Migrating Models: A Decentralized View on Federated Learning*

Péter Kiss[1][0000−0001−6941−2095] and Tomáš Horváth[1,2][0000−0002−9438−840X]

[1] Department of Data Science and Engineering
ELTE – Eötvös Loránd University, Faculty of Informatics
Budapest, H-1117 Budapest, Pázmány Péter sétány 1/C., Hungary
[2] Pavol Jozef Šafárik University, Faculty of Science, Institute of Computer Science
Jesenná 5, 040 01 Košice, Slovakia

Abstract. Federated learning (FL) researches attempt to alleviate the increasing difficulty of training machine learning models, when the training data is generated in a massively distributed way. The key idea behind these methods is moving the training to locations of data generation, and periodically collecting and redistributing the model updates. We present our approach for transforming the general training algorithm of FL into a peer-to-peer-like process. Our experiments on baseline image classification datasets show that omitting central coordination in FL is feasible.

Keywords: Federated Learning · Peer-to-Peer · Neural Networks

1 Introduction

The goal of supervised machine learning (ML) is to predict some missing values or attributes (labels) of data points given observed values (input features) using some model of the data distribution.

Given this model, the learning process usually corresponds to empirical risk minimization, that aims at finding its parameters $\mathbf{w} \in \mathbb{R}^d$, that are able to minimize a loss function l over the training data, which tells us how the learned model distribution differs from the data distribution.

Nowadays, without doubt, artificial neural networks (NNs) are the most popular ML models due to their applicability for a wide range of tasks and to the end-to-end nature of their learning process. In this paper we focus on federated training of this class of models (specifically for image classification). However, as we will indicate, the proposed approach can be applied, with more or less restrictions, to a broader family of ML models as well.

In case of a general loss surface, as that one of NNs, training happens using versions of mini-batch gradient descent, where the weights $\mathbf{w}$ are iteratively

* This work was partially supported by the project "Application Domain Specific Highly Reliable IT Solutions" financed by the National Research, Development and Innovation Fund of Hungary (TKP2020-NKA-06).

changed (moved) in the negative direction of gradients computed from the loss on a random subset (*mini-batch* $\mathcal{B}$) of training data:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \nabla l^{\mathcal{B}}(\mathbf{w}_t). \tag{1}$$

This training method requires huge amount of training data and computational resources. In fact, without the loss on generality, we can say that the more the data and training time the better the model becomes.

As the amount of data to be processed has been growing at a higher rate than the computation and storage power of machines (where the training takes place), parallelization of learning gained more and more importance. *Data parallel* distributed training methods of NNs [7, 23, 5] focused, at first place, on data center based learning and the exploitation of multi-core or multi-GPU architectures but later gave a framework for distributed learning in data centers.

Formally, the setup of distributed ML can be described as follows: given a set of nodes $\mathcal{V} = \{v^1, v^2, \ldots, v^K\}$ with $K = |\mathcal{V}|$ and n data points allocated into sets $\mathcal{D}^k$ of indices of data points stored at nodes v^k $(1 \leq k \leq K)$ with $n^k = |\mathcal{D}^k|$ being the number of data points at the node v^k. Without the loss of generality we usually assume that $\mathcal{D}^k \cap \mathcal{D}^l = \emptyset$ whenever $l \neq k$, thus $n = \sum_{k=1}^{K} n^k$. The task at hand, defining the local loss for node v^k as $l^k(\mathbf{w}) = \frac{1}{n^k} \sum_{i \in \mathcal{D}^k} l_i(\mathbf{w})$, is to solve the following optimization problem:

$$\min_{\mathbf{w} \in \mathbb{R}^d} l(\mathbf{w}) = \sum_{k=1}^{K} \frac{n^k}{n} l^k(\mathbf{w}) \tag{2}$$

where $l_i(\mathbf{w})$ denotes the loss on ith data point $(\mathbf{x}_i, y_i)$ given the parametrization $\mathbf{w}$, with $y_i \in \mathcal{C}$, where $\mathcal{C}$ stands for the range of the missing values, in our case the set of classes of images.

To solve the problem in Equation 2 for non convex losses, as that of in cases of NNs, the most widely used methods are different versions of distributed mini-batch gradient descent, where a coordinator collects and aggregates the gradients (Equation 1) coming from the k different nodes:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \frac{1}{K} \sum_{k=1}^{K} \nabla l^{\mathcal{B}^k}(\mathbf{w}_t). \tag{3}$$

After the aggregation is performed, the new model with parameters $\mathbf{w}_{t+1}$ will be redistributed to the worker nodes, and this loop continues until a reasonably good global performance is reached.

The idea of FL [19] was that we could exploit the computation power of the huge amount of user devices for the training, where the data is generated anyway. This way we can better protect users' privacy and, in the same time, save the data centers from the not negligible burden of storing and processing such huge amount of data.

The setup of FL, thus, differs from the traditional distributed mini-batch gradient descent (MBGD) based training in the following characteristics: (i)

the number of nodes can be much larger than the average number of training instances stored on a single node, (ii) the data on each node can be drawn from a different distribution, and, (iii) the number of data instances on different nodes may vary by orders of magnitude.

