CHALMERS
UNIVERSITY OF TECHNOLOGY

PENS: Leveraging Data Heterogeneity in Federated Learning

A Decentralized Federated Learning Approach to Create Personalized Models

Master's Thesis in Data Science and AI

GUSTAV KARLSSON

Master's Thesis in Engineering Mathematics and Computational Science

NOA ONOSZKO

DEPARTMENT OF MATHEMATICAL SCIENCES

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021

Master's Thesis 2021

PENS: Leveraging Data Heterogeneity in Federated Learning

A Decentralized Federated Learning Approach to Create Personalized Models

GUSTAV KARLSSON
NOA ONOSZKO

Department of Mathematical Sciences
Division of Applied Mathematics and Statistics
Chalmers University of Technology
Gothenburg, Sweden 2021

PENS: Leveraging Data Heterogeneity in Federated Learning
A Decentralized Federated Learning Approach to Create Personalized Models
GUSTAV KARLSSON
NOA ONOSZKO

Supervisor at company: Edvin Listo Zec, RISE Sweden
Supervisor at university: Moritz Schauer, Department of Mathematical Sciences
Examiner: Tobias Gebäck, Department of Mathematical Sciences

Master's Thesis 2021
Department of Mathematical Sciences
Division of Applied Mathematics and Statistics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2021

PENS: Leveraging Data Heterogeneity in Federated Learning
A Decentralized Federated Learning Approach to Create Personalized Models
Gustav Karlsson
Noa Onoszko
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

Federated learning (FL) is a decentralized machine learning technique where training is done cooperatively by exchanging model weights or gradients instead of sharing the raw data between the cooperating devices (clients). Classical FL algorithms such as federated averaging work best in the special case when the data is IID over clients. In this work, we address the problem of data heterogeneity in federated learning. We propose a decentralized federated learning (DFL) algorithm termed **Pe**rformance-based **N**eighbour **S**election Federated Learning Algorithm (`PENS`), that effectively leverages the data heterogeneity over clients. `PENS` is a cooperative communication-based algorithm where clients communicate with other clients that have a similar data distribution. Specifically, model performance is used as a proxy for data similarity as no raw data is allowed to be shared among clients. Experiments on the CIFAR-10 dataset show that this communication scheme results in higher model accuracies than if clients communicate randomly with each other. The method is robust for different numbers of participating clients as long as the local datasets are sufficiently large.

Keywords: decentralized federated learning, federated learning, data heterogeneity, personalization, distributed machine learning, gossip learning, privacy, image classification

Acknowledgements

We would like to extend our deepest gratitude to our supervisor Edvin Listo Zec for giving us the opportunity to write this thesis project at RISE. We would like to thank him for all of his valuable feedback and guidance during all parts of the project. Special thanks should also go to the rest of the Deep Learning research group at RISE, including Olof Mogren, John Martinsson and Leon Sütfeld, for much appreciated discussions, and for providing the computational resources needed to finish this project. We would also like to thank our Chalmers supervisor, Moritz Schauer for his insightful suggestions. Finally, we also wish to thank our dear friend and fellow student Joel Lidin for providing moral support and valuable advice during the project.

Gustav Karlsson & Noa Onoszko, Gothenburg, May 2021

Contents

1 Introduction

1.1 Background

During the last decade, the development of computing capabilities in our mobile devices has made it possible to deploy advanced machine learning systems for applications such as image recognition and recommender systems. At the same time, in recent years there has been an increased focus on privacy for the users. The introduction of GDPR in the EU, which regulates how companies and organizations manage data from the users [1], is one example of this. Users have also become more aware of the amount of data they are sharing on a daily basis. This development is in large part driven by the increased amount of technology in our homes. With more and more smart devices such as smartphones, smart TVs and cars we are constantly sharing large amounts of data with companies, often without thinking about it. However, with new legal regulations and increased awareness among users, organizations will have to come up with new solutions in order to protect users' privacy.

This is especially true in the field of machine learning where data is essential to develop good models. Historically, the data has been collected from a set of local devices such as smartphones. This data has then been used to train models on a central server. However, if people do not feel like sharing sensitive data, organizations will need a different approach to be able to train their models and still achieve high performance. The combination of increased computational power and the increased privacy concerns has motivated the development of distributed machine learning systems. One such approach is federated learning (FL) [2]. In this framework there exist a possibly large number of users who are connected to a central server. Traditionally, users collaboratively train a global machine learning model by training locally and communicating model weights or gradients with the server. This way, no raw data is shared between the local devices and the central server. However, as communication between the local users and the central server is considered to be expensive, the server's bandwidth can become a bottleneck in the system. Even more importantly, the use of a global model limits the ability to create personalized models.

One approach that deals with these challenges is decentralized federated learning

(DFL). In this framework, the central server is eliminated, letting the local users communicate with each other directly. One crucial aspect of DFL is the communication scheme between the users - how many and which users should each user communicate with, and at what frequency? In previous research, the users to communicate with has mainly been selected either at random or by a predefined communication graph [3], [4], [5], [6]. This aspect becomes even more important if the data is heterogeneous across users, that is, if users have different data generating distributions. For instance, one group of users might be more interested in sports articles while another group is more interested in fashion articles. On the one hand, each user might not have enough local data to successfully train a machine learning model. On the other hand, if the heterogeneity among the users is not taken into consideration, we are not able to create personalized models based on the users' data.

1.2 Problem Formulation

Consider a network of N individual clients, $\mathcal{C} = \{C_1, ..., C_N\}$, as illustrated in Figure 1.1. Each client C_i represents an entity consisting of data and a machine learning model. We define a client C_j that is connected by a directed edge from client C_i to be a neighbour of client C_i. Note that the graph is directed and hence, it does not necessarily have to be the case that two clients are each other's neighbours.

Let us assume there are M different data distributions, $\mathcal{P} = \{P_1, ..., P_M\}$. Each client $C_i \in \mathcal{C}$ has access to n independent feature-label pairs, $\mathcal{Z} = \{z_i^1, ..., z_i^n\}$, where $z_i^l := (x_i^l, y_i^l)$, x denotes the features and y the label. Moreover, $z_i^l \sim P_{k_i}$, where $k_i \in \{1, ..., M\}$ denotes the cluster index for client C_i. In this way, the clients are partitioned into M disjoint clusters, $\mathcal{K} = \{K_1, ..., K_M\}$.

The goal for each client is to obtain a model that performs well on a given task, on its own data. No central server exists and it is not allowed to share data between clients. Instead, the clients are allowed to share model weights. For a given client, C_i, consider a model with parameters, w_i. In mathematical terms, the problem for client C_i then becomes to minimize its local objective function,

$$f_i(w_i) = \mathop{\mathbb{E}}_{z \sim P_{k_i}} [L(w_i; z)], \tag{1.1}$$

where $L(w_i; z)$ is the loss of model w_i for datapoint z. Note that clients belonging to the same cluster will have the same objective function.

1.3 Project Aims

The aim of this report is to contribute to new knowledge in decentralized federated learning. By building upon previous research and introducing new algorithms, we hope to give new insights into the field. More specifically, the aim is to investigate different approaches to improve the performance of decentralized federated learning when data is heterogeneous across clients.

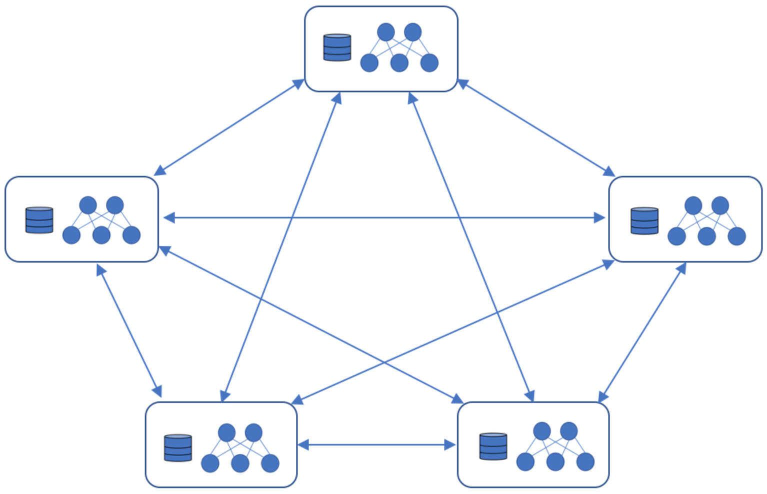


Figure 1.1: A fully connected network where each client has the possibility to communicate with every other client in the graph. Each client has its own local dataset and its own model.

1.4 Limitations

In this study, the primary focus is to investigate different design choices in DFL, especially those regarding the communication scheme. As the goal is to increase performance by leveraging data heterogeneity, we are interested in the performance relative to baselines and not the absolute performance. For this reason, we have used a relatively small neural network and no exhaustive hyperparameter optimization has been performed as this would be too time-consuming.

Generalization is a central concept in machine learning. In a decentralized setting, this can mean different things. One possibility is to strive for generalization on all clients' data. Another goal could be that each client has a model that generalizes to future local data, that is, that the model performs well on current and future data points generated by the client. These two types of generalization abilities are very different, so it might be difficult to achieve both at the same time. In this work, we limit ourselves to trying to achieve the second type of generalization, where each client's model performs well on the local data.

