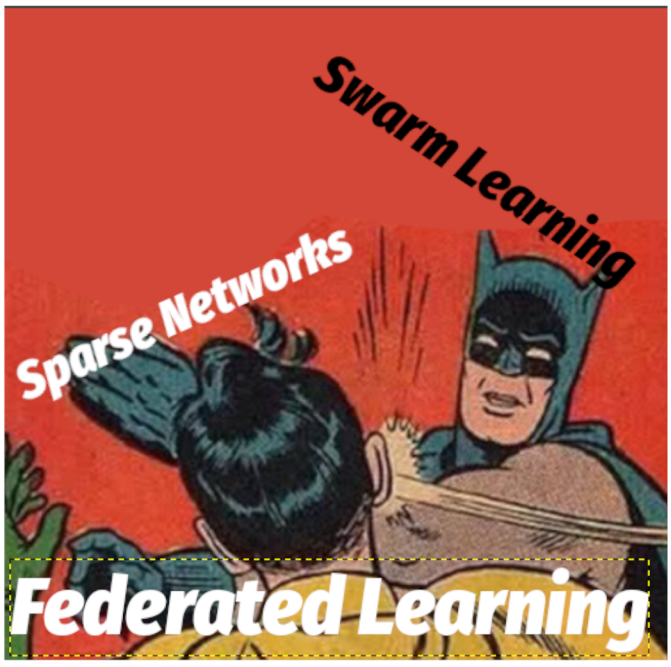
SwarmAvg: A Novel Approach to Fully Distributed Machine Learning

Josh Pattman

April 25, 2023



Abstract

Federated learning is a technique that allows a machine learning model to be trained on data distributed across multiple data islands. This approach protects privacy by keeping the data decentralized, meaning that sensitive data does not need to ever be shared. Swarm learning is a similar technique that eliminates the need for a central server, ensuring that not only the data, but also the communication, is completely decentralized. Current Swarm Learning algorithms rely on blockchain to distribute the shared global model, however this may be a poor choice for certain scenarios closely associated with swarms. In this paper, a novel swarm learning technique called SwarmAvg is presented which operates without a blockchain. The algorithm is validated against federated learning in various scenarios, and also in some situations where federated learning cannot be applied. The benefits and drawbacks of operating Swarm Learning without a blockchain are also discussed, presenting some interesting reasons why one might choose to use SwarmAvg over other distributed machine learning techniques.

Statement of Originality

- I have read and understood the ECS Academic Integrity information and the University's Academic Integrity Guidance for Students.
- I am aware that failure to act in accordance with the Regulations Governing Academic Integrity may lead to the imposition of penalties which, for the most serious cases, may include termination of programme.
- I consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to verify whether my work contains plagiarised material, and for quality assurance purposes.

You must change the statements in the boxes if you do not agree with them.

We expect you to acknowledge all sources of information (e.g. ideas, algorithms, data) using citations. You must also put quotation marks around any sections of text that you have copied without paraphrasing. If any figures or tables have been taken or modified from another source, you must explain this in the caption and cite the original source.

I have acknowledged all sources, and identified any content taken from elsewhere.

If you have used any code (e.g. open-source code), reference designs, or similar resources that have been produced by anyone else, you must list them in the box below. In the report, you must explain what was used and how it relates to the work you have done.

I have not used any resources produced by anyone else.

You can consult with module teaching staff/demonstrators, but you should not show anyone else your work (this includes uploading your work to publicly-accessible repositories e.g. Github, unless expressly permitted by the module leader), or help them to do theirs. For individual assignments, we expect you to work on your own. For group assignments, we expect that you work only with your allocated group. You must get permission in writing from the module teaching staff before you seek outside assistance, e.g. a proofreading service, and declare it here.

I did all the work myself, or with my allocated group, and have not helped anyone else.

We expect that you have not fabricated, modified or distorted any data, evidence, references, experimental results, or other material used or presented in the report. You must clearly describe your experiments and how the results were obtained, and include all data, source code and/or designs (either in the report, or submitted as a separate file) so that your results could be reproduced.

The material in the report is genuine, and I have included all my data/code/designs.

We expect that you have not previously submitted any part of this work for another assessment. You must get permission in writing from the module teaching staff before re-using any of your previously submitted work for this assessment.

I have not submitted any part of this work for another assessment.

If your work involved research/studies (including surveys) on human participants, their cells or data, or on animals, you must have been granted ethical approval before the work was carried out, and any experiments must have followed these requirements. You must give details of this in the report, and list the ethical approval reference number(s) in the box below.

My work did not involve human participants, their cells or data, or animals.

Contents

1 Background **6**
- 1.1 Federated Learning . . . 7
- 1.2 Blockchain . . . 7
- 1.3 Swarm Learning . . . 8

2 Problem and Goals **10**
- 2.1 Problem . . . 11
  - 2.1.1 The Problem of Privacy . . . 11
  - 2.1.2 The Problem of Performance . . . 11
- 2.2 Goals . . . 11
  - 2.2.1 To Design an Novel Algorithm for Swarm Learning . . . 11
  - 2.2.2 To Implement the Algorithm . . . 11
  - 2.2.3 To Test Performance in Situations Where FedAvg Performs Well . . . 12
  - 2.2.4 To Test Performance in Situations Where FedAvg Performs Badly . . . 12

3 Detailed Design **13**
- 3.1 The Job of a Node . . . 14
- 3.2 A Blockchain-less Algorithm . . . 14
- 3.3 Keeping Track of Training: The Training Counter . . . 14
- 3.4 Combining Neighbouring Models . . . 15
  - 3.4.1 Simple Method: Averaging . . . 15
  - 3.4.2 Advanced Method: Averaging With Synchronisation Rate . . . 15
  - 3.4.3 Ignoring Old Models: Training Counter Filtering . . . 16
- 3.5 Behaviour in Sparse Networks . . . 16
  - 3.5.1 Lightweight Approach: Passive Convergence . . . 16
  - 3.5.2 Complex Approach: Relay . . . 17
- 3.6 The Complete Algorithm . . . 17

4 Implementation **18**
- 4.1 Machine Learning Problem . . . 19
  - 4.1.1 The Dataset . . . 19
  - 4.1.2 The Model . . . 19
- 4.2 Implementing Federated Learning . . . 19
  - 4.2.1 Development of the Federated Learning Algorithm . . . 19
  - 4.2.2 Evaluation of this Federated Learning Implementation . . . 20
- 4.3 Implementing the First Prototype . . . 20
  - 4.3.1 Development of the First Prototype . . . 20
  - 4.3.2 Evaluation of the First Prototype . . . 20
- 4.4 Implementing the Final Algorithm . . . 21

4.4.1 Development of the Final Algorithm . . . 21
4.4.2 Evaluation of the Implementation of the Final Algorithm . . . 21

5 Testing Strategy and Results **22**
5.1 Strategies for Gathering of Results . . . 23
5.1.1 Chosen Metric: Accuracy . . . 23
5.1.2 Number of Simulated Nodes . . . 23
5.1.3 Repeated Testing for Noise Reduction . . . 23
5.1.4 Configuring the Algorithm . . . 23
5.1.5 Data Provided to Each Node . . . 23
5.1.6 Training Steps and Epochs . . . 24
5.2 Scenario 1: Densely Connected Network . . . 24
5.2.1 Dense Network Results . . . 24
5.3 Scenario 2: Sparsely Connected Network . . . 27
5.3.1 Sparse Network Results . . . 28
5.4 Scenario 3: Densely Connected Network with Class Restrictions . . . 30
5.4.1 Dense Network with Class Restrictions Results . . . 31
5.5 Scenario 4: Sparsely Connected Network with Class Restrictions . . . 32
5.5.1 Sparse Network with Class Restrictions Results . . . 33

6 Critical Evaluation **34**
6.1 Comparison of SwarmAvg to Other Methods . . . 35
6.1.1 Comparison of SwarmAvg to FedAvg . . . 35
6.1.2 Comparison of SwarmAvg to SwarmBC . . . 35
6.2 Pitfalls of this Study . . . 36
6.2.1 The Small Simulated Number of Nodes . . . 36
6.2.2 The Lack of Testing Overheads . . . 36
6.2.3 The Lack of Simulated Internet Lag . . . 36

7 Project Management **37**
7.1 Time Management . . . 38
7.2 Changing Plans . . . 38
7.3 Risk Assessment . . . 38
7.3.1 Personal Issues . . . 38
7.3.2 Hardware Failure . . . 39
7.3.3 Algorithm Does Not Work . . . 39

8 Conclusion and Future Work **40**
8.1 Conclusion . . . 41
8.2 Future Work . . . 41

A SwarmAvg Algorithm **42**
A.1 Single Training Step . . . 42
A.2 Model Received Event . . . 43

B Network Generation Algorithm **44**

C Machine Learning Model **45**

D Gantt Charts **46**
D.1 Gantt - Interim 48
D.2 Gantt - Final 50

Chapter 1

Background

Machine learning is an exceedingly vital tool in modern society. Nonetheless, as the issues which machine learning endeavours to resolve become more intricate, the possibility of utilizing a single centralized intelligence diminishes. One possible remedy to this problem is the distribution of data and computation amongst multiple nodes. Transitioning from centralized approaches to decentralized approaches also offers numerous advantages. For example, it has been proven mathematically that the accuracy of a distributed system surpasses that of a centralized one [1]. The following sections will provide an overview of some of the state-of-the-art approaches used to distribute machine learning amongst multiple nodes.

1.1 Federated Learning

Many modern machine learning algorithms require large volumes of diverse data to achieve optimal performance. Real-world data is often distributed among multiple nodes that are unable to share it with each other due to privacy regulations such as GDPR [2], reducing the amount of data available to train models on and negatively impacting trained performance [3]. Federated Learning (FL) [4] is a technique in machine learning that aims to train a single model using all available data across nodes without requiring any data to be shared among them.

There are a multitude of published FL frameworks [5], each with different merits and drawbacks for certain use cases. Federated Averaging (FedAvg) [6] is a commonly used yet simple framework, which splits training into iterations where three steps take place:

1. A copy of the current model is sent to each node from the central server.
2. Each node performs some training with their copy of the model and their own private data.
3. The trained models from each node are sent back to the server to aggregate into the new server model.

The server model is improved over time, beyond what could be achieved by simply training on a single nodes data.

As it does not require data to be shared between nodes, FL is naturally beneficial for privacy sensitive tasks compared to conventional machine learning where the data is aggregated in a central location [7]. Additionally, as FL performs training on multiple nodes in parallel, it can make better use of available training resources in situations where processing power is distributed among multiple nodes.

One branch of FL is distributed federated learning (DFL). This algorithm works in a very similar fashion to FL, but replaces the central server with an elected leader in a network of nodes [8]. DFL has been shown to increase fault tolerance and security over FL.

1.2 Blockchain

Blockchain technology enables a network of nodes to record transactions on a shared ledger in a fully decentralized manner [9]. In a conventional blockchain, any participating

node is permitted to add a transaction to the ledger, but once a transaction is executed, it becomes immutable and irreversible. Although blockchains are primarily associated with cryptocurrency and financial transactions, any type of data, including neural network weights, can be recorded as a transaction.