Since here the process is distributed over geographically widespread network, instead of a data center, the cost of the communication becomes a real problem. Thus, nodes execute multiple updates on their local models before they send their "new" models (or, equivalently, the differences to the old models) to the coordinator:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{n}{n^k} \sum_{k=1}^{K} \Delta^{(k)}, \text{ with } \Delta^{(k)} = \sum_{i=0}^{r} \nabla l^{\mathcal{B}^k_{t_i}}(w^k_{t_i}), \tag{4}$$

where $\mathbf{w^k_{t_{i+1}}} = \mathbf{w^k_{t_i}} - \eta \nabla l^{\mathcal{B}^k_{t_i}}(\mathbf{w}^k_{t_i})$, $\mathbf{w}^k_{t_0} = \mathbf{w}_t$ and $r = \beta \cdot n^k / |\mathcal{B}|$ the number of local updates, for an epoch number β, and batch size $|\mathcal{B}|$.

In Federated Averaging (FedAvg) [23], that is most probably the most widely used method of FL, the global model is distributed across only a random subset of nodes, each being chosen with probability γ. In [23] it has been empirically shown that with a value $\gamma = 0.1$ this method reaches, if not outperforms, the convergence rate of full aggregation, despite of the very significant communication and computation savings.

1.1 Related Work

Since, according to the problem statement of FL, the number of nodes K is extremely large, the management of these nodes even with the sub-setting that is introduced in FedAvg might be a challenging task. Consequently one of the most apparent practical issue of FedAvg is delays that stem from the centralized synchronous nature of the algorithm. Delays are introduced by *overcrowded channels* around the coordinator and by so-called *struggling nodes*, both leading to a slow-down in the training process. To solve this problem a variety of techniques have been proposed providing strategies to orchestrate updates, such that, applying *traditional scheduling techniques* [34]; using *federated client selection* [24] that prefers nodes with the best communication and computation capabilities; dynamically adapting of training scheme to available resources [31]; or allowing *asynchronous communication patterns* [6]. Another viable way for reducing communication burden seeks to decrease the size of data to be communicated through some kind of *quantization* [25, 27, 9, 1, 2]. Methods for compressing updates also include techniques that build on characteristics of NN training, such that pruning updates [10] through variational dropout [18] or evaluating the importance of parameter layers in NNs [6].

Assuming the convexity of the loss function, a range of innovative methods has been proposed, mostly built on the "communication-efficient distributed dual coordinate ascent" (CoCoA) framework using dual optimization ([16]), with the main goal to minimize the number of communication rounds during the learning.

A different perspective to deal with communication difficulties is to *decentralize the training* across multiple parameter servers as it has been already proposed in DistBelief [7]. Going further, for convex loss functions, a number of completely peer-to-peer gossip-based [8] asynchronous algorithms has been proposed such as [11], to mention only one example. Dual optimization has been used in peer-to-peer setup as well, for example the alternating direction method of multipliers [4, 32] in [28, 3].

A second important issue in FL is the degradation of performance, which might be caused by *weight divergence* [38, 36], resulting from averaging the updates. An interesting approach to tackle the problem is the so-called neuron matching [36], to mention only one work. Another cause of the dropped performance might be the fact, that updates applied on the common model of FL can be viewed as a MBGD with updates being computed over extremely large "mini batches", that can cause serious generalization gaps [22].

In general, higher number of participating nodes, bigger local batch sizes and stronger divergence in the local data distributions lead to degraded performance and, especially for more complex tasks and models, they often prevent the system to learn any interpretable consensus model.

1.2 Our Contribution

.

The two migrating model (MM) approaches we present in this paper can be derived from FedAvg in the following way: (i) splitting the coordinator into multiple smaller processes, that is, multiple "coordinators" collect the updates from a smaller amount of nodes (per coordinator) and (ii) instead of having a fixed node for the coordinator, the models to be trained are passed from node to node, each adding its update when it "owns" the model. Finally, (iii) to simulate the model averaging step, the updates are written in a buffer and their average will be applied on the model with a predefined frequency. (iv) We will also discuss a special case of the MM approach, in which we omit the buffer and apply the updates without delay.

The contributions of the presented MM approach are:

1. reduction of the complications of centralized FL through evening the communication burden over the whole network by simulating FedAvg in a peer-to-peer environment;
2. by reducing the number of updates used for model averaging (or excluding averaging in the special case) and, thus, using smaller effective batch sizes (at the price of more biased updates) it is possible to dramatically reduce the general communication and computation cost in exchange for, according to our empirical results, only a slightly worst training efficiency;
3. using a simple mechanics to involve those nodes in the training process, that promise the most performance gain, incentivating training on as diverse data as possible (closer to iid. wrt. whole distribution);

4. any node can initiate an optimization of new models, so the network optimizes various models in parallel, completely asynchronously.

For inference, the nodes can use some of the previously seen models, potentially in an ensemble fashion to obtain a kind of "global model", with competitive performance to FedAvg, at least according to our experiments. For performance evaluation at first place we wanted to compare our method to FedAvg over at least as many nodes, as in our methods, thus we used bagging ensembles, along with measurements for the performance of a single model.

In a real-world scenario, a *tracker* can be deployed to provide information about the network, the various models and their migration within the network.

