers.

3.1.2 CIFAR-10 5 Cluster Label Shift

CIFAR-10 was also split into 5 clusters for our CIFAR-10 5 cluster experiments. For these experiments, we used 100 clients equally split into groups of 20 clients. Each cluster only received data with two different labels and there was no overlap of labels between clusters. Each client has 400 images for training and 100 for validation, and the experiments ran for 300 communication rounds. The model architecture is the same as for the other CIFAR-10 shift. This experiment setting aims to study how communication strategies and similarity metrics perform when there are more clusters and a random communication strategy is less likely to select the correct cluster by chance.

3.1.3 Synthetic Problem

To create an adversarial synthetic problem we use linear regression. Here we assume that each cluster is defined by its clients having data on the form $y_i = \langle x_i, \theta_j^* \rangle + \varepsilon_i$, for client i and cluster j where ε_i denotes normally distributed noise independent of the data x. Essentially, each cluster has an underlying data generating process defined by θ_j^* which a model for that cluster aims to learn from the data. The model is a single fully connected linear layer, aiming to imitate the data generation process which is equivalent to linear regression. If θ_j^* varies significantly across clusters, merging models from different clusters would lead to suboptimal models.

To construct the clusters, we generate three distinct $\theta_j^* \in \mathbb{R}^d$, each drawn from a uniform distribution. The feature matrix $X \in \mathbb{R}^{n \times d}$ is populated by entries sampled uniformly from the range $[-10, 10]$, forming a d dimensional vector with n samples for each client. Finally, the response variable y_i is computed as $y_i = \langle x_i, \theta_j^* \rangle$. We use mean squared error (MSE) as the loss function, setting $d = 10$ and $n = 50$ for each client. This synthetic setup enables a controlled exploration of the efficacy of similarity metrics in a setting characterized by distinct, non-overlapping data distributions.

3.1.4 Fashion-MNIST Covariate Shift

The Fashion-MNIST dataset contains 70000 28x28 grayscale images of fashion products from 10 categories. For our experiments using Fashion-MNIST, the clients were split into four clusters of different sizes, where each cluster is characterized by a specific rotation angle of their images x. This setup not only tests the impact of a covariate shift such as image rotation in a decentralized learning setting but also incorporates varying cluster sizes to mimic realistic client distributions: 70, 20, 5, and 5 clients per cluster. The rotation degrees set for these clusters are $0°, 180°, 10°$ and $350°$, respectively. This arrangement creates three clusters with relatively similar conditions and one notably distinct cluster - the $180°$ rotation. We run this experiment for 300 communication rounds. The neural network used for this setting is a small CNN with two convolutional layers, described in greater detail in Appendix A.

3.1.5 Fashion-MNIST and MNIST Combined

To create an experiment setting with more significant data shifts, we combined the Fashion-MNIST dataset with the MNIST dataset. The MNIST dataset contains 70000 28x28 grayscale images of handwritten numbers. It has ten labels corresponding to the numbers from zero to nine. The clients are split into two clusters of 50 clients for a total of 100 clients. Each cluster only gets data from one of the datasets (Fashion-MNIST or MNIST), where the labels from Fashion-MNIST are upshifted by 10 to not collide with the labels from MNIST. All clients have 20 outputs since they are not aware of which cluster they belong to. We run this experiment for 300 communication rounds. Both the same CNN as the one used for the Fashion-MNIST covariate shift experiments and a Multi-Layered Perceptron is used for this setting.

3.1.6 CIFAR-100 Label Shift

A more difficult version of CIFAR-10 is the CIFAR-100 dataset, which contains images similar to those of CIFAR-10 but with 100 different labels. This experiment aimed to investigate how well decentralized learning strategies hold up for more difficult problems like this one, and how initializing local models with a pretrained model influences performance. This is relevant, because in a real-world implementation of a decentralized learning framework, it would probably be beneficial for learning to initialize models with pretrained weights, should those exist. We therefore use ResNet-18 for this setting, which is a deep convolutional neural network with 18 layers (17 convolutional layers and 1 fully connected layer at the end). ResNet-18 has approximately 11.7 million trainable parameters and is suitable for image classification tasks. The pre-trained ResNet-18 is trained on the ImageNet dataset [4]. ImageNet is a large and widely used dataset in the field of computer vision, containing over 14 million images categorized into over 20,000 categories.

The shift implemented for this setting is a label shift into three clusters of sizes 26, 13, and 13 for a total of 52 clients. We decreased the number of clients in these experiments because training the ResNet-18 and storing multiple versions of it is more computationally demanding than for the networks in the other settings. Exact documentation of which labels of CIFAR-100 are included in each cluster can be found in Appendix A.

Chapter 4

Experimental Results

4.1 Exploring Label Shift with the CIFAR-10 Dataset

We first apply the different decentralized learning strategies on the CIFAR-10 dataset, with the Animal/Vehicle label shift. After hyperparameters were tuned for each communication strategy and similarity metric (the details of which can be found in Appendix A), three runs across different random seeds were produced, and the average testing accuracies of these runs are presented in Table 4.1 and Figure 4.1. For random, oracle and no communication there is no `FedSim` since there are no priors $p_i(k)$.

Communication strategy & Similarity metric		Cluster 1 Animals		Cluster 2 Vehicles		Mean	
		FedAvg	FedSim	FedAvg	FedSim	FedAvg	FedSim
Baselines	Random	52.88	-	69.94	-	59.71	-
	Oracle	54.39	-	74.05	-	62.25	-
	No comm	30.14	-	49.21	-	37.77	-
DAC	Inv loss	53.97	53.77	73.10	72.05	61.62	61.08
	Cos grad	52.28	51.06	68.62	68.73	58.81	58.13
	Cos weight	51.96	52.02	69.90	71.11	59.14	59.66
	L2	52.14	50.62	70.39	68.15	59.44	57.63
PANM	Inv loss	54.13	-	73.85	-	62.02	-
	Cos grad	46.73	-	61.26	-	52.54	-

Table 4.1: Cluster-wise test accuracies for different communication strategies, similarity metrics, and averaging methods on the CIFAR-10 Animal/Vehicle label shift setting.

As can be seen in the results, the random communication baseline performs well compared to the no communication baseline, but falls a few percentage points short of the testing performance of oracle communication, meaning there is performance to be gained from a good communication strategy. Only inverse loss as a similarity metric with `DAC` or `PANM` has test accuracies that can be considered better than random, even though that result is not significant over the experiment runs presented.

In Figures 4.3, 4.4 and 4.5 the similarity score of each client in a cluster to all other clients is plotted, along with the average validation accuracy per cluster before and after merging, over each communication round. The similarity score plotted is a true measurement of the similarity, and not the estimated and outdated similarity measurements a client actually possesses from the communication strategy. This information is not used in the communication between clients,

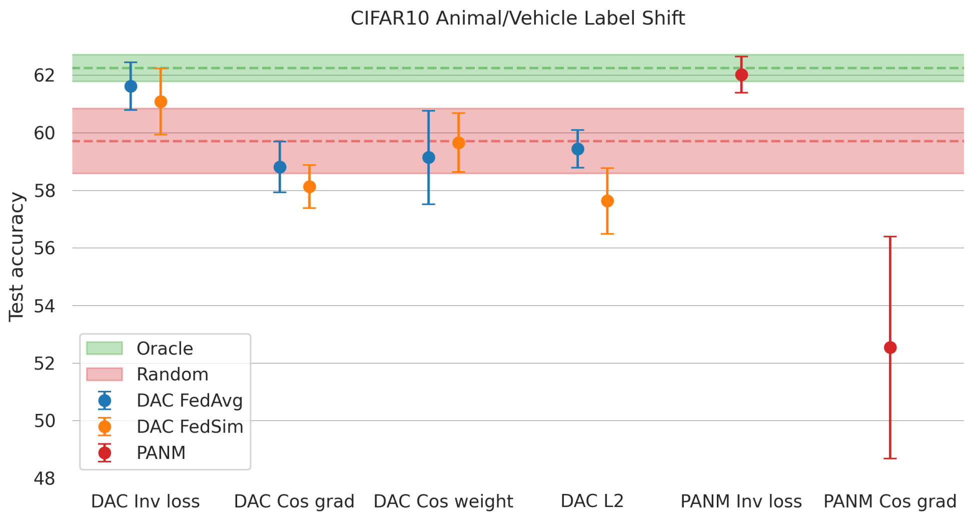
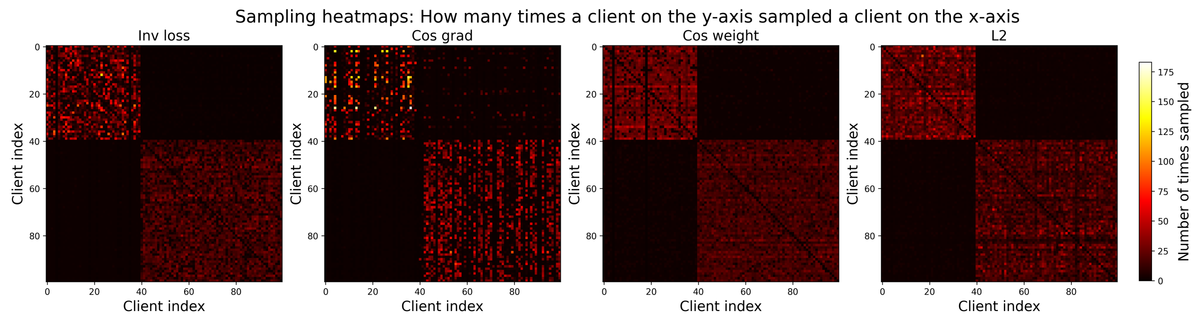


Figure 4.1: Test accuracies for different communication strategies, similarity metrics, and averaging methods on the CIFAR-10 Animal/Vehicle label shift setting.

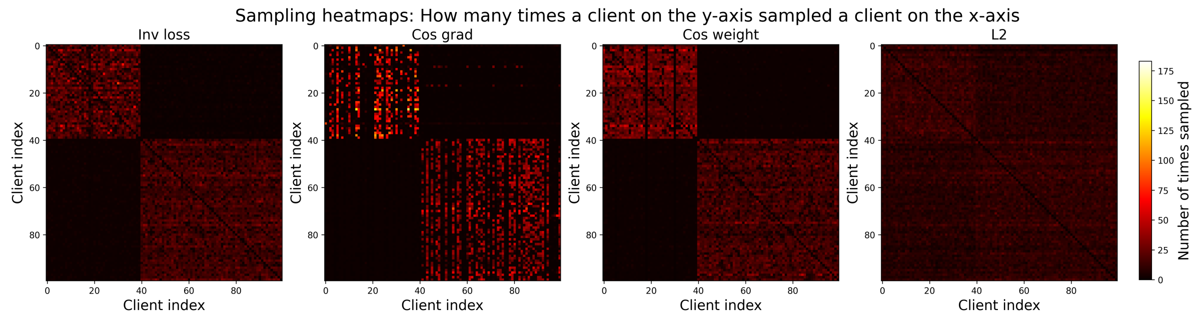
but used for analyzing and evaluating the overall system. This is because actually measuring all similarities in each communication round is too expensive, communication-wise. The goal of a communication strategy is to estimate these true similarities as accurately as possible while minimizing the amount of communication done with other clients. These true similarities show when it is possible for a client to distinguish between clusters; if the true similarities overlap, even an optimal strategy could not distinguish between clusters.

For `DAC` with inverse loss we see that a client could theoretically distinguish between clusters since all similarities of clients in its own cluster are higher than those of the other cluster. The fact that the algorithm does distinguish between clusters is practically reflected in the average validation accuracy before and after exchange for each cluster, since after a few rounds the average validation accuracy increases after each merge. We see that for all the runs using `DAC`, as well as `PANM` with Inv loss, this is the case and clients benefit from merging since they can identify their own cluster. However, we see that random is only outperformed with Inv loss for both communication strategies, even though the other similarity metrics also identify their clusters well for `DAC`, as is illustrated in the communication heatmaps in Figure 4.2.