When a blockchain is scaled up, it may encounter performance issues. For instance, the Bitcoin network typically processes only 7 transactions per second on average [10]. High confirmation delay and performance inefficiency are two primary issues related to this paper when scaling up a blockchain [10]. These issues could prove catastrophic in a system like a swarm of robots, where it is crucial to always have the latest data and optimize limited processing power.

1.3 Swarm Learning

Swarm learning (SL) is a subcategory of FL which operates in a completely distributed and decentralised manner. SL enables the collaboration of nodes to learn a shared global model, however in contrast to FL, a central server is never used. SL also does not use leader election, so all nodes on the network are given the same importance.

In SL, the model on which new training is performed is known as the global model. However, unlike FL where the global model is stored in a central location, the global model in SL does not materially exist, but is instead a concept which is agreed upon by the nodes in the network.

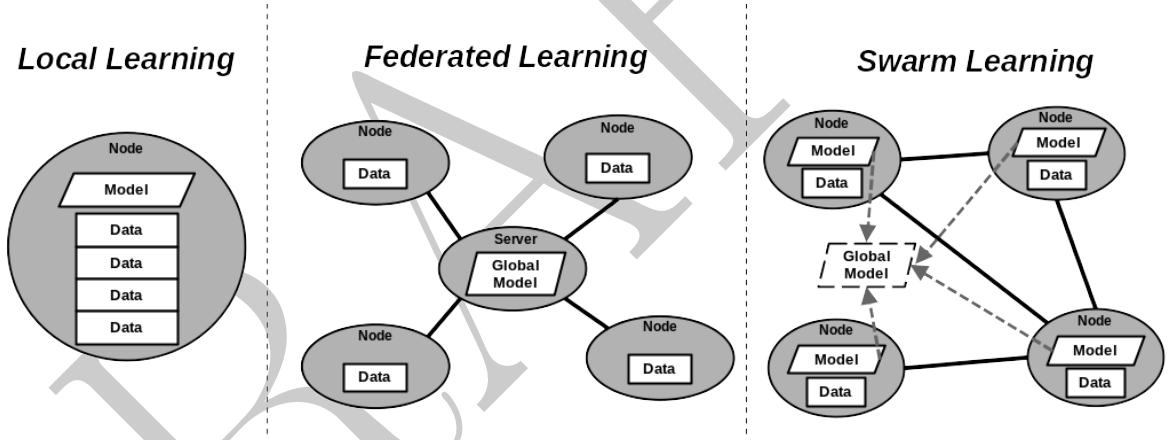


Figure 1.1: Diagram of different learning algorithms. Each *Node* indicates a single training machine, and each line denotes a connection between two machines, along which the model can be shared. In the swarm learning diagram, the dashed lines show that each local model is an approximation of the global model.

One SL algorithm, referred to by this paper as SwarmBC, uses a blockchain to store the global model [11]. In this version of SL, training is performed by repeating the following steps:

1. A node obtains a copy of the global model from the blockchain.
2. The node performs some training with their copy of the model and their own private data.
3. The updated model is merged with the latest blockchain model and sent back to the blockchain for other nodes to use.

SL exhibits many of the same benefits of FL over conventional learning, but it also improves upon FL in some aspects. As is often the case when comparing decentralised algorithms to their centralised counterparts [12], the absence of a central server theoretically makes SL more resilient and robust to failures than centralized FL approaches such as FedAvg. The absence of a central server in SL not only means that the system does not entirely rely on a single node, but also that it is less sensitive to disruptions in connections between nodes. In FL, if a connection between two nodes ceases to exist, the node can no longer participate in learning without some form of tunnelling. In contrast, in SL, even if a connection between two nodes is lost, the two connected nodes are still likely to be connected to other operational nodes, enabling the system to continue functioning normally.

The utilization of FL with leader election effectively addresses the challenge of ensuring system robustness, as it enables any node to serve as a replacement for the server in the event of network disconnection. Nevertheless, it is important to note that a swarm of nodes is not often completely interconnected, and the connections between the nodes are often subject to changes. This dynamic and sparse network topology can present significant complexities when it comes to leader election, leading to an increase in overhead [13]. The lack of a leader election process in SL makes it a more viable option in such situations.

Finally, a swarm of nodes can offer greater scalability compared to its centralized counterpart, particularly if agents are restricted to communicating with only their nearby neighbours [14]. This is primarily because in such cases, there is no need for all communications to travel to a single node or server that may not be able to handle the significant volume of traffic. In a swarm, where nodes are limited to a certain number of close neighbours, the size of the swarm becomes less significant, since a single node would not experience a difference between a small or large swarm, as it would have the same number of neighbours. Consequently, SL is typically more scalable than FL.

Chapter 2

Problem and Goals

2.1 Problem

In machine learning, there exists some issues that arise from trying to use conventional techniques in the real world.

2.1.1 The Problem of Privacy

It is common for data to be spread across multiple locations, referred to as data islands. Traditionally, all of this data would be consolidated into a single centralized server to facilitate the process of machine learning. However, it may not be possible to do so, given the potential conflict with privacy legislation. This leaves two options to the data scientist who is looking to train a model: either a single model per data island, likely with inferior performance, or use an algorithm that would allow the different data islands to collaborate and collectively train a model.

Consider this scenario where many different hospitals wish to train a model to detect an illness in a patient. However, due to the obligation to maintain patient confidentiality, the medical data of patients cannot be shared with any of the other hospitals. This means that, despite likely having superior performance, a model trained on all data across all hospitals is not feasible, as a consequence patients would not have the highest quality medical care possible.

2.1.2 The Problem of Performance

In general, it has been observed that larger machine learning models coupled with more data typically lead to better performance. However, the use of conventional approaches for training these large models necessitates the requirement of a powerful computer. Most entities, however, do not have access to such a computer. Nevertheless, they may have access to multiple, lower-power computers. For instance, during non-working hours, a company may have hundreds of computers in its offices that are not being used, thus providing a pool of unused processing power.

2.2 Goals

2.2.1 To Design an Novel Algorithm for Swarm Learning

In this project, the primary aim is to design a novel SL algorithm. This algorithm should be based on FedAvg and *Swarm Learning*. However, the algorithm should be fully decentralised, and it should operate without a blockchain. The decision to not use a blockchain was made due to the disadvantages stated in Section 1.2.

2.2.2 To Implement the Algorithm

The primary purpose of the algorithm is to test its viability when compared to other techniques, not to be as efficient as possible for real world usage. Te other important feature of this implementation is that it is easy to replicate by a data scientist who wished to use SL in their next product. For this reason, when implementing the algorithm, an easy-to-understand programming language should be used, and the focus should be on readability rather than absolute performance.

2.2.3 To Test Performance in Situations Where FedAvg Performs Well

FedAvg is designed to work in situations where each node has a reliable direct connection to the server. It should be shown that the new algorithm can perform well in similar situations, where each node has many stable connections to other nodes in the network.

2.2.4 To Test Performance in Situations Where FedAvg Performs Badly

FedAvg may not be effective when the server has unreliable connections to the nodes, or when nodes are halted. Additionally, if the server stops working, FedAvg is unable to proceed. The goal is to demonstrate that the new algorithm can achieve successful results in scenarios where nodes frequently drop out of the network, thus making it possible to use the algorithm in cases where FedAvg may not be viable.

Chapter 3

Detailed Design

3.1 The Job of a Node

In SL, a node is an agent responsible for facilitating the improvement of the global model. The global model is an abstract concept representing the consensus of all nodes in the network. In the proposed version of SL, referred to as SwarmAvg, each node maintains its own model, known as the local model, which is an approximation of the global model. However, in SwarmAvg, the consensus algorithm used is repeated averaging, not blockchain. This means that at the start of each training step, every node may start with a slightly different model. As training progresses and performance begins to plateau, each node's local model should not only converge towards a minima, but also towards each other. Over time, each nodes local model better approximates the global model, and the global model becomes a better solution to the problem.

Each node in the network possesses a confidential dataset that is not disclosed to any other nodes. In order to train the global model, nodes fit their own local model of their local dataset. In order to maintain consistency between local and global models, a combination procedure is conducted following each round of local training, which involves the integration of neighbouring nodes models into the local model.

The actions in each training step for SwarmAvg are as follows:

1. Fit local model to local dataset.
2. Send local model to all neighbours.
3. Combine neighbouring models into local model, using one of the methods presented in Section 3.4.

In addition, each node retains a local cache of the most recent models of its neighbouring nodes. This cache is updated each time a neighbouring node transmits its model to the node in question, instead of being updated on-demand during the combination step. The reasoning behind this decision is elaborated upon in greater detail in Section 4.3.1.

3.2 A Blockchain-less Algorithm

SwarmBC employs a blockchain as the mechanism for distributing the global model, whereas SwarmAvg utilizes a variation of averaging with its neighbours and does not use a blockchain. This decision was made due to the long transaction confirmation time of large blockchains, which could adversely affect training as nodes continuously upload their latest networks to the network. If this process were to take an excessive amount of time, each node would be training on an outdated model.

Furthermore, the inefficiency of blockchains with respect to performance is another reason for not using them in SwarmAvg. SwarmAvg is designed for deployment in large swarms, with each agent possibly having low processing power. If a blockchain were utilized, some of the processing power that could be devoted to training would instead be utilized for validating transactions. In scenarios where processing power is limited, it would be practical to avoid the use of blockchains.

3.3 Keeping Track of Training: The Training Counter

A vital aspect of SwarmAvg, specifically the combination step, involves evaluating the performance of a local model. The conventional approach would involve testing each

model using an independent test set. However, due to the inability to exchange test sets among nodes, this approach is not feasible as it would result in non-comparable scores for each model. In order to circumvent this problem, this paper presents a heuristic metric referred to as the "training counter," which serves as an approximation of the level of training for a network by estimating the number of training steps performed on a given model.

The training counter can be changed in one of two manners. Firstly, the counter is incremented by 1 when the local model is trained on the local dataset, indicating that an additional step of training has been performed. Following the combination step, the training counter is also updated to reflect the combination method that was utilized. For instance, if the neighbouring models were averaged, the training counter would be updated to represent the average of all neighbouring nodes' training counters.

3.4 Combining Neighbouring Models

The combination step is a crucial component of SwarmAvg. During this step, a node merges its local model with those of its neighbours, producing an updated estimate of the global model. This paper presents multiple methods for performing the combination step.

In the below equations, $\mu(x)$ denotes the function $mean(x)$. All models are assumed to be 1-dimensional arrays, meaning that mathematical operations such as add $(+)$ and multiply $(*)$ can be performed in an element-wise fashion. To achieve this constraint in the context of neural networks, a simple flattening operation is sued on the weight matrices.

3.4.1 Simple Method: Averaging

The most rudimentary approach to combination is to compute the average model between the local model and the models of all neighbouring nodes.

$$localModel \leftarrow \mu(localModel \cup neighborModels)$$

This technique is utilized in FedAvg, which is the most simplistic form of FL. The benefit of this method is that it necessitates no hyper-parameters, which means that the data scientist needs to do less tuning. However, this attribute can also be viewed as a drawback, as it affords less flexibility in terms of customization for particular tasks.