2 Migrating Models (MM)

Let us assume that nodes $v^1, v^2, \ldots, v^{K'} \in V$ participating in the training are organized into a graph $G(V, E)$ and the models with parameters $\mathbf{w}^1, \ldots, \mathbf{w}^{K'}$ ($K' \leq K$) are travelling along the edges $e \in E$ of the graph.

The initial phase of the learning algorithm starts with random initialization of K' model parameter sets $\mathbf{w}^1, \ldots, \mathbf{w}^{K'}$ at a subset of nodes (in our simulation, K' is a hyper-parameter, in the real world it might change dynamically). These initial weights need not be aligned across the nodes. Different initializations may even help to explore a bigger portion of the parameter space. As long as the input and output dimensions are aligned across the system, any kind of models can be used in the presented MM approach, whose training is done by MBGD (i.e. not only NNs).

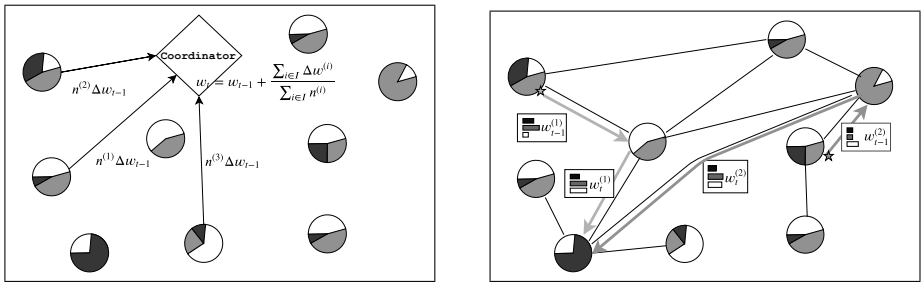

Fig. 1. Visualisation of FedAvg (left) and MM (right) approaches. Pie charts represent data distributions at nodes with colors corresponding to classes. FedAvg randomly picks some nodes, refines the common model on them, then adds the averaged update to it. In MM, a node initiates a model (indicated by stars referring to two different models), trains it on the local data and computes the belief vector about its performance (rectangle with confidence per class as the colored bars). In the following step the model will be moved to the next available node with the largest expected gain.

Updating the models, as visualized in the Figure 1, happens in a completely asynchronous manner: When a model has been trained on a partial set of data,

it looks for an appropriate next location to move, preferably a node, where the data is likely to help improving the model.

Choosing next location. To decide to which node should a model $\mathbf{w}^k$ be moved for further training, each model maintains a *belief vector* $\mathbf{p}^k \in \mathbb{R}^C$, that corresponds to a guess for its (per class) performance on the global data for the C classes.

All the entries of $\mathbf{p}^k$ start with 0 and, after training on a dataset, the belief vector is updated according to the Algorithm 1. The idea is to set believes to performance of the updated model on an i.i.d. test subset of local data, if the class is present, and discount the values of the absent ones, since we expect this performance to decrease. After obtaining the new belief vector, a new location for the model $\mathbf{w}^k$ is chosen, based on data sets residing at the neighboring nodes $N_G(v^k)$ from its current host node v^k. Selection of the next node to move the model to, is based on an expected performance gain, that is described in the Algorithm 2, where I is the indicator function returning 1 if its parameter is true. In this step the node broadcasts its belief vector to the neighbours, who will push up the expected performance on classes that they possess (we set it to 1 for simplicity), and send the sum of the entries back. The intuition is that the highest sum promises the most performance gain.

Algorithm 1 Updating the belief vector $\mathbf{p}^k$ of the model $\mathbf{w}^k$ based on F1-scores $\mathbf{t}^k$ on a test set of the given node k, with a discount rate ξ

procedure BELIEFUPDATE($\mathbf{p}^k = (p_1^k, p_2^k, \ldots, p_C^k), \mathbf{t^k} = (t_1^k, t_2^k, \ldots, t_C^k)$)
  $\mathbf{p^k} \leftarrow (1 - \xi)\mathbf{p^k}$
  for $c \leftarrow 1$ to C **do**
    $p_c^k \leftarrow \max\{p_c^k, t_c^k\}$
  return $\mathbf{p}^k$

Algorithm 2 Finding the best node to migrate the model to

procedure GETMAXBENEFITNODE($v^k, \mathbf{p}^k = (p_1^k, p_2^k, \ldots, p_C^k)$)
  $j^* \leftarrow \arg\max_{j|v^j \in N_G(v^k)} \underbrace{\sum_{c \in \mathcal{C}} \max\{I(\exists i \in \mathcal{D}^j(y_i = c)), p_c^k\}}_{\text{benefit at the node } v^j \text{ (computed at the node } v^j)}$
  return v^{j^*}

Update buffer. To simulate the aggregation step, for each model $\mathbf{w}^k$ we specify a buffer size σ^k that defines that, in a given "training round" r, how many nodes the model should visit before the aggregated (weighted averaged) update will be applied on its parameters $\mathbf{w}_{r-1}^k$. By updating the believes before each model

relocation, we hope to get an aggregated update more similar to one that would have been resulted from an i.i.d. training run (wrt. general distribution).

