

Swarm Learning: Fully Distributed Machine Learning

Josh Pattman

April 8, 2023

Contents

1	Background	4
1.1	Federated Learning	5
1.2	Swarm Learning	5
1.3	Blockchain	7
2	Problem and Goals	8
2.1	Problem	9
2.1.1	Privacy	9
2.1.2	Performance	9
2.2	Goals	9
2.2.1	Design an Novel Algorithm for Swarm Learning	9
2.2.2	Implement the Algorithm	10
2.2.3	Test Performance in Situations Where Federated Learning Performs Well	10
2.2.4	Test Performance in Situations Where Federated Learning Performs Badly	10
3	Analysis and Specification of the Solution to the Problem	11
4	Detailed Design	12
4.1	Node	13
4.2	No Blockchain	13
4.3	Training Counter	14
4.4	Model Combination Methods	14
4.4.1	Averaging	14
4.4.2	Averaging With Synchronisation Rate	15
4.4.3	Filtering By Training Counter	15
4.5	Sparse Network Behaviour	16
4.5.1	Passive Convergence	16
4.5.2	Relay	16

5	Implementation	18
5.1	Dataset and Machine Learning Model	19
5.1.1	Dataset	19
5.1.2	Model	19
5.2	Federated Learning	19
5.2.1	Algorithm	20
5.2.2	Evaluation	20
5.3	Prototype	20
5.3.1	Algorithm	20
5.3.2	Evaluation	21
5.4	Final	21
5.4.1	Algorithm	21
5.4.2	Evaluation	21
6	Testing Strategy and Results	23
6.1	Methods	24
6.1.1	Metrics	24
6.1.2	Data Collection	24
6.1.3	Node Counts	24
6.1.4	Algorithm Configurations	24
6.1.5	Data Per Node	24
6.2	Dense Network Performance	25
6.3	Dense Network Performance with Node Dropout	28
6.4	Sparse Network Performance	28
6.5	Sparse Network Performance with Dropout	28
7	Critical Evaluation	29
7.1	Comparison of SL to FL	30
7.1.1	Advantages of using SL over FL	30
7.1.2	Disadvantages of using SL over FL	30
8	Project Management	31
8.1	Time Management	32
8.2	Changing Plans	32
8.3	Risk Assessment	33
8.3.1	Personal Issues	33
8.3.2	Hardware Failure	33
8.3.3	Algorithm Does Not Work Work	34
9	Conclusion and Future Work	35

A	Machine Learning Model	36
B	Gantt Charts	37
B.1	Gantt - Interim	38
B.2	Gantt - Final	39

Chapter 1

Background

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 [1], reducing the amount of data available to train models on and negatively impacting trained performance [2]. Federated Learning (FL) [3] 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 [4], each with different merits and drawbacks for certain use cases. Federated Averaging (FedAvg) [5] 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 improves over time, and after a certain number of steps, the training is complete and the server model is the final output.

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 **CITE THIS STATEMENT**. 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 **CITE THIS STATEMENT**.

One branch of FL is distributed federated learning (DFL). This algorithm works in a very similar way to FL, but replaces the central server with an elected leader in a network of nodes [6]. DFL has been shown to increase fault tolerance and security over FL.

1.2 Swarm Learning

Swarm learning (SL) [7] is a subcategory of FL which operates in a completely distributed and decentralised manner. SL still allows a network of nodes to

collaborate to learn a shared model, but at no point is a central server used or a leader chosen. In contrast to FL, SL methods usually use a blockchain based system to coordinate a global model.

TODO: Change this image

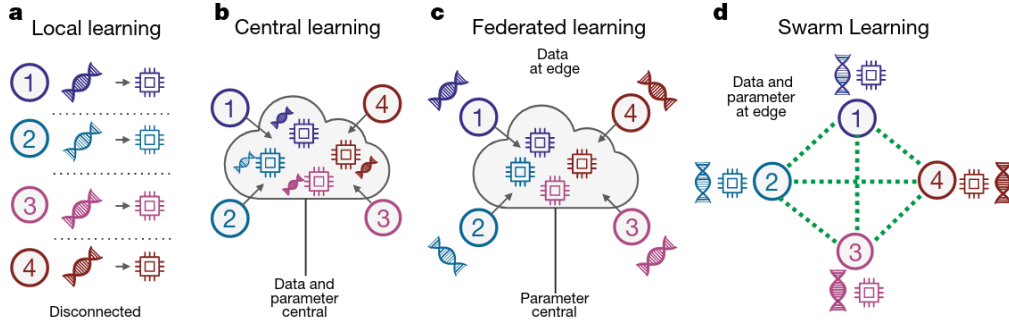


Figure 1.1: Diagram of swarm learning compared to other type of learning taken from [7]