3.4.2 Advanced Method: Averaging With Synchronisation Rate

A more complex approach to combination is to compute the average model of all neighbours, then compute the weighted average between that model and the local model.

$$localModel \leftarrow (1 - \alpha) * localModel + \alpha * \mu(neighborModels)$$

The synchronisation rate, denoted as α, indicates the degree to which each node adjusts its local model to align with the global model. If α is set too low, each node's model in the network will diverge, resulting in each node becoming trapped at a local minima. On the other hand, if α is too high, the progress achieved by a given node will be discarded at each averaging step, which can result in slower learning.

This method is beneficial over vanilla averaging as it provides some resistance to variation in the number of neighbours of a node. When using averaging, the amount of change made by the local node that persists into the next training step is proportionate to the number of neighbours which were combined. However, when using this method, the amount of change made that persists will be constant, no matter how many neighbours that are present. This may increase the stability of training, especially if the situation which SwarmAvg is deployed involves a dynamic set of neighbours.

3.4.3 Ignoring Old Models: Training Counter Filtering

A potential modification to the previously mentioned combination algorithms involves filtering based on the training counter. Specifically, a node may only include its neighbour models if they meet the following statement:

$$neighborTrainingCounter + \beta \geq localTrainingCounter$$

The training offset β is the amount the training counter of a neighbour can be behind the local training counter before it is ignored.

A compelling reason to allow training counter filtering is the increased fault tolerance. Consider the situation where node A has received many model updates from many nodes, one of which being node B. However, node B goes offline and no longer is sending model updates. Without training counter filtering, node A will continue to combine the outdated node B model with it's own for as long as it is training. However, if training counter filtering is enabled, after a number of training steps the outdated model B updates will be ignored.

An issue with training counter filtering pertains to the presence of runaway nodes. These nodes possess a substantially higher training counter compared to all other nodes in the network, meaning that when filtering is applied, they are left with no neighbours to utilise in the combination step. Consequently, a runaway node may start to overfit on its own training data, as this is the only data it is exposed to, thereby leading to decreased local performance, as well as potential performance reductions in the rest of the network. To address this problem in SwarmAvg, each node must wait until it has obtained at least γ viable neighbours prior to performing the combination step. Although this measure prevents individual nodes from becoming runaway nodes, groups of size γ still have the potential to become runaway as a unit. Nevertheless, if γ is roughly equivalent to the number of neighbours and all neighbours train at a comparable rate, the issue is minimised.

3.5 Behaviour in Sparse Networks

Given the sparsely connected nature of distributed scenarios, it is often the case that nodes only have direct connections to a small subset of their neighbours. This is a situation where FL struggles, however the author hypothesises that SwarmAvg should be able to deal with this situation.

3.5.1 Lightweight Approach: Passive Convergence

An approach to deal with a sparsely connected network is to use the swarm learning algorithm without any modifications. This approach is effective due to the use of averaging

as a combination method. When a node tries to update the global model, its changes will propagate through the network slowly, over many training iterations, even to nodes that are not directly connected. This approach has the advantage of requiring no extra data transmission, resulting in significantly less data traffic compared to other methods.

However, this method also has certain theoretical drawbacks. Consider a scenario where the network is comprised of several sparsely connected groups of nodes, where each node in a group is densely connected to other nodes within that group. In this case, it is possible that each group may learn a distinct solution to the problem. This is inefficient because instead of functioning as a cohesive network, there are multiple smaller networks acting somewhat independently of each other, potentially leading to a decrease in overall performance.

3.5.2 Complex Approach: Relay

A solution to this could be to relay any received model updates, which means that as long as each node has at least one path to reach all other nodes, the network will behave as a dense network. This approach offers theoretical immunity to changes in network topology, but in practice, the network's performance may still decrease compared to a truly dense network due to slower communication times between non-connected nodes.

The main disadvantage of this approach is the drastic increase in network traffic, which in turn will lead to longer model transfer times. If the swarm learning algorithm is applied to a low power network, such as an IoT network, the increase in network traffic may not be feasible at all. This relay approach can also be applied to federated learning with the same advantages and disadvantages. For these reasons, Relay will not be discussed further.

3.6 The Complete Algorithm

The complete pseudocode algorithm for SwarmAvg can be found at Appendix A. It has multiple parts which are described below.

The update loop, which can be found at Appendix A.1, is the section of the algorithm which runs continuously during the time which a node is running. It takes care of training and synchronising the local model. The provided code represents what happens in a single update step, meaning that it should be run in a loop that terminates once a stop condition, such as target accuracy, has been reached.

The model received event, which can be found at Appendix A.2, is run on the local node every time a remote node sends the local node a model update. This event takes care of updating the local model cache, to ensure the local node has the most up-to-date information. The model update should contain the model of the remote node and also the remote nodes training counter.

Chapter 4

Implementation

4.1 Machine Learning Problem

In order to evaluate SwarmAvg, it was necessary to create both a problem and a model to supply the algorithm with updates. Nevertheless, it should be noted that the algorithm is intended to function with any model and dataset. Presented below are the selected model and dataset.

4.1.1 The Dataset

Initially, the dataset utilized for experimentation was the MNIST dataset, which encompasses 60000 greyscale 28x28 labelled images of digits from 0 to 9. This dataset was selected for its simplicity, requiring no pre-processing or data cleaning prior to training, and due to its availability as a built-in component of the chosen machine learning framework, Keras.

However, upon implementation it was determined that this dataset was too simple for the application of machine learning, as a single node could reach near peak accuracy after a single epoch, rendering it ill-suited for swarm learning, an algorithm designed to function across multiple training epochs.

To address this issue, the MNIST dataset was replaced with MNIST-fashion (henceforth referred to as MNIST-F), a drop-in replacement dataset containing 10 classes of different items of clothing. MNIST-F is known to be more challenging [15]. To further increase the complexity of the problem, in several experiments each agent was only provided with a small subset of the entire dataset, resulting in less training per epoch, and therefore meaning that an agent would require more epochs to achieve the same performance.

4.1.2 The Model

The Keras machine learning framework in Python was utilized to implement the model due to its reputation for being both simple and straightforward. All experiments made use of the same model, which is outlined in Appendix C and is a small convolutional neural network. The model was tested on the MNIST-F dataset and was able to attain an accuracy score of above 90 percent when trained on the entire dataset using local learning; this result is on par with the accuracy reported in the original paper [15], making the model suitable for use.

4.2 Implementing Federated Learning

FedAvg was chosen for comparison of performance against SwarmAvg. In order to ensure fairness of the comparison, it was necessary to implement FedAvg from scratch using the same language framework as SwarmAvg.

4.2.1 Development of the Federated Learning Algorithm

The implemented algorithm worked in the same manner as the algorithm described in the FedAvg paper. However, one modification was made: at the start of each timestep instead of choosing N random nodes to perform training, all nodes were chosen. This means that every available node will perform training at every timestep, which should result in the

best possible performance for federated learning, especially give that only 10 nodes could be run at once.

Initially, a REST API was utilised to transfer the model between the server and the client. However, this was deemed unnecessary and was supplanted for two reasons. Firstly, the decision was made to measure performance against epochs trained instead of performance against time, which is discussed in further detail in the results section. Secondly, the REST API approach was much slower than the method chosen to replace it, yet it resulted in the same performance measurements when gauged in terms of epochs trained. As a substitute for the REST API, a system of functions was implemented. When a node sent a model to another node, it simply called a function on that node. This was abstracted away from the main algorithm code, thus meaning that a drop-in REST replacement could b added at a later date. This significantly accelerating training, and enabled a greater number of training runs to be conducted, resulting in more data being collected.

4.2.2 Evaluation of this Federated Learning Implementation

This implementation of federated learning is simple yet effective. Though certain simplifications have been made, they should not interfere with the performance of the model in this scenario. It should only be used for testing performance against epochs trained, not against time taken, as the model transfer layer has been simplified.

4.3 Implementing the First Prototype

It was decided to make an initial, less streamlined, prototype as a proof-of-concept before spending a lot of time creating the final algorithm. This prototype was created to be very modifiable so that changes could easily be made and tested.

4.3.1 Development of the First Prototype

This algorithm was a simplified version of that described in the design section. The primary difference was that when a node performed its synchronisation step, it would request the models from each neighbour instead of utilising its cached versions of their models. This had the consequence of slowing down training, as the synchronisation step could not be completed until all nodes had responded. Furthermore, this algorithm did not incorporate the training counter, leading to the absence of training counter filtering, β and γ.

4.3.2 Evaluation of the First Prototype

This step was beneficial for the progression of the project, as it enabled the author to form the ideas detailed in the design process, which were then implemented in the subsequent step. However, due to the abundance of superfluous code, the algorithm was inefficient and performed poorly. For this reason, it was decided not to record the results of this method.

4.4 Implementing the Final Algorithm

In this implementation, the findings of the prototype were taken and built upon. The code was streamlined for the purpose of testing performance. However, the code was still designed to be reusable and easy to read.

4.4.1 Development of the Final Algorithm

The algorithm is as described in the design section. However, there was one discrepancy which needed to be tested: whether to send model updates to neighbours before or after performing the combination step. Both were tested but pushing model updates before combination seemed to be the more effective method when comparing the accuracy. The author hypothesises that this is due to more of the local nodes training progress being preserved, meaning that a local nodes training has more of an affect on its neighbours.

The back end for distributing models was implemented as an interface which abstracts the details of the distribution away from the main algorithm code. For this implementation, the same distribution strategy as the previously implemented FedAvg was used: calling local functions that simulated a web connection.

4.4.2 Evaluation of the Implementation of the Final Algorithm

Overall, this implementation of the proposed swarm learning method is satisfactory. It is efficient, reusable and simple to understand and use. Nevertheless, it is not yet suitable for real world applications, as it lacks any security features and there is limited error handling, being designed for use in a controlled testing environment. However, it would not be challenging to incorporate a backend for this code, allowing for communication over the internet.

Chapter 5

Testing Strategy and Results

5.1 Strategies for Gathering of Results

5.1.1 Chosen Metric: Accuracy

The metric chosen for the following experiments was accuracy, due to its comprehensible nature. Despite the fact that an unbalanced dataset presents one of the significant drawbacks of using accuracy, this concern is irrelevant in the case of MNIST-F since it is a balanced dataset. The accuracy of each node is computed after each training step using the test subset of MNIST-F, and none of the nodes are ever provided access to the test set for training.

5.1.2 Number of Simulated Nodes

The experiments were conducted using 10 nodes, with the exception of the server in cases where FedAvg was employed. The decision of how many nodes to simulate was based on the highest node count attainable without causing inconsistencies and crashes due to resource depletion of the training machine.

5.1.3 Repeated Testing for Noise Reduction

The training process for each experiment was conducted five times, and the resulting accuracies of every node were recorded. To mitigate the impact of training noise on the performance graphs, the accuracy value for each time step was calculated as the median accuracy across all nodes and runs at that time step. As there were 10 nodes, this resulted in the graphs representing median of 50 nodes.

5.1.4 Configuring the Algorithm

In each of the following experiments, the algorithm was configured using a specific set of parameters $\alpha\beta\gamma$. These parameters were obtained heuristically by making an initial guess, testing, and then fine-tuning them until a satisfactory outcome was reached. However, it is important to note that an exhaustive investigation into the optimal parameter configuration for a particular type of problem is not within this papers scope, meaning that it is plausible that swarm learning could yield better results with more precisely tuned parameters.