Algorithm 3 Migrating models – Experimental algorithm
K' – number of models
β, η – number of epochs and the learning rate at local training

procedure TRAINING(K', β, η)
  initialize graph $G(V, E)$
  $v^1, \ldots, v^{K'} \leftarrow$ Pick K' nodes randomly from G
  initialize belief vectors $\mathbf{p}^1, \ldots, \mathbf{p}^{K'} \leftarrow \mathbf{0}$
  initialize buffer sizes $\sigma^1, \ldots, \sigma^{K'} \leftarrow 1$
  initialize $\mathbf{w}_0^1, \ldots, \mathbf{w}_0^{K'}$ randomly
  $r \leftarrow 0$
  repeat
    for all $k \in \{1, \ldots, K'\}$ **do** in parallel
      $\delta \leftarrow 0$
      count $\leftarrow 0$
      $l_r^k \leftarrow$ accuracy$(\mathbf{w}_r^k, \mathcal{D}^k)$ ▷ Test accuracy at current node's test set
      $\sigma^k \leftarrow$ UpdateBufferSize(l_r^k, σ^k) ▷ Extend the buffer size if needed
      for $s \leftarrow 1$ to σ^k **do** ▷ Collecting the σ^k updates
        $\delta \leftarrow \delta +$ ClientUpdate$(k, \mathbf{w}_r^k, \beta, \eta) * n^k$
        count $\leftarrow$ count $+ |D^k|$
        $\mathbf{t}^k \leftarrow$ Test$(\mathcal{D}^k, \mathbf{w}_r^k)$ ▷ Test the F1-score of $\mathbf{w}$ on $\mathcal{D}^k$
        $\mathbf{p}^k \leftarrow$ BeliefUpdate$(\mathbf{p}^k, \mathbf{t}^k)$
        $v^k \leftarrow$ GetMaxBenefitNode$(v^k, \mathbf{p}^k)$
      $\mathbf{w}_{r+1}^k \leftarrow \frac{\delta}{\text{count}}$
    $r \leftarrow r + 1$
  until stop

Algorithm 4 Migrating models – Client update

procedure CLIENTUPDATE($k, \mathbf{w}, \beta, \eta$)
  $\mathbf{w}' \leftarrow \mathbf{w}$
  $B \leftarrow$ split$\mathcal{D}^k$ to set of batches
  for each local epoch i from 1 to β **do**
    for all $\mathcal{B} \in B$ **do**
      $\mathbf{w}' \leftarrow \mathbf{w}' - \eta \nabla^{\mathcal{B}} l(\mathbf{w})$
  return $\mathbf{w}'$

To keep the computation and communication costs low, we start from $\sigma^k = 1$ for all $k = 1, \ldots, K'$. Then, during the training, we monitor the improvement of the loss functions l^k and, based on some heuristics, we extend the buffer if we

experience that performance is not improving at the expected rate[3]. The MM method is described in the Algorithm 3 and 4.

Simple Migrating Models (sMM) As a specific case of MM, we also experimented with the simplest possible setup, denoted here as sMM, where $\sigma = 1$. In this case, at the price of highly biased updates, we can further reduce the communication and training costs of MM for a single update. Besides, there are two factors which are believed to decrease the performance of FedAVG such that (i) using large batch sizes and (ii) performing model averaging. In sMM, which for models with MBGD training is equivalent to a single node MBGD, both of these factors are excluded. It might be worth to note, that since we are not averaging over the updates of the nodes, sMM can be applied for different training methods, consequently different models as well (for example decision trees).

3 Experiments

Data sets and models. Experiments were run with three image classification task such that `MNIST`, `Fashion-MNIST` and `CIFAR-10`. The number of nodes for the two simplest cases, `MNIST` and `Fashion-MNIST`, was set to 200. For `CIFAR-10`, due to heavy computational load and the danger of stagnation in learning process, the number of nodes was set to 50. Similarly to the experimental settings in [23], 90% of each data set have been divided into equally sized one or two class chunks. 10% of this data, that has been assigned to each node, was assigned to a local test set (with the same distribution as in the local train set). The remaining 10% of the data was retained for evaluating the overall performance of the algorithms.

For the above mentioned three image classification tasks two types of NNs were utilized: For `MNIST` and `Fashion-MNIST` data sets we used a fully connected single hidden layer network (FCN) [4] For the `CIFAR-10` data set we used a convolutional neural network (CNN) [5]. Experiments with FedAvg and the proposed MM and sMM approaches were performed.

[3] A simple heuristic we have used here was to extend the buffer size if the exponential moving average of the model performance on new datasets (on the new nodes) shows no improvement after θ steps, where θ is a hyper-parameter.

[4] Following the Keras reference model for `MNIST` (not available anymore):
input: 784 dimension vector (=28×28) $\rightarrow$ dropout layer$\rightarrow$ dense with 128 units $\rightarrow$ sigmoid activation $\rightarrow$ dropout $\rightarrow$ dense with 10 units $\rightarrow$ softmax.