DFL algorithms, ours included, are meant to be used by clients in a decentralized setting. To simplify the implementation, the code for the proposed algorithms is run on one computing unit and clients are iterated over sequentially, instead of acting in parallel. As a consequence, we do not cover the topic of different bandwidth and computational power among clients.

The designed algorithms are generic in the way that they could be used for any problem within supervised learning. However, we only evaluate the algorithms on image classification. Therefore, it is not certain how well the algorithms will perform in other applications.

1.5 Contributions

Our primary contributions are 1) a new method for selecting neighbours in decentralized federated learning, using model performance as a proxy for data similarity; 2) the proposal of a decentralized federated learning algorithm that achieves good personalization performance for a different number of clients in the presence of data heterogeneity.

2

Theory and Related Work

In this chapter, we present the theory required to understand the proposed algorithms in the thesis. First, a brief introduction to convolutional neural networks is given. Secondly, we introduce federated learning and different approaches in decentralized federated learning. Finally, prior research about data heterogeneity in federated learning is presented.

2.1 Convolutional Neural Networks

Convolutional neural networks (CNNs) are special types of artificial neural networks (ANNs) that are inspired by animal visual cortices. Just like the human eye and brain, CNNs are excellent at detecting local details and are therefore often used in tasks such as image analysis, pattern recognition and natural language processing [7]. The main difference between CNNs and traditional ANNs is that CNNs use a convolutional kernel. The kernel is used to extract local features from input data. Moreover, the kernel is shared over all input data which reduces the number of trainable parameters in the network [8]. In contrast, when using an ANN every pixel in an image would be connected to a separate neuron. Even for fairly small images, this would require a huge amount of parameters, making the neural network slow to train. This is an important difference between CNNs and ANNs, and what makes CNNs the more suitable option for image classification.

A CNN typically consists of three different types of layers - convolutional, pooling and fully connected [8]. The convolutional and pooling layers are unique to the CNN and extract features from the input data. The fully connected layers, typically placed at the end of the CNN, then map these features to an output. A CNN can therefore be seen as an extension of an ANN which only consists of fully connected layers. The different types of layers will be explained in more detail in the following chapters.

2.1.1 Convolutional Layer

As the name indicates, the convolutional layer is based on the mathematical operator called convolution. Given two functions, $f, g : \mathbb{Z}^2 \to \mathbb{R}$, the convolution between these at a point $\vec{k} \in \mathbb{Z}^2$ is defined as

$$(f * g)[k_1, k_2] = \sum_{n_1 \in \mathbb{Z}} \sum_{n_2 \in \mathbb{Z}} f(k_1 - n_1, k_2 - n_2) g(n_1, n_2). \tag{2.1}$$

For a convolution layer, f represents the input data coming from the preceding layer and g represents the convolutional kernel. Note that both f and g in the case of convolutional layers usually have finite support, making the number of elements in equation 2.1 fairly small. The output from a convolutional layer can be seen as a weighted average of the input data with the convolutional kernel as weights. The kernel then traverses through the input feature map creating the output feature map which is then passed on to the next layer. This process is illustrated in Figure 2.1. The size of the output feature map depends on the kernel size and how one deals with the boundaries of the input feature map. In the example in Figure 2.1, no special treatment is given to the boundaries which will reduce the size of the feature map. Note that one may use several kernels for the same convolutional layer [9].

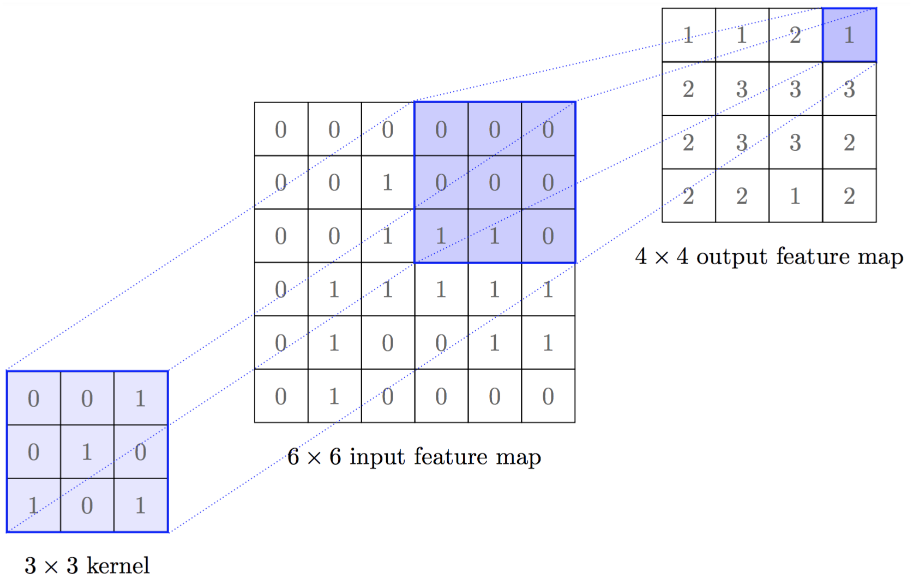


Figure 2.1: The convolutional kernel traverses through the input feature map and outputs a weighted average of the input data.

2.1.2 Pooling Layer

A pooling layer, usually placed after a convolutional layer, collects statistics about each neuron's local neighbourhood [9]. Two of the most commonly used pooling operations are average pooling and max pooling. Max pooling enhances edges by extracting the maximum value from each neuron's neighborhood. An example of max pooling is shown in Figure 2.2. Average pooling, on the other hand, smoothens the feature maps by extracting the average value from each neuron's local neighborhood. Another important consequence of a pooling layer is that it reduces the spatial dimension of the image data. By doing so, fewer parameters are required in the subsequent steps of the network. Moreover, the lower image resolution enables the following convolutional layer to extract features in a different length scale than before.

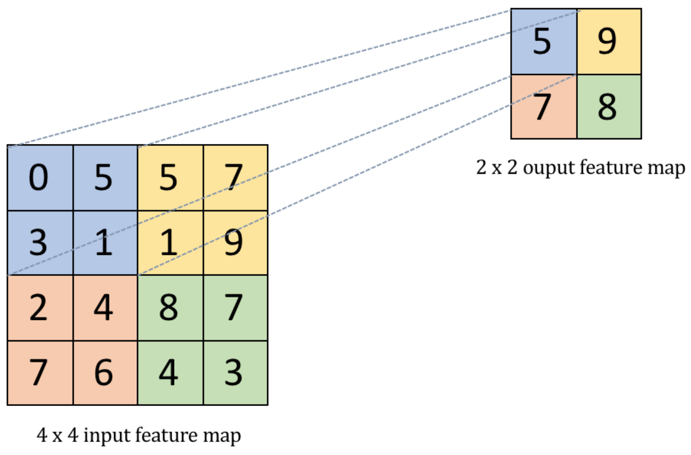


Figure 2.2: Resulting output feature map after applying a 2×2 max pooling layer on the input feature map.

2.1.3 Fully Connected Layer

As mentioned earlier, one or more fully connected layers are typically placed at the end of a CNN [9]. In a fully connected layer, every input node is connected to every output node by a trainable weight. This is illustrated in Figure 2.3. Moreover, every node has a trainable bias associated with it. In mathematical terms, this can be expressed as

$$y_j = \sum_i W_{ij} x_i + b_j, \tag{2.2}$$

where W_{ij} denotes the weight from input node i to output node j, and b_j denotes the bias at node j.

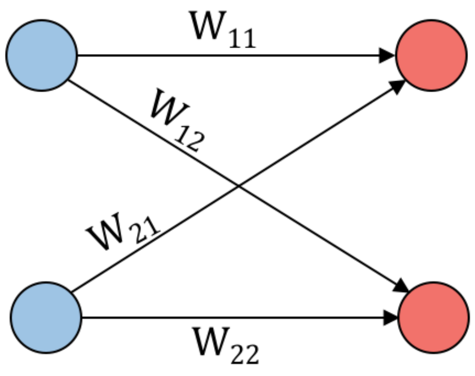


Figure 2.3: A fully connected layer with two input nodes and two output nodes.

2.2 Federated Learning

The term federated learning was coined in 2016 by McMahan et al. [2]. In this paper, a new class of methods is introduced where data from multiple sources are used to create a model without ever storing the data at a central server. Instead, the training is done in the following steps:

1. A global machine learning model is initialized on a central server.
2. The global model is sent to all participating clients.
3. A randomly selected subset of the clients train the model locally for a number of epochs.
4. The newly trained models are sent to the central server.
5. The server computes the new global model as a weighted average of the received, newly trained models, with the size of the corresponding local datasets as weights. Return to step 2 and repeat for a given number of communication rounds.

McMahan et al. [2] also show that in step 1, it is important how one initializes the models as this will affect the averaging. When the models are initialized with the same parameters a lower loss is achieved than when the models are initialized with different parameters.