5.1.5 Data Provided to Each Node

The experiments evaluate the algorithm's performance using three levels of data volume per node. These levels are considerably smaller than the full MNIST-F dataset not only to increase problem difficulty, but also as the algorithm is intended for scenarios where each nodes access to data is restricted. To create a subsection of data for each node, a random sampling with replacement method was used to select the desired number of datapoints. During the initial training phase, each node performs a single sampling of its dataset, after which that nodes data subset remains constant. The three levels of data volume can be seen in Table 5.1.

5.1.6 Training Steps and Epochs

Due to the limited size of the dataset, a single node executes more than one epoch of training in each training step. The number of epochs carried out by a node per training step will be referred to as Epochs per Step (EPS). Empirical testing has indicated that both SL and FL exhibit improved performance with higher EPS, at times surpassing the gains from increasing the number of training steps. Moreover, the utilization of higher EPS was favoured due to its reduced training time, compared to increasing the number of training steps. The three levels of data volume with their respective EPS are shown in table 5.1

Dataset Size	EPS
1000	5
100	10
25	20

Table 5.1: The different levels of dataset size and EPS that were tested

5.2 Scenario 1: Densely Connected Network

A crucial experiment for evaluating the performance of the SL algorithm involves assessing its performance under optimal circumstances, specifically within a network of nodes wherein each node is directly connected to every other node. In FL, the analogous topology involves direct connections between each node and the server. This comparison is significant as it facilitates a direct evaluation of the SL and FL algorithms under their respective ideal conditions.

5.2.1 Dense Network Results

The following are the results obtained from the execution of the training script. Each graph displays the data for SwarmAvg in red, and FedAvg in black.

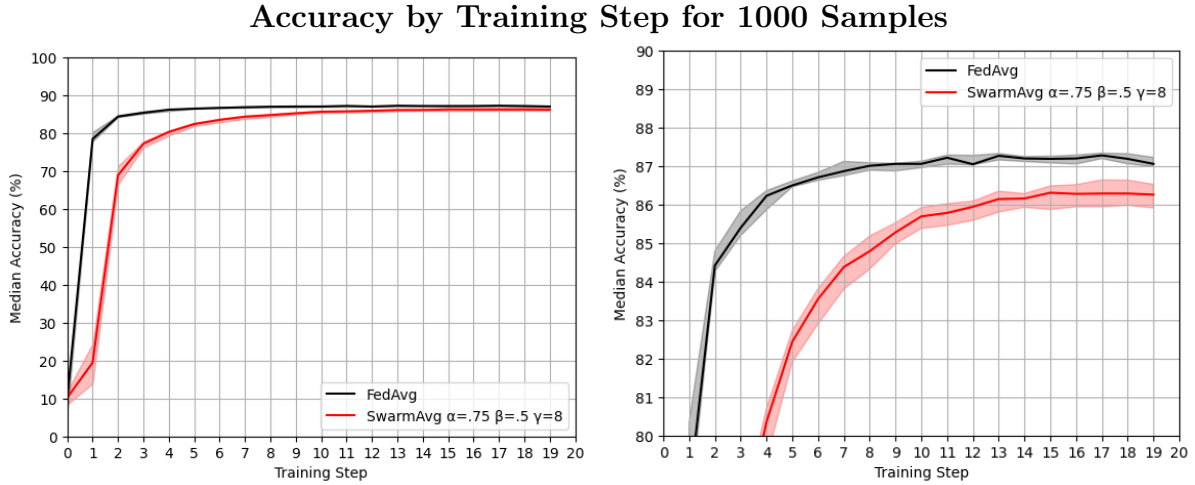


Figure 5.1: Median accuracy by training step across 5 repeats. Each node has 1000 random data samples from MNIST-F. The shaded region surrounding each line represents the upper and lower quartile. The right-hand graph represents the same graph but zoomed in.

The results depicted in Figure 5.2 demonstrate that the SwarmAvg algorithm exhibits a slower convergence rate compared to FedAvg, trailing by at least 1 epoch for the duration of the test. Despite this, both methods ultimately achieve a similar level of accuracy, with less than a percent difference.

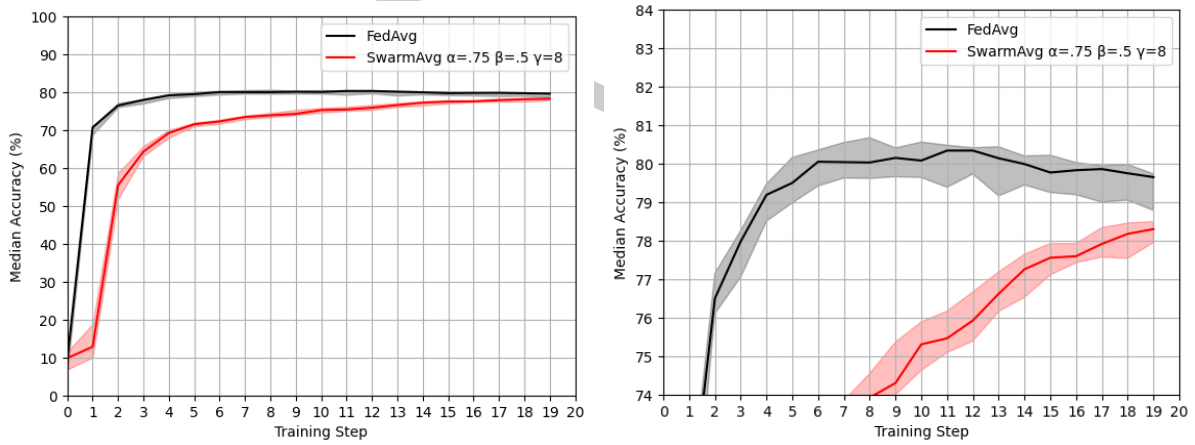


Figure 5.2: Median accuracy by training step across 5 repeats. Each node has 100 random data samples from MNIST-F. The shaded region surrounding each line represents the upper and lower quartile. The right-hand graph represents the same graph but zoomed in.

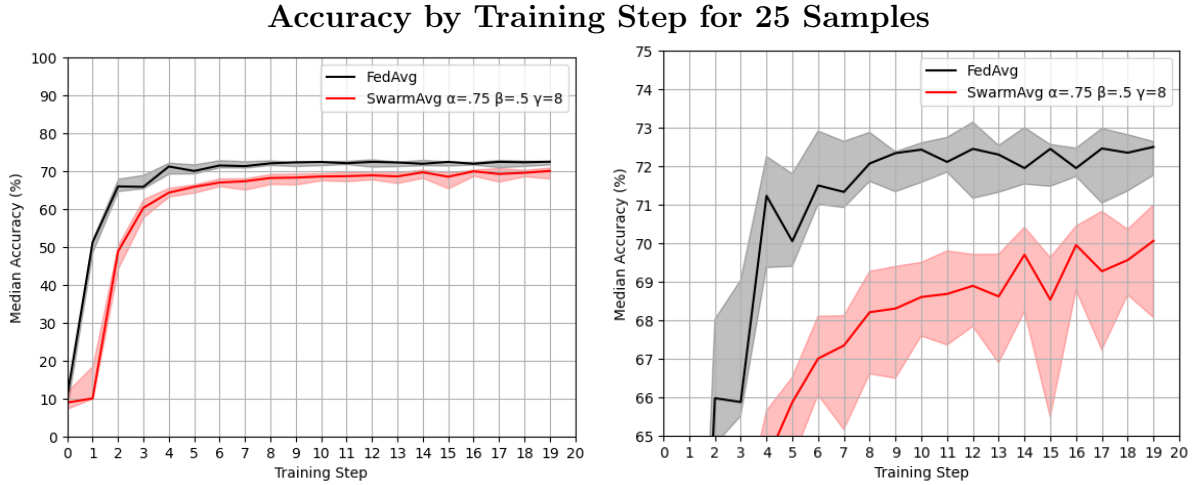


Figure 5.3: Median accuracy by training step across 5 repeats. Each node has 25 random data samples from MNIST-F. The shaded region surrounding each line represents the upper and lower quartile. The right-hand graph represents the same graph but zoomed in.

The trend of SwarmAvg ending with a lower accuracy than FedAvg continues in Figure 5.3. Both algorithms also have a much lower accuracy than what they achieved previously, due to the drastic decrease in available data.

Similarly to Figure 5.2, Figure 5.3 shows that SwarmAvg concludes with a lower accuracy than FedAvg. Furthermore, both algorithms exhibit a considerable decline in accuracy when compared to their previous performances with higher volumes of data.

The most noticeable impact resulting from reducing the volume of data is the significant decrease in the accuracy of both algorithms, as expected. However, it is also evident that the reduction in data affects SwarmAvg and FedAvg to a similar degree, as both of their peak accuracies stay within a similar range of each other in all tests. The difference in peak accuracy between the two algorithms was quite small, often within a 2 percent margin.

One of the prominent challenges associated with SwarmAvg is its slower convergence rate compared to FedAvg, particularly from the outset. SwarmAvg consistently takes a longer time to attain its peak accuracy. This may be attributed to the asynchronous nature of the nodes in SwarmAvg, which implies that some nodes that conduct training before others may have lower accuracy than expected.

It is worth noting that in all these evaluations, SwarmAvg has a slightly higher interquartile range than FedAvg, indicating that FedAvg is more consistent in terms of accuracy. However, this difference is minor.

One interesting feature of both SwarmAvg and FedAvg is that both algorithms seem to be affected by overfitting much less what would be expected. Especially with just 25 data samples per node, the effects of overfitting would usually be far more severe in a single node. However, this does not mean that the algorithms do not overfit, but it may indicate that the process of overfitting for them is drastically slowed down.

5.3 Scenario 2: Sparsely Connected Network

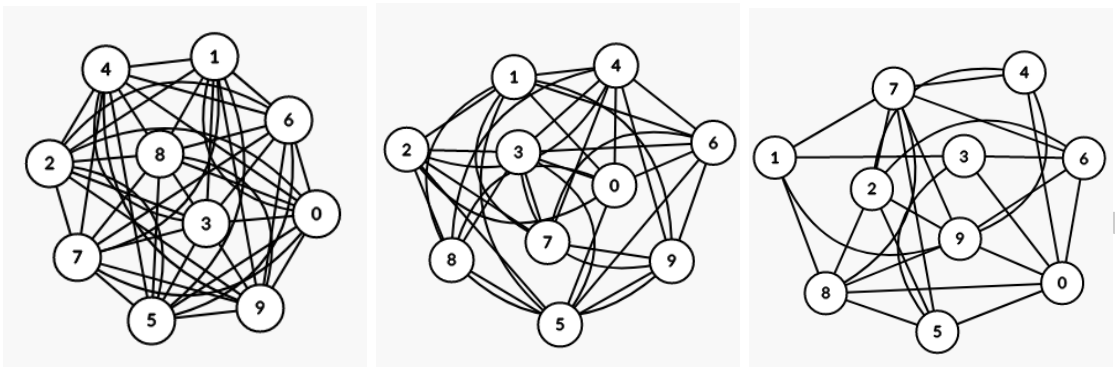
In reality, it is uncommon for each node to be linked with every other node. To simulate a more realistic scenario, a technique was employed to generate a network of nodes with a specific density. It is important to note that, moving forward, density refers to an artificial metric and is not associated with the physical definition of density. When density is set to 0, the network is minimally linked, meaning that each node has at least one indirect path to every other node, but the minimal number of connections required to accomplish this exist. When density is set to 1, all nodes are connected to one another in a dense fashion. Since this measure may not provide a straightforward indicator of network density, two additional metrics will be provided: Mean Minimum Hops (MMH) and Mean Connections per Node (MCPN). MMH denotes the mean minimum number of transitions required to get from one node to another in the network. MCPN denotes the average number of connections a given node possess, for a network of 10 nodes.