[5] Following the Keras reference model for `CIFAR-10`(not available anymore):
input: $32 \times 32 \times 3$ image$\rightarrow$
2d convolution with 32 3×3 filter and same padding and ReLU$\rightarrow$ 2d convolution with 32 3×3 filter and same padding and ReLU$\rightarrow$ 2×2 maxpooling $\rightarrow$ Droput$\rightarrow$ 2d convolution with 64 3×3 filter and same padding and ReLU$\rightarrow$ 2d convolution with 64 3×3 filter and same padding and ReLU $\rightarrow$ 2×2 maxpooling $\rightarrow$ Droput $\rightarrow$ dense with 512 units and ReLU $\rightarrow$ Dropout $\rightarrow$ dense with 10 units $\rightarrow$ softmax

Hyper-parameters , such that the size $|\mathcal{B}|$ of batches b, the number of epochs β and the learning rate η of the optimization process were tuned using grid search with the following values: $\beta \in \{1, 3, 5, 10\}$, $|\mathcal{B}| \in \{10, 32, 64\}$, $\eta \in \{1, 0.1, 0.01\}$ when we used FCNs and $\eta \in \{0.1, 0.001, 0.0001\}$ for the case of CNNs. Besides these, we run tests with different maximal buffer sizes (number of gradients to be collected) $\sigma \in \{1, 3, 5\}$ and also tested the general performance of bagging ensembles of a given number of models $K' \in \{1, 4\}$. Here, bagging refers to predictions resulting from averaging unnormalized per-class activations of a selected subset of the models $\mathbf{w}^1, \mathbf{w}^2, \ldots, \mathbf{w}^{K'}$, that are trained in our network. These hyper-parameters were chosen in a way, that the communication and computation costs should be upper bounded by that one of FedAvg. We defined the number of maximum collected node updates such that the number of communication rounds will be always upper bounded by that of FedAvg [6].

Dropout. Due to the strongly skewed nature of the local data sets, after each training round the sMM models tend to overfit on the local data. Thus, a strong regularization is necessary. Therefore, we applied dropout, as described in [13], with a high probability: 0.25, 0.25, 0.5 for the layers of the used CNN, and 0.5 for the single hidden layer of the used FCN. For the control experiments, we did not use any dropout since we found out that it has a bad impact on FedAvg. Thus, for the buffered learning we decreased the dropout rate π of the model $\mathbf{w}^k$ by a factor σ^k, i.e. $\pi^k = \frac{\pi}{\sigma^k}$.

Graph. We used a full graph topology, where for simulating a more realistic network, we randomly picked 5 nodes and chose the most promising relocation targets among those. For the discount rate at the belief update step we used $\xi = 0.05$.

3.1 Results

Results are summarized in Table 1 and visualized in Figures 2, 3 and 4. The data series of these figures show the average performance of the 10 best results for the four versions of federated learning, namely, (i) MM with ensembles (denoted as MM), (ii) the FedAvg baseline (FedAvg), (iii) sMM with ensembles (sMM), and, (iv) MM with only one single model (Single MM). On the left side of these figures the performance comparison in terms of accuracy is shown, the middle charts depict the communication costs (i.e. how many times the weights of the model had to be forwarded) while on the right side the corresponding computational costs of these algorithms are showed.

The computation costs has been calculated for a training round r by multiplying N_r, the number of nodes participating in the training in the round (that is $N_r = K'$, with K' being our initial choice in sMM, $N_r = \sum_{k=1}^{K'} \sigma_r^k$ in MM, while

[6] For the buffer size extension we set a threshold for improvement to $\theta = 15$, that is, if the accuracy did not improve in the last 15 relocations then we extend the number of models to aggregate.

$N_r = K' = \gamma K$ for FedAvg), and the number of epochs β. Thus, the accumulated computation costs at round r are computed by $\text{cost}_r^{\text{comp}} = \text{cost}_{r-1}^{\text{comp}} + N_r * \beta$ The communication cost values were calculated in a similar way, such that $\text{cost}_r^{\text{comm}} = \text{cost}_{r-1}^{\text{comm}} + \omega_r$, where ω is the number of transmissions of the model weights, that is necessary for updating all trained models. For sMM, $\omega_r = N_r$ and, for MM, $\omega_r = \sum_{k=1}^{K'} \sigma_r^k$. That equals to the number of participating nodes in a training round, because collecting the gradients from σ_r^k nodes means the same number of transmissions. The value of ω for FedAvg is, on the other hand, $2 * N_r$ since each active node has to first acquire the recent model and then to send back the updates. At these values, we calculated the number of transmissions necessary only for the training and did not include the costs of broadcasting the common models for inference to each node.

The performance of the algorithms is measured via accuracy, since for the MM and sMM methods averaging ensembles were used for making predictions.

Results have shown that the sMM algorithm is very simple and viable method to train NNs at a very low computational and communication cost. Its performance, however, stays under the performance of the FedAvg baseline, even with ensembles.

The MM approach closed this gap in a trade-off for increasing costs. We believe, however, that producing a similar performance to FedAvg with a decentralized algorithm, such that the proposed MM, is promising[7] [8].