. The DNA icon denotes the location of data and the CPU icon denotes the location the model is stored

In SL, nodes learn in a similar fashion to FL in training steps. However, the nodes do not need to have their training steps synchronised. Following are the typical training steps of a swarm learning node:

1. A node obtains a copy of the current 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

The model on the blockchain will improve over time, and after a certain number of steps the blockchain model is the final output.

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 [8], the absence of a central server makes SL more resilient to failures than centralized FL approaches such as FedAvg. The lack of a leader election protocol also means that SL may be better suited to tasks where networks of nodes are sparsely connected, as leader election in dynamic networks is a complex

problem and can add large amounts of overhead [9]. Additionally, the removal of the need for a server in SL reduces the likelihood of performance bottlenecks for very large swarms.

1.3 Blockchain

TODO: Write this out

Blockchain is ... [10]

Blockchain has problems with scalability [11].

Blockchain is inefficient and slow [11].

Chapter 2

Problem and Goals

2.1 Problem

In the modern world of machine learning, there exist some problems which are difficult to solve with conventional and centralised, machine learning approaches.

2.1.1 Privacy

It is common for data to be spread across multiple locations, known 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 train a model.

For example, due to the obligation to maintain patient confidentiality, the medical data of patients stored in a hospital cannot be shared with any other entity. However, if a model was trained on all of the data across all sources, it could be of great benefit to many people.

2.1.2 Performance

In general, it has been observed that larger machine learning models coupled with more data typically lead to better performance ***CITE THIS STATEMENT***. However, the use of conventional approaches for training such big 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 an opportunity to utilise the unused processing power.

2.2 Goals

2.2.1 Design an Novel Algorithm for Swarm Learning

In this project, the primary aim is to design a novel swarm learning 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.

2.2.2 Implement the Algorithm

The algorithm should be implemented for the primary purpose of testing its performance, meaning it should be the minimal viable implementation. It should be implemented in an easy-to-understand programming language, making it simple for someone to reproduce.

2.2.3 Test Performance in Situations Where Federated Learning Performs Well

It has been demonstrated that Federated Learning is effective in settings where each node has a reliable connection to the server *CITE THIS STATEMENT*. 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 Test Performance in Situations Where Federated Learning Performs Badly

Federated learning may not be effective when the server has unreliable connections to the nodes, or when nodes are halted *CITE THIS STATEMENT*. Additionally, if the server stops working, federated learning 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 federated learning may not be viable.

Chapter 3

Analysis and Specification of the Solution to the Problem

TODO: Figure out what goes here

Chapter 4

Detailed Design

4.1 Node

In swarm learning, a node is an agent responsible for facilitating the improvement of the global model. Each node maintains a local model, which is an approximation of the global model that is stored locally. However, the global model is an abstract concept representing of the average of all local models across all nodes. As training progresses and performance approaches a plateau, the global model gradually converges towards each node's local model.

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 steps in each training loop are as follows:

1. Fit local model to local dataset
2. Send local model to all neighbours
3. Combine neighbouring models into local model

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 at the instant of the combination step. The reasoning behind this decision is elaborated upon in greater detail in the implementation section.

4.2 No Blockchain

TODO: Write this out

- Neural nets are heuristic - they don't need to be exact
- Its just overhead
- Blockchain can have situations where data is lost (branches) **VALIDATE THIS STATEMENT**
- SL with averaging more robust against malicious agents than SL with blockchain **VALIDATE THIS STATEMENT**

4.3 Training Counter

A vital aspect of the swarm learning algorithm, 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 of a network by estimating the number of epochs 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 epoch 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 represent the average of all neighbouring nodes' training counters.