In order to conduct thorough testing on a variety of potential deployment scenarios, several different densities were tested for each count of data. The densities that were tested, along with their corresponding MMH and MCPN, are presented in Table 5.2.

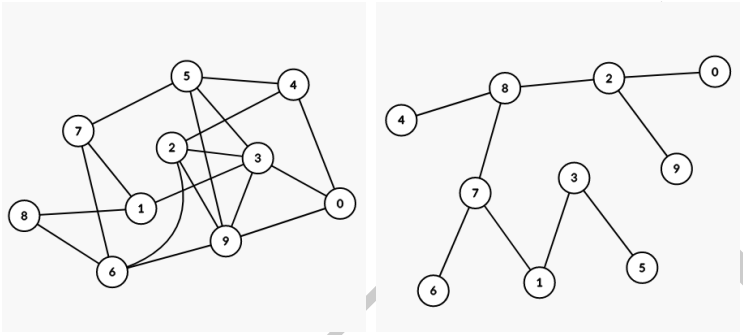
Density	MMH	MCPN
1	1.0	9.0
0.75	1.2	7.2
0.5	1.4	5.4
0.25	1.7	3.6
0	3.0	1.8

Table 5.2: The statistics for different density levels

In order to assist the reader with visualizing the various density levels, some sample networks that were generated for each density can be found in Figure 5.4.



(a) *density* = 1.0 (b) *density* = 0.75 (c) *density* = 0.5



(d) *density* = 0.25 (e) *density* = 0.0

Figure 5.4: Example networks of nodes generated for each density level, visualised using the tool at *https://csacademy.com/app/graph_ editor*. Each time a simulation is started, a new random network is generated for that simulation.

This section does not include any testing of FedAvg. In scenarios where not every node is directly connected to the server node, FedAvg has two potential options: ignore all nodes which are not directly connected, or attempt to relay the model updates through connected nodes. As mentioned in Section 3.5.2, relaying may not always be the optimal solution. Ignoring nodes is also not a good option, as data is wasted. Therefore, in this test, the decision was made to solely evaluate SwarmAvg.

5.3.1 Sparse Network Results

Presented below are the results obtained from executing the testing script. The density of each line on the graph is indicated by the colour, with a green hue representing higher density and a red hue indicating lower density. For all tested densities, the value of γ was set to $round_{down}(MCPN) - 1$, which ensures that each node can progress only after waiting for all but one of its neighbours. Notably, failure to reduce γ for less dense networks would cause several nodes to wait for a number of neighbours that cannot be achieved. Due to the random nature of the graph generation, there will still be some nodes who do not have γ neighbours, but this should be rare and the loop to wait for γ neighbours will terminate after a certain number of tries anyway, ensuring that training can progress.

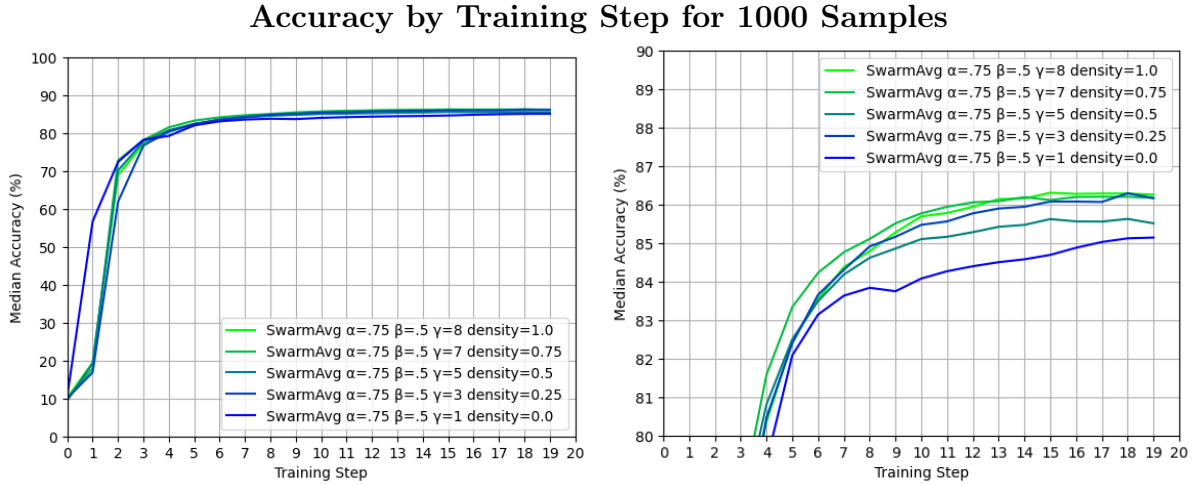


Figure 5.5: Median accuracy by training step across 5 repeats. Each node has 1000 random data samples from MNIST-F. Quartiles are not shown. The right-hand graph represents the same graph but zoomed in. The more green a line is, the higher its density

An observation that can be made of Figure 5.5 is that the different network densities perform very similarly. The final accuracy for all of the tests fell within a 2 percent range, with the higher densities occupying the higher end.

One interesting feature of the Figure 5.5 is that the lowest density demonstrates faster convergence initially that the other densities. The author hypothesises that this may be due to groups of nodes forming local sub-networks which converge on their own sub-sets of data faster, but more slowly combine into the network as a whole as training progresses.

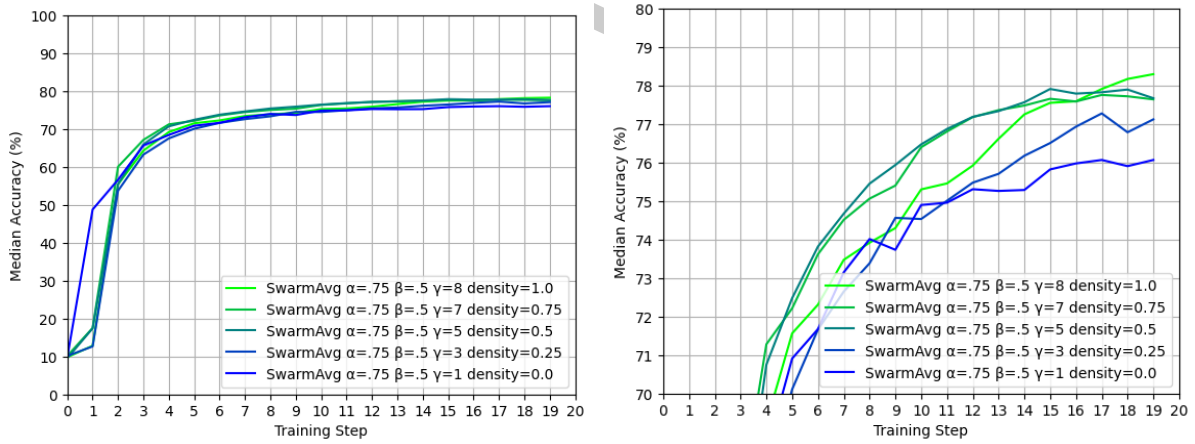


Figure 5.6: Median accuracy by training step across 5 repeats. Each node has 100 random data samples from MNIST-F. Quartiles are not shown. The right-hand graph represents the same graph but zoomed in. The more green a line is, the higher its density

In Figure 5.6, there is a greater spread of final accuracies, about 2.5 percent, than is shown in Figure 5.5. The more noticeable feature of Figure 5.6 is that the final accuracies of all densities are approximately 8 percent lower than their counterparts with more data available in figure 5.5.

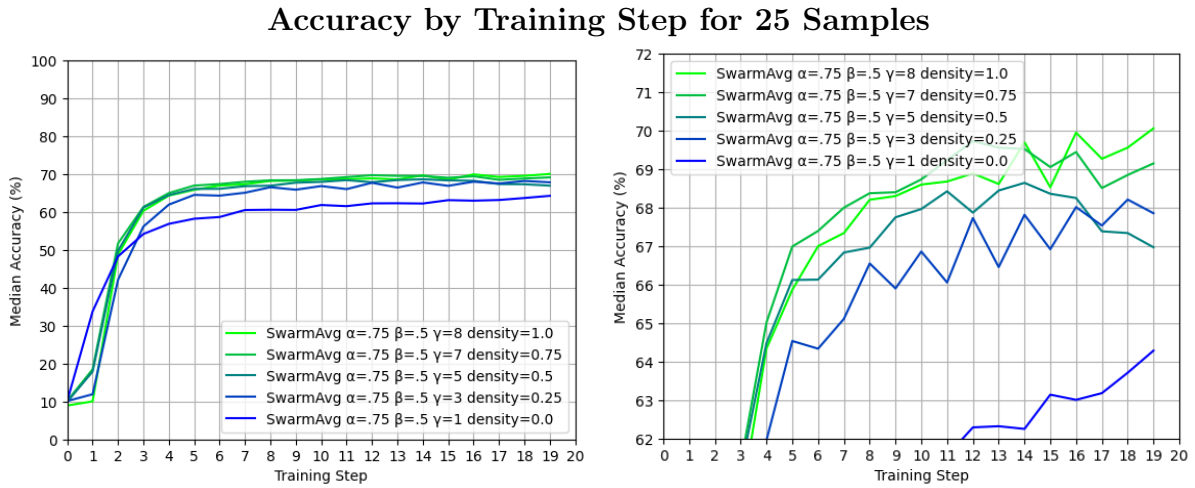


Figure 5.7: Median accuracy by training step across 5 repeats. Each node has 25 random data samples from MNIST-F. Quartiles are not shown. The right-hand graph represents the same graph but zoomed in. The more green a line is, the higher its density

With just 25 data samples, the tests depicted in Figure 5.7 show a much higher variation in final accuracies than the previous two figures. There is also a much higher degree of noise in this figure than any other, suggesting that training may be much less stable with this small number of data samples to train on.

It should also be noted that the lowest density is still exhibiting the behaviour of outperforming the other densities in the initial steps, however it appears that this effect has become slightly less prominent.

In general, altering the network density of nodes has a small impact on the training of the nodes within the network. Decreasing the density of nodes typically leads to a lower final accuracy. This effect becomes more noticeable as the data volume provided to each node decreases, as demonstrated by the increased variability in training displayed in Figure 5.7 in comparison to Figures 5.5 and 5.6.

The networks possessing a density of 0 attain their optima at a faster rate than those with higher densities. Figure 5.7 illustrates that the lower densities reach convergence marginally quicker than higher densities, yet are surpassed by the latter towards the end of training.

5.4 Scenario 3: Densely Connected Network with Class Restrictions

Decentralized machine learning algorithms, such as FL and SL, commonly involve a scenario where each node possesses a dataset whose distribution does not align with the overall data distribution across all nodes. In order to simulate such a scenario, a situation was created where each node was given access to only three out of ten classes of MNIST-F. However, it should be noted that each node was provided a unique set of three classes, and all ten classes were included at least once. The tests in this experiment were conducted in a densely connected network of nodes, thereby allowing the evaluation of FL as well. Furthermore, the FL nodes were subject to the same class-limiting constraints.