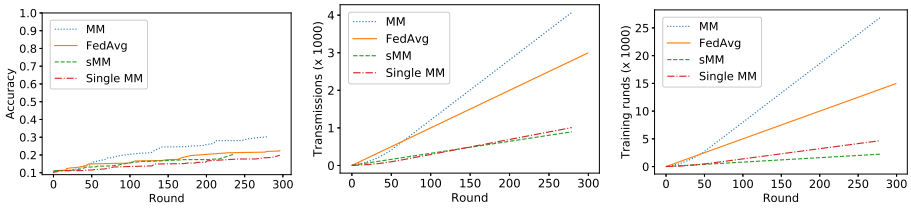

Fig. 2. Accuracy, Communication cost and Training cost on `CIFAR-10`

[7] The accuracy, along with the communication and computation costs, of ensemble of MMs exceeds that one of FedAvg. The reason for that is that, due to resource intensity of `CIFAR-10` training, the number of participating nodes have been reduced to 50, which means fewer models have been used in FedAvg, while the hyper-parameters (K' and maximum σ) of sMM and MM have been kept unchanged.

[8] Hyper-parameter search was not performed for the γ parameter of FedAvg, $\gamma = 1/10$ was used according to [23]. The main reason was our limited computation possibilities. However, since the settings for γ also affect sMM and MM (e.g. the buffer size or the ensemble count), it is possible that the gap between the communication and computation costs might be different in case of a large-scale hyper-parameter search.

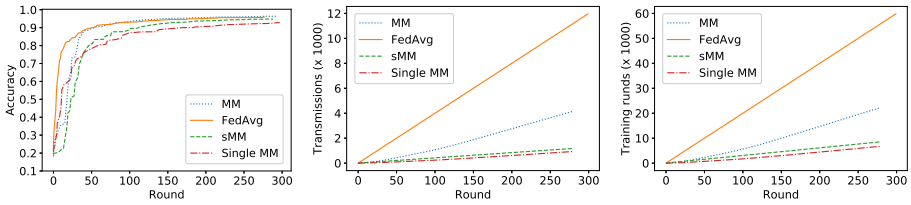

Fig. 3. Accuracy, Communication cost and Training cost on `MNIST`

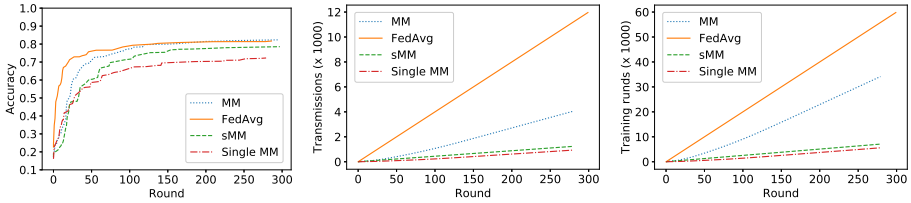

Fig. 4. Accuracy, Communication cost and Training cost on `Fashion-MNIST`

	Accuracy				Parameter transmission ($\times 10^4$)				Training epochs($\times 10^4$)			
	Fed Avg	MM	sMM	Single MM	Fed Avg	MM	sMM	Single MM	Fed Avg	MM	sMM	Single MM
C10	0.22	**0.30**	0.2	0.19	14.95	26.90	**2.24**	4.67	2.99	4.07	**0.89**	1.00
MN	0.95	**0.96**	0.94	0.92	59.8	22.27	8.54	**6.74**	11.96	4.14	1.17	**0.92**
F-M	0.81	**0.82**	0.78	0.72	59.8	34.14	7.08	**5.59**	11.96	4.03	1.23	**0.93**

Table 1. The averages of best 10 results in terms of accuracy of each methods, with the corresponding communication and computations costs. Bold fonts denote the best values i.e. best accuracy, lowest communication and computation cost.

4 Discussion

Besides the promising results of the experiments, a few issues have to be mentioned concerning the presented methods, though. These are the following:

Ensembles A disadvantage of the proposed methods is the increased resource demand of the ensembles on inference at the nodes. This could be alleviated trough distillation ([12]), however, at the price of additional computation load. Another question is, how a node acquires the necessary number of high quality models for inference. In our experiment, we trained the exact number of models that constitute the predictor and used each model in the tested ensemble.

Stagnation The potentially advantageous effect of omitting model averaging is probably balanced out with the loss of its regularization effect. Despite of the strong regularization, the local models overfit on the most recent data sets.

This leads, in a significant number of cases, to a stagnating ensemble accuracy. However, we experienced a similar phenomenon in the case of FedAvg as well. An analysis for the poor performance of FedAvg in strongly skewed data sets can be found for example in [15] or in [29].

Real world usage In our experiment, we have been working with a fully connected graph, that is naturally unrealistic in real world scenarios. The distribution of data over the nodes, along with the number of nodes, is pretty far from the characteristics given by [19]. However, using similar set-ups is a common practice as we have seen in the literature.

4.1 Convergence

The proposed models can be viewed as generalizations of FedAvg: For $K' = 1$ and a constant gradient buffer size $\sigma = \gamma K$, in the case of a fully connected graph, the MM method becomes equivalent to FedAvg. Moreover, also in the fully connected case, $\gamma = 1/K$ (in case of FedAvg) and $K' = 1$ and $\pi = 0$ (for MM), leads to the same method, apart from involvement of the parameter server at FedAvg. Finally, if we limit $\sigma = 1$ for MM, we get the sMM algorithm.