4.4 Model Combination Methods

The combination step is a crucial component of the swarm learning algorithm. 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, which are evaluated in the results section.

In the below equations, $\mu(x)$ denotes the function $mean(x)$, t_x denotes $training_counter_x$, and m_x denotes $model_x$.

4.4.1 Averaging

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

$$m_{local+1} = \mu(\{m_{local}\} \cup m_{neighbours})$$

This technique is utilized in FedAvg, which is the most simplistic form of federated learning. The benefit of this method is that it necessitates no hyper-parameters, which translates to less tuning required by the programmer. However, this attribute can also be viewed as a drawback, as it affords less flexibility in terms of customization for particular tasks.

4.4.2 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.

$$m_{local+1} = (1 - \alpha) * m_{local} + \alpha * \mu(m_{neighbours})$$

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. The implications of adjusting this parameter are discussed further in the results section.

4.4.3 Filtering By Training Counter

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:

$$t_{neighbour} + \beta \geq t_{local}$$

The training offset β is the amount the training counter of a neighbour can be behind the local training counter before it is ignored. Different values of β are explored in the results section.

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 run-away 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, these nodes start to overfit on their own training data, as their model is only exposed to this data, thereby leading to decreased overall performance, as well as potential performance reductions in the rest of the

network. To address this problem in the proposed swarm learning algorithm, 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.

4.5 Sparse Network Behaviour

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 SL should be able to deal with this situation.

4.5.1 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.

4.5.2 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, so will not be discussed further.

Chapter 5

Implementation

5.1 Dataset and Machine Learning Model

5.1.1 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, a drop-in replacement dataset containing 10 different items of clothing. MNIST-fashion is known to be more challenging ***CITE THIS STATEMENT*** [<https://arxiv.org/abs/1708.07747>]. 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.

5.1.2 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 A and is a small convolutional neural network. The model was tested on the MNIST fashion dataset and was able to attain an accuracy score of above 90 percent when trained on the entire dataset; this result is on par with the accuracy reported in the original paper ***CITE THIS STATEMENT*** [<https://arxiv.org/abs/1708.07747> (same as above)], making the model suitable for use.

5.2 Federated Learning

Federated learning was chosen for comparison of performance against swarm learning. In order to ensure fairness of the comparison, it was necessary to implement the federated learning algorithm from scratch, using the same frameworks and language as the swarm learning algorithm.

5.2.1 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. Nevertheless, 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 significantly accelerating training. This enabled a greater number of training runs to be conducted, resulting in more data being collected.

5.2.2 Evaluation

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.

5.3 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.

5.3.1 Algorithm

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 concept of a training counter, meaning that training counter filtering, β and γ were all absent.

5.3.2 Evaluation

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 time consuming. For this reason, it was decided not to record the results of this method.

5.4 Final

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

5.4.1 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 back end for distributing models was implemented as an interface which abstracts how the back end works away from the main algorithm code. In the results section, the same distribution strategy as federated learning was used: calling local functions that simulated a web connection.

5.4.2 Evaluation

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 6

Testing Strategy and Results

TODO: Write this section out properly

6.1 Methods

6.1.1 Metrics

For each graph, the x axis signifies number of epochs of training, and the y axis signifies accuracy. Accuracy is calculated after the synchronisation step, by checking the number of correct predictions on an unseen test set.

6.1.2 Data Collection

For each set of parameters to the algorithm, training was repeated 5 times, and the accuracies for each node from every run were recorded. For each timestep, the accuracy was taken to be the median accuracy between all nodes across all runs at that timestep. This helped to reduce noise in the performance metrics.

6.1.3 Node Counts

All experiments were run with 10 nodes, excluding the server when using federated learning. This number was chosen as it was the highest node count that could be achieved without the training machine’s resources being exhausted and crashing.

6.1.4 Algorithm Configurations

In the each of the following experiments, the algorithm is configured with a specific set of parameters. These parameters were found heuristically, by making an initial guess and testing, then tweaking the parameters until a good result was found. Doing a full investigation into the optimal set of parameters for a specific type of problem is not in the scope of this paper, meaning that it is entirely possible that swarm learning could perform better with more well tuned parameters.

