Designing and Testing a Novel Idea Based on Federated Learning: Swarm Learning

Josh Pattman

March 6, 2023

Contents

Chapter 1

Problem and Goals

Chapter 2

Literature Review

Chapter 3

Project Management

Chapter 4

Analysis and Specification of the Solution to the Problem

Chapter 5

Detailed Design

5.1 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.

5.2 No Blockchain

- 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***

5.3 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.

5.4 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 $model_x$.

5.4.1 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 hyperparameters, 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.

5.4.2 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.

5.4.3 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 be behind the local training counter before it is ignored. Different values of β are explored in the results section.

An issue with training counter filtering pertains to the presence of runaway nodes. These nodes possess a substantially higher training counter relative 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.

5.5 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. To address this issue, this paper proposes several solutions.

5.5.1 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 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 training on the same problem with less data, potentially leading to divergence between groups and resulting in a decrease in overall performance.

5.5.2 Active Relay

The second proposed approach to address the sparsely connected networks is 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. It is also vital that a model update is only kept and relayed if it has a higher training counter than the cached update that the local node already stores. If this rule is not followed, infinite loops can occur in the network, where a model update is repeatedly sent to the same 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.

5.5.3 Partial Active Relay

A third method proposed to address sparsely connected networks is partial active relay, which strikes a balance between the previous two methods. This approach works similarly to active relay, but introduces a parameter δ. Upon receiving a model update, a node decides whether to relay the update, with the probability of relaying being δ. This parameter allows for customisation of the algorithm to prioritise network traffic or convergence speed, without committing exclusively to either option.

When using Partial Active Relay, it is necessary to incorporate training counter filtering. Unlike Active Relay or Passive Convergence, where a node always receives the most recent model update from its neighbours, Partial Active Relay introduces a chance that a model update may not be relayed to distant non-neighbouring nodes, even if an older model update was. As a result, the receiving node may combine an outdated model with its local model, causing slower training. However, training counter filtering ensures that old model updates are ignored, preventing this issue.

Chapter 6

Implementation

6.1 Dataset and Machine Learning Model

6.1.1 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.

6.1.2 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.

6.2 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.

6.2.1 Algorithm

6.2.2 Back-end Distributor

6.2.3 Evaluation

6.3 Prototype

6.3.1 Algorithm

6.3.2 Back-end Distributor

6.3.3 Evaluation

6.4 Final

6.4.1 Algorithm

6.4.2 Back-end Distributor

when using push on train rather than pull on sync, the latest diffs are more preserved

6.4.3 Evaluation

Chapter 7

Testing Strategy and Results

7.1 Methods

7.1.1 Data Collection

multiple runs, average of every node

7.1.2 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.

7.1.3 Node Counts

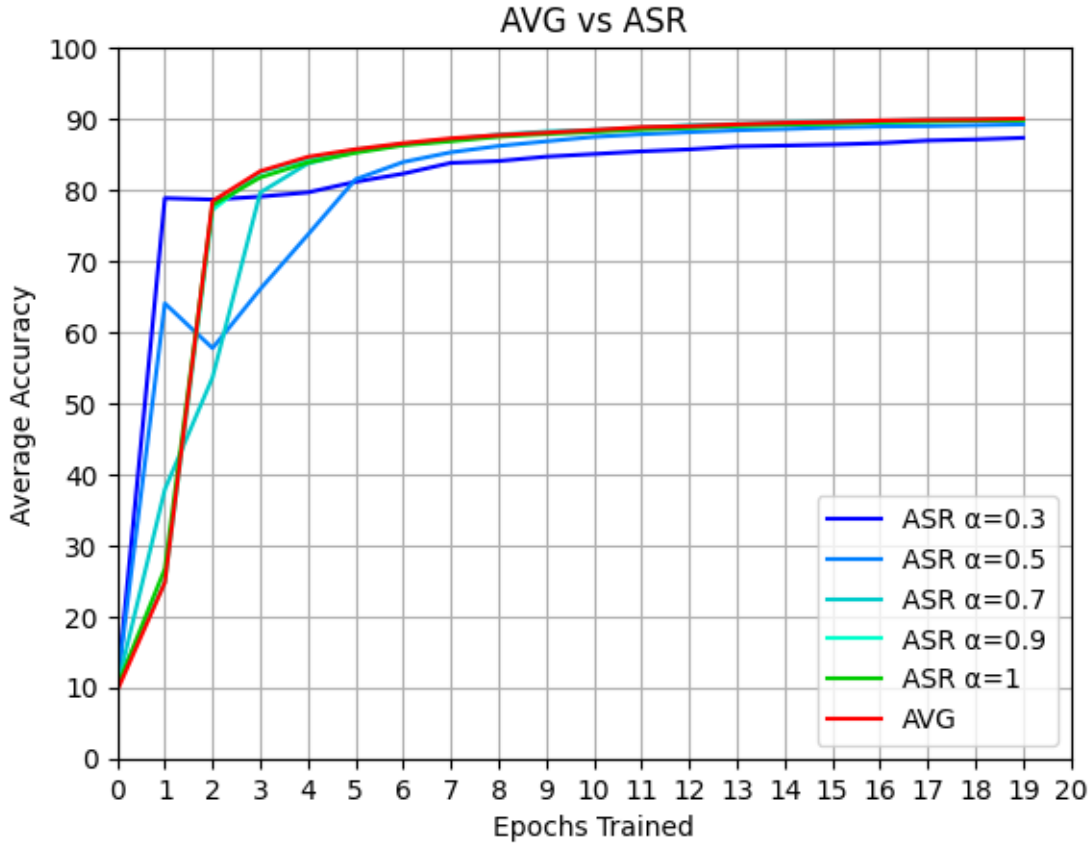
All experiment use 10 nodes unless otherwise stated. All experiments show the average accuracy of each 10 nodes x 5 experiments = 50 nodes

7.2 Experiments

7.2.1 Combination Methods: Averaging vs Averaging with Synchronisation Rate

The purpose of this experiment was to compare the effectiveness of *averaging* (AVG) to *averaging with synchronisation rate* (ASR). Below are some details of the experiment:

- All nodes were started at the same time
- When using ASR, the parameter α was varied
- The parameter β set to 0
- The parameter γ was set to 8
- The network is dense



The lower values of ASR increase performance faster at the start, but then struggle to achieve the same perf as the higher values/AVG after some time. Hypothesis: this is because nodes become de-synced, then the amount of change per epoch is greater than the change per sync, so networks stay at local minima.

AVG performs the same as ASR 0.9 because there are 10 nodes, therfor they are mathematically identical.

ASR 1 performs very well too. Hypothesis: As model updates are sent before the sync step, no training is being lost, as all training done in this step is being stored on neighbour nodes.

From now on, ASR 0.9 will be used. It provides the same performance as AVG when in a dense network, but hypothetically will provide more stability when used in a sparse network with varying numbers of neighbours.

7.2.2 Combination Methods: Varying β

The purpose of this experiment is to find the relationship between β and performance. Below are some details of the experiment:

- All nodes were started at the same time
- ASR 0.9 was used
- The parameter β varied
- The parameter γ was set to 8
- The network is dense

Chapter 8

Critical Evaluation

Chapter 9

Conclusion and Future Work

Chapter 10

References

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(),
```