The reason for this could be that even the slightest out-of-cluster communication quickly brings the performance down to the level of random communication. As can be seen in the similarities in Figures 4.3 and 4.4, Inv loss has the greatest separation of similarities between clusters, minimizing the chance of out-of-cluster communication. All other similarity metrics have similarities that do not differ as much between clusters, especially in the earlier rounds. This is reflected in significant reductions in the after-exchange validation accuracy in the early rounds, which likely stem from out-of-cluster communication. This could be the reason for the random-like performance of these similarity metrics. Only `PANM` with Inv loss has strong early rounds, likely because the early rounds of `PANM` sample more "safely" than those of `DAC`, which sample more



(a) `FedAvg`

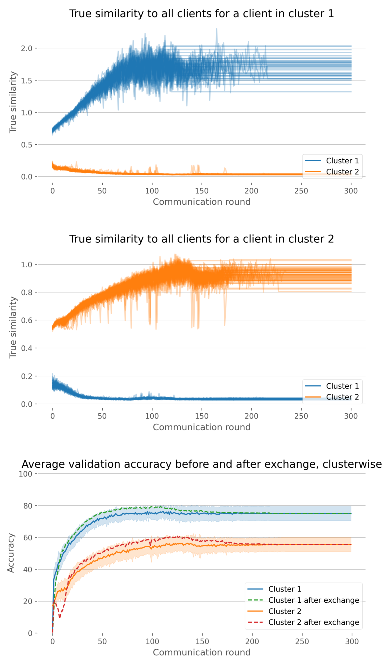


(b) `FedSim`

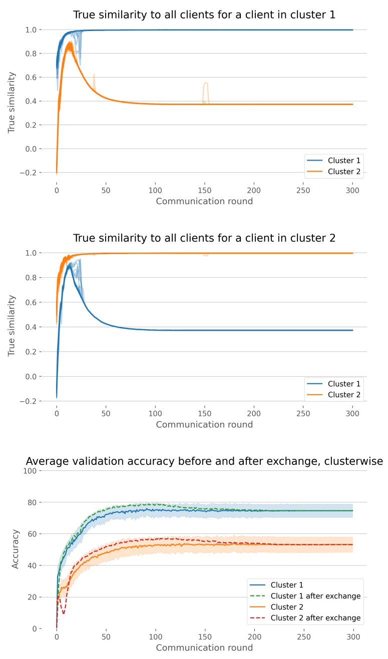
Figure 4.2: CIFAR-10 Animal/Vehicle label shift communication heatmaps. Each row represents a client i, and column j on that row is how many times client i has sampled client j.

randomly initially but in exchange gain more information about the network.

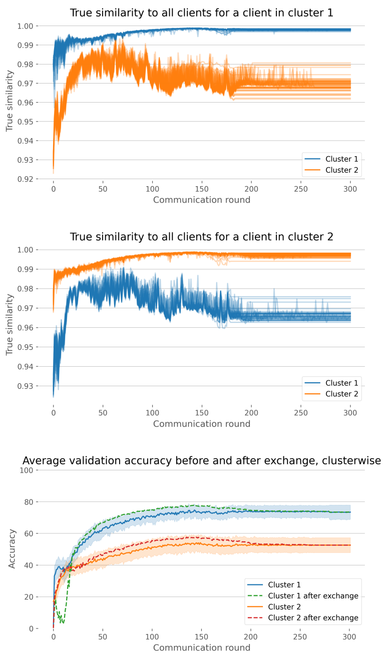
`PANM` with Cos grad performs by far worst of all in this expermiment, and fails most significantly after 50 rounds which is where the strategy of `PANM` changes. The second step in `PANM`s strategy is quite sensitive to how it is initialized and relies on the first part correctly identifying which cluster a client belongs to. What likely happens is that most clients correctly identify their cluster but a few wrongly identify their cluster. This phenomena is discussed in 4.2. We also see that after 50 communication rounds all similarity scores go to one, which is the maximum of Cos grad, implying that all models are extremely similar to all others. A global model that tries to generalize to all local objectives is created, but notably still performs significantly worse than random. Once the similarities collapse as for `PANM` with Cos grad, no communication strategy is going to be able to correctly identify their clusters. Inv loss is less susceptible to this, as it in a sense compares a model to its local data (which is unchanging), while the other similarity metrics compare models with models. Models can be corrupted through bad merging, rendering even the best similarity metric useless if it compares models with models. Inv loss is partially shielded to this, as its similarity estimates are based on the true and unchanging local data. If a model does not classify the local data well, it is objectively not good to merge with. However that same model can be very similar to the local model of that client, meaning that a model-to-model similarity metric such as Cos grad gives that model a high similarity.



(a) Inv loss `DAC`

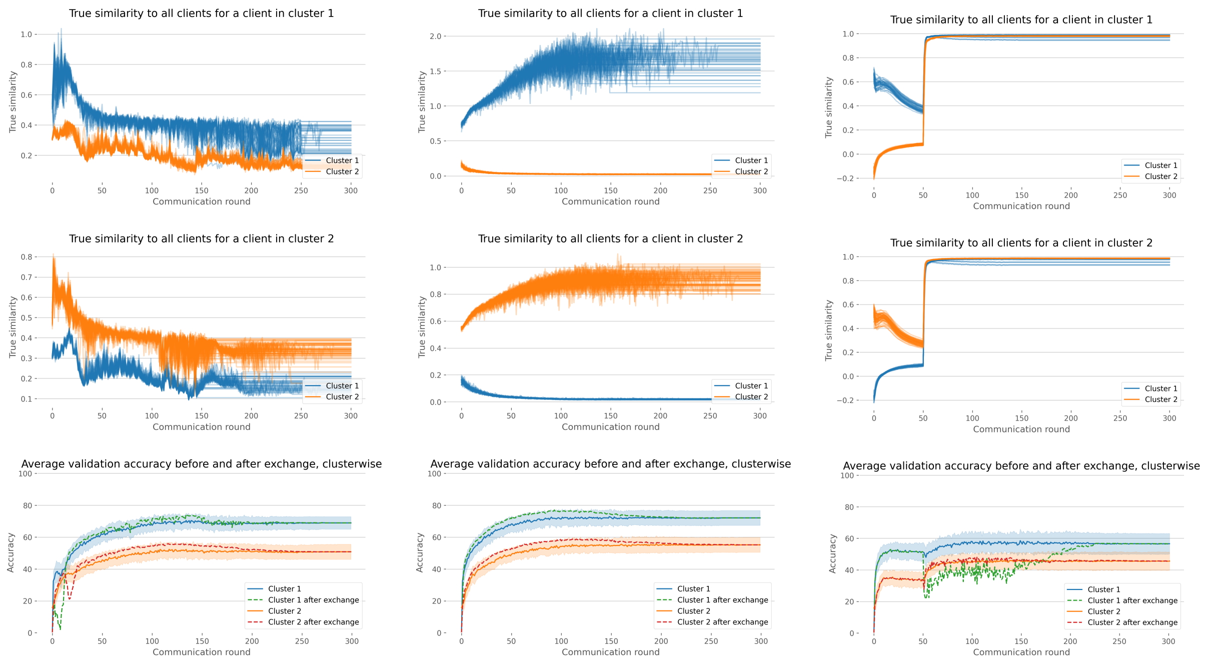


(b) Cos grad `DAC`



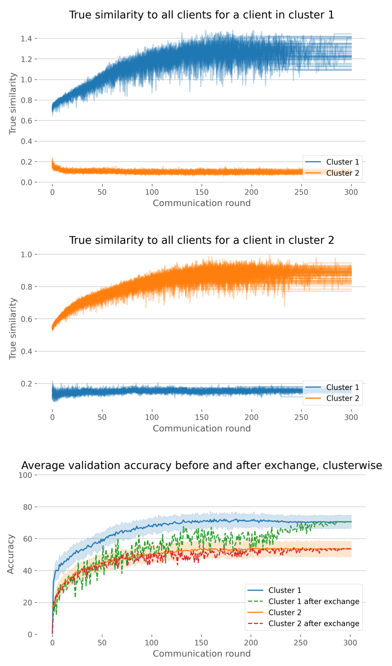
(c) Cos weight `DAC`

Figure 4.3: For runs on the CIFAR-10 dataset, with the Animal/Vehicle label shift with different similarity metrics, the similarity for a client in cluster 1 and a client in cluster 2 to all other clients. The clusterwise validation accuracy before and after merge at each communication round for each similarity metric and exchange strategy is included for comparison. `FedAvg` is used for all methods.

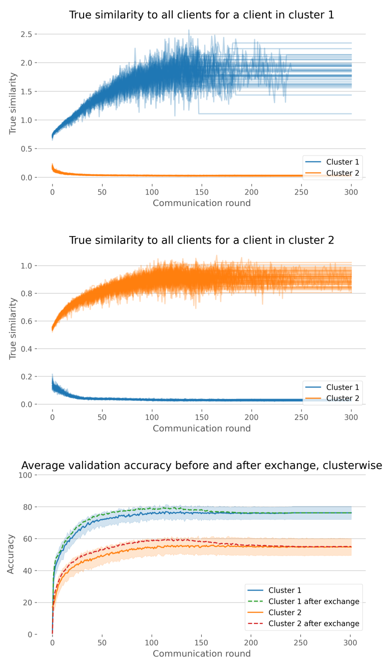


(a) L2 `DAC` (b) Inv loss `PANM` (c) Cos grad `PANM`

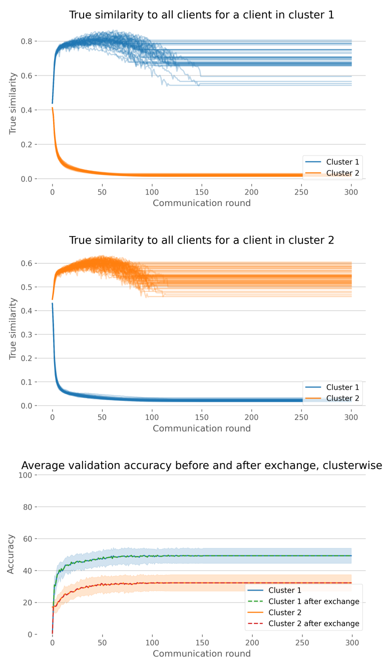
Figure 4.4: For runs on the CIFAR-10 dataset, with the Animal/Vehicle label shift with different similarity metrics and communication strategies the similarity for a client in cluster 1 and a client in cluster 2 to all other clients. The cluster-wise validation accuracy before and after merge at each communication round for each similarity metric and exchange strategy is included for comparison. `FedAvg` is used for all methods.



(a) Random communication



(b) Oracle communication



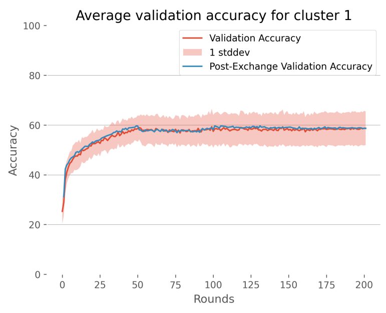
(c) No communication

Figure 4.5: Oracle/random/no communication runs on the CIFAR-10 dataset, with the Animal/Vehicle label shift. Here the similarity is calcualted with inverse loss for a client in cluster 1 and a client in cluster 2 to all other clients. The clusterwise validation accuracy before and after merge at each communication round for each similarity metric and exchange strategy is included for comparison. `FedAvg` is used for all communication strategies.

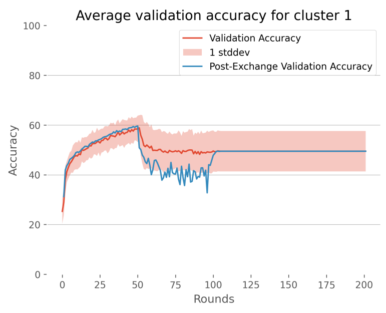
4.2 Cluster Death

In our experiments, we identified a critical issue, termed *Cluster Death*, which significantly impacts the performance of decentralized learning systems. Cluster Death occurs a few rounds into training when the majority of clients correctly identify their cluster membership, but a few clients mistakenly believe they belong to a different cluster. This leads all clients in the cluster where a few clients have wrongly identified themselves to lose accuracy. This is because all clients who have correctly identified their cluster still communicate with these "delusional" clients. Importantly, this arises when clients in the other cluster generally communicate correctly, which is the reason performance does not decrease as much with random communication. With random communication, a client from cluster 1 may communicate with a client from cluster 2, but that client has communicated with many clients from cluster 1 previously, reducing the adversarial effect.

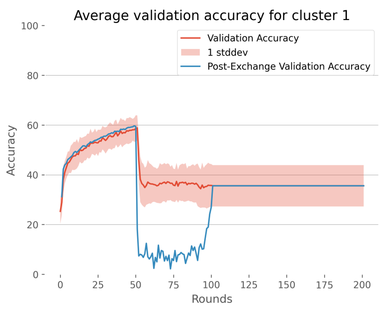
To replicate this behavior, we conducted an artificial experiment using the CIFAR-10 dataset with an Animal/Vehicle label shift. We let all clients have the oracle communication strategy initially, and after 50 rounds $x = 1, 5, 25$ clients in cluster 1 become "delusional", and start to communicate solely with the wrong cluster. Presented in Figure 4.6 are average validation accuracies for cluster 1 for each of these scenarios. It becomes evident that these scenarios are very detrimental to the performance of the whole cluster, even when only a few clients communicate incorrectly. The validation accuracy goes down, especially immediately after the exchange, and never recovers to the levels it had before round 50. The decrease in performance is compounded by the fact that clients that communicate correctly receive bad models from the "delusional" clients.



(a) 1 delusional client



(b) 5 delusional clients



(c) 25 delusional clients

Figure 4.6: Impact of varying numbers of delusional clients on cluster validation accuracy. Each subfigure represents a scenario with 1, 5, and 25 misidentified clients, respectively.

Cluster death can happen in realistic scenarios with DAC. Say that one or several clients start communicating outside their cluster by sampling low-probability clients or having erroneously estimated similarities. This will "poison" the local model of those clients with models from outside their cluster. This will result in them having higher similarities to clients outside their cluster. Also, other clients within that cluster will, based on earlier measures of similarity, continue to communicate with these clients, leading the poisoned models to spread within that cluster. With `PANM`, the situation could be even worse, as a client in the second phase of the algorithm can be of the understanding that it *belongs* to the wrong cluster, causing it to *exclusively* communicate outside its cluster. We belive this is what happened in the `PANM` with Cos grad runs for the CIFAR-10 Animal/Vehicle label shift.