6.1.5 Data Per Node

Each experiment tests the algorithm performance with three different volumes of data per node. These are all much less than the full MNIST dataset as the algorithm is expected to be use in situations where data assess for a single node is limited. A subsection of the data is taken for each node by

randomly sampling the number of datapoints at the start of training. Each nodes data never changes throughout training. Multiple nodes can contain the same datapoint, and on the flip side, a datapoint is not guaranteed to be seen by a node.

Due to the small size of the dataset, each training loop a node will perform more than one epoch of training. The number of epochs of training a node performs per training step is know as Epochs per Step (EPS). Through testing, it was found that both SL and FL benefit from higher EPS, sometimes more so than increasing the number of training steps.

The three levels of data volume with their respective EPS are shown in table 6.1

- 6000 - This is 1/10th of the whole dataset. There are 10 nodes. - 2 epochs per update
- 1000 - Lower amount of data - 5 epochs per update
- 100 - Extreme case - ? epochs per update

Dataset Size	EPS	Reason
6000	2	As there are 10 nodes, it was decided that each node should be tested with 1/10th of the dataset
1000	5	A much lower volume of data was tested to reflect the anticipated use case of SL - training models where each node has very small amounts of data
100	15	The extreme case was tested to see how well the algorithms perform in undesirable conditions

Table 6.1: The different levels of dataset size and EPS that were tested

6.2 Dense Network Performance

An important benchmark for the swarm learning algorithm is to test the performance in the best possible case: a network of nodes where every node is directly connected to every other node. The equivalent topology for federated learning is that every node is directly connected to the server.

The parameters for the SL algorithm were chosen based on the authors past experience with testing the algorithm. The parameters are shown in table 6.2.

P	Value	Reason
α	0.75	Low enough to allow nodes to maintain a small variation but not so low that the nodes diverge indefinitely
β	0.5	Allows a small amount of nodes looking back, but high value is not needed as every node will be running at approximately the same speed.
γ	8	All nodes will always be connected to 9 other nodes, so higher is better. There is room for 1 node to be skipped to prevent deadlock.

Table 6.2: The chosen parameters for the SL algorithm

For SL, the nodes were all configured to connect to every other node in the network. For FL, each node was configured to have a direct connection to the server.