5.4.1 Dense Network with Class Restrictions Results

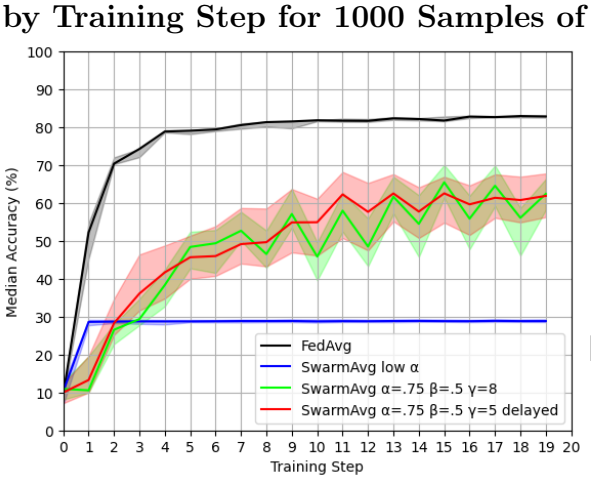


Figure 5.8: Median accuracy by training step across 5 repeats. Each node has 1000 random data samples from MNIST-F. Quartiles are depicted as the shaded area. Each node has access to a unique set of 3 classes out of 10

During the course of the experiment, it was determined that SwarmAvg required some adjustments in order to function properly. One issue that was particularly noticeable was a significant fluctuation in accuracy during the training process. The author posited that this may have been due in part to the synchronization of nodes in the network. To address this concern, the decision was made to decrease the value of γ and introduce a delay in the node start times, in order to desynchronize their training. The chosen delay time was 2 seconds. Figure 5.8 illustrates these two techniques, with the delayed training and reduced gamma method depicted in red, and the original method depicted in green. It is evident that the application of delayed training and reduced gamma had a positive impact in minimizing the oscillations, although they were not entirely eliminated.

While attempting to adjust the parameters, another issue was encountered which is known as divergent training. This phenomenon occurs when nodes begin to learn completely separate models and start to disregard the models of their neighbouring nodes. It was observed that a low value of α was sufficient to cause this problem. When divergent training occurred, the nodes failed to achieve an accuracy level higher than 30 percent, which is expected if a node only trains on three out of the ten available classes. This effect is shown as the blue line of Figure 5.8.

It is evident that SwarmAvg consistently exhibits a significantly lower level of accuracy compared to FedAvg. Additionally, the inter-quartile range of SwarmAvg is noticeably higher.

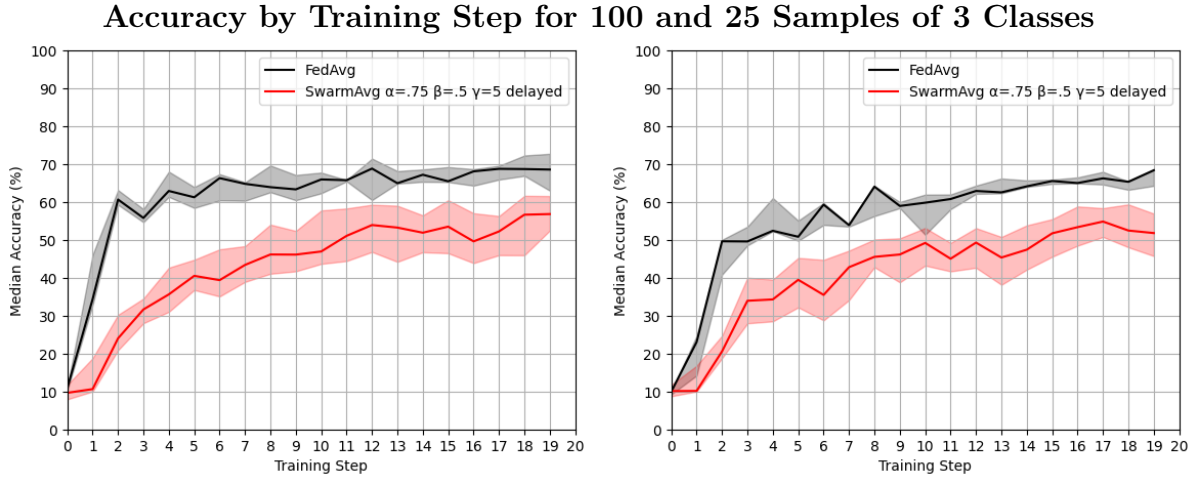


Figure 5.9: Median accuracy by training step across 5 repeats. On the left, each node has 100 random data samples from MNIST-F, on the right each node has 25. Quartiles are depicted as the shaded area. Each node has access to a unique set of 3 classes out of 10

When performing training on a lower sample count, as depicted in Figure 5.9, the oscillations in SwarmAvg seem to have been drastically reduced when compared to 5.8. However, both SwarmAvg and FedAvg have lower accuracies by the end of training than they did with 1000 samples.

It is evident that the reduction of data volume per node has a detrimental impact on the performance of SwarmAvg in general. However, in these restricted class tests, this effect is less pronounced compared to previous tests where each node had access to all possible classes. It should be noted that SwarmAvg encounters accuracy oscillations, which can be partially attributed to the synchronized training steps of nodes. Nevertheless, by decreasing the value of γ and staggering the startup of nodes, these oscillations can be mitigated. Moreover, reducing the data volume may also aid in reducing the effect of these oscillations, possibly implying that nodes are being trained excessively in a single step.

5.5 Scenario 4: Sparsely Connected Network with Class Restrictions

This scenario was designed to test the SwarmAvg algorithm to its limits. It not only involved limiting data to each node, but also limiting the classes each node had access to in the same fashion as Scenario 3, and also constraining the networks to be sparsely connected.

5.5.1 Sparse Network with Class Restrictions Results

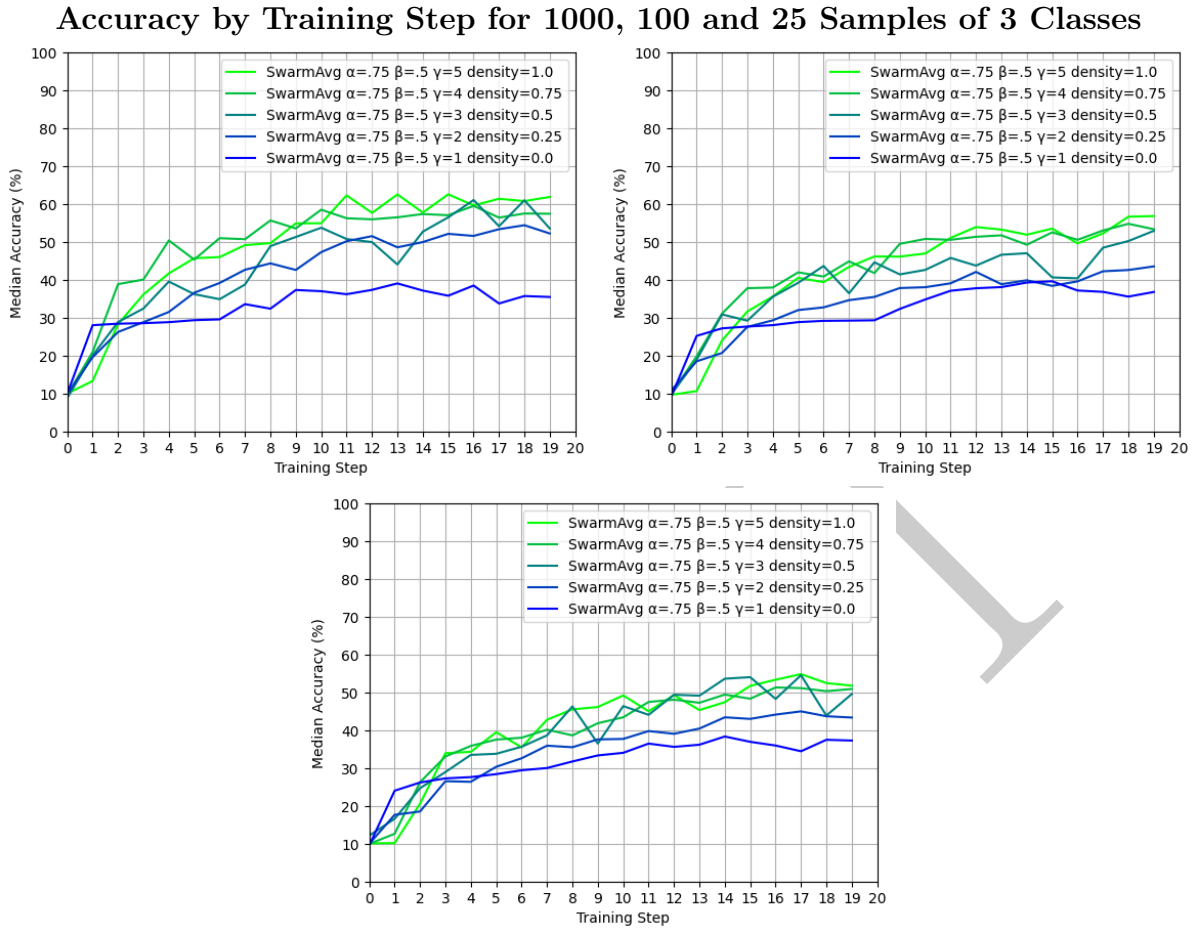


Figure 5.10: Median accuracy by training step across 5 repeats. On the top left, each node has 1000 random data samples from MNIST-F, on the top right each node has 100, and on the bottom, each node has 25. Quartiles are depicted as the shaded area. Each node has access to a unique set of 3 classes out of 10. The more green a line is, the higher the network density.

In all three tests, it is clear that a higher density is highly correlated with higher performance in this scenario. It is also noteworthy that with a network density of 0, no matter which volume of data was presented, the accuracy curve was very similar. However, as the density of the network increased, so did the degree at which it was effected by the reduction in data.

TODO: analyse more

Chapter 6

Critical Evaluation

6.1 Comparison of SwarmAvg to Other Methods

Although SwarmAvg has demonstrated promising results, it remains important to conduct a comparative analysis against established methods. This will enable individuals seeking to incorporate distributed machine learning into their future projects to make and informed decisions whether SwarmAvg is the algorithm that they should use.

6.1.1 Comparison of SwarmAvg to FedAvg

TODO: Write this out

- Can prevent a single slow node from bottlenecking the whole process
- No central server so more fault tolerant to node dropouts
- Can still function if all nodes do not have direct connections to all other nodes (in the case of FL every node must have access to the server)
- Less data transmission ($O(n)$ rather than worst case $O(n^2)$). This effect is less prominent if SL is sparse
- FL is a Less complex algorithm -> fewer parameters to tune -> easier to fit to a problem
- Slower to converge

6.1.2 Comparison of SwarmAvg to SwarmBC

Despite the fact that this paper did not perform tests on SwarmBC, it is still possible to compare the theoretical advantages of both methods over one another, based on a comprehensive understanding of the inner workings of each.