2.2.1 Decentralized Federated Learning

In decentralized federated learning, there exists no central server. Instead, the clients have to communicate directly with each other to exchange information. A variety of different approaches have been studied in DFL and most of them follow a similar pattern to centralized FL. However, no global model can be used as no central server exists. Instead, each client has its own local model. For a given client, the training process can be described with the following steps:

1. Initialize a local machine learning model.
2. Select a set of clients and collect their models.
3. Merge with the selected group of clients.
4. Train the model locally for a number of epochs.
5. Return to step 2 and repeat for a given number of communication rounds.

Note that although most proposed algorithms follow the above structure, there might be different ways of doing DFL. Step 3 described above is, similarly to centralized FL, most commonly done by taking a weighted average of the received model parameters (including its own model). However, the client selection differs a lot between different approaches. One approach used in [10] is to use a network graph where each client only communicates with its one-hop neighbours. However, it is not trivial how one would create such a graph. Another approach is to select all available clients at every round [5]. In this way, each client gets a lot of new information at each communication round. However, it can be very time-consuming to communicate with all other clients. Thus, this approach is best suited when working with a limited number of clients.

There are also strategies that incorporate randomness in the client selection step. One example of this is Gossip Learning [11], [12]. In this framework, the client sends its model to a randomly selected peer. When a client receives a model, it updates its current model by merging it with the received one. This client will later send its updated model to a new, randomly selected peer. In this way, the initial model will perform a random walk over the network. Note that this process is fully asynchronous as the clients are not required to merge or send their models at the same time. Furthermore, note that, contrary to earlier mentioned approaches, only 2 models are merged at the same time in gossip learning.

One aspect that differs slightly from centralized FL is the model initialization in step 1. While common initialization of models might be a trivial task in centralized FL, this can be more challenging in DFL where no central server is used. In centralized FL, a central server makes sure that a global model is distributed to all clients before the local training begins. Hence, all clients start with the same model at the beginning of each communication round. In DFL, this is not possible except at the very beginning of training. When new clients join the federation, the old clients will all have different models parameters which makes common initialization impossible.

2.2.2 Non-IID Data in Federated Learning

The local dataset for a given client corresponds to the usage of that user's personal device. Each user has their own preferences and will therefore be exposed to different data points. The distribution of the local data for a given client will therefore most likely differ from the data distribution of the entire population. Because of the natural heterogeneity among users in the decentralized setting and the potential to develop personalized models, non-IID data has become one of the most researched aspects in federated learning.

To understand in which situations data heterogeneity might arise, let us first assume that there exists a set of clients, where $\mathcal{Q}$ denotes the distribution over available clients. Here, $\mathcal{Q}$ gives the probability of drawing a client $C_i \in \mathcal{C}$, when a subset of clients is selected. Furthermore, let $P_i(x, y)$ denote the local data distribution for client $C_i \sim \mathcal{Q}$, where x denotes the features and y the labels. In this setting, there are a few different kinds of data heterogeneity that should be considered. First of all, the distributions for two different clients, P_i and P_j can differ. Secondly, the local data distribution, P_i for a given client C_i as well as the distribution over available clients, $\mathcal{Q}$, might differ over time. Although both are important cases of heterogeneity, we will mainly focus on the former.

There are a few different scenarios where $P_i \neq P_j$. To discuss these further, first note that $P_i(x, y)$ can be rewritten as $P_i(y|x)P_i(x)$ or $P_i(x|y)P_i(y)$. It is now possible to identify four types of heterogeneity in the data distribution [13].

- *Feature distribution skew*: $P_i(x) \neq P_j(x)$ while $P_i(y|x) = P_j(y|x)$. For instance, images from users in Spain might have a higher pixel intensity than images taken in Sweden.
- *Label distribution skew*: $P_i(y) \neq P_j(y)$ while $P_i(x|y) = P_j(x|y)$. One example of this is when the distribution of labels varies between different subgroups of the population. For instance, people in Asia might eat different kinds of fruit than what is eaten in Europe.
- *Same label, different features*: $P_i(y) = P_j(y)$ while $P_i(x|y) \neq P_j(x|y)$. In this case, the features, x varies even though the label, y is the same. On example of this is that a picture of a house might look very different depending on where and in what conditions the image is taken. For instance, a house in New York will, due to economic, cultural and geographic differences, look vastly different from a house in Sápmi.
- *Same features, different label*: $P_i(x) = P_j(x)$ while $P_i(y|x) \neq P_j(y|x)$. Even though the features are the same, they might be interpreted differently in different subgroups of the population.

Furthermore, a real-world dataset will likely contain a combination of the above-mentioned effects.

There have been plenty of articles studying the effects of data heterogeneity in federated learning. For instance, McMahan et al. [2] show that federated averaging performs well on heterogeneous data in the case of label distribution skew, although more communication rounds are required than if the data was IID. These results have been disputed as [14] shows that federated averaging struggles to achieve the same performance for non-IID data as for IID data. Moreover, the federated averaging algorithm generates a global model. Hence, the algorithm is limited in the way that it can not create personalized models based on the local data distribution of each client. One common approach in the machine learning field to achieve personalization is to first train the model on global data and then fine-tune it using a local dataset. This approach is used in transfer learning [15] and meta-learning [16], and was first introduced in distributed learning by Wang et al. [17]. First, a global model is obtained by using the standard federated averaging algorithm introduced by [2]. By fine-tuning this global model on the local datasets, personalization could be achieved, which resulted in higher overall accuracy.

Ghosh et al. [18] continue the research on data heterogeneity in FL by addressing the issue of feature distribution skew. This is simulated by rotating CIFAR-10 and MNIST images, creating disjoint data clusters, and assigning data points from only one cluster to each client. A new framework is proposed where k global models are initialized, each representing one cluster. In an iterative process, clients are assigned to clusters based on the loss function of the global models and these models are then updated by merging the local models that correspond to the assigned clients. The idea behind this algorithm is to both capture the personalization aspect and to take advantage of an increased amount of data by only merging with clients that have the same feature distribution. Another centralized FL algorithm proposed by Listo

Zec et al. [19] handles the problem with data heterogeneity by using a mixture of experts. First, a global model is trained with federated averaging. Then, specialist models are generated by fine-tuning the global model locally without cooperation. In the final step, a weighted average of the specialist model and the global model is trained with a neural network where a gating function is used to control the weight of each model. In this step, the weights of the gating function and specialist model are updated, whereas the global model weights are frozen.

Most existing research in decentralized federated learning is focused on obtaining a global model [20], [21], [22], [23]. Although this makes sense when looking for a consensus amongst the clients, there might be situations where one wants to minimize the loss for local data. Moreover, the ability to create personalized models becomes more natural in decentralized federated learning as there is no need for a global model. Roy et al. [5] showed that a server-less peer-to-peer approach is able to achieve higher performance on non-IID data than what is possible with federated averaging. Each client collects models from all other clients and merges its model with those by taking a weighted average. This is followed by local training and then the process is repeated a number of times. Hence, no special consideration is taken in regards to the data heterogeneity. This illustrates that DFL has a natural ability to perform well in the case of data heterogeneity.

Almeida and Xavier [24] propose a more involved solution in which they account for similarities between the clients. Each client aims to find a model which minimizes both the loss on the local dataset and the differences with its neighbours' model weights. A network matrix W is used where W_{ij} controls the level of similarity that we want client C_i and C_j to have. Thus, for a given client, W controls which neighbours should be considered close. Furthermore, Almeida and Xavier state that W_{ij} can be chosen based on the similarity between client C_i's and client C_j's local datasets. Vanhaesebrouck et al. [25] and Bellet et al. [26] have achieved promising results on non-IID data with similar approaches using a similarity graph as a regularization factor. However, the authors do not provide a suggestion on how to calculate this similarity measure in a decentralized setting where the data is private.

Zantedeschi et al. [27] extend previously mentioned research by learning the similarity graph along with the model. In this approach, the objective function consists of both model weights and weights from the similarity graph. The algorithm is divided into two parts. First, given a fixed similarity graph W, personalized models are trained in a similar way to what is described by Almeida and Xavier [24]. Then, the graph weights W_{ij} are optimized given fixed model weights. In this step, clients expand their neighbourhood by randomly communicating with new clients that do not currently belong to their direct neighbourhood (the clients with the highest similarity score in the current similarity graph). The similarity graph is then updated by applying gradient descent to the objective function. These two steps are then alternated until convergence is reached. Results show that the algorithm manages to approximately identify the correct clusters among the clients.

In conclusion, there has been plenty of research on DFL and on data heterogeneity but not many articles have studied how to leverage data heterogeneity to create personalized models. Therefore, we aim to propose a new method for leveraging non-IID data to achieve personalization in DFL.

3

Methods

3.1 Dataset

Federated learning is a problem-agnostic technique for decentralized machine learning. However, the literature mainly deals with FL in a supervised learning setting and image classification is not uncommon. Similar to previous work, we chose to evaluate the different FL methods on an image classification problem, using the CIFAR-10 [28] dataset. The dataset contains color images of ten common objects, see Figure 3.1, and is partitioned into a training set of 50,000 images and a test set of 10,000 images. Each image has three color channels and is of dimension 32x32. Despite the low resolution, the best performing classifier to date has achieved 99.7% accuracy [29]. The images are preprocessed only by normalizing each image channel so that the pixel values for each image are $N(0.5, 0.5)$-distributed. This is done to improve learning since this will for example make gradients have more similar sizes.