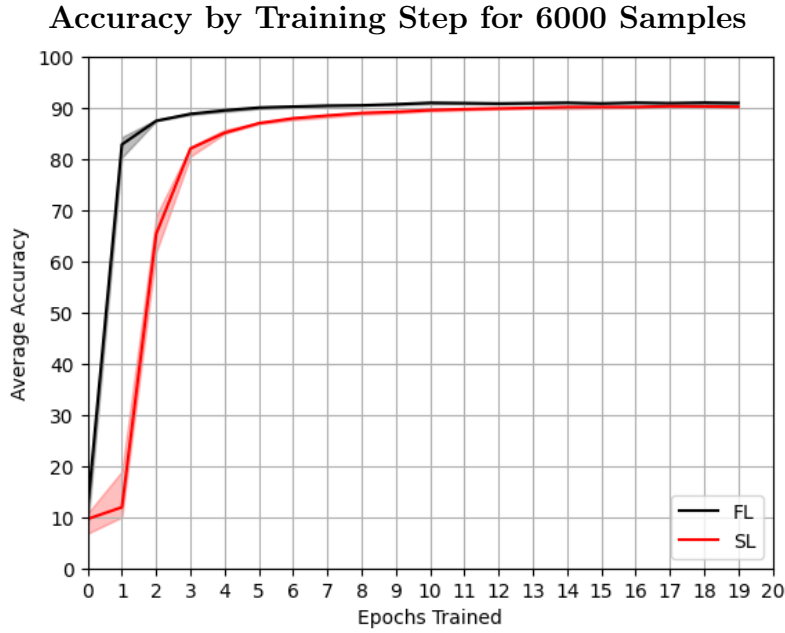


Figure 6.1: Comparing Accuracy of FL and SL with 6000 Data Samples per Node

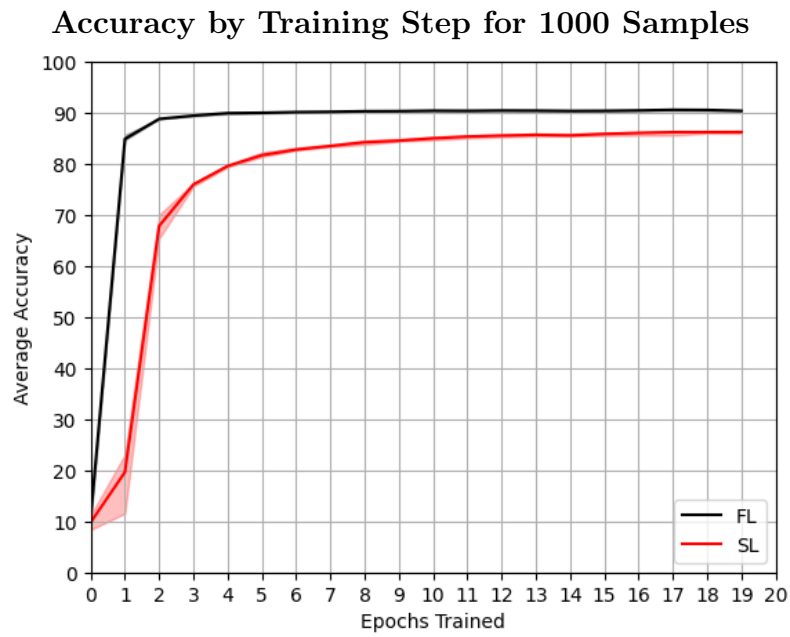


Figure 6.2: Comparing Accuracy of FL and SL with 1000 Data Samples per Node

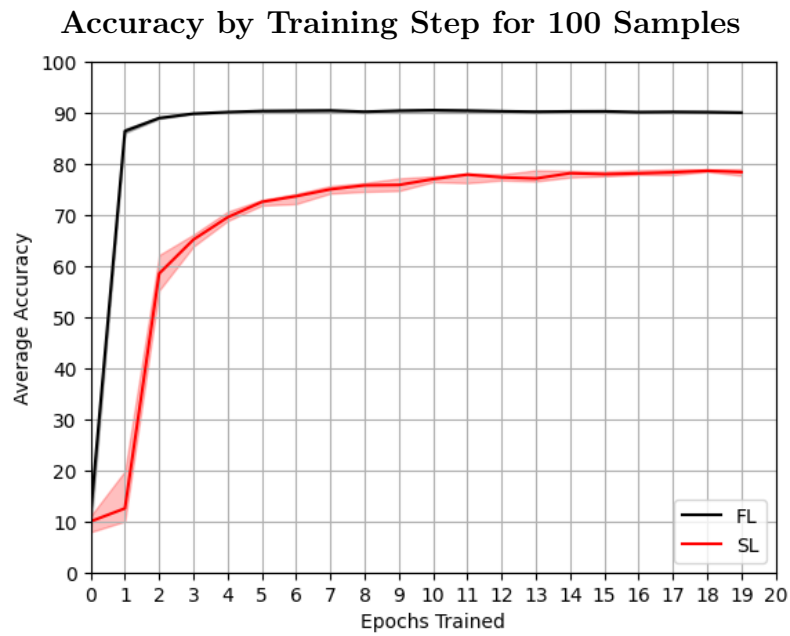


Figure 6.3: Comparing Accuracy of FL and SL with 100 Data Samples per Node

TODO: Describe the above graphs

6.3 Dense Network Performance with Node Dropout

the same but this time dropout nodes. Try with and without filtering to show that it increases fault tolerance

6.4 Sparse Network Performance

can only test SL here (i guess you could take average number of neighbours and do fl on that too)

6.5 Sparse Network Performance with Dropout

Chapter 7

Critical Evaluation

7.1 Comparison of SL to FL

This report has show that SL has the capability to be a competitor to FL in a distributed learning situation. SL can also be used in some situations where FL cannot. However, there are some clear disadvantages to using SL as opposed to FL. *TODO: Make this flow better*

7.1.1 Advantages of using SL over FL

The main advantage to using SL over FL is the ability for SL to function on networks where every node does not have connections to every other node.