One of the primary drawbacks of utilizing SwarmAvg as opposed to SwarmBC is the potential occurrence of divergent training, which arises when individual nodes or groups of nodes begin to learn significantly different models from each other during SwarmAvg execution. Consequently, when merging the models from these groups, they prove incompatible and cause a decrease in overall performance. This detrimental effect may persist throughout the entire training process, leading to significantly worse outcomes than anticipated. Several factors can lead to divergent training, with low settings for α and γ, coupled with a sparsely-connected network, being highly correlated with its occurrence during testing. SwarmBC does not suffer from this issue, as the blockchain ensures that all nodes agree on the same model. However, during testing, divergent training was a rare phenomenon, and its remedy was a simple parameter tuning process upon detection.

One drawback of utilizing the SwarmAvg algorithm, as opposed to SwarmBC, is the absence of security features. The utilization of blockchain technology enables the straightforward authentication of nodes utilizing different algorithms that are frequently well-documented and integrated into the blockchain framework. On the other hand, SwarmAvg lacks any built-in authentication mechanism. As a result, to employ it securely, a separate stand-alone authentication system would need to be implemented. Failure to utilize authentication could result in a malicious node gaining entry and compromising the training process.

The rationale behind the development of SwarmAvg was to establish a more agile and efficient SL algorithm in comparison to the existing technologies. SwarmAvg surpasses SwarmBC in terms of computational efficiency, as nodes are not mandated to verify transactions. In contrast, Blockchain involves a comparatively substantial overhead as opposed to simply averaging the models of a nodes neighbours. Consequently, SwarmBC necessitates higher processing power to operate for the same amount of training. This overhead can impede the speed of training on a resource-limited system, such as a swarm of drones, as a considerable amount of time is consumed in validating transactions, rather than performing training. Thus, SwarmAvg may be a more fitting choice than SwarmBC for swarms where processing power is a concern.

TODO: SwarmAvg in theory may actually allow semi separated islands of nodes to learn slightly different models without fully diverging - this may be beneficial if the distribution of data changes. Cant do this with blockchain

6.2 Pitfalls of this Study

6.2.1 The Small Simulated Number of Nodes

A significant concern regarding this study is the limited number of nodes simulated during testing. Several drawbacks of FL in comparison to SL, such as server bottlenecking, are not apparent when testing on a limited number of nodes. Although SwarmAvg may have the potential to be utilized for training in large swarms, this particular scenario has not yet been evaluated.

6.2.2 The Lack of Testing Overheads

In the course of testing the SwarmAvg algorithm, it was determined that the assessment of its performance should be based on training steps rather than time. Although it would have been preferable to evaluate performance in relation to time, this was not feasible due to limitations in resources. The GPUs utilized in this study are primarily optimized for running a single GPU program effectively, whereas this research required multiple nodes to perform training simultaneously. As a result, it was noted during testing that training times varied significantly, rendering time-based measurement impractical due to high levels of noise.

6.2.3 The Lack of Simulated Internet Lag

The algorithm proposed is designed for usage over the internet. However, the study solely evaluated its performance within a simulated environment. It is worth noting that a fundamental distinction between the simulated environment and the real world is the absence of a time gap between the sending of a model update by one node and its reception by another. The deliberate omission of this time gap in the simulation aimed to increase the speed of simulations. However, this critical aspect of the algorithm's functionality remains untested as a result.

Chapter 7

Project Management

7.1 Time Management

To assist with planning and organisation of this project, Gantt charts were used. These helped to visualise which tasks needed doing, and also if the project progress was ahead or behind schedule. The Gantt for the interim report can be found in appendix D.1 and the final report Gantt can be found at D.2.

On the Gantt charts, it was decided to not count weekends as working days. This was chosen as it represented the fact that the author had other work to do during the course of the project.

The presented charts represent the final iteration of planning. As described in the following section, changes occurred throughout the project and the Gantt charts were adapted accordingly.

7.2 Changing Plans

At the time of writing the interim report, the project plan was very different to what is presented in this report. Originally, the plan was to implement SL and then to simulate real world problems such as uneven data distribution and sparse networks. However, during the process of implementing the SL algorithm it was apparent that the SL algorithm itself would take a very large amount of time to develop and test. For this reason, the decision was made to refocus the project onto the development of the SL algorithm, and if enough time was available after this, testing a single real world problem may be possible.

Plans were also changed on a smaller scale somewhat regularly. This is due to the experimental nature of this project, and the fact that it was impossible to predict some problems during the planning phase. For example, the parameters β and γ were not planned for at the start of the project, but whilst experimenting problems arose which seemed like they could be fixed by those parameters. This meant that another implementation had to be created which included those parameters.

The small changes to the plan were not an issue due to the inclusion of many buffer zones in the planned time allocation, such as *Bugfixing Time*. This meant that an overrun task would not affect too many tasks moving forwards, as it could be caught up on in a buffer zone.

7.3 Risk Assessment

The risk assessment was created early on in the project to mitigate any of the major risks. However, none of the described problems arose to an extent which their mitigation plan needed to be followed.

7.3.1 Personal Issues

Description

Personal issues which cause the author to be unable to do work, such as illness.

Risk Calculations

Severity (1-5): 3
Likelihood (1-5): 3
Overall Risk (1-25): **9**

Mitigation

The codebase will be designed such that individual sections and modules have minimal dependencies on other sections. This means that, even if the author is unable to work for a period of time, some less critical sections can be omitted without significantly impacting the rest of the project.

7.3.2 Hardware Failure

Description

Failure on the authors local computer of any kind, such as a graphics card or storage breakage.

Risk Calculations

Severity (1-5): 4
Likelihood (1-5): 2
Overall Risk (1-25): **8**

Mitigation

The project will be regularly backed up to GitHub. If a core component of the work computer breaks, the author has access to a personal laptop and the Zepler Labs. The deep learning environment along with dependencies is backed up to the authors Google Drive in the form of a docker image, so that switching to a new computer would be a smooth process.

7.3.3 Algorithm Does Not Work

Description

The SL algorithm does not function as well as expected.

Risk Calculations

Severity (1-5): 5
Likelihood (1-5): 1
Overall Risk (1-25): **5**

Mitigation

It may be possible to shift the project away from SL and onto distributed FL with leader election. FL is more commonly used and therefore has more literature, meaning that it is more likely to be an achievable goal to implement it.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

TODO: Conclude what is SwarmAvg
TODO: Conclude the findings + advantages
TODO: Conclude the disadvantages
TODO: Conclude why you would want to use SwarmAvg instead of another method

8.2 Future Work

In the future, a vital step would be to test SwarmAvg on a significantly larger number of nodes, exceeding the current 10 node limit. The aforementioned constraint was established due to limitations on the machine used for training. However, if a data scientist were to design and implement an internet-enabled back-end to the given code, it would be feasible to distribute numerous nodes among multiple training machines, thus enabling the simulation of larger swarms. It would be particularly intriguing to observe the performance of SwarmAvg in scenarios that entail extremely large swarms, but only involve minute amounts of data and transmissions exclusively between a few neighbours for each node who dynamically change over time - a common occurrence in swarm robotics.

A second aspect of future research that could enhance the potential of SwarmAvg involves conducting an in-depth examination of optimal parameters for different situations. For the present study, parameters had to be chosen heuristically to meet testing purposes within the time constraints, potentially resulting in suboptimal outcomes as compared to what could have been achievable.

Finally, the scenarios in which SwarmAvg was tested where each node had access to only a subset of the classes left much to be desired. SwarmAvg performed far worse than FedAvg in these situations, however it remains plausible that with future research and modifications, that SwarmAvg may be able to handle these situations better.

Appendix A

SwarmAvg Algorithm

A.1 Single Training Step

Algorithm 1 Training Step - Called Repeatedly in a Loop

```
1: TRAIN(localModel, localData)
2: for all n ∈ neighbors do
3:     SENDTO((localModel, localTrainingCounter), n)
4: end for
5: for x ∈ range(maxSyncWaits) do
6:     neighborModels ← ∅
7:     for all n ∈ neighbors do
8:         model, traningCounter ← CACHELOOKUP(n)
9:         if trainingCounter + β ≥ localTrainingCounter then
10:            APPEND(neighborModels, model)
11:        end if
12:    end for
13:    if length_neighborModels ≥ γ then
14:        if syncronisationMethod = "AVG" then
15:            localModel ← μ(localModel ∪ neighborModels)
16:        else if syncronisationMethod = "ASR" then
17:            localModel ← (1 − α) ∗ localModel + α ∗ μ(neighborModels)
18:        end if
19:    else
20:        continue
21:    end if
22:    SLEEP(syncWaitTime)
23: end for
```

A.2 Model Received Event

Algorithm 2 Model Received Event - Called When a Model Update is Received from a Remote Node

Input: $neighbour$, $nModel$, $nTrainingCounter$

1: **if** InCache($neighbour$) **then**
2: $\quad$ $\_, nTraningCounterOld \leftarrow$ CacheLookup(n)
3: $\quad$ **if** $nTrainingCounter > nTraningCounterOld$ **then**
4: $\quad\quad$ SetCache($neighbour$, $(nModel, nTrainingCounter)$)
5: $\quad$ **end if**
6: **else**
7: $\quad$ SetCache($neighbour$, $(nModel, nTrainingCounter)$)
8: **end if**

Appendix B

Network Generation Algorithm

Algorithm 3 Network Generator - Used to generate networks with specific density

Input: *density*

1: ▷ Generate a random minimally connected graph

Appendix C

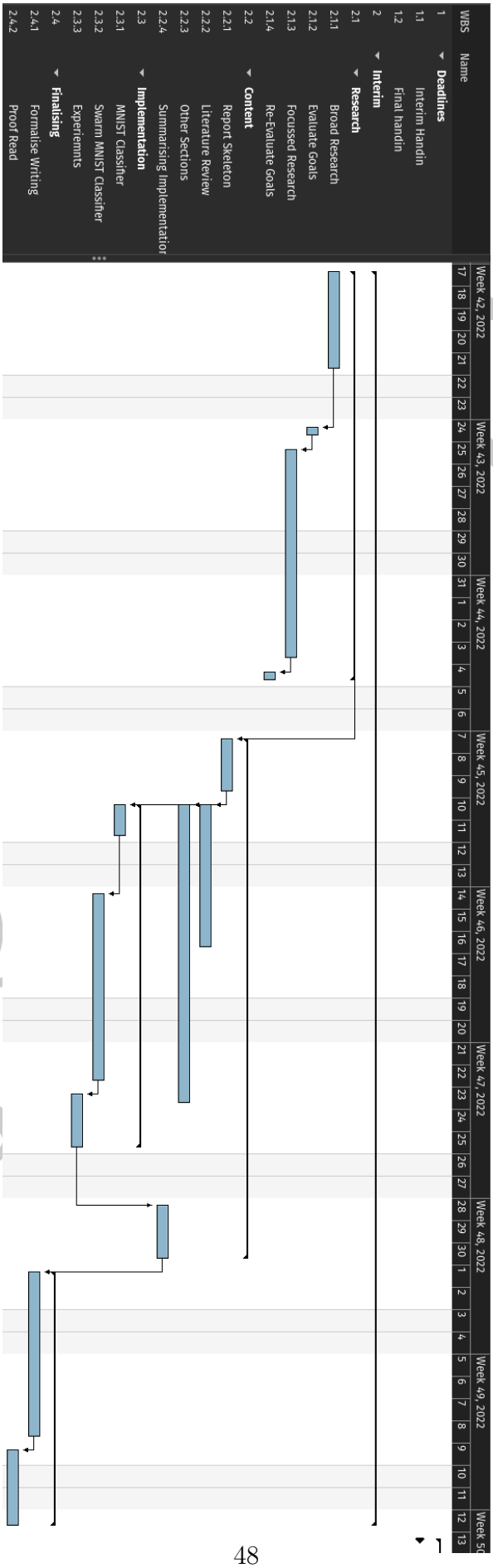
Machine Learning Model

```
inp = Input((28,28))
out = Reshape((28,28,1))(inp)
out = Conv2D(16, (3,3), activation="relu")(out)
out = Conv2D(16, (3,3), activation="relu")(out)
out = Flatten()(out)
out = Dense(256, activation="relu")(out)
out = Dense(128, activation="relu")(out)
out = Dense(10, activation="sigmoid")(out)
model = Model(inputs=inp, outputs=out)
model.compile(
        optimizer="adam",
        loss=SparseCategoricalCrossentropy(),
        metrics=[SparseCategoricalAccuracy()]
)
```