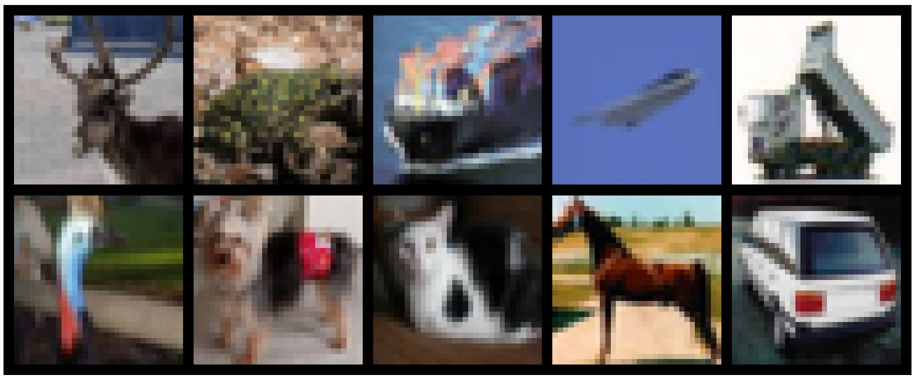


Figure 3.1: CIFAR-10 sample images. The figure shows images from each class in the dataset.

3.2 Network Architecture

The network used is a small convolutional neural network with three convolutional layers, each followed by max pooling, and two fully connected layers. The full network architecture is shown in Table 3.1. The network has 73418 trainable parameters and assuming a float is 24 bytes, takes approximately 1.8MB of space.

Table 3.1: Network architecture

Layer	#Channels	Filter Size	Activation	Output Size	#Parameters
Input Layer	-	-	-	32x32x3	-
Convolutional	32	3x3	ReLU	30x30x32	896
Max Pooling	1	2x2	-	15x15x32	-
Convolutional	64	3x3	ReLU	13x13x64	18496
Max Pooling	1	2x2	-	6x6x64	-
Convolutional	64	3x3	ReLU	4x4x64	36928
Max Pooling	1	2x2	-	2x2x64	-
Fully Connected	-	-	Linear	64	16448
Fully Connected	-	-	Softmax	10	650

3.3 Algorithms

Below, we present the implemented DFL algorithms. Note that all algorithms are described from the perspective of a single client. This means that each client will follow the given algorithm. Moreover, to avoid duplication, a function defined in one algorithm might be used in others without repeating the definition again. Values for the hyperparameters used in the algorithms and during training are shown in Table 3.2 and Table 3.3. As a note on notation: In the algorithms descriptions, sets are denoted by calligraphic letters and lists are denoted by brackets.

3.3.1 `Random`

First, we introduce a new DFL method, termed `Random` which is inspired by [10] and [5]. However, instead of selecting the same clients for each communication round, a random subset of clients is selected at each round. It is an iterative algorithm that switches between local training and averaging model parameters with other clients, which has great similarities with gossip learning. However, in contrast to gossip learning where a client sends its model to other clients, the client receives models from a number of other clients. In the averaging step, it is common to weigh the model of each client with the size of its local dataset. We omit it here from the algorithm description since we will work with balanced data, that is, each client has the same number of data points. For a full description of the method, see Algorithm 1.

3.3.2 Improvements for Non-IID Data

`Random` does not take data heterogeneity into account. To leverage this to create personalized models, four new algorithms were developed: `Greedy` in Algorithm 2, `EpsilonGreedy` in Algorithm 3, `PENS` in Algorithm 4 and `RandomWeighted` in

Algorithm 1 Random

```
 1: Initialize network weights $w$
 2: $w \leftarrow$ TrainLocally($w$)
 3: for round $t$ in 1,2,... do
 4:     Select $m_{\text{selected}}$ clients $\mathcal{C}_t \subset \mathcal{C}$ randomly
 5:     $w \leftarrow$ ModelAveraging($w, \mathcal{C}_t$)
 6:     $w \leftarrow$ TrainLocally($w$)
 7: procedure ModelAveraging($w$, $\mathcal{C}'$)
 8:     Receive models $\vec{w}$ from clients $\mathcal{C}'$
 9:     $\vec{w} \leftarrow \vec{w} \cup \{w\}$
10:     $w \leftarrow \frac{1}{m_{\text{selected}+1}} \sum_{w' \in \vec{w}} w'$
11:     return $w$
12: procedure TrainLocally($w$, $E$)
13:     for each local epoch 1 to E do
14:         for batch $b \in \mathcal{B}$ do
15:             $w \leftarrow w - \eta \nabla L(w; b)$
16:     return $w$
```

Algorithm 5. In all four methods, the model performance on other clients is used as a tool to improve the averaging process, either by selecting models based on accuracy or weighing the models with the accuracy as in `RandomWeighted`. All algorithms build upon `Random`. For `Greedy`, `EpsilonGreedy` and `PENS`, the main difference from `Random` is in line 4 in Algorithm 1 where clients now are selected based on data similarity instead of randomly. A client sends its model to a number of other clients and requests that they evaluate it on their local data. The client then receives classification accuracies and from this point, the algorithms differ. `Greedy` then selects the subset of clients that report the highest accuracies and the averaging is carried out with those. `EpsilonGreedy` does the same thing, except some of the selected clients are swapped out randomly.

As a further improvement, we propose `PENS`, short for **Pe**rformance-based **N**eighbour **S**election Federated Learning Algorithm. It consists of two steps: in the first step, the goal is to find the right clients to communicate with and in the second step, the `Random` algorithm is executed to train the model while collaborating with the clients gathered in the first step. The idea is to find clients that have similar data, by using accuracy as a proxy. The first step is similar to the `Greedy` algorithm, as clients are selected greedily using the accuracies that they report. The difference is that in each round, before the local training, the client sampling and subsetting process is repeated a number of times, and the selected clients at each repetition are added to a list of communication history. After the final round, a subset of the clients present in the communication history is selected as the client population for step 2. The clients that occur more frequently than the expected count, if clients were chosen randomly, are selected. Finally, in step 2, `Random` is executed with the constraint that the client now only communicates with clients in the new client population.

In the `RandomWeighted` method, the clients are still selected randomly. The main difference from `Random` is instead in the averaging process. The accuracies collected from evaluating the model on other clients' data are here used as weights in the averaging process. This is illustrated in Algorithm 5.

Model initialization can either be done with identical or different random seeds across clients. To investigate the effect of using identical or different random seeds, all experiments are done with and without identical random seeds.

Algorithm 2 Greedy

```
 1: Initialize network weights $w$
 2: $w \leftarrow$ TrainLocally($w$)
 3: for round $t$ in 1,2,... do
 4:     $\mathcal{C}^t_{\text{selected}}, \mathcal{C}^t_{\text{sampled}} \leftarrow$ SelectclientsGreedily()
 5:     $w \leftarrow$ ModelAveraging($w, \mathcal{C}^t_{\text{selected}}$)
 6:     $w \leftarrow$ TrainLocally($w$)
 7:
 8: procedure SelectclientsGreedily()
 9:     Select $m_{\text{sample}}$ clients $\mathcal{C}_{\text{sampled}} \subset \mathcal{C}$ randomly
10:     $\vec{A} \leftarrow []$
11:     for client in $\mathcal{C}_{\text{sampled}}$ do
12:         Send $w$ to client and request that it evaluates $w$ on its training data
13:         Receive classification accuracy $a_i$ from client
14:         Append $a_i$ to $\vec{A}$
15:     Select $m_{\text{selected}}$ clients $\mathcal{C}_{\text{selected}} \subset \mathcal{C}_{\text{sampled}}$ with the $m_{\text{selected}}$ highest accuracies
        in $\vec{A}$
16:     return $\mathcal{C}_{\text{selected}}, \mathcal{C}_{\text{sampled}}$
```

Table 3.2: Hyperparameters used in the `Random`, `Greedy`, `EpsilonGreedy` and `RandomWeighted` where E denotes the number of local epochs and lr denotes the learning rate.