We especially notice cluster death in `DAC` when a client has a priors-vector p with a small number of values significantly different from zero (fewer than the number of clients sampled). This can arise in the early rounds when a client does not have information about many other clients in the network, or with high values of the temperature τ. The random sampling from the priors will sample these few high probability clients, but also a few essentially randomly, increasing the probability of cluster death.

To address Cluster Death, we developed `FedSim`, a technique designed to merge weights more cautiously and reduce the negative impact of out-of-cluster communication. Further exploration of adaptive sampling and robust similarity measures could also mitigate this issue. `FedSim` diminishes the weighting of low-similarity clients, effectively causing the merge to only include clients that have a high similarity score even if there is only a few. The drawback to `FedSim` is that a client effectively merges with fewer clients and gets less information which diminishes the positive aspects of gaining information from the network.

4.3 Widening the Gap Between Oracle and Random with More Clusters

Due to of the small difference observed between random and oracle performance for the Animal/Vehicle label shift, we conducted further experiments on CIFAR-10 with 5 label-shifted clusters to test the performance of different decentralized learning methodologies when clients were less likely to randomly sample their own clusters. At this point, we decided to discontinue using the PANM algorithm due to its stability issues with the Cosine Gradient similarity metric, which can be seen in presented results but also in our experience with tuning the algorithm in different variations of the experiment settings. For example, we reproduced some of the shifts on CIFAR-10 tested in [5], and found the performance of `PANM` with Cos grad to remain unstable. Instead, we focus on `DAC`. As before, we tuned the hyperparameters and conducted three experiments with different random seeds for `DAC`, using each similarity metric with their optimal settings, as well as for the baseline methods.

Figure 4.7 illustrates that DAC significantly outperforms random sampling across all similarity metrics, whether using `FedAvg` or `FedSim`. Figure 4.7 and Table 4.2 show the testing accuracy for `DAC` using different similarity metrics and averaging methods. Figure 4.7 shows the standard deviation across runs and Table 4.2 shows the testing accuracy for each cluster.

Since there are more clusters there is more to gain from a good algorithm compared to random sampling. This is reflected in the difference in testing accuracy between oracle and random (a difference of almost 5 percentage points, see Table 4.2). Accuracies are generally higher for this setting than the other label shift, likely because there are only two labels in each generating distribution. If a client learns this fact well enough, the problem essentially becomes a binary classification problem, which is easier to solve with a higher accuracy than the multiclass classification problem that arises in the Animal/Vehicle label shift setting.

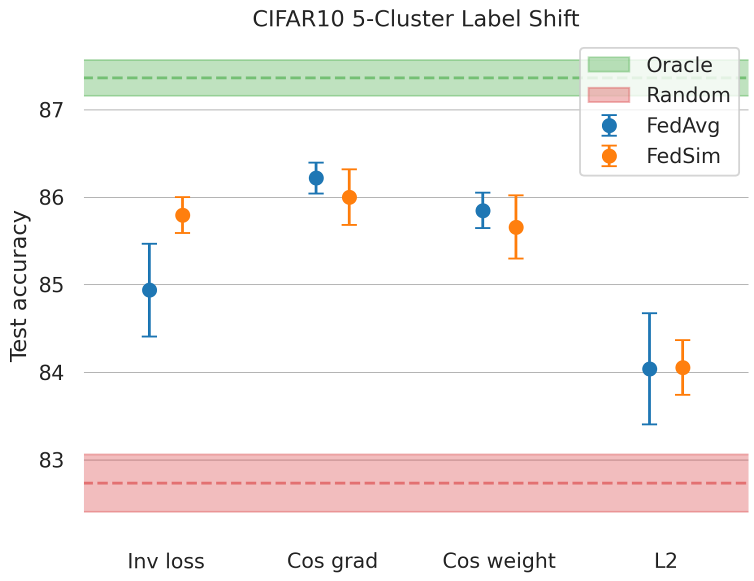


Figure 4.7: The testing accuracies for different similarity metrics and averaging methods. CIFAR-10, 5 cluster label shift.

These results show that `DAC` with any of the tested similarity metrics outperforms random. However, all similarity metrics also fall significantly short of oracle, meaning there is room for improvement. To further explore the limitations and potential of `DAC`, we move on to a more adversarial problem, where sampling of out-of-cluster clients is more detrimental.

Communication strategy & Similarity metric		Cluster 1		Cluster 2		Cluster 3	
		FedAvg	FedSim	FedAvg	FedSim	FedAvg	FedSim
Baselines	Random	87.34	-	73.99	-	79.13	-
	Oracle	91.66	-	78.64	-	83.25	-
	No comm	82.31	-	66.93	-	74.07	-
DAC	Inv loss	88.1	90.86	77.83	76.77	82.14	81.36
	Cos grad	91.3	90.86	77.21	77.06	81.91	82.38
	Cos weight	91.01	91.23	77.46	76.51	81.16	81.45
	L2	89.29	89.21	74.79	75.3	79.85	79.67

Communication strategy & Similarity metric		Cluster 4		Cluster 5		Mean	
		FedAvg	FedSim	FedAvg	FedSim	FedAvg	FedSim
Baselines	Random	88.76	-	84.46	-	82.73	-
	Oracle	92.2	-	91.06	-	87.36	-
	No comm	83.84	-	78.25	-	77.08	-
DAC	Inv loss	88.37	91.23	88.26	88.77	84.94	85.8
	Cos grad	91.44	91.11	89.23	88.59	86.22	86.0
	Cos weight	91.42	91.5	88.19	87.6	85.85	85.66
	L2	89.93	90.02	86.35	86.07	84.04	84.06

Table 4.2: Test accuracies for different communication strategies, similarity metrics, and averaging methods on the CIFAR-10 5-cluster label shift setting.

4.4 A More Adversarial Setting: A Synthetic Problem

To investigate the robustness of `DAC`, we designed a more adversarial experimental setting. To achieve this, we want the local objective of a cluster to be completely different than the local objective of the other cluster/clusters. While in the problem settings previously examined a client could gain from communicating out-of-cluster by learning, for example, low-level features that are shared between all images regardless of the label, we here aim to create a problem setting where merging out-of-cluster has no upsides. This led us to create the synthetic problem described in Section 3.1.3. Because of the simple nature of this problem, experiments run much faster than for the other settings. This allowed us to tune the learning rates and temperatures simultaneously, as described in Appendix A. For the optimal hyperparameters for each setting, each experiment was recreated with 15 different random seeds. Also, we explored the influence of the size of the local datasets on the performance of the similarity metrics. For 50 training examples per client, the results are presented in Figures 4.3 and 4.8.

Table 4.3 shows a large difference in loss between the oracle and random strategies for this setting. This is likely because it is exclusively detrimental to sample out-of-cluster in this setting. This is reinforced by random communication being a lot worse than not communicating which is not the case for the CIFAR-10 label shift problem, where random is better than not communicating, as shown in Table 4.1. For this more adversarial problem, `FedSim` seems to help similarity metrics perform better than `FedAvg`. We can see this for the in the results in Table 4.3.

Communication strategy & Similarity metric		Cluster 1		Cluster 2		Cluster 3		Mean	
		FedAvg	FedSim	FedAvg	FedSim	FedAvg	FedSim	FedAvg	FedSim
Baselines	Random	1196.04	-	1251.50	-	2036.90	-	1494.84	-
	Oracle	9.54	-	9.29	-	9.46	-	9.43	-
	No comm	28.76	-	26.80	-	35.22	-	30.26	-
DAC	Inv loss	32.91	14.87	30.40	14.58	31.77	15.01	31.69	14.82
	Cos grad	10.19	10.40	10.32	10.22	10.44	10.29	10.32	10.30
	Cos weight	10.29	10.37	10.25	10.41	10.48	10.13	10.34	10.30
	L2	21.97	10.85	19.77	10.92	21.56	10.78	21.10	10.85

Table 4.3: Test losses for different communication strategies, similarity metrics, and averaging methods on the synthetic problem setting.

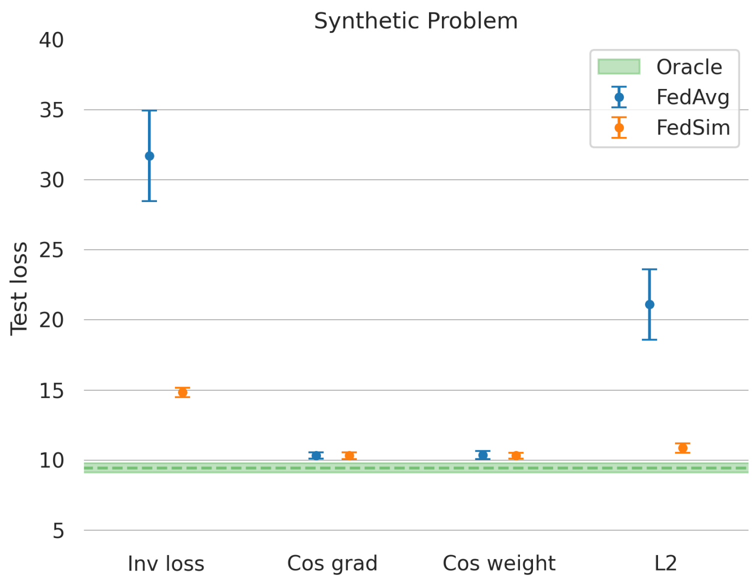
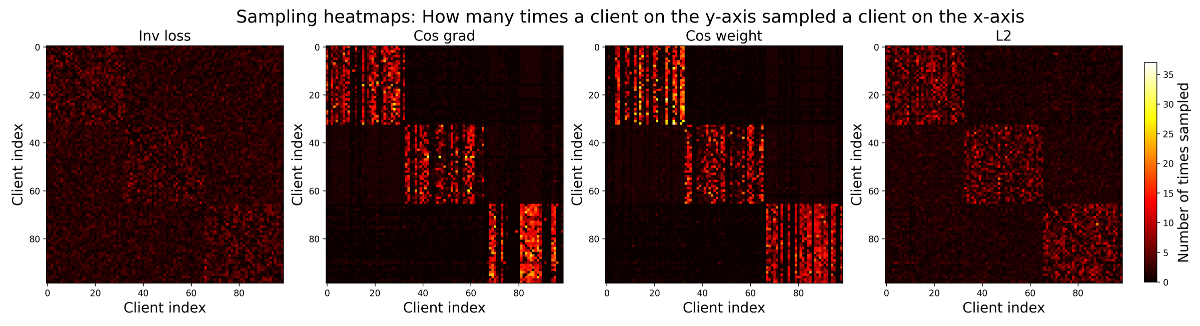
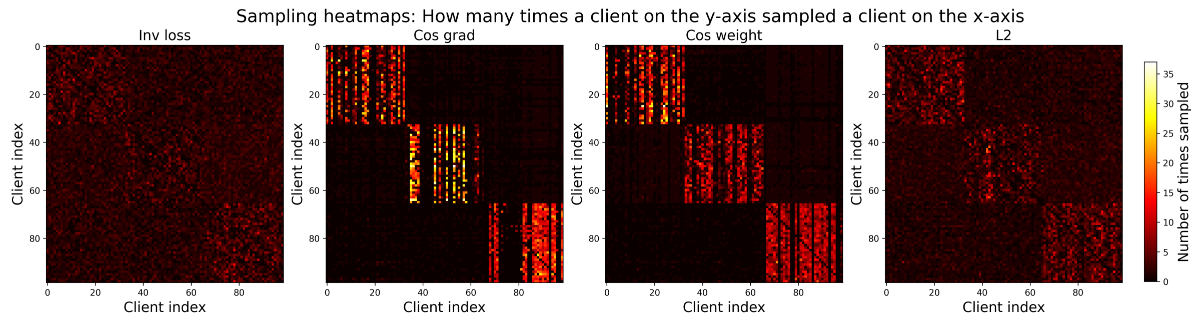


Figure 4.8: The testing losses for different communication strategies, similarity metrics and averaging methods in the synthetic problem setting. Random and No comm are not included because they are off the scale.

The results indicate that cosine similarity metrics outperform Inv loss and L2 metrics in this adversarial setting. `FedSim` appears to mitigate performance loss to some extent compared to `FedAvg`, particularly for the Inv loss and L2 metrics. Both Inv loss and L2 struggle to identify their clusters more than Cos grad and Cos weight, as can be seen in the heatmaps in Figure 4.9. Also illustrated in the heatmaps is a phenomenon that can arise when using `DAC`, in this setting as well as others. The heatmaps contain several black vertical lines, indicating that some clients are never sampled by any other client. This is detrimental to the overall performance of the clients, as not all data available in the network is utilized. The reason for this phenomenon is the initialization of `DAC`. In the first round, clients sample other clients randomly. In the next round, a client only has similarity scores to these and to potential two-step neighbors of them, meaning they will either sample the same clients again, or clients that those clients have sampled. This means that if a client is not sampled by anyone in the first round, which can happen by chance, especially if there are many clients and $n_{sampled}$ is low, that client will never be sampled



(a) `FedAvg`



(b) `FedSim`

Figure 4.9: Heatmaps for the synthetic problem. The first 33 clients belong to cluster 1, the next 33 to cluster 2, and the last 33 to cluster 3.

thoughout the course of training. This relates to the fact that the initial round of `DAC` is worse at exploration of the network than other communication strategies such as `PENS` which `PANM` is based on, and has a tendency to latch on to the clients it sampled randomly in the first round rather than to gain more information about the network. These are some of the shortfalls of `DAC`.