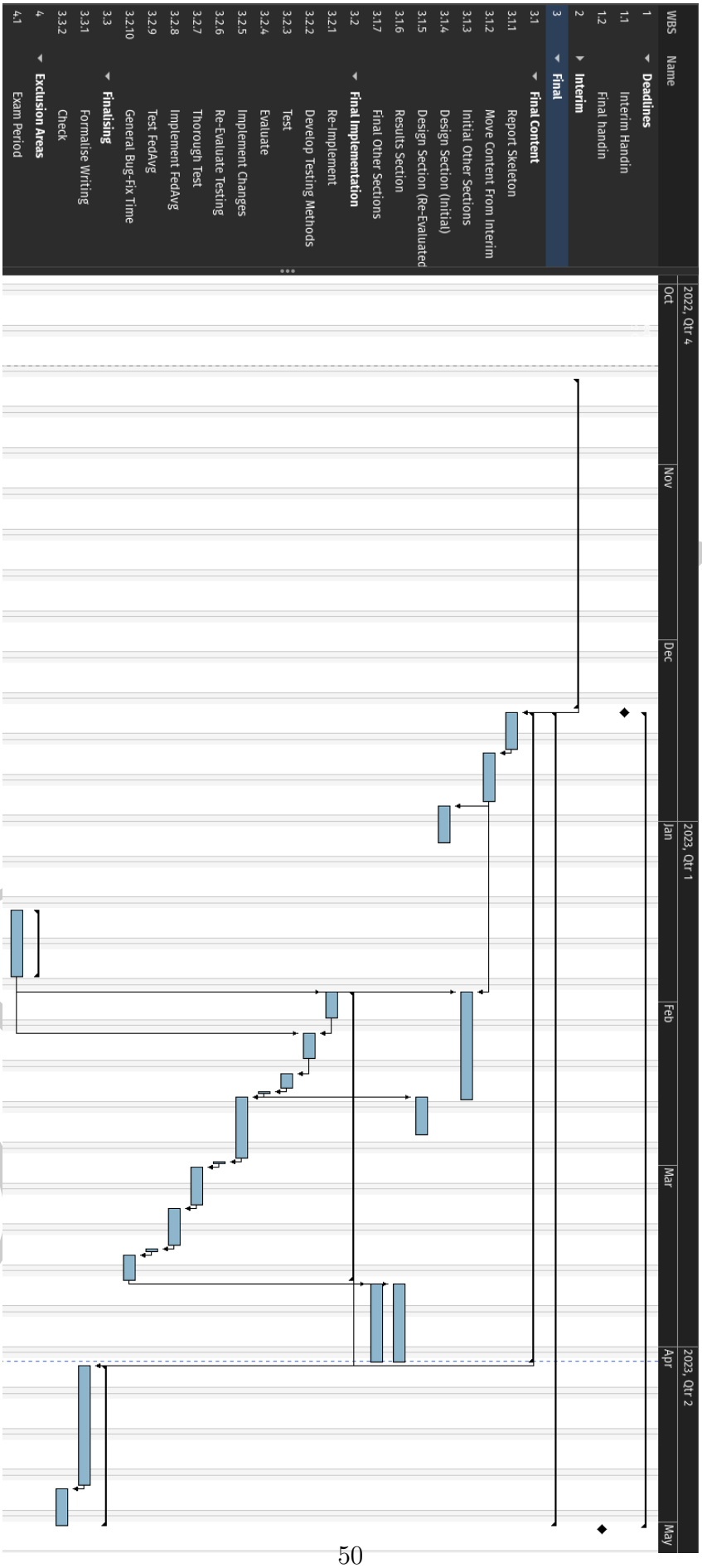
Appendix D

Gantt Charts

D.1 Gantt - Interim



D.2 Gantt - Final



Bibliography

[1] W. L. Shuya Ke, "Distributed multi-agent learning is more effectively than single-agent,"

[2] M. Kop, "Machine learning & eu data sharing practices," Stanford-Vienna Transatlantic Technology Law Forum, Transatlantic Antitrust . . . , 2020.

[3] G. A. Kaissis, M. R. Makowski, D. Rückert, and R. F. Braren, "Secure, privacy-preserving and federated machine learning in medical imaging," *Nature Machine Intelligence*, vol. 2, no. 6, pp. 305–311, 2020.

[4] C. Zhang, Y. Xie, H. Bai, B. Yu, W. Li, and Y. Gao, "A survey on federated learning," *Knowledge-Based Systems*, vol. 216, p. 106775, 2021.

[5] Q. Li, Z. Wen, Z. Wu, S. Hu, N. Wang, Y. Li, X. Liu, and B. He, "A survey on federated learning systems: vision, hype and reality for data privacy and protection," *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[6] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, "Communication-efficient learning of deep networks from decentralized data," 2016.

[7] V. Mothukuri, R. M. Parizi, S. Pouriyeh, Y. Huang, A. Dehghantanha, and G. Srivastava, "A survey on security and privacy of federated learning," *Future Generation Computer Systems*, vol. 115, pp. 619–640, 2021.

[8] J.-H. Chen, M.-R. Chen, G.-Q. Zeng, and J.-S. Weng, "Bdfl: a byzantine-fault-tolerance decentralized federated learning method for autonomous vehicle," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 9, pp. 8639–8652, 2021.

[9] D. Yaga, P. Mell, N. Roby, and K. Scarfone, "Blockchain technology overview," tech. rep., oct 2018.

[10] D. Yang, C. Long, H. Xu, and S. Peng, "A review on scalability of blockchain," in *Proceedings of the 2020 The 2nd International Conference on Blockchain Technology*, ICBCT'20, (New York, NY, USA), p. 1–6, Association for Computing Machinery, 2020.

[11] S. Warnat-Herresthal, H. Schultze, K. L. Shastry, S. Manamohan, S. Mukherjee, V. Garg, R. Sarveswara, K. Händler, P. Pickkers, N. A. Aziz, S. Ktena, F. Tran, M. Bitzer, S. Ossowski, N. Casadei, C. Herr, D. Petersheim, U. Behrends, F. Kern, T. Fehlmann, P. Schommers, C. Lehmann, M. Augustin, J. Rybniker, J. Altmüller, N. Mishra, J. P. Bernardes, B. Krämer, L. Bonaguro, J. Schulte-Schrepping, E. De Domenico, C. Siever, M. Kraut, M. Desai, B. Monnet, M. Saridaki, C. M.

Siegel, A. Drews, M. Nuesch-Germano, H. Theis, J. Heyckendorf, S. Schreiber, S. Kim-Hellmuth, P. Balfanz, T. Eggermann, P. Boor, R. Hausmann, H. Kuhn, S. Isfort, J. C. Stingl, G. Schmalzing, C. K. Kuhl, R. Röhrig, G. Marx, S. Uhlig, E. Dahl, D. Müller-Wieland, M. Dreher, N. Marx, J. Nattermann, D. Skowasch, I. Kurth, A. Keller, R. Bals, P. Nürnberg, O. Rieß, P. Rosenstiel, M. G. Netea, F. Theis, S. Mukherjee, M. Backes, A. C. Aschenbrenner, T. Ulas, A. Angelov, A. Bartholomäus, A. Becker, D. Bezdan, C. Blumert, E. Bonifacio, P. Bork, B. Boyke, H. Blum, T. Clavel, M. Colome-Tatche, M. Cornberg, I. A. De La Rosa Velázquez, A. Diefenbach, A. Dilthey, N. Fischer, K. Förstner, S. Franzenburg, J.-S. Frick, G. Gabernet, J. Gagneur, T. Ganzenmueller, M. Gauder, J. Geißert, A. Goesmann, S. Göpel, A. Grundhoff, H. Grundmann, T. Hain, F. Hanses, U. Hehr, A. Heimbach, M. Hoeper, F. Horn, D. Hübschmann, M. Hummel, T. Iftner, A. Iftner, T. Illig, S. Janssen, J. Kalinowski, R. Kallies, B. Kehr, O. T. Keppler, C. Klein, M. Knop, O. Kohlbacher, K. Köhrer, J. Korbel, P. G. Kremsner, D. Kühnert, M. Landthaler, Y. Li, K. U. Ludwig, O. Makarewicz, M. Marz, A. C. McHardy, C. Mertes, M. Münchhoff, S. Nahnsen, M. Nöthen, F. Ntoumi, J. Overmann, S. Peter, K. Pfeffer, I. Pink, A. R. Poetsch, U. Protzer, A. Pühler, N. Rajewsky, M. Ralser, K. Reiche, S. Ripke, U. N. da Rocha, A.-E. Saliba, L. E. Sander, B. Sawitzki, S. Scheithauer, P. Schiffer, J. Schmid-Burgk, W. Schneider, E.-C. Schulte, A. Sczyrba, M. L. Sharaf, Y. Singh, M. Sonnabend, O. Stegle, J. Stoye, J. Vehreschild, T. P. Velavan, J. Vogel, S. Volland, M. von Kleist, A. Walker, J. Walter, D. Wieczorek, S. Winkler, J. Ziebuhr, M. M. B. Breteler, E. J. Giamarellos-Bourboulis, M. Kox, M. Becker, S. Cheran, M. S. Woodacre, E. L. Goh, J. L. Schultze, C.-. A. S. (COVAS), and D. C.-. O. I. (DeCOI), "Swarm learning for decentralized and confidential clinical machine learning," *Nature*, vol. 594, pp. 265–270, Jun 2021.

[12] J. C. Varughese, R. Thenius, T. Schmickl, and F. Wotawa, "Quantification and analysis of the resilience of two swarm intelligent algorithms.," in *GCAI*, pp. 148–161, 2017.

[13] J. Augustine, T. Kulkarni, and S. Sivasubramaniam, "Leader election in sparse dynamic networks with churn," in *2015 IEEE International Parallel and Distributed Processing Symposium*, pp. 347–356, IEEE, 2015.

[14] M. Dorigo, M. Birattari, and M. Brambilla, "Swarm robotics," *Scholarpedia*, vol. 9, no. 1, p. 1463, 2014. revision #138643.

[15] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," 2017.