Method	E	Batch size	lr	m_{selected}	m_{sample}	(ε, λ)
`Random`	3	8	10^{-3}	20	-	-
`Greedy`	3	8	10^{-3}	20	-	-
`EpsilonGreedy`	3	8	10^{-3}	4	20	$\{(0.2, 1), (0.5, 1), (1, 0.99)\}$
`RandomWeighted`	3	8	10^{-3}	20	-	-

Algorithm 3 Epsilon-Greedy

```
 1: Initialize network weights $w$
 2: $w \leftarrow$ TrainLocally($w$)
 3: for round $t$ in 1,2,... do
 4:     $\mathcal{C}_t^{\text{selected}} \leftarrow$ SelectclientsEpsilonGreedily($t$)
 5:     $w \leftarrow$ ModelAveraging($w, \mathcal{C}_t^{\text{selected}}$)
 6:     $w \leftarrow$ TrainLocally($w$)
 7:
 8: procedure SelectclientsEpsilonGreedily($t$)
 9:     Select $m_{\text{sample}}$ clients $\mathcal{C}_{\text{sampled}} \subset \mathcal{C}$ randomly
10:     $\vec{A} \leftarrow []$
11:     for client in $\mathcal{C}_{\text{sampled}}$ do
12:         Send $w$ to client and request that it evaluates $w$ on its training data
13:         Receive classification accuracy $a_i$ from client
14:         Append $a_i$ to $\vec{A}$
15:     Select $m_{\text{selected}}$ clients $\mathcal{C}_{\text{selected}} \subset \mathcal{C}_{\text{sampled}}$ with the $m_{\text{selected}}$ highest accuracies
        in $\vec{A}$
16:     Sample $n_{\text{swap}}$ from the $\text{Bin}(m_{\text{selected}}, \lambda^t \varepsilon)$ distribution
17:     Select $n_{\text{swap}}$ clients $\mathcal{C}_{\text{swap}}$ in $\mathcal{C}_{\text{selected}}$ randomly
18:     $\mathcal{C}_{\text{new population}} \leftarrow (\mathcal{C}_{\text{sampled}} \setminus \mathcal{C}_{\text{selected}}) \cup \mathcal{C}_{\text{swap}}$
19:     Select $n_{\text{swap}}$ clients $\mathcal{C}_{\text{new}}$ in $\mathcal{C}_{\text{new population}}$ randomly
20:     $\mathcal{C}_{\text{selected}} \leftarrow (\mathcal{C}_{\text{selected}} \setminus \mathcal{C}_{\text{swap}}) \cup \mathcal{C}_{\text{new}}$
21:     return $\mathcal{C}_{\text{selected}}$
```

Algorithm 4 PENS

```
1: **Step 1: Collect communication history**
2: Initialize network weights $w$
3: $\vec{C}_{\text{selected}} \leftarrow []$
4: $\mathcal{C}_{\text{sampled}} \leftarrow \varnothing$
5: $w \leftarrow \textsc{TrainLocally}(w)$
6: for round $t$ in 1,2,... do
7:     for round j in 1 to $n_{\text{samplings}}$ do
8:         $\mathcal{C}^t_{\text{selected}}, \mathcal{C}^t_{\text{sampled}} \leftarrow \textsc{SelectclientsGreedily}()$
9:         Append $\mathcal{C}^t_{\text{selected}}$ to $\vec{C}_{\text{selected}}$
10:        $\mathcal{C}_{\text{sampled}} \leftarrow \mathcal{C}_{\text{sampled}} \cup \mathcal{C}^t_{\text{sampled}}$
11:    $w \leftarrow \textsc{ModelAveraging}(w, \mathcal{C}^t_{\text{selected}})$
12:    $w \leftarrow \textsc{TrainLocally}(w)$
13: $\mathcal{C}_{\text{neighbours}} \leftarrow \textsc{SelectNeighbours}(\vec{C}_{\text{selected}}, \mathcal{C}_{\text{sampled}})$
14:
15: **Step 2: Train with a subset of clients**
16: Perform Algorithm 1, only communicating with clients in $\mathcal{C}_{\text{neighbours}}$
17:
18: procedure $\textsc{SelectNeighbours}(\vec{C}_{\text{selected}}, \mathcal{C}_{\text{sampled}})$
19:    $T \leftarrow \frac{|\vec{C}_{\text{selected}}|}{|\mathcal{C}_{\text{sampled}}|}$
20:    $\mathcal{C}_{\text{neighbours}} \leftarrow \varnothing$
21:    for client in $\mathcal{C}_{\text{sampled}}$ do
22:        $c \leftarrow$ number of occurrences of client in $\vec{C}_{\text{selected}}$
23:        if $c > T$ then
24:            $\mathcal{C}_{\text{neighbours}} \leftarrow \mathcal{C}_{\text{neighbours}} \cup \{\text{client}\}$
25:    return $\mathcal{C}_{\text{neighbours}}$
```

Algorithm 5 Weights

```
1: Initialize network weights $w$
2: $w \leftarrow$ TRAINLOCALLY($w$)
3: for round $t$ in 1,2,... do
4:     Select $m_{\text{selected}}$ clients $\mathcal{C}_t \subset \mathcal{C}$ randomly
5:     $w \leftarrow$ WEIGHTEDMODELAVERAGING($w, \mathcal{C}_t$)
6:     $w \leftarrow$ TRAINLOCALLY($w$)
7: procedure WEIGHTEDMODELAVERAGING($w$, $\mathcal{C}'$)
8:     $\vec{A} \leftarrow []$
9:     for client in $\mathcal{C}'$ do
10:        Send $w$ to client and request that it evaluates $w$ on its training data
11:        Receive classification accuracy $a_i$ from client
12:        Append $a_i$ to $\vec{A}$
13:    Receive models $\vec{w}$ from clients $\mathcal{C}'$
14:    Evaluate $w$ on the local validation set and denote the accuracy by $a$
15:    $\vec{w} \leftarrow \vec{w} \cup \{w\}$
16:    $\vec{A} \leftarrow \vec{A} \cup \{a\}$
17:    $w \leftarrow \sum_{(a',w') \in (\vec{A},\vec{w})} \frac{a'}{a} w'$, where $a = \sum_{a' \in \vec{A}} a'$
18:    return $w$
```

Table 3.3: Hyperparameters used for PENS for different number of clients. E denotes the number of local epochs and lr denotes the learning rate for the SGD optimizer that was used during training.

Clients	E	Batch size	lr	m_{selected}	m_{sample}	$n_{\text{samplings}}$
50	2	8	10^{-3}	1	5	10
200	10	8	10^{-3}	4	20	10
1000	50	8	10^{-3}	4	20	10

3.4 Experimental Setup

The algorithms described in the previous section are described from the perspective of one client. To simplify the coding, instead of making a parallel implementation where clients send requests to each other, as described in the algorithms, the communication and training were carried out sequentially with respect to clients. First, all clients trained locally in a sequence. Then, one client averaged with a number of other clients, followed by the same procedure for another client and so on, until all clients have averaged their models. In the averaging step, clients are iterated over in random order with new random order for each communication round.

To simulate heterogeneous data across clients, the training set and test set were partitioned into two datasets ($M = 2$), each of equal size. The images in one of them were rotated 180° and the other dataset was left unchanged and we denote their generating distributions by P_1 and P_2. Half of the clients were assigned data points from the rotated dataset and the other half got the untransformed images, creating two client clusters C_1 and C_2. The clients were given an equal number of data points from the training set, and the whole training set was used. With for example 50,000 train data points and 200 clients ($N = 200$), each client was assigned 250 data points ($n = 250$). The data points of each client were split into a training set and a validation set. The size of the training set and validation set for the different number of clients is shown in Table 3.4. The training set was used to train the models and to evaluate other clients' models. The validation set was used for early stopping. Early stopping was carried out on each client separately, using a moving average of the validation accuracy. Training was stopped when the moving average of the validation accuracy had not increased for a number of communication rounds, determined by the patience, or when the maximal number of communication rounds had been reached. See Table 3.5 for details. When a client stops early, it goes back to its model at a previous time step where the maximal accuracy was achieved. From the point of stoppage, the client no longer updates its own model but it still participates in the training of other models by evaluating other clients' models on its dataset and sending its own model to other clients. The training continues either until all clients have stopped early, or if the maximum number of total local epochs for each client has been reached.

The algorithms were implemented in Python 3.8.3 and PyTorch 1.7.1 [30] was used for the machine learning parts. Each experiment was repeated four times in order to get information about the statistical uncertainty of the results. The experiments were carried out on single v100-sxm2-32gb and RTX 2080 TI GPUs.

Table 3.4: Size of the training set and validation set for different number of clients.

Number of clients	50	200	1000
training set size	875	125	25
Validation set size	125	125	25

Table 3.5: Hyperparameters related to the length of training.

Hyperparameter	**Value**
Maximum number of rounds	333
Early stopping patience	50
Early stopping window size	5

3.5 Evaluation

After training, each client's model was evaluated on the test set that corresponds to its own data partition. The entire CIFAR-10 test set consisting of 10,000 images was used for each client and depending on the client's cluster it was left unchanged or rotated. This way, the model's personalization ability was measured, that is, how well it generalized to unseen examples from its own data generating distribution.

Two evaluation metrics were used: classification accuracy and communication cost. The classification accuracy is a performance metric that measures the ratio of correctly classified images. The communication cost measures the amount of communication between clients. We define the communication cost of a client as the number of times it receives a model or sends its own model to another client. We report the average communication cost over all clients participating in an experiment.

Table 3.6: Baselines used to evaluate the performance of the algorithms.

Name	**Explanation**
`Local`	One client trains without federation on the same number of data points as the algorithm it is compared with, e.g. 125.
`CentralIIDHalfData`	One client trains on half of the dataset, without rotating any images. We call this central training since all data that is used is stored at one place.
`CentralNonIID`	One client trains on the full dataset, where half of the images are rotated and half are unchanged.
`Random`	The `Random` algorithm with the same number of clients and data points as the algorithm it is compared with.
`Oracle`	`Random` but where each client only communicates with clients that belong to the same cluster.