Uncertain convergence To carry out a thorough analysis of convergence of NN training in a FL setting involves many variables which make giving meaningful guarantees extremely hard. A lot of effort [31, 20, 21, 39, 26, 30, 33, 17, 35] have been carried out in this direction, however, due to the complexity of the problem, all of them make certain restrictions. Even for the case of convex optimization, there are assumptions such that iid data distribution across the nodes or all the devices being active (the latter is equivalent to FedSGD). The latter assumption was made in [17, 35, 31], while authors of [39, 26, 30, 33] build on both. [20] and [21] provides convergence analysis for true FedAvg with non-iid data, but for the case of a strongly convex optimization objective, not applicable for the case of NNs.

Convex case Following the reasoning of [21], for convex optimization, the convergence rate of FedAvg is $\mathcal{O}(\frac{1}{n})$, where n is the total number of data points which contribute to the optimization. According to their analysis, to achieve an accuracy of ϵ, the number of training round to execute is

$$\frac{n}{E} = \mathcal{O}\left[\frac{1}{\epsilon}\left(\left(1+\frac{1}{K'}\right)E\overline{||\nabla l||^2} + \frac{\sum_{k=1}^{K}{\gamma^k}^2\overline{Var(\nabla l^k)} + \Gamma + \overline{||\nabla l||^2}}{E}\right)\right] \quad (5)$$

where $E = \beta|\mathcal{B}|$ is the number of data points, from which the updates are computed between two communication rounds, γ^k is the probability of selecting the node k for the update round, $\Gamma = F^* - \sum_{k=1}^{K} p^k F^{k^*}$ quantifies "non-iid-ness" that is the difference of globally optimal loss and the local optima. $\overline{Var(\nabla l^k)}$ is a bound for variance, while $\overline{||\nabla l||^2}$ stands for squared norm of local stochastic gradients.

4.2 Privacy

An important challenge in FL, and distributed machine learning (ML) in general, is the question of privacy of potentially sensitive data. In our setup we see the following two major points for potential attacks:

Forward inference If we use the method of Algorithm 2, it is feasible to combinatorically infer classes of data held at the candidate nodes, granting an additional vulnerability to our method compared to FedAvg. However, applying a random noise over the indicator function, apart from some extreme situations, considerably decreases the vulnerability of the proposed approach.

Backward inference According to our best knowledge, as it is summarized in [14], attacks on privacy in distributed ML build on regular and frequent update messages following the same routes. Peer-to-peer nature of the training process adds an additional complexity to deal with, making it necessary to have access to all communication channels of the attacked node to achieve similar effectiveness of an attack. Gradient leakage attacks [37] can be executed having access to a single update vector, currently however they work only on single batch updates, or multiple batches with very few examples included. For both cases the tracker can be used to ensure legitimacy of routes, for example, to avoid building loops around the target or even to redraw the connection graph from time to time.

5 Conclusion

We presented our approach to alleviate the challenges imposed by the federated learning setup and, in general, distributed machine learning systems. The key idea of the proposed approach is that, instead of the transmission of model updates, the models themselves travel to the location of the data, evening the communication needs across the network.

With this (almost complete) decentralization of the learning process, the synchronization problems and straggler effect can be bypassed as well as communication burden at the parameter servers is not present anymore.

Our experiments have shown that similar performance can be achieved compared to the federated averaging baseline, however, with less communication and computational cost (at the price of using ensembles of small number of models at prediction phase). Based on the results of our experiments and the fact that the proposed approach is a generalization of the popular federated averaging approach, the presented work is worth further research.

References

1. Aji, A.F., Heafield, K.: Sparse communication for distributed gradient descent. arXiv preprint arXiv:1704.05021 (2017)

2. Alistarh, D., Grubic, D., Li, J., Tomioka, R., Vojnovic, M.: Qsgd: Communication-efficient sgd via gradient quantization and encoding. In: Advances in Neural Information Processing Systems. pp. 1709–1720 (2017)
3. Bellet, A., Guerraoui, R., Taziki, M., Tommasi, M.: Personalized and private peer-to-peer machine learning. arXiv preprint arXiv:1705.08435 (2017)
4. Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J., et al.: Distributed optimization and statistical learning via the alternating direction method of multipliers. Foundations and Trends® in Machine learning **3**(1), 1–122 (2011)
5. Chen, J., Pan, X., Monga, R., Bengio, S., Jozefowicz, R.: Revisiting distributed synchronous sgd. arXiv preprint arXiv:1604.00981 (2016)
6. Chen, Y., Sun, X., Jin, Y.: Communication-efficient federated deep learning with layerwise asynchronous model update and temporally weighted aggregation. IEEE Transactions on Neural Networks and Learning Systems (2019)
7. Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Senior, A., Tucker, P., Yang, K., Le, Q.V., et al.: Large scale distributed deep networks. In: Advances in neural information processing systems. pp. 1223–1231 (2012)
8. Dimakis, A.G., Kar, S., Moura, J.M., Rabbat, M.G., Scaglione, A.: Gossip algorithms for distributed signal processing. Proceedings of the IEEE **98**(11), 1847–1864 (2010)
9. Dryden, N., Moon, T., Jacobs, S.A., Van Essen, B.: Communication quantization for data-parallel training of deep neural networks. In: 2016 2nd Workshop on Machine Learning in HPC Environments (MLHPC). pp. 1–8. IEEE (2016)
10. Du, W., Zeng, X., Yan, M., Zhang, M.: Efficient federated learning via variational dropout (2018)
11. Hegedűs, I., Danner, G., Jelasity, M.: Gossip learning as a decentralized alternative to federated learning. In: IFIP International Conference on Distributed Applications and Interoperable Systems. pp. 74–90. Springer (2019)
12. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531 (2015)
13. Hinton, G.E., Srivastava, N., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.R.: Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580 (2012)
14. Hitaj, B., Ateniese, G., Perez-Cruz, F.: Deep models under the gan: information leakage from collaborative deep learning. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 603–618 (2017)
15. Hsu, T.M.H., Qi, H., Brown, M.: Measuring the effects of non-identical data distribution for federated visual classification. arXiv preprint arXiv:1909.06335 (2019)
16. Jaggi, M., Smith, V., Takác, M., Terhorst, J., Krishnan, S., Hofmann, T., Jordan, M.I.: Communication-efficient distributed dual coordinate ascent. In: Advances in neural information processing systems. pp. 3068–3076 (2014)
17. Khaled, A., Mishchenko, K., Richtárik, P.: First analysis of local gd on heterogeneous data (2019)
18. Kingma, D.P., Salimans, T., Welling, M.: Variational dropout and the local reparameterization trick. In: Advances in Neural Information Processing Systems. pp. 2575–2583 (2015)
19. Konečnỳ, J., McMahan, H.B., Ramage, D., Richtárik, P.: Federated optimization: Distributed machine learning for on-device intelligence. arXiv preprint arXiv:1610.02527 (2016)
20. Li, T., Sahu, A.K., Zaheer, M., Sanjabi, M., Talwalkar, A., Smith, V.: Federated optimization in heterogeneous networks (2018)