TODO: Write this out

- Can prevent a single slow node from bottlenecking the whole process
- No central server so more fault tolerant
- 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)

7.1.2 Disadvantages of using SL over FL

TODO: Write this out

- Less data transmission ($O(n)$ rather than worst case $O(n^2)$). This effect is less prominent if SL is sparse
- Less complex algorithm -> fewer parameters to tune

Chapter 8

Project Management

8.1 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 B.1 and the final report Gantt can be found at B.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.

8.2 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.

8.3 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.

8.3.1 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.

8.3.2 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.

8.3.3 Algorithm Does Not Work 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 9

Conclusion and Future Work

Appendix A

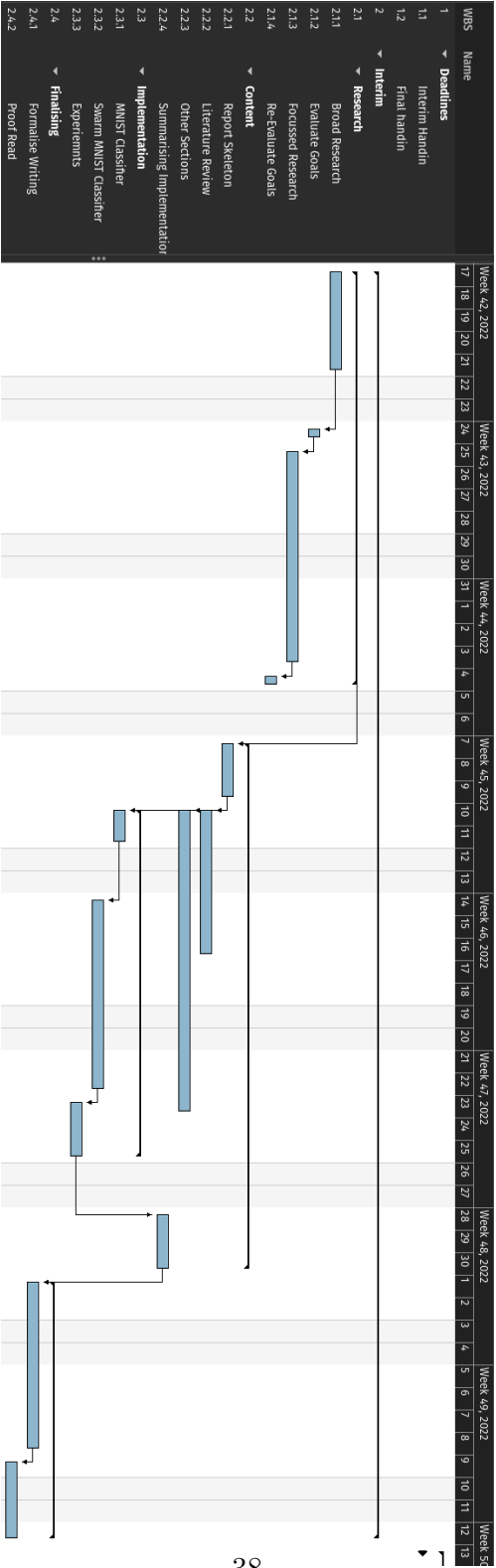
Machine Learning Model

```
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(128, activation="relu")(out)
out = Dense(10, activation="sigmoid")(out)
model = Model(inputs=inp, outputs=out)
model.compile(
    optimizer="adam",
    loss=SparseCategoricalCrossentropy(),
    metrics=[SparseCategoricalAccuracy()])
```