To evaluate the performance of the algorithms, several baselines were used, see Table 3.6. `Local` is an easy baseline to beat. It is, however, possible for an FL algorithm to perform worse since there is no guarantee that averaging models improves performance. The performance of `Oracle` is supposedly an upper bound for any DFL algorithm given the data partitioning, assuming that communicating with clients from the other data partition is detrimental. `CentralNonIID` represents the case where all data is stored centrally but no data partitioning has taken place and `CentralIIDHalfData` is central training with half of the datapoints without any rotation. The accuracies of these baselines are also upper bounds if we make the rather plausible assumption that central training is better than FL with the same total number of data points across clients. They are however interesting to compare as they show how much worse the possible upper bound baseline, `Oracle`, performs. `Random` is also on the list for one important reason. Namely, it does not take into account that the data can be heterogenous over clients. Therefore, the hope is that the other algorithms that do take this into account will perform better.

4

Results

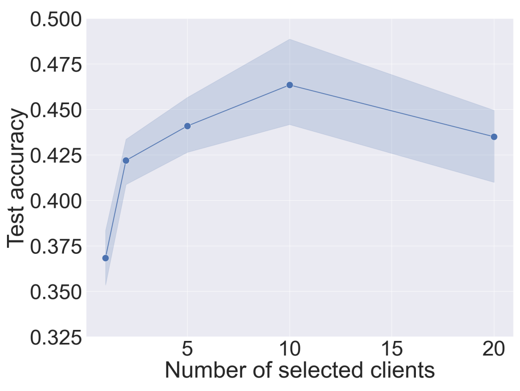
In this chapter, we present the results for all algorithms and baselines mentioned in the previous chapter in terms of performance and communication. First, we present results from varying some of the hyperparameters in `Random` using IID data. Then, we present results for all baselines and algorithms on non-IID data, both with regards to performance and communication patterns among clients.

Note that all presented accuracies below are averages taken over the clients' individual performances on the test dataset. From this point on, this average accuracy will be denoted test accuracy. 95% confidence intervals over four identical repetitions are reported for each experiment.

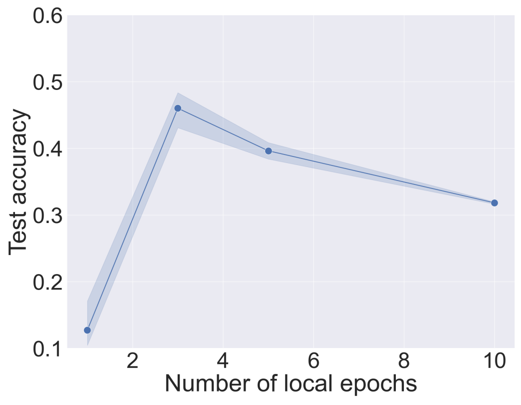
4.1 `Random` on IID data

First, we introduce the proposed `Random` algorithm by studying how the number of selected clients, the number of local epochs and the learning rate affect the performance. Note that this should not be considered as an exhaustive hyperparameter search. Instead, one parameter is varied at a time while keeping all other parameters fixed. The fixed values used are taken from the `Random` row in Table 3.2. The results can be seen in 4.1. The results indicate that out of the tested values, the best values are 10 for the number of selected clients, 3 for the number of local epochs and 10^{-3} for the learning rate. It is interesting to see how poor the performance is when only using one local epoch. The accuracy seems to increase as the number of selected clients increases. This is expected as each client gets a larger exchange of information at each communication round. However, Figure 4.1a indicates a drop in performance when exceeding ten selected clients.

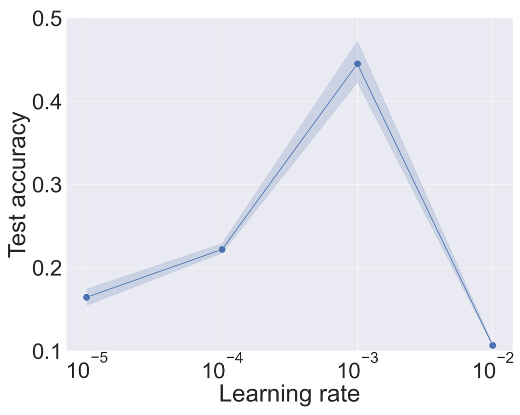
In the following, a learning rate of 10^{-3}, three local epochs and 20 clients to communicate are used. Table 4.1 shows how `Random` compares to `Local` and central training with these values for the hyperparameters. Central training was performed on the full training set. Note that `Random` performs much better than the local models which indicate that the federation is working. However, the achieved test accuracy is much lower than for central training, see Table 4.1.



(a)



(b)



(c)

Figure 4.1: Test accuracies with 95% confidence intervals when varying the number of selected clients (a), the number of local epochs (b) and the learning rate (c). Note that the x-axis in (c) is in logarithmic scale.

Table 4.1: Test accuracy with corresponding 95% confidence interval for `Local`, `Random` and central training on all datapoints.

`Local`	`Random`	Central training
24.6 ± 0.1	46.0 ± 3.9	69.3 ± 1.0

4.2 Comparison of Methods on Partitioned Data

Table 4.2 shows the accuracy and communication cost for all implemented methods and baselines. When using independent model initialization all algorithms that take data heterogeneity into account got higher mean accuracies than `Random` and `PENS` performs almost as well as `Oracle`. For common initialization, however, only `RandomWeighted` and `PENS` got higher mean accuracies than `Random`. Overall, `PENS` got the highest accuracy in both cases. However, the communication cost is almost three times as high compared to `Random`.

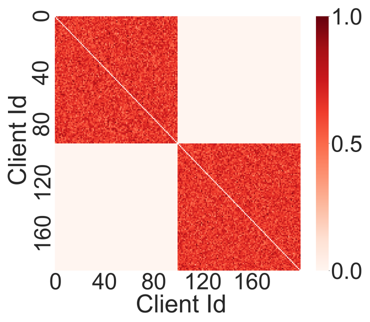
One interesting observation from Table 4.2 is that the clients achieve higher accuracy when using common model initialization. Moreover, the variability in accuracy seems to be lower for the DFL algorithms when using common model initialization. It is also worth pointing out that there seems to be a positive correlation between test accuracy and communication cost.

Table 4.2: Test accuracy and communication cost for all methods and baselines on partitioned data. Each score shows the mean and a 95% confidence interval for four identical experiments but with different random seeds. The methods in gray are upper baselines and have access to more information than the other methods. Both independent and common model initialization are included. Values for ε and λ for the `EpsilonGreedy` methods are indicated as (ε, λ).

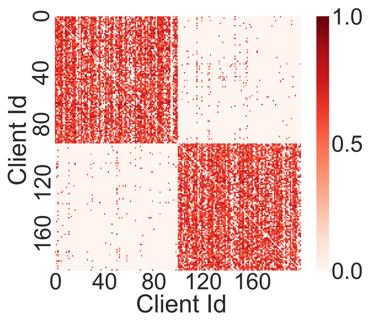
Method	Independent initialization		Common initialization	
	Acc. (%)	Com. cost	Acc. (%)	Com. cost
`CentralIIDHalfData`	65.1 ± 0.9	0	65.1 ± 0.9	0
`CentralNonIID`	62.5 ± 0.3	0	62.5 ± 0.3	0
`Oracle`	43.7 ± 1.8	6560 ± 430	50.0 ± 1.1	5980 ± 320
`PENS`	$\mathbf{42.9 \pm 1.0}$	16130 ± 510	$\mathbf{47.7 \pm 2.0}$	15310 ± 190
`EpsilonGreedy`$(0.5, 1)$	40.8 ± 2.9	7190 ± 430	42.1 ± 1.8	5490 ± 500
`EpsilonGreedy`$(1, 0.99)$	39.5 ± 2.5	6950 ± 550	42.8 ± 0.9	5540 ± 530
`EpsilonGreedy`$(0.2, 1)$	39.2 ± 1.9	6820 ± 690	41.1 ± 2.2	5160 ± 450
`Greedy`	38.2 ± 1.7	6630 ± 720	40.8 ± 0.7	4940 ± 200
`RandomWeighted`	38.2 ± 4.5	6630 ± 960	45.4 ± 1.1	5910 ± 80
`Random`	37.4 ± 1.8	6410 ± 720	44.1 ± 1.6	5650 ± 240
`Local`	24.5 ± 0.1	0	24.7 ± 0.7	0

Figure 4.2 shows the communication pattern for a selection of the methods in Table 4.2 when using independent model initialization. The color indicates how often a

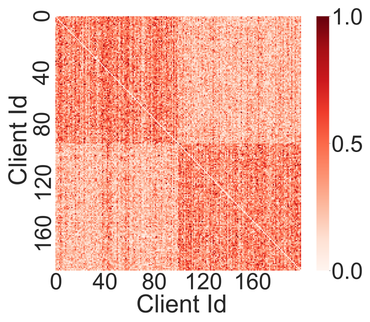
client has communicated with another client where a dark color means that the two clients have communicated frequently. The clients are sorted so that all clients in cluster 1 have client ID 0-99 and the remaining clients, belonging to cluster 2, have client ID 100-199. This can be seen in Figure 4.2a, where it is clear that each client only exchanges information with clients belonging to the same cluster. The opposite is shown in Figure 4.2f where all clients communicate with each other, even clients belonging to different clusters. It is worth pointing out that `PENS` has a similar communication pattern to `Oracle`, indicating that this algorithm effectively selects the correct clients to communicate with. Note, however, that most of the clients in Figure 4.2b communicate with some clients belonging to the wrong cluster. `EpsilonGreedy` seems to struggle more with the client selection, as can be seen in Figure 4.2d and Figure 4.2c. Not surprisingly, a higher value of the exploration factor ε yields a communication pattern more similar to `Random`. Another interesting feature, which is especially visible in Figure 4.2e, is the vertical stripes which indicate that some clients are selected more frequently than others. Note also that there seems to be a correlation between Figure 4.2 and Table 4.2 in that the methods in which the clients successfully find other clients in the same cluster also achieve higher accuracy.