`FedSim` minimizes the adversarial effect of selecting clients to sample outside ones cluster, if the similarity to those is lower than the clients within-cluster, which is the case here. The difference in similarity is just not large enough for `FedAvg` to be effective. This confirms the importance of correct sampling for this problem setting. The reason for Inv loss struggling to identify clusters is likely because the loss is very high initially, on the order of thousands. Taking the inverse of such a large number will yield very small differences, so even if the clusters exhibit differences in the early rounds, `DAC` will have trouble distinguishing them. L2 seems to be a less relevant metric than the cosine similarities for this model in this setting. It is worth noting that Inv loss and L2 had higher optimal learning rates than the cosine similarities. This is likely because these metrics had more to gain from training harder in the first round, so that the clusters differentiated themselves more. The cosine similarities had less trouble differentiating, meaning they could use a lower learning rate that would increase performance of the local training while still sampling within-cluster.

Provided in Figures 4.10 and 4.11 are the relationships between the size of the training dataset for each client and the average test loss. As can be seen in these results, the loss generally

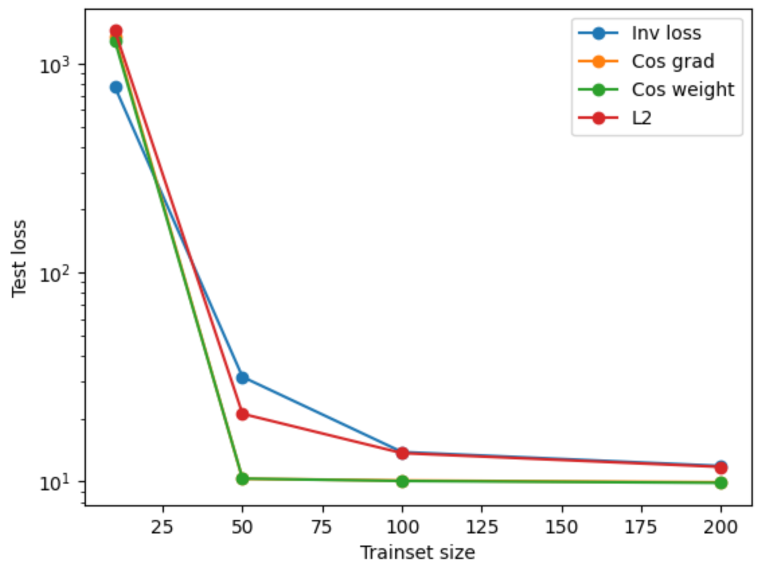


Figure 4.10: Relationships between the size of the training dataset for each client and the average test loss for each similarity metric using `DAC` with `FedAvg`.

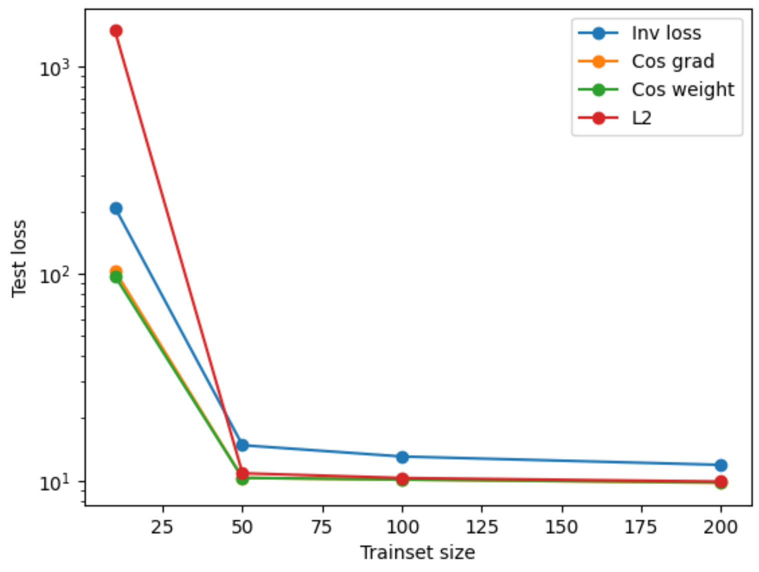


Figure 4.11: Relationships between the size of the training dataset for each client and the average test loss for each similarity metric using `DAC` with `FedSim`.

decreases as the size of the training set increases, which is to be expected. It can also be seen that Inv loss and L2 struggle more than Cos grad and Cos weight when `FedAvg` is used, for all measured training set sizes, except for 10, where all similarity metrics perform equally bad. However, when `FedSim` is used, L2 improves to almost match the performance of the cosine similarities. Inv loss improves too, especially for the low training set sizes. This could imply that `FedSim` is suitable in scenarios when there is little available local data, or when sampling out-of-cluster is especially detrimental to model performance.

4.5 Applying `DAC` to a Covariate Shift-Setting

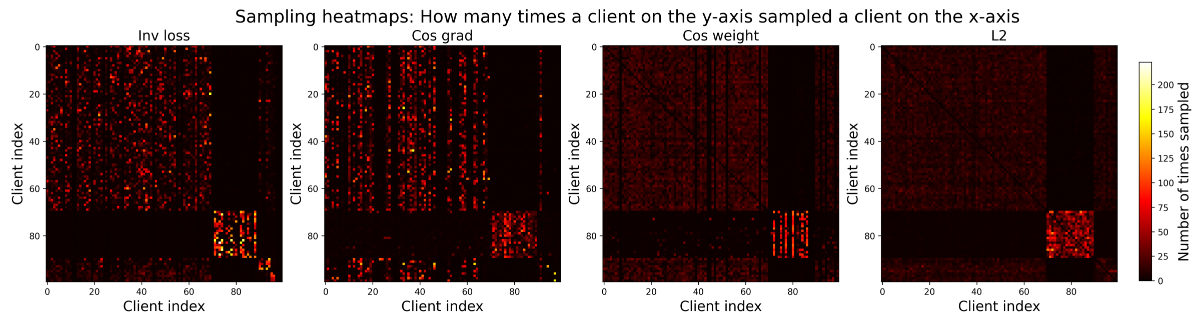
Another problem we studied was a 4-cluster problem using the Fashion-MNIST dataset. For this problem we implement a covariate shift, meaning all clients have the same distribution of labels but their images are augmented to be different between clusters. We created the covariate shift by rotating the images, with three clusters having a maximum of 20° difference in rotation, while cluster 2 had a rotation difference of at least 170° from any other cluster. Cluster 2 is also smaller in size in regard to the number of clients.

This setup indicates that a random communication strategy is effective for clusters 1, 2, and 3, as randomly selected clients are likely to belong to similar clusters. This is reflected in the testing accuracies in Table 4.4, where random communication is as good as oracle communication for these clusters. The difference in testing accuracy in cluster 2 between random and oracle in Table 4.4 is much more significant. Additionally, almost all similarity metrics outperform random communication for cluster 2, as depicted in Figure 4.13. Heatmaps for this problem

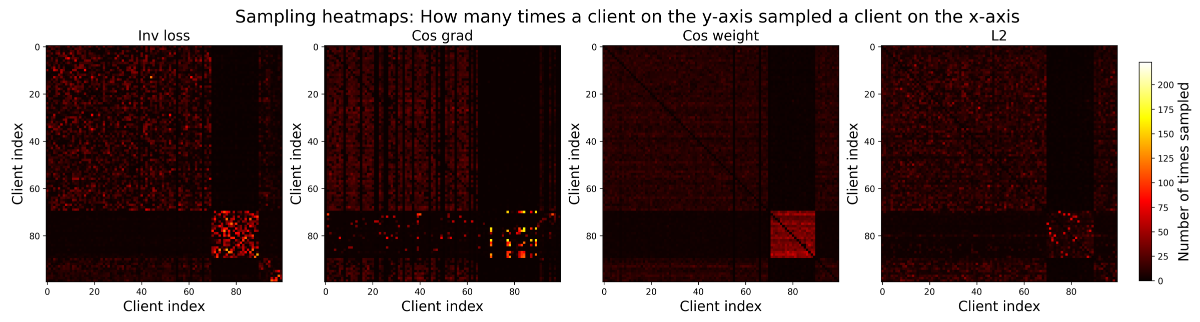
Communication strategy & Similarity metric		Cluster 1		Cluster 2		Cluster 3		Cluster 4		Mean	
		FedAvg	FedSim	FedAvg	FedSim	FedAvg	FedSim	FedAvg	FedSim	FedAvg	FedSim
Baselines	Random	85.11	-	79.53	-	82.36	-	81.89	-	83.70	-
	Oracle	86.11	-	85.19	-	82.15	-	82.60	-	85.55	-
	No comm	76.36	-	75.86	-	76.23	-	76.47	-	76.26	-
DAC	Inv loss	85.49	85.51	83.73	84.72	82.44	82.64	82.31	83.02	84.83	85.08
	Cos grad	85.61	85.32	83.68	77.02	82.57	82.11	81.59	81.85	84.87	83.32
	Cos weight	85.81	85.24	83.59	84.67	82.74	82.34	82.35	82.13	85.04	84.82
	L2	85.54	85.37	80.17	80.87	82.66	82.21	82.43	82.26	84.17	84.16

Table 4.4: Test accuracies for different communication strategies, similarity metrics, and averaging methods on the Fashion-MNIST covariate shift setting.

are presented in Figure 4.12. As can be seen, the three clusters which are similar sample from each other quite extensively, while avoiding the cluster with upside-down images.



(a) FedAvg



(b) FedSim

Figure 4.12: Heatmaps for DAC on the Fashion-MNIST covariate shift experiment setting.

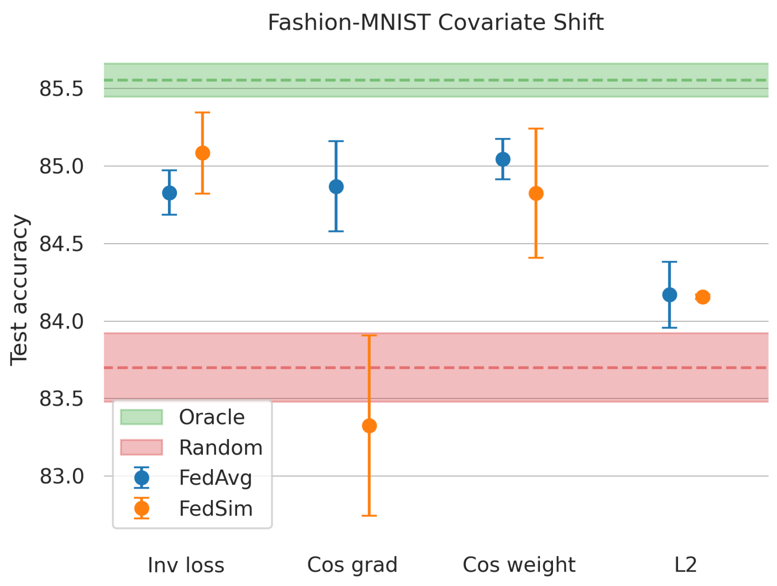


Figure 4.13: The testing accuracies for different similarity metrics and averaging methods for Fashion-MNIST covariate shift.

4.6 Fashion-MNIST and MNIST Combined

In Table 4.5 we see the results from three runs on the Fashion-MNIST and MNIST combined experiment with tuned hyperparameters. The difference between oracle and random communication is not very large. There is still a significant difference between random and cosine similarity of gradients and cosine similarity of weights with `FedAvg`, as shown in Figure 4.14. There is also an increase in testing accuracy when using inverse loss with `FedSim` compared to `FedAvg`. Because the difference between oracle and random is so small, it could be assumed

Communication strategy & Similarity metric		Cluster 1		Cluster 2		Mean	
		FedAvg	FedSim	FedAvg	FedSim	FedAvg	FedSim
Baselines	Random	97.01	-	84.05	-	90.53	-
	Oracle	97.68	-	85.56	-	91.62	-
	No comm	89.29	-	75.15	-	82.22	-
DAC	Inv loss	97.11	96.96	82.55	84.38	89.83	90.67
	Cos grad	97.29	96.91	84.99	84.54	91.14	90.72
	Cos weight	97.22	96.72	84.83	84.52	91.03	90.62
	L2	97.05	97.04	83.92	84.56	90.49	90.80

Table 4.5: Test accuracies for different communication strategies, similarity metrics, and averaging methods on the Fashion-MNIST + MNIST combined setting.