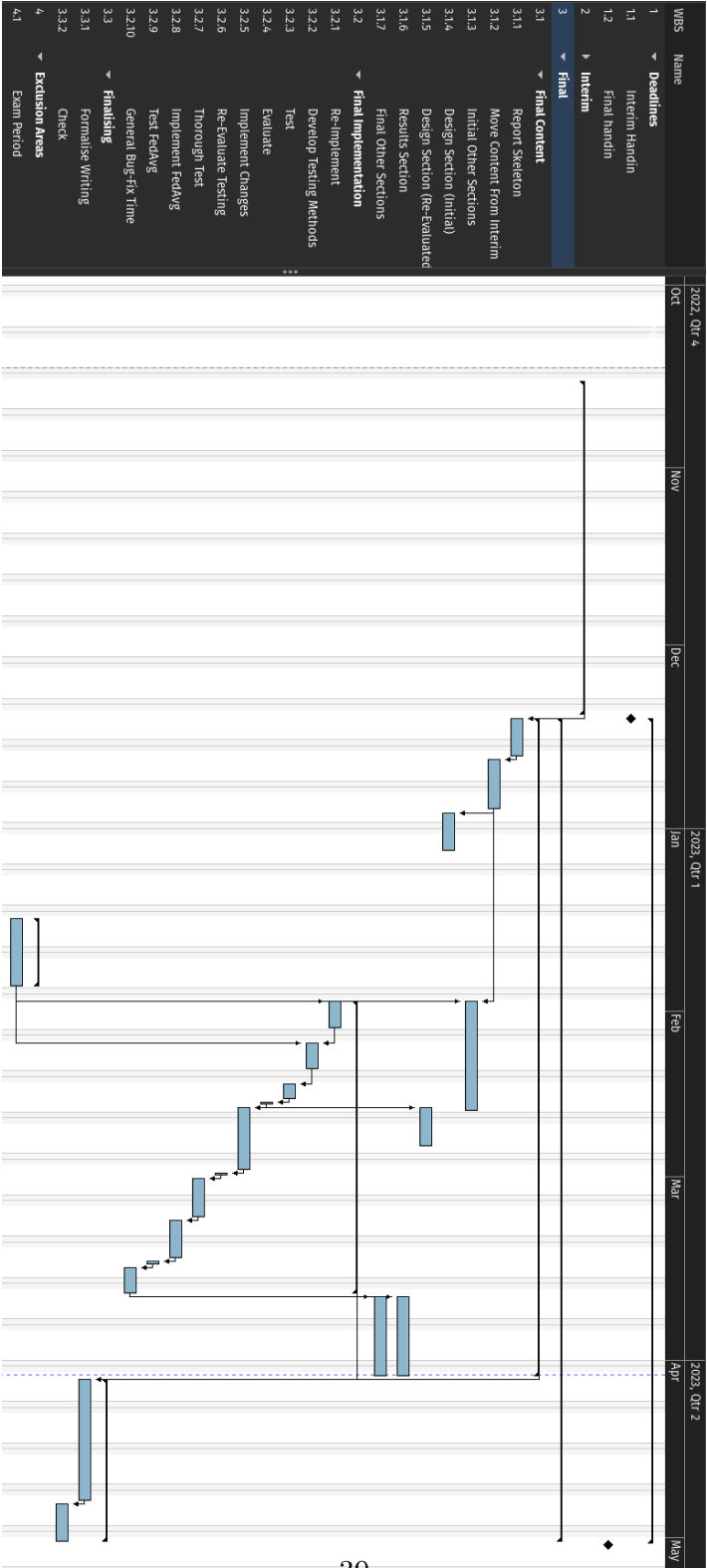
Appendix B

Gantt Charts

B.1 Gantt - Interim



B.2 Gantt - Final



Bibliography

- [1] M. Kop, “Machine learning & eu data sharing practices,” Stanford-Vienna Transatlantic Technology Law Forum, Transatlantic Antitrust ..., 2020.
- [2] 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.
- [3] 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.
- [4] 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.
- [5] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, “Communication-efficient learning of deep networks from decentralized data,” 2016.
- [6] 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.
- [7] S. Warnat-Herresthal, H. Schultze, K. L. Shastri, 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. Rybníček, 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. Sridhara, 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. Wiczorek, 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.
- [8] 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.
- [9] 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.
- [10] D. Yaga, P. Mell, N. Roby, and K. Scarfone, “Blockchain technology overview,” tech. rep., oct 2018.
- [11] D. Yang, C. Long, H. Xu, and S. Peng, “A review on scalability of blockchain,” in *Proceedings of the 2020 The 2nd International Con-*

ference on Blockchain Technology, ICBCT'20, (New York, NY, USA),
p. 1–6, Association for Computing Machinery, 2020.