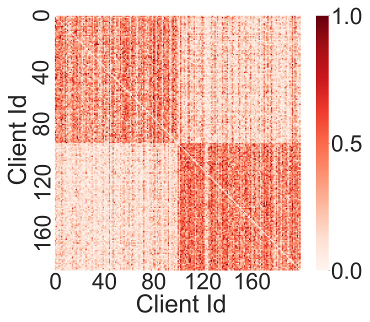
(a) `Oracle`



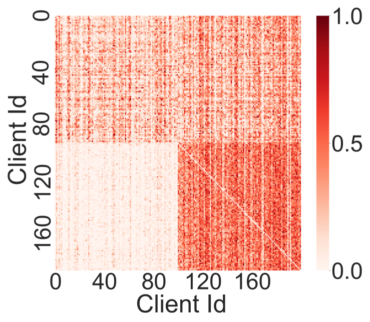
(b) `PENS`, step 2



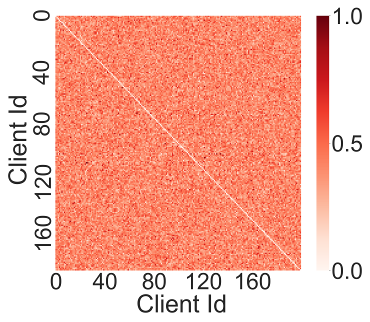
(c) `EpsilonGreedy`$(0.5, 1)$



(d) `EpsilonGreedy`$(0.2, 1)$



(e) `Greedy`

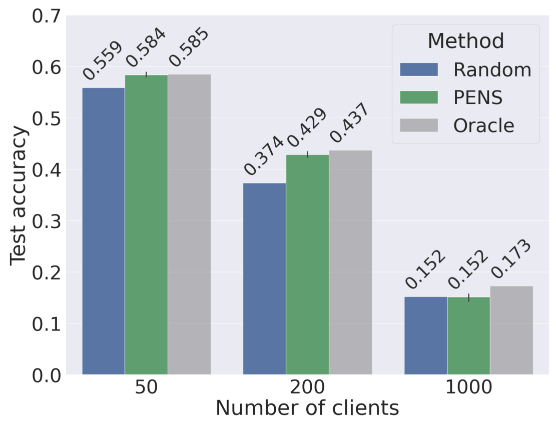


(f) `Random`

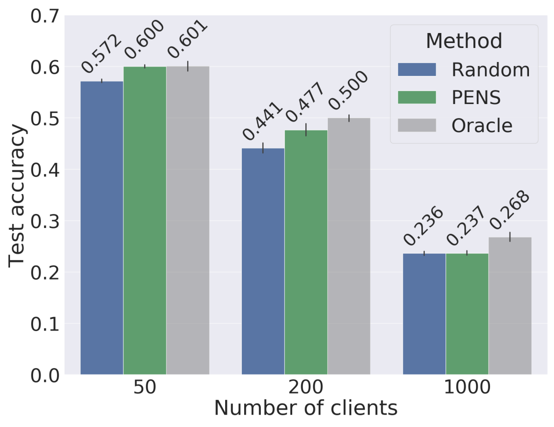
Figure 4.2: Heatmaps showing the communication pattern between clients where a value close to 1 at pixel (i, j) means that client C_j has collected the model from client C_i frequently. The clients are sorted so that clients with ID 0-99 belong to the first cluster and clients with ID 100-199 belong to the second cluster.

4.3 PENS for Different Number of Clients

In Figure 4.3, the best performing algorithm, `PENS`, is evaluated for different number of clients. The experiments are done with both independent and common initialization. By comparing Figure 4.3a and Figure 4.3a, it is clear that common model initialization yields better performance. What is more interesting is that this seems to be more important when the size of the local datasets is small, which is the case when using a large number of clients. Moreover, `PENS` performs significantly better than `Random` if there are 50 or 200 clients. In particular, note that the accuracy for `PENS` and `Oracle` are nearly identical in the case of 50 clients. However, for 1000 clients, the method does not beat `Random`.



(a) Independent model initialization



(b) Common model initialization

Figure 4.3: Test accuracies for `Random`, `PENS` and `Oracle` in the case of 50, 200 and 1000 clients. The vertical lines illustrate 95% confidence intervals over the four repetitions.

Table 4.3 shows two important statistics related to the selection of neighbours for step 2 in `PENS`. One of them is the precision for neighbour selection, which we define as the fraction of clients that a client selects as neighbours for step 2 that belong to the correct cluster. The other one is the recall for neighbour selection, that is, the number of correctly selected neighbours divided by the total number of correct clients. A precision of 100% would mean that the client only has selected clients from its own cluster and a recall of 100% would mean that all clients in the same cluster are selected, and possibly some clients from the other cluster. We report the average of these statistics over all clients. In general, the vast majority of selected neighbours belong to the correct cluster. However, the ability to select neighbours correctly seems to deteriorate as the number of clients grows.

Table 4.3: Precision and recall for the neighbour selection in step 1, averaged over all clients. 95% confidence intervals over four identical repetitions are reported.

Clients	Independent initialization		Common initialization	
	Precision (%)	Recall (%)	Precision (%)	Recall (%)
50	100.0	80.8	100.0	67.6
200	97.8	71.2	99.7	67.4
1000	82.2	80.6	82.2	80.2

5 Discussion

The results indicate that it is possible to achieve higher accuracies on non-IID data with DFL when communication is not random. Out of the algorithms proposed, PENS performs best. Its accuracy is close to the accuracy of Oracle which indicates that the neighbour selection method is effective. This can also be seen in that the precision and recall for neighbour selection is high for 50 and 200 clients, see Table 4.3. Moreover, this is illustrated in Figure 4.2b, where there are two distinct squares indicating communication with clients in the correct cluster. These squares are also visible in the heatmaps for Greedy and EpsilonGreedy, but are not as clear as for PENS. This indicates that the clients communicate with clients from the other cluster more often than in PENS. Communication with clients from the wrong cluster is likely detrimental as the model of a client from another cluster probably is more distant in weight space than one from the same cluster. The reason for PENS's success in selecting correct clients could be that the number of local epochs is 10, compared to 3 local epochs in the Greedy-based methods. More local training likely makes the models within a client group become more similar, which makes correct client selection more likely. Although many local epochs make the models in the same group more easily distinguishable, it makes all models diverge more as they overfit to their local data. This means that the averaging will make the models worse than if fewer local epochs were used, resulting in low accuracy, which is illustrated in Figure 4.1b This is acceptable in step 1 of PENS, as we only use the communication history, but for the Greedy-based methods this would be detrimental. Another possible advantage of PENS is that during training in step 2, the number of clients to communicate with is 20 instead of 4 in the Greedy-based methods. This is likely advantageous since many selected neighbours correlate positively with high accuracy in the case of IID data, see Figure 4.1a.

A drawback with all algorithms is that their communication costs are higher than that of Random. The Greedy-based methods are not too costly, increasing the communication by up to 12 %, but do not offer much of an improvement in performance either. It is important to note that Random has access to the same communication budget as all other methods, except PENS. The reason that the Greedy-based methods have a slightly higher communication cost is because early stopping occurs later than in Random. PENS, on the other hand, has a communication cost roughly 2.5 times as large as Random but in return yields a significant improvement in performance. How to reduce communication costs for PENS would be an interesting topic for future research, as the performance is already quite good. With relatively small models of 1.8MB as in this work, the average client would have to communicate roughly 27GB worth of models. This is the communication cost for both step 1 and step 2 and in a practical application, it is possible that step 2, that is, the training, is repeated regularly whereas step 1 is done less frequently. As step 2 in `PENS` consists of `Random`, this would mean that the communication cost could be similar to `Random`. It is however not clear whether step 1 has to be performed frequently or not.

`PENS` got higher accuracy than `Random` for 50 and 200 clients. For 1000 clients, it performed similarly to `Random`. We see in Table 4.3 that the precision for neighbour selection is much lower than for 200 and 50 which plays a large part in this. This is likely due to the small number of data points per client. In the 1000 clients case, each client only had 50 data points of which 25 were used for training and 25 for validation and early stopping. With 25 data points, a client could have a very skewed data distribution in label space that could be similar to the label distribution of some clients in the other client group. Then these clients could have similar models, despite the difference in feature space due to the rotation, leading to incorrect communication. If this is correct, a large number of clients in itself is not a problem but rather a small number of local data points.