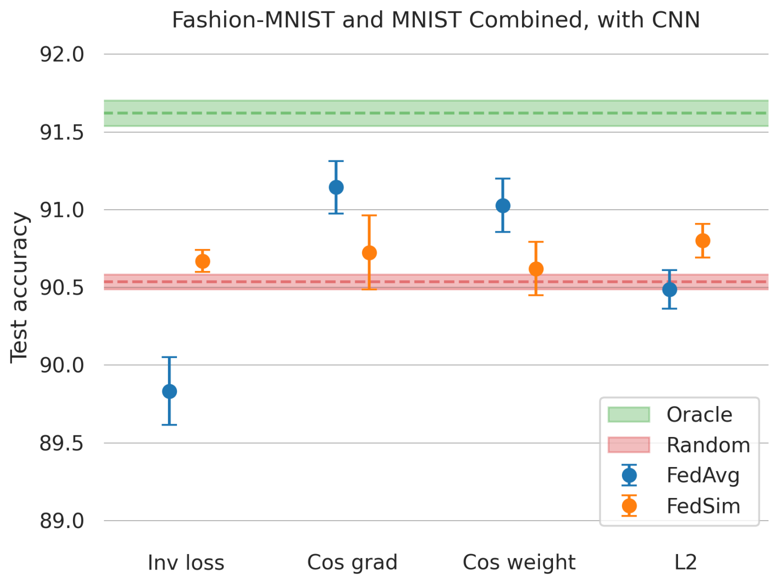
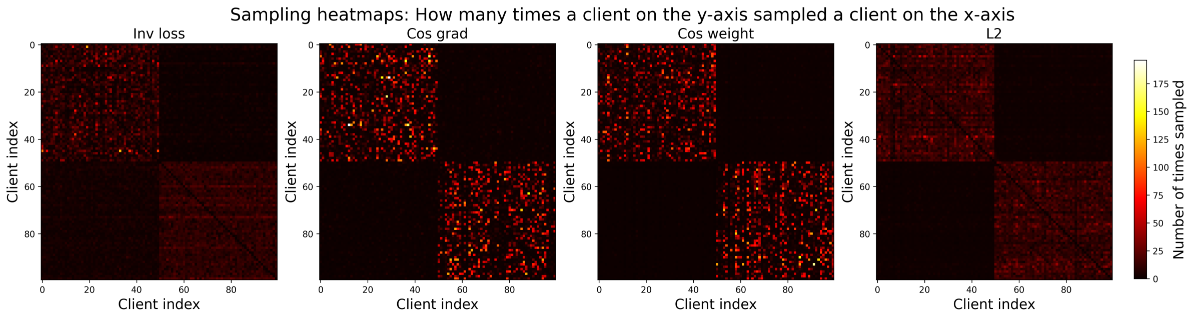
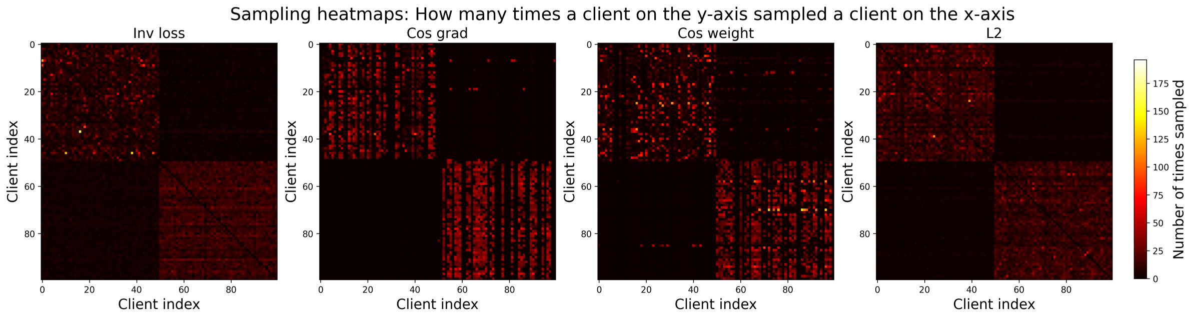


Figure 4.14: The testing accuracies for different similarity metrics and averaging methods in the MNIST and Fashion-MNIST combined experiment setting with a convolutional neural network.

that the convolutional network is capable enough to learn both local objectives of each cluster, leading to the high performance of random. To simulate a more difficult problem, we replaced



(a) `FedAvg`



(b) `FedSim`

Figure 4.15: Heatmaps for the Fashion-MNIST and MNIST combined experiment setting, using a CNN.

the convolutional neural network used in this setting with a less capable *Multi-Layered Perceptron* (MLP), and redid the hyperparameter tuning and reproduction of runs on the optimal hyperparameters, in the hopes of making the gap between random and oracle communication larger. The results of these experiments are presented in Figure 4.16. As can be seen in these

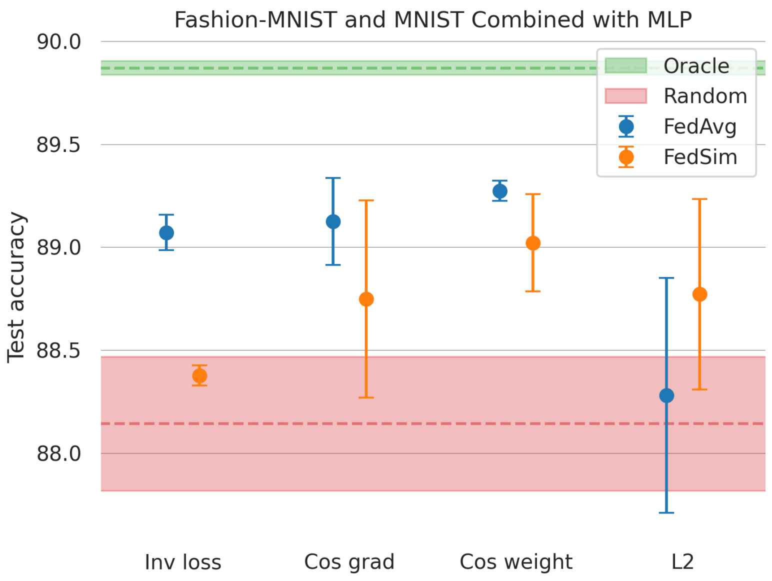


Figure 4.16: The testing accuracies for different similarity metrics and averaging methods in the MNIST and Fashion-MNIST combined experiment setting with a multi-layered perceptron.

results, the MLP solves both local objectives almost as well as the CNN, indicating that an MLP with fewer trainable parameters could have been better for what we wanted to achieve. However, these results still succeed in showing that `DAC` performs similarly on different neural network architectures for the same problem setting with all similarity metrics, with the exception of Inv loss with `FedAvg`. The reason for this could be insufficient tuning of hyperparameters for the setting with the CNN.

4.7 Pretrained Models on CIFAR-100

To investigate how the similarity metrics perform for larger models, and to see if using a pretrained model influences the performance of different similarity metrics, we implemented a network of clients with ResNet-18 models on the CIFAR-100 dataset. Because this setting is more computationally complex than the other ones mentioned in this paper, we opted to only test the similarity metrics on `DAC` with `FedAvg`. After tuning the learning rate and temperature τ, we obtained the results found in Tables 4.6 and 4.7, and Figures 4.17 and 4.18 for the optimal values of the hyperparameters for each similarity metric. The heatmaps for one run on each similarity metric are presented in Figure 4.19.

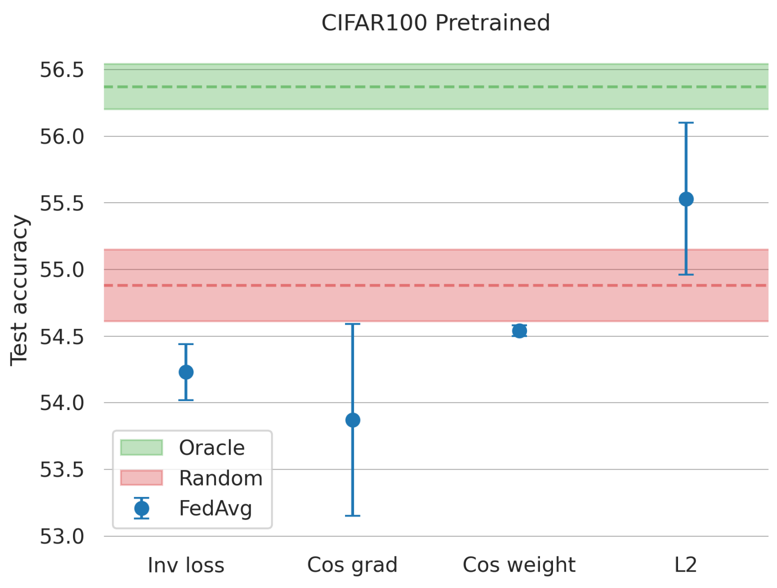


Figure 4.17: The testing accuracies for different similarity metrics and averaging methods on CIFAR-100 with a pretrained ResNet-18.

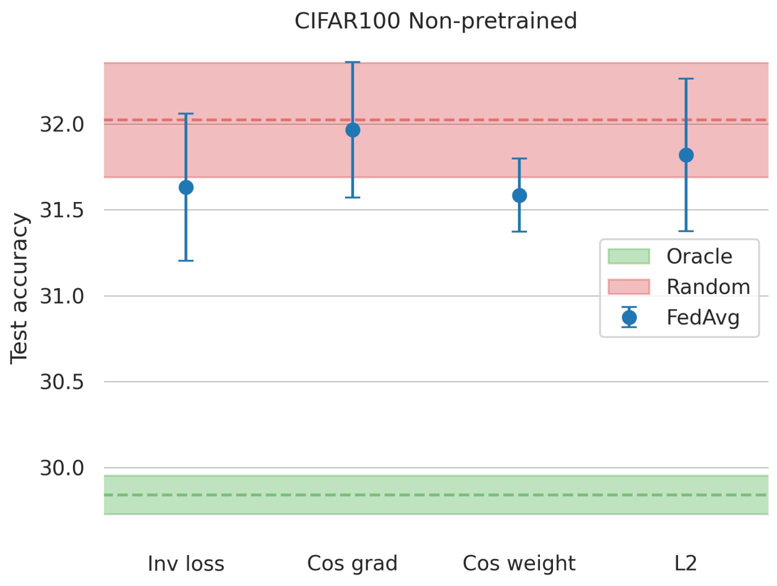


Figure 4.18: The testing accuracies for different similarity metrics and averaging methods on CIFAR-100 with a randomly initialized ResNet-18.

Communication strategy & Similarity metric		Cluster 1	Cluster 2	Cluster 3	Mean
Baselines	Random	49.44	63.11	57.53	54.88±0.27
	Oracle	50.33	65.87	58.95	56.37±0.17
	No comm	20.99	41.04	34.94	29.49±0.03
DAC	Inv loss	48.16	63.07	57.53	54.23±0.21
	Cos grad	47.96	62.77	56.79	53.87±0.72
	Cos weight	48.80	63.09	57.46	54.54±0.04
	L2	50.31	63.44	58.07	55.53±0.57

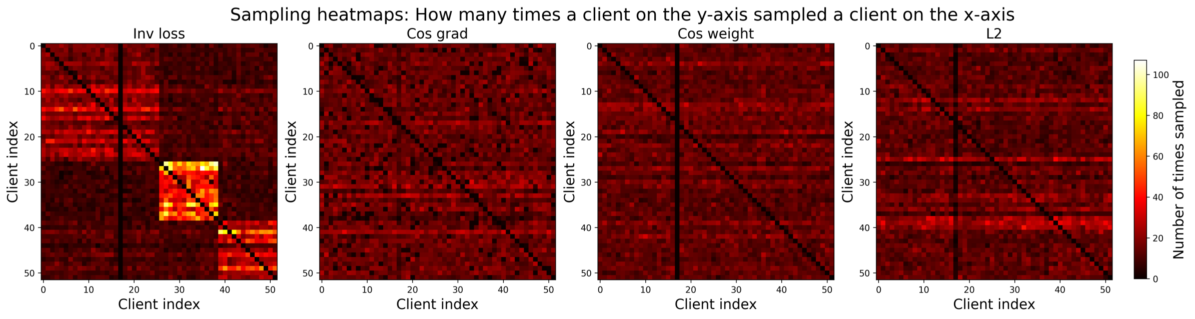
Table 4.6: Test accuracies in percentages of communication strategies and similarity metrics across clusters for a pretrained ResNet-18 on CIFAR-100 using `FedAvg`. The standard deviation in the "Mean" column is across runs with different random seeds.

Communication strategy & Similarity metric		Cluster 1	Cluster 2	Cluster 3	Mean
Baselines	Random	27.07	39.51	34.44	32.02±0.33
	Oracle	25.82	36.83	30.89	29.84±0.11
	No comm	10.76	23.09	18.54	15.79±0.21
DAC	Inv loss	26.47	39.45	34.14	31.63±0.43
	Cos grad	27.07	39.15	34.58	31.97±0.39
	Cos weight	26.54	38.97	34.29	31.58±0.21
	L2	26.64	39.36	34.64	31.82±0.44

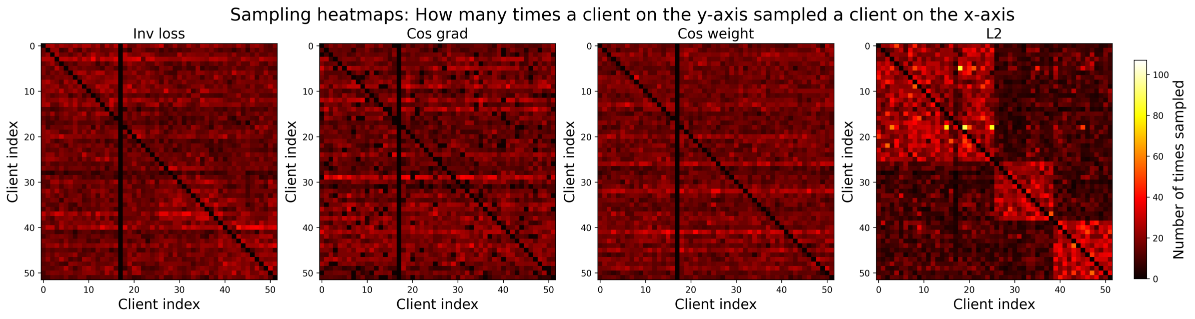
Table 4.7: Test accuracies in percentages of communication strategies and similarity metrics across clusters for a randomly initialized ResNet-18 on CIFAR-100 using `FedAvg`. The standard deviation in the "Mean" column is across runs with different random seeds.