21. Li, X., Huang, K., Yang, W., Wang, S., Zhang, Z.: On the convergence of fedavg on non-iid data. arXiv preprint arXiv:1907.02189 (2019)
22. Masters, D., Luschi, C.: Revisiting small batch training for deep neural networks. arXiv preprint arXiv:1804.07612 (2018)
23. McMahan, H.B., Moore, E., Ramage, D., Hampson, S., et al.: Communication-efficient learning of deep networks from decentralized data. arXiv preprint arXiv:1602.05629 (2016)
24. Nishio, T., Yonetani, R.: Client selection for federated learning with heterogeneous resources in mobile edge. CoRR **abs/1804.08333** (2018)
25. Seide, F., Fu, H., Droppo, J., Li, G., Yu, D.: 1-bit stochastic gradient descent and application to data-parallel distributed training of speech dnns. In: Interspeech (2014)
26. Stich, S.U.: Local sgd converges fast and communicates little (2018)
27. Strom, N.: Scalable distributed dnn training using commodity gpu cloud computing. In: Sixteenth Annual Conference of the International Speech Communication Association (2015)
28. Vanhaesebrouck, P., Bellet, A., Tommasi, M.: Decentralized collaborative learning of personalized models over networks (2017)
29. Wang, H., Kaplan, Z., Niu, D., Li, B.: Optimizing federated learning on non-iid data with reinforcement learning. In: IEEE INFOCOM 2020-IEEE Conference on Computer Communications. pp. 1698–1707. IEEE (2020)
30. Wang, J., Joshi, G.: Cooperative sgd: A unified framework for the design and analysis of communication-efficient sgd algorithms (2018)
31. Wang, S., Tuor, T., Salonidis, T., Leung, K.K., Makaya, C., He, T., Chan, K.: Adaptive federated learning in resource constrained edge computing systems. In: IEEE INFOCOM 2018-IEEE Conference on Computer Communications (2018)
32. Wei, E., Ozdaglar, A.: On the o (1= k) convergence of asynchronous distributed alternating direction method of multipliers. In: 2013 IEEE Global Conference on Signal and Information Processing. pp. 551–554. IEEE (2013)
33. Woodworth, B., Wang, J., Smith, A., McMahan, B., Srebro, N.: Graph oracle models, lower bounds, and gaps for parallel stochastic optimization (2018)
34. Yang, H.H., Liu, Z., Quek, T.Q., Poor, H.V.: Scheduling policies for federated learning in wireless networks. IEEE Transactions on Communications (2019)
35. Yu, H., Yang, S., Zhu, S.: Parallel restarted sgd with faster convergence and less communication: Demystifying why model averaging works for deep learning. Proceedings of the AAAI Conference on Artificial Intelligence **33**, 5693–5700 (2019)
36. Yurochkin, M., Agarwal, M., Ghosh, S., Greenewald, K., Hoang, T.N., Khazaeni, Y.: Bayesian nonparametric federated learning of neural networks. arXiv preprint arXiv:1905.12022 (2019)
37. Zhao, B., Mopuri, K.R., Bilen, H.: idlg: Improved deep leakage from gradients. arXiv preprint arXiv:2001.02610 (2020)
38. Zhao, Y., Li, M., Lai, L., Suda, N., Civin, D., Chandra, V.: Federated learning with non-iid data. arXiv preprint arXiv:1806.00582 (2018)
39. Zhou, F., Cong, G.: On the convergence properties of a k-step averaging stochastic gradient descent algorithm for nonconvex optimization. Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (2018)