Initializing the clients' models with different or common random seeds turned out to have a significant effect on performance for all algorithms and baselines. With independent initialization and given the highly parameterized nature of the models, it is not obvious that DFL would even work as there is a chance that all models would converge towards different local optima. However, the results in the previous chapter show that at least some models indeed do converge to the same local optimum despite different starting points. The reason why independent model initialization results in lower accuracies is however not clear, although it would be reasonable if the models end up in more clusters in weight space than in the case of common initialization.

The presented algorithms were mainly evaluated using 200 clients and with data from one or two distributions. To further evaluate the robustness of our methods it would be interesting to test both `Random` and `PENS` in different environments. For instance, one could evaluate the methods for a different number of clients or using a different number of data partitions. In a real-world scenario, there will probably exist unbalancedness, both in the number of data samples per client and in the size of the client clusters. Moreover, the clusters might not be as well defined as in this constructed environment. By varying these parameters it would be possible to see how robust `Random` and `PENS` are to changes in the environment, and how well the methods will perform in a real-world scenario.

Moreover, it would be of great interest to test the implemented methods in other domains such as natural language processing. Some of the fundamental ideas of the algorithms are also generic and could therefore be applied to other methods in decentralized federated learning. For instance, an initial filtering of neighbours based on data similarity could also be applied before gossip learning. By doing so, one could compare the performance of `Random` with gossip learning, both in the case of homogeneous and heterogeneous data.

6

Conclusion

Our experiments indicate that having clients communicate with other clients with similar data distributions yields better results than if clients communicate with each other randomly. Out of the five proposed algorithms, `PENS` performed best but it also had the highest communication cost. `PENS` seems to perform well for different number of participating clients as long as the local datasets are sufficiently large. It also works both when the models are initialized with unique and common random seeds. Since `PENS` requires a lot of communication to achieve good results, it would be an interesting topic for future research to investigate how to reduce the communication cost. `PENS` was only tested on an image classification problem and with one type of data heterogeneity. It would be interesting to investigate if `PENS` performs well on other problems and when data distributions differ in other ways across clients.

Bibliography

[1] B. Wolford, "What is GDPR, the EU's new data protection law?." `https://gdpr.eu/what-is-gdpr/`, n.d. Accessed: 2020-05-06.

[2] H. B. McMahan, E. Moore, D. Ramage, and B. A. y Arcas, "Federated Learning of Deep Networks using Model Averaging," *CoRR*, vol. abs/1602.05629, 2016.

[3] A. Agrawal, D. D. Kulkarni, and S. B. Nair, "On Decentralizing Federated Learning," in *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 1590–1595, 2020.

[4] C. Li, G. Li, and P. K. Varshney, "Decentralized Federated Learning via Mutual Knowledge Transfer," *CoRR*, vol. abs/2012.13063, 2020.

[5] A. G. Roy, S. Siddiqui, S. Pölsterl, N. Navab, and C. Wachinger, "BrainTorrent: A Peer-to-Peer Environment for Decentralized Federated Learning," *CoRR*, vol. abs/1905.06731, 2019.

[6] A. Lalitha, O. C. Kilinc, T. Javidi, and F. Koushanfar, "Peer-to-peer Federated Learning on Graphs," *CoRR*, vol. abs/1901.11173, 2019.

[7] G. W. Lindsay, "Convolutional Neural Networks as a Model of the Visual System: Past, Present, and Future," *Journal of Cognitive Neuroscience*, p. 1–15, Feb 2020.

[8] J. Koushik, "Understanding Convolutional Neural Networks," *CoRR*, vol. abs/1605.09081, 2016.

[9] B. Mehlig, "Artificial Neural Networks," *CoRR*, vol. abs/1901.05639, 2019.

[10] S. Savazzi, M. Nicoli, and V. Rampa, "Federated Learning With Cooperating Devices: A Consensus Approach for Massive IoT Networks," *IEEE Internet of Things Journal*, vol. 7, p. 4641–4654, May 2020.

[11] I. Hegedűs, G. Danner, and M. Jelasity, "Decentralized learning works: An empirical comparison of gossip learning and federated learning," *Journal of Parallel and Distributed Computing*, vol. 148, pp. 109–124, 2021.

[12] L. Giaretta and S. Girdzijauskas, “Gossip Learning: Off the Beaten Path,” in *2019 IEEE International Conference on Big Data (Big Data)*, pp. 1117–1124, 2019.

[13] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, R. G. L. D’Oliveira, S. E. Rouayheb, D. Evans, J. Gardner, Z. Garrett, A. Gascón, B. Ghazi, P. B. Gibbons, M. Gruteser, Z. Harchaoui, C. He, L. He, Z. Huo, B. Hutchinson, J. Hsu, M. Jaggi, T. Javidi, G. Joshi, M. Khodak, J. Konečný, A. Korolova, F. Koushanfar, S. Koyejo, T. Lepoint, Y. Liu, P. Mittal, M. Mohri, R. Nock, A. Özgür, R. Pagh, M. Raykova, H. Qi, D. Ramage, R. Raskar, D. Song, W. Song, S. U. Stich, Z. Sun, A. T. Suresh, F. Tramèr, P. Vepakomma, J. Wang, L. Xiong, Z. Xu, Q. Yang, F. X. Yu, H. Yu, and S. Zhao, “Advances and Open Problems in Federated Learning,” *CoRR*, vol. abs/1912.04977, 2019.

[14] K. Hsieh, A. Phanishayee, O. Mutlu, and P. B. Gibbons, “The Non-IID Data Quagmire of Decentralized Machine Learning,” *CoRR*, vol. abs/1910.00189, 2019.

[15] M. Oquab, L. Bottou, I. Laptev, and J. Sivic, “Learning and Transferring Mid-level Image Representations Using Convolutional Neural Networks,” in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1717–1724, 2014.

[16] C. Finn, P. Abbeel, and S. Levine, “Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks,” *CoRR*, vol. abs/1703.03400, 2017.

[17] K. Wang, R. Mathews, C. Kiddon, H. Eichner, F. Beaufays, and D. Ramage, “Federated Evaluation of On-device Personalization,” *CoRR*, vol. abs/1910.10252, 2019.

[18] A. Ghosh, J. Chung, D. Yin, and K. Ramchandran, “An Efficient Framework for Clustered Federated Learning,” *ArXiv*, vol. abs/2006.04088, 2020.

[19] E. L. Zec, O. Mogren, J. Martinsson, L. R. Sütfeld, and D. Gillblad, “Federated learning using a mixture of experts,” *CoRR*, vol. abs/2010.02056, 2020.

[20] J. C. Duchi, A. Agarwal, and M. J. Wainwright, “Dual Averaging for Distributed Optimization: Convergence Analysis and Network Scaling,” *IEEE Transactions on Automatic Control*, vol. 57, p. 592–606, Mar 2012.

[21] I. Colin, A. Bellet, J. Salmon, and S. Clémençon, “Gossip Dual Averaging for Decentralized Optimization of Pairwise Functions,” in *Proceedings of The 33rd International Conference on Machine Learning* (M. F. Balcan and K. Q. Weinberger, eds.), vol. 48 of *Proceedings of Machine Learning Research*, (New York, New York, USA), pp. 1388–1396, PMLR, 20–22 Jun 2016.

[22] X. Lian, W. Zhang, C. Zhang, and J. Liu, “Asynchronous Decentralized Parallel

Stochastic Gradient Descent," *arXiv*, vol. abs/1710.06952, 2018.

[23] H. Tang, X. Lian, M. Yan, C. Zhang, and J. Liu, "D^2: Decentralized Training over Decentralized Data," *CoRR*, vol. abs/1803.07068, 2018.

[24] I. Almeida and J. Xavier, "Djam: Distributed Jacobi Asynchronous Method for Learning Personal Models," *IEEE Signal Processing Letters*, vol. 25, p. 1389–1392, Sep 2018.

[25] P. Vanhaesebrouck, A. Bellet, and M. Tommasi, "Decentralized Collaborative Learning of Personalized Models over Networks," *CoRR*, vol. abs/1610.05202, 2016.

[26] A. Bellet, R. Guerraoui, M. Taziki, and M. Tommasi, "Personalized and Private Peer-to-Peer Machine Learning," in *AISTATS 2018 - 21st International Conference on Artificial Intelligence and Statistics*, (Lanzarote, Spain), pp. 1–20, Apr. 2018.

[27] V. Zantedeschi, A. Bellet, and M. Tommasi, "Communication-Efficient and Decentralized Multi-Task Boosting while Learning the Collaboration Graph," *CoRR*, vol. abs/1901.08460, 2019.

[28] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Tech. Rep. 0, University of Toronto, Toronto, Ontario, 2009.

[29] P. Foret, A. Kleiner, H. Mobahi, and B. Neyshabur, "Sharpness-Aware Minimization for Efficiently Improving Generalization," *CoRR*, vol. abs/2010.01412, 2020.

[30] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.