Notably, for the non-pretrained networks, the oracle communication strategy performs worse than random, unlike all other experiments. This could be due to the lack of available data. ResNet-18 is a large network with around 11 million trainable parameters, and we introduce it here in a setting with comparatively small clusters, and few clients in total. Since the network is not pretrained, it has to learn low-level features to be able to classify images correctly, and it does this better with more diverse data, which it gets from random communication, even though some of the clients communicated with have irrelevant labels. Likely, extending the clusters with more clients would eventually lead to oracle outperforming random.

For the pretrained networks, oracle outperforms random with the same cluster sizes. This could be because those neural networks already have good low-level feature recognition, and therefore essentially only have to fine-tune their final layers to the new label distribution that they get from their local data and from within their cluster. In this setting, only the inverse L_2-distance loss outperforms random, while the others perform significantly worse. In the heatmaps presented in Figure 4.19b, it can be seen that only the L2 similarity metric finds significant cluster-structure. No metric reaches the performance of oracle. This could partly be due to insufficient hyperparameter-tuning (due to long experiment run-times the hyperparameter search was done more coarsely than for other settings), but also the nature of the problem and the architecture of the network.



(a) Non-pretrained



(b) Pretrained

Figure 4.19: Heatmaps for the CIFAR-100 label shift experiment setting.

In the non-pretrained setting, `DAC` with any similarity metric performs similar to random if not slightly worse. As can be seen in the heatmaps in Figure 4.19a, the communication pattern of most similarity metrics is similar to a random one. Inv loss does find significant structure, but still samples out-of-cluster. Likely, this out-of-cluster sampling ruins the gains made from sampling within-cluster, as we have seen for the CIFAR-10 experiments. This implies that `DAC` in this setting is not able to identify the clusters well enough, even though it would not be beneficial to do so.

Chapter 5

Discussion

5.1 Main Findings

Testing various decentralized learning methodologies on different datasets and data shifts has provided novel insights into the factors influencing the performance of these strategies. We have identified specific settings in which different similarity metrics and averaging methods perform well and where they do not. We have found that both the communication strategies `DAC` and `PANM` are able to outperform random communication for certain similarity metrics and averaging methods in certain settings. However, which similarity metric and averaging method performs best varies significantly between problem settings. For example, inverse training loss is the strongest similarity metric for the CIFAR-10 Animal/Vehicle label shift, while the similarity metrics based on cosine similarities significantly outperform inverse training loss in the synthetic problem setting. We also introduced the novel merging technique `FedSim` which outperforms the existing `FedAvg` in adversarial settings and when each client has few local data points, but can fall short of `FedAvg` in easier settings and when the difference in data distributions is small between clients. We illustrate the shortfalls of the communication strategy `PANM` with experimental results and formulate the phenomenon which we call *cluster death*. Furthermore, we present experiment settings where random communication outperforms any other communication strategy, such as the CIFAR-100 label shift with a randomly initialized neural network. This shows that properties of the network of clients such as availability of data, size of clusters, and characteristics of the networks used have a large influence on which decentralized learning strategy is optimal. For each communication strategy, hyperparameters such as the number of local epochs, learning rate, temperature τ, and the number of clients sampled per round significantly influence performance. We observed that tuning these parameters is crucial for optimizing the performance of decentralized learning strategies.

5.2 Impact and Future Work

Our findings show that optimizing many different local objectives across clients with non-IID data using decentralized learning methodologies is a challenging problem, with no one-size-fits-all solutions. However, in most experimental settings in this study, we identified a combination of communication strategy, similarity metric, and averaging method that outperforms random communication and significantly exceeds the performance of no communication. Any real-world implementation of a decentralized learning system would require the testing of many different approaches, along with hyperparameter tuning for each approach. Because of this fact and the novelty of the concept, decentralized deep learning has not yet been implemented in any real-world applications. We believe the insights and results gained from this study have made significant advancements in this field, but many unanswered questions remain. For example, given the shortfalls of `PANM` and `DAC` illustrated in this study, there is room for better communication strategies that are more adaptive to different characteristics of the clients in the network and their local objectives, and require less hyperparameter-tuning to use. Also, it remains to be seen if the insights gained from these experiment settings hold up in a real-world scenario. Real scenarios where it is desirable to implement decentralized learning strategies introduce some new aspects not tested by us, such as heavily skewed local dataset sizes, a larger number of clients, less defined clusters, and that clients only participate in the communication when and if they are willing, such as for example when a smartphone is charging and connected to WiFi.

Other studies such as [11] have shown that introducing a small set of globally shared data significantly diminishes the negative impact of clients having non-IID data when using federated learning. This concept could perhaps be applied in the decentralized setting, but might be hard to accomplish if the local objectives of clients differ too much.

In this work, model averaging has consisted of averaging all weights of the neural networks. However, it could make sense to only include certain layers of the model in the averaging. For example, the final layers of a model is highly fine-tuned to the local objective of each client, while earlier layers find features that are more general to all data in the network. Therefore it could be beneficial to only average earlier layers, and to keep the final layers locally for each client.

Furthermore, the communication strategies tested in this study do not make use of all information available to them to make decisions on which clients to communicate with. For example, clients could use the difference in validation accuracy or validation loss from before a merging of models to after the merge to determine the quality of that selection of sampled clients. While it is not possible to know which of the sampled clients contributed positively to the averaging, sampling many sets of clients over many rounds can yield useful information on which clients generally contribute to a good averaging of models.

5.3 Limitations

Several limitations to this work should be addressed. For example, the results show that the average test accuracy of clients can differ significantly for the same experiment settings depending on the random seed used, meaning these algorithms are sensitive to random chance, such as which clients sample which in the first couple of rounds when communication is largely random. For most experiment settings, we produced simulations with three different random seeds, due to the computational expense of each experiment run and our limited access to computing resources. Ideally, we would have more experiments for each setting to further improve the strength of our results, as was done for the synthetic problem, where each setting was tested on 15 different random seeds. Another consequence of our limited computing resources was that we did not fully test the `PANM` communication strategy in all experiment settings. For example, it would have been interesting to test `PANM` with `FedSim`, to see if `FedSim` could mitigate some of the stability issues with `PANM`. We instead chose to focus on applying `DAC` to a larger set of experiment settings.

A limitation of decentralized learning in general is that privacy may not be entirely preserved with model sharing. Studies have shown that partial training data can be reconstructed from model parameters in certain settings [3].

Another limitation is our use of curated datasets like CIFAR and MNIST, and applying shifts to these that may not reflect real-world distributions of data. Because of this, there are likely significant differences between our experimental setup and what an actual application of decentralized learning would look like. However, we made efforts to reproduce our results over many different datasets and with many different shifts, such as label shifts (for CIFAR-10 and CIFAR-100), covariate shifts (Fashion-MNIST), and concept shifts (Fashion-MNIST and MNIST combined and the synthetic problem). The range of potential applications of decentralized learning also stretches beyond the task of image classification, which we predominantly focused on in this work. Other machine learning tasks have different datasets and models, and the dynamics which govern effective decentralized communication may be different.

5.4 Conclusion

We believe that this thesis has shed light on how decentralized learning methodologies work as well as shown that they can be an effective way of solving tasks where data is distributed in a non-IID fashion among many devices. We have demonstrated cases when strategies do and do not work effectively, and have introduced variations to existing communication strategies that can mitigate their shortfalls. Because more and more problems are solved with machine learning approaches, and ever more data that can be considered private is created, there is a place in the future for decentralized learning. In addition to creating specialized machine learning models in a privacy-preserving manner, decentralized learning can be effective in scenarios where computational resources are scarce or costly, as it distributes the computational load. We envision a future where different entities and stakeholders control their own data while collaborating to improve machine learning models through collective learning.

Bibliography

[1] European Parliament and Council of the European Union. Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation), 2016.

[2] Python Software Foundation. Python programming language, 2024.

[3] Niv Haim, Gal Vardi, Gilad Yehudai, Ohad Shamir, and Michal Irani. Reconstructing training data from trained neural networks. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 22911–22924. Curran Associates, Inc., 2022.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[5] Zexi Li, Jiaxun Lu, Shuang Luo, Didi Zhu, Yunfeng Shao, Yinchuan Li, Member, Zhimeng Zhang, Yongheng Wang, and Chao Wu. Towards effective clustered federated learning: A peer-to-peer framework with adaptive neighbor matching. *IEEE Transactions on Big Data*, 2022.

[6] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. *AISTATS*, 2017.

[7] Noa Onoszko, Gustav Karlsson, Olof Mogren, and Edvin Listo Zec. Decentralized federated learning of deep neural networks on non-iid data. 2021.

[8] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019. NeurIPS 2019.

[9] Sweden. Dataskyddslagen [Data Protection Act]. SFS 2018:218, 2018. Available online: `http://www.riksdagen.se`.

[10] Edvin Listo Zec, Ebba Ekblom, Martin Willbo, Olof Mogren, and Sarunas Girdzijauskas. Decentralized adaptive clustering of deep nets is beneficial for client collaboration. 2022.

[11] Y Zhao, M Li, L Lai, N Suda, D Civin, and V Chandra. Federated learning with non-iid data. *IEEE 38th International Conference on Data Engineering*, 2022.

Appendices

Appendix A

Experimental Setup and Hyperparameter Tuning

A.1 Models

A.1.1 CIFAR-10 Experiments

For the experiment settings with CIFAR-10, we used a neural network with an architecture described in listing 1.

Listing 1 Pytorch description of the CNN used for the CIFAR10 experiments.

```
simple_CNN(
  (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (conv2_drop): Dropout2d(p=0.1, inplace=False)
  (fc1): Linear(in_features=400, out_features=120, bias=True)
  (dropout): Dropout(p=0.5, inplace=False)
  (fc2): Linear(in_features=120, out_features=84, bias=True)
  (output): Linear(in_features=84, out_features=10, bias=True)
  (activation): LogSoftmax(dim=None)
)
Total number of trainable parameters: 62006
```

A.1.2 Synthetic Problem

The model used for the synthetic problem is a single fully connected linear layer and is described in listing 2.

Listing 2 Pytorch description of the model used for the synthetic problem.

```
LinearRegression(
  (linear): Linear(in_features=10, out_features=1, bias=True)
)
Total number of trainable parameters: 11
```

A.1.3 Fashion-MNIST/MNIST

For the Fashion-MNIST experiment setting we used a neural network with an architecture described in listing 3. For the Fashion-MNIST and MNIST combined experiment setting, the same network was used, but with 20 output features instead of 10, as this setting has 20 labels. Also, an MLP with two fully connected layers was used, which is described in listing 4.

Listing 3 Pytorch description of the CNN used for the Fashion-MNIST and Fashion-MNIST + MNIST problems

```
fashion_CNN(
  (conv1): Conv2d(1, 16, kernel_size=(3, 3), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
  (fc1): Linear(in_features=800, out_features=64, bias=True)
  (output): Linear(in_features=64, out_features=10, bias=True)
  (activation): LogSoftmax(dim=1)
)
Total number of trainable parameters: 56714
```

Listing 4 Pytorch description of the MLP used for the Fashion-MNIST and Fashion-MNIST + MNIST problems

```
Classifier(
  (fc1): Linear(in_features=784, out_features=512, bias=True)
  (fc2): Linear(in_features=512, out_features=20, bias=True)
  (dropout): Dropout(p=0.2, inplace=False)
)
Total number of trainable parameters: 412180
```

A.1.4 CIFAR-100 Experiments

For the CIFAR-100 experiments, we used pretrained and randomly initialized versions of the ResNet-18 architecture, with a total of 11,227,812 trainable parameters.

A.2 Shifts

A.2.1 CIFAR-10 Animal/Vehicle Label Shift

For the CIFAR-10 animal/vehicle label shift how data is split between the clusters is shown in Table A.1.

A.2.2 CIFAR-100 Label Shift

For the CIFAR-100 label shift experiments the data is split into three clusters and which labels are in which cluster is presented in Table A.2.

Cluster 1 (60 % of data, 60 clients)	**Cluster 2** (40 % of data, 40 clients)
Bird (2)	Airplane (0)
Cat (3)	Automobile (1)
Deer (4)	Ship (8)
Dog (5)	Truck (9)
Frog (6)	
Horse (7)	

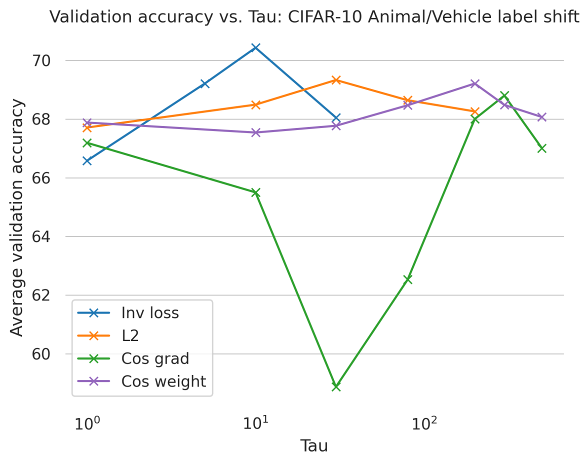
Table A.1: Labels and their index in each cluster in the CIFAR-10 Animal/Vehicle Label Shift experiment setting.

Cluster 1 (50% of data, 26 clients)	**Cluster 2** (25% of data, 13 clients)	**Cluster 3** (25% of data, 13 clients)
Aquatic mammals	Flowers	Household electrical devices
Fish	Food containers	Household furniture
Reptiles	Fruit and vegetables	Large man-made outdoor things
Insects	Trees	Vehicles 1
Large carnivores	Large natural outdoor scenes	Vehicles 2
Large omnivores and herbivores		
Medium-sized mammals		
Non-insect invertebrates		
Small mammals		
People		

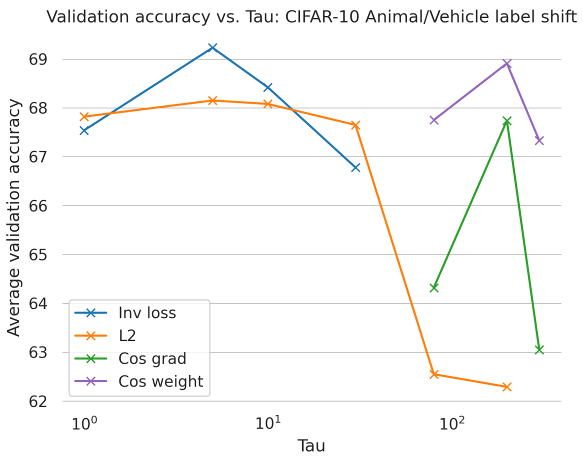
Table A.2: Superclasses of labels in each cluster in the CIFAR-100 Label Shift experiment setting

A.3 τ-tuning

For each experiment setting, the optimal value of the temperature hyperparameter τ for `DAC` was searched for by running experiments with different values of τ. These are presented in Figures A.1-A.6.

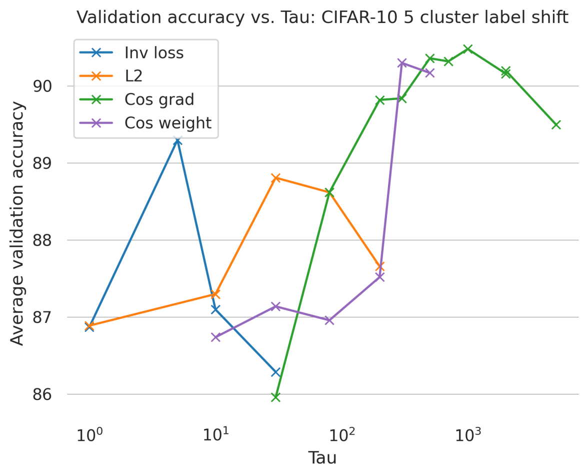


(a) FedAvg

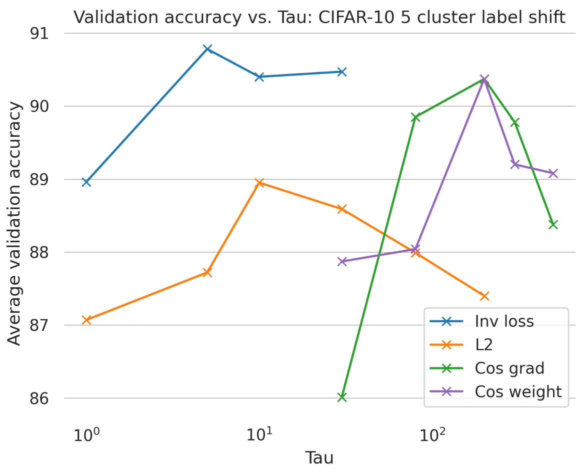


(b) FedSim

Figure A.1: Tau tuning for all similarity metrics and averaging methods for the CIFAR-10 Animal/Vehicle label shift.

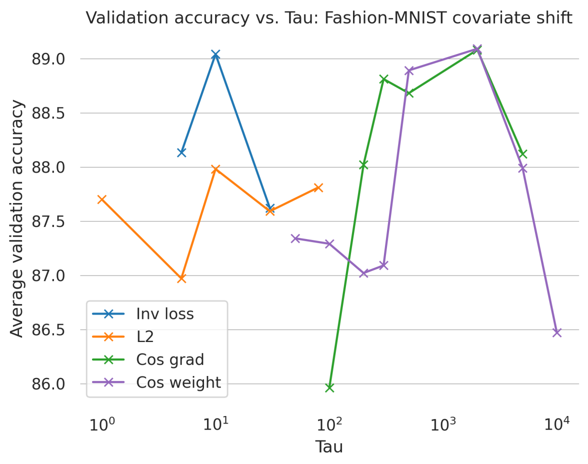


(a) FedAvg

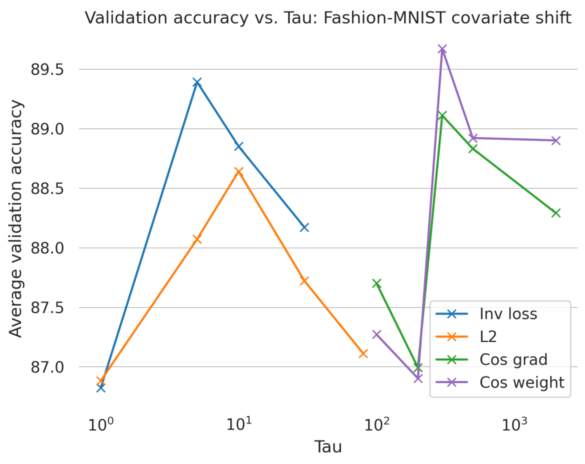


(b) FedSim

Figure A.2: τ-tuning for all similarity metrics and averaging methods for the CIFAR-10 5 cluster label shift.

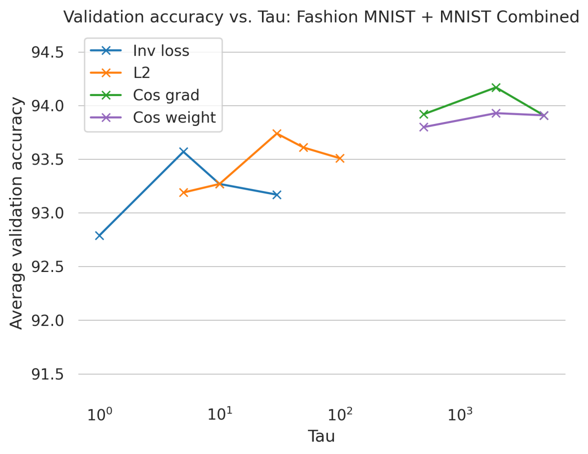


(a) `FedAvg`

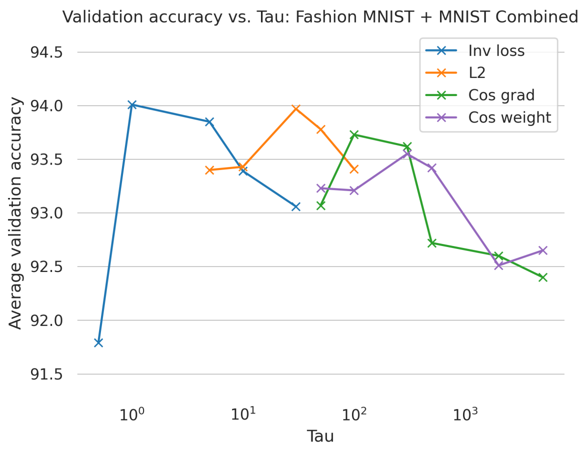


(b) `FedSim`

Figure A.3: τ-tuning for all similarity metrics and averaging methods for the Fashion-MNIST covariate shift.

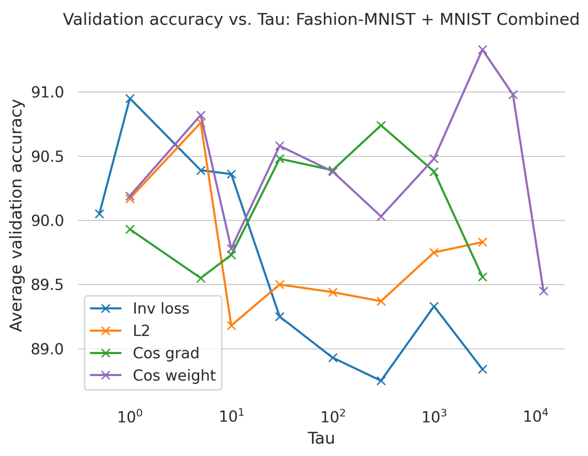


(a) `FedAvg`

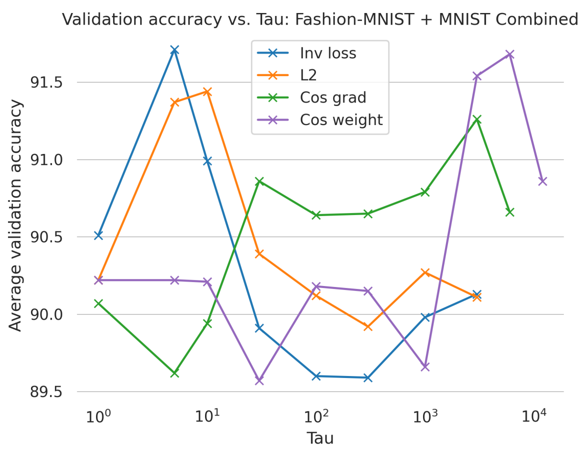


(b) `FedSim`

Figure A.4: τ-tuning for all similarity metrics and averaging methods for the combined Fashion-MNIST and MNIST problem, using the CNN.

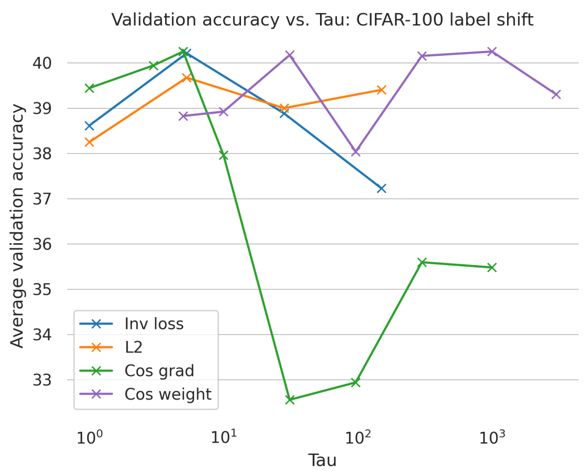


(a) `FedAvg`

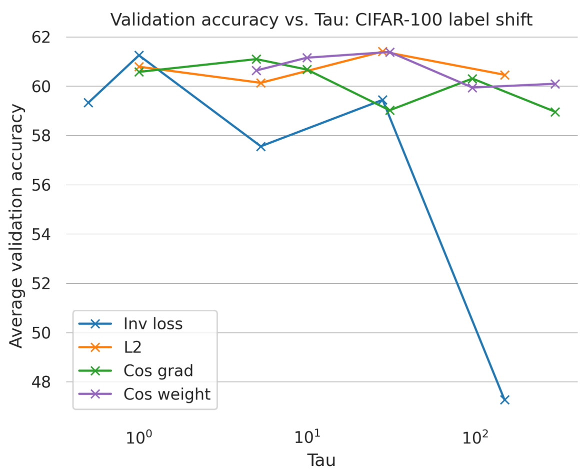


(b) `FedSim`

Figure A.5: τ-tuning for all similarity metrics and averaging methods for the combined Fashion-MNIST and MNIST problem, using the MLP.

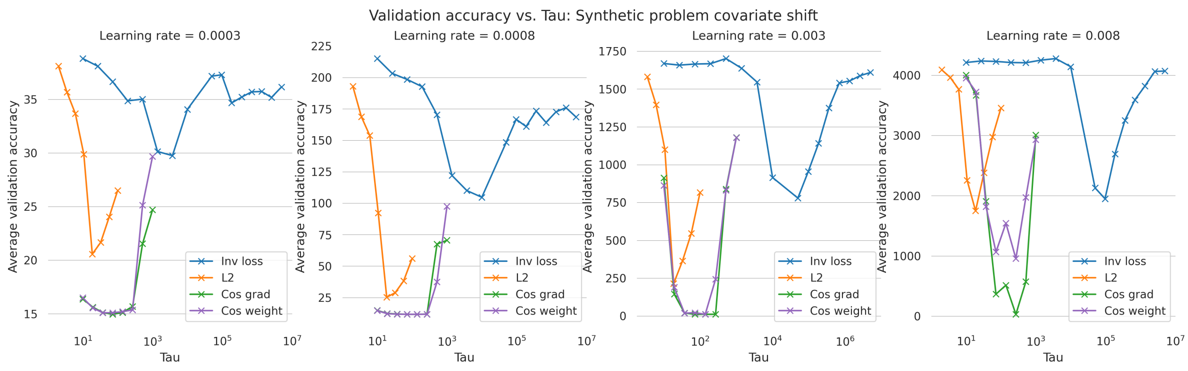


(a) Randomly initialized ResNet-18



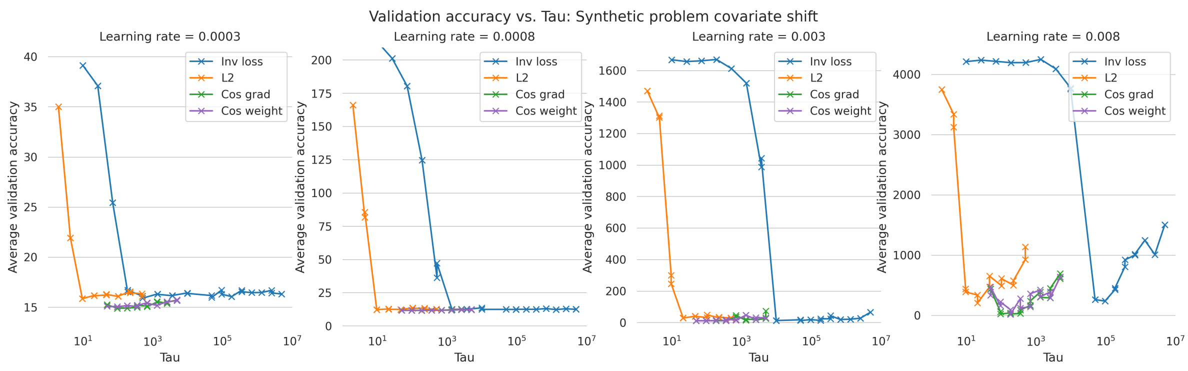
(b) Pre-trained ResNet-18

Figure A.6: τ-tuning for all similarity metrics on pretrained and non-pretrained models for the CIFAR-100 problem.



(a) `FedAvg`

Small Learning rate Large



(b) `FedSim`

Figure A.7: τ-tuning for all similarity metrics and different learning rates for the synthetic problem.

A.4 Summary of Hyperparameter Settings

A summary of used hyperparameters for each experiment setting can be found in Table A.3

Hyperparameters	Experiment name						
	CIFAR-10 animal/vehicles label shift	CIFAR-10 5 cluster label shift	Synthetic problem	Fashion-MNIST covariate shift	Fashion-MNIST and MNIST combined	CIFAR-100 label shift	
Learning rate	0.001	0.001	0.003/0.008	0.0003	0.001	0.0001	
No comm learning rate	0.0001	0.0001	0.008	$5 \cdot 10^{-5}$	$5 \cdot 10^{-5}$	0.0001	
No. of local epochs	1/5/10	1	1	1	1	1	
No. of rounds	300	300	50	300	300	270	
No. of clients	100	100	99	100	100	52	
No. of clients per cluster	40/60	20/20/20/20/20	33/33/33	70/20/5/5	50/50	26/13/13	
Train data size per client	400	400	50	500	400	400	
Validation data size per client	100	100	100	100	100	100	
Early stopping rounds	50	50	50	50	50	50	
No. of neighbors sampled	5	5	5	4	5	5	
No. of labels	10	10	na	10	20	100	
No. of runs per similarity	3	3	15	3	3	3	
Optimizer	Adam	Adam	Adam	Adam	Adam	Adam	
Optimal τ:						pre-trained	not pre-trained
Inv loss, `FedAvg`	10	5	10000	10	1	1	5
Cos grad, `FedAvg`	300	1000	140	2000	2000	30	5
Cos weight, `FedAvg`	200	300	140	2000	2000	30	30
L2, `FedAvg`	30	30	19	10	30	30	5
Inv loss, `FedSim`	5	5	5000	5	1	na	
Cos grad, `FedSim`	200	200	140	300	100	na	
Cos weight, `FedSim`	200	200	140	300	300	na	
L2, `FedSim`	5	10	19	10	30	na	

Table A.3: Summary of all experiment settings used, as well as the found optimal hyperparameters for each experiment.

Appendix B

Personalized Adaptive Neighbor Matching (PANM)

PANM divides its algorithm into two stages. The first stage consists of random communication to collect information on other clients in the network. The second stage uses the information gained in the first stage to only communicate with clients in its own cluster.

The first stage, NSMC, which stands for *Neighbor Selection Based on Monte Carlo*, works by calculating the similarity to x new random clients and comparing them to the similarity scores of the x sampled clients from the last round. Then NSMC picks the top x similarity scores from these (if it is the first round it samples x random clients and calculates their similarity scores). This makes it so that after a few rounds NSMC is prone to merge with the same clients over and over again, see the function NSMC in Algorithm 2.

The second stage starts after T_1 rounds which is a hyperparameter. This stage is called NAEM which stands for *Neighbor Augmentation Based on EM-GMM* and EM-GMM stands for *Expectation Maximization of Gaussian Mixture Model.* From NSMC each client has x clients that a client is quite sure of belongs to their own cluster. NAEM uses this fact and assumes that the similarity of these "tried and tested" clients and other clients in their own cluster are normally distributed and have a different distribution than all clients from other clusters. This can be mathematically formulated for a client i as

$$\begin{gathered} s_i(p) \sim \mathcal{N}(\mu_0, \sigma_0^2), \quad s_i(q) \sim \mathcal{N}(\mu_1, \sigma_1^2) \\ \forall p \in \mathbb{N}_i^*, \quad q \in \overline{\mathbb{N}_i^*}, \end{gathered} \tag{B.1}$$

where $\mathbb{N}_i^*$ refers to the true neighbors of client i (all clients in the same cluster as client i), and $\overline{\mathbb{N}_i^*}$ refers to the false neighbors of client i (clients not in the same cluster as client i). It is assumed that $\mu_0 > \mu_1$ for a reasonable similarity metric. NAEM then uses this assumption when sampling new clients. this is done by assuming that the previous rounds samples are from the same cluster as the sampling client. This means, mathematically, that the samples from the previous round belongs to the normal distribution with the higher mean ($\mathcal{N}(\mu_0, \sigma_0^2)$). NAEM then picks x new random clients that it assumes are not part of its cluster (they belong to $\mathcal{N}(\mu_1, \sigma_1^2)$). Since both these groups of clients have similarity scores we can write the probability that a client j belongs to the probability distribution $\mathcal{N}(\mu_r, \sigma_r^2))$ as

$$\Pr\big(\text{client } j \text{ belongs to } \mathcal{N}(\mu_r, \sigma_r^2) \big| s_i(j)\big) = \beta_r \cdot \mathcal{N}(s_i(j) \mid \mu_r, \sigma_r^2), \tag{B.2}$$

where β_r refers to the overall probability that $s_i(J)$ is generated by distribution $\mathcal{N}(\mu_r, \sigma_r^2)$. If we construct the function

$$\gamma_{j,r} = \begin{cases} 1, & \text{if } j \text{ belongs to distribution } \mathcal{N}(\mu_r, \sigma_r^2) \\ 0, & \text{otherwise} \end{cases} \tag{B.3}$$

we can now estimate μ_r, σ_r, β_r for each client j, notated as $\hat{\mu}_r, \hat{\sigma}_r, \hat{\beta}_r$. These estimates become

$$\hat{\mu}_r = \frac{\sum_{j \in n_1 \cup n_2} \gamma_{j,r} s_i(j)}{n_r}, \quad \hat{\beta}_r = \frac{n_r}{|n_1| + |n_2|}, \quad \hat{\sigma}_r^2 = \frac{\sum_{j \in n_1 \cup n_2} \gamma_{j,r}(s_i(j) - \hat{\mu}_r)^2}{n_r}, \tag{B.4}$$

here n_r refers to the number of clients belonging to $\mathcal{N}(\mu_r, \sigma_r^2)$). Using these estimates each client j gets a probability to belong to a specific distribution and it "moves" to the distribution that it gave the highest probability to (for example a clients goes from being in n_1 to n_2). The estimates are then repeated until there is a round where no clients changes which distribution they gave the highest probability to (in other words the same clients that where in n_1 and n_2 the previous loop are there now). `NAEM` is formulated in Algorithm 3. Since `PANM` uses a lot of notation it is summarized in Table B.1. The whole `PANM` algorithm can be seen in Algorithm 2 and Algorithm 3.

Notation	Meaning
n	Total number of clients
k	Size of aggregation neighbor list
l	Size of neighbor candidate list
τ	Round interval of NAEM in the second stage
N_i^t	Neighbor list of client i in round t
B_i^t	Neighbor bag of client i in round t
C_i^t	Neighbor candidate list of client i in round t
S_i^t	Selected neighbors in EM-step
H_i^t	Neighbor estimation list in EM-step of client i

Table B.1: Specific notations for the `PANM` algorithm

Algorithm 2 Personalized Adaptive Neighbor Matching (`PANM`)

```
Input: $n, k, l, T_1, T_2, \eta, E, \tau, w_0, \mathbf{W}^0 = \{\mathbf{w}_0^1 = \mathbf{w}_0, i \in [n]\}$;
Output: $W^{T_1+T_2}, B$

Initialize empty neighbor lists $N_i$
for each round $t = 1, \ldots, T_1 + T_2$ do
    for each client $i$, $i \in [n]$ in parallel do
        Compute $E$ epochs of local training:
        $w_{i,t} \leftarrow w_{i,t-1} - \eta \nabla F_i(w_{i,t-1})$
        if $t \leq T_1$ then                                  ▷ First stage: Neighbor Selection
            $N_i^t \leftarrow \texttt{NSMC}\left(N_i^{t-1}, i\right)$
            $\mathbf{w}_i^t \leftarrow \text{Aggregation}\left(N_i^t, \mathbf{w}_i^{t-\frac{1}{2}}\right)$
        else                                                  ▷ Second stage: Neighbor Augmentation
            $B_i^{T_1+1} = N_i^{T_1}$
            if $t \equiv 0 \pmod{\tau}$ then
                $B_i^t \leftarrow \texttt{NAEM}\left(B_i^{t-1}, i\right)$
                $N_i^t \leftarrow \text{RandomSample}\left(B_i^t\right)$
                $\mathbf{w}_i^t \leftarrow \text{Aggregation}\left(N_i^t, \mathbf{w}_i^{t-\frac{1}{2}}\right)$
            else
                $B_i^t \leftarrow B_i^{t-1}$
                $N_i^t \leftarrow \text{RandomSample}\left(B_i^t\right)$
                $\mathbf{w}_i^t \leftarrow \text{Aggregation}\left(N_i^t, \mathbf{w}_i^{t-\frac{1}{2}}\right)$


function $\texttt{NSMC}(N, i)$:
    $N \leftarrow N \cup \text{RandomSample}\left(A_i \setminus N\right)$      ▷ $A_i$ is the set of all clients except $i$
    for each client in $N$ do
        Compute $s_i(j)$                                      ▷ $j$ is the clients indices in $N$
    Sort $N$ in descending order of $s_i(j)$-score
    $N \leftarrow N[1 : k]$                                   ▷ Slice the list to get top $k$ clients
    return $N$
```

Algorithm 3 Neighbor Augmentation Based on EM-GMM (`NAEM`)

```
function NAEM($B, i$):
    $C_0 \leftarrow \text{RandomSample}\,(A_i \setminus B)$        ▷ $A_i$ is the set of all clients except $i$
    $C_1 \leftarrow B$
    for each client in $C_0 \cup C_1$ do
        Compute $s_i(j)$
    while the clients in $C_0$ and $C_1$ changes do
        for $r = 0, 1$ do
            $\hat{\mu}_r \leftarrow \frac{\sum_{j \in C_r} s_i(j)}{|C_r|}$
            $\hat{\sigma}_r^2 \leftarrow \frac{\sum_{j \in C_r} (s_i(j) - \hat{\mu}_r)^2}{|C_r|}$
            $\hat{\beta}_r \leftarrow \frac{|C_r|}{|C_0 \cup C_1|}$
        for $j \in C_0 \cup C_1$ do
            if $\hat{\beta}_0 \Phi\left(\frac{s_i(j) - \hat{\mu}_0}{\hat{\sigma}_0}\right) > \hat{\beta}_1 \Phi\left(\frac{s_i(j) - \hat{\mu}_1}{\hat{\sigma}_1}\right)$ then
                $C_0 \leftarrow C_0 \cup j$
                $C_1 \leftarrow C_1 \setminus j$
            else
                $C_1 \leftarrow C_1 \cup j$
                $C_0 \leftarrow C_0 \setminus j$
    $B \leftarrow C_1$
    return $B$
```

"Mm, I get high with a little help from my friends"

— Ringo Starr

Master's Theses in Mathematical Sciences 2024:E28
ISSN 1404-6342

LUTFMA-3536-2024

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/