Loss-based D-Cliques Optimization

Karim Hadidane
karim.hadidane@epfl.ch
Supervised by: Erick Lavoie, erick.lavoie@epfl.ch

Scalable Computing Systems, SaCS
Ecole Polytechnique Fédérale de Lausanne, Switzerland

Abstract—Federated learning aims to collaboratively train a machine learning model between different edge devices or nodes while avoiding data privacy concerns about transferring data. The convergence speed of machine learning models trained in a Federated learning fashion is, however, significantly affected by data heterogeneity, in particular in a fully decentralized setting. We focus on D-Clique topology\cite{1}, where nodes are grouped in sparsely interconnected cliques such that the label distribution in a clique is representative of the global label distribution. Our work optimizes the construction of D-cliques in parallel with model training to preserve the convergence speed while avoiding any information leak on the label distribution inside cliques. We propose DC-SGD, an extension of D-SGD to optimize the cliques concurrently with learning. DC-SGD exchanges nodes’ model parameters in order to change the system topology and create cliques that are more representative of the global label distribution.

I. INTRODUCTION

In the recent years, machine learning has shifted to data-intensive workloads and its algorithms were involved in training models using huge data volumes. Those volumes are fragmented on many storage devices, forming distributed databases. Some problems deal naturally with data distributed on many storage nodes or user devices, such as learning an image classifier on user-taken photos. Such problems require high-quality data that live on edge devices such as smartphones. Due to privacy concerns and data regulation laws, raw data are not allowed to be exchanged between nodes.

Federated learning is a collaborative learning framework that leverage machine learning services, making them effective by exploiting the distributed databases without exchanging raw data. FL is a promising solution to enable many parties to jointly train a machine learning model while keeping their local data decentralized. It is based on exchanging model parameters between nodes. We focus here, on Federated learning in a decentralized fashion, where each node has the knowledge of only a subset of participating nodes, i.e. their neighbors.

One major challenge that FL is facing is high difference between data distributions across nodes. The data generated and/or located on the different participating nodes are usually non independently and identically distributed, non-IID. This is due to the fact that local data depends on usage and production patterns specific to each participant.

Non-IIDness can be seen in more than one form. The skew can be between features i.e comparing \( P_{D_i}(X) \) and \( P_{D_j}(X) \) between node \( i \) and \( j \), or between labels i.e \( P_{D_i}(Y) \), or between joint distributions i.e \( P_{D_i}(X,Y) \). We focus on our work on the label distribution skew. An important type of data heterogeneity is label distribution skew. A common example is classifying images from many nodes, where each node has only one class in their data.

Work from Hsieh et al. \cite{2} shows that skewed data labels are a fundamental and pervasive problem for decentralized learning, causing significant accuracy loss across many ML applications, and that the degree of data skew is a key determinant of the difficulty of the problem.

Several solutions have been proposed to deal with highly skewed non-IID data such as sharing local device data or creating server-side proxy data\cite{3}. However, those methods may not be realistic because exchanging local data violates the key privacy assumption of Federated Learning.

D-Clique topology, proposed in \cite{1}, considers the trade-off between the nodes communication and the data heterogeneity in the FL system, and was able to find a good compromise between the two. The idea behind the D-Clique approach is to create sub-networks of fully-connected nodes, called cliques, such that the label distribution inside a clique is close global distribution. Our work tackles the optimization of the cliques. In particular, we design a dynamic D-Clique optimization algorithm run concurrently with the training. Our algorithm uses the performance of the node models, summarized in its loss, to change nodes memberships.

Compared to other Clique construction algorithms described in \cite{1}, in particular Greedy Swap, our algorithm does not require revealing any information about label distributions inside the cliques. Also, it avoids the overhead of decoupling the cliques construction and the training of the models. Instead, it works on randomly constructed cliques and optimizes them concurrently in the training process. Furthermore, Greedy Swap requires sharing the global label distribution to all nodes prior to the cliques construction\cite{7}. Our algorithm does not require the prior knowledge of the global distribution at the node level.

Also, DC-SGD is purely dependent on the model performances. Therefore, D-Clique optimization can be further generalized to decentralized learning in an unsupervised setting. We empirically prove that when using the loss of the models as the proxy to the skew, we still can achieve an important gain in the skew and convergence speed.

The rest of this paper is organized as follows. We first introduce the notation that we use in Section 2. We then describe the problem setting and D-Cliques in section 3 and 4. Section 5 describes our dynamic clique optimization using models performances. We design experiments to empirically test...
the robustness of our approach in Section 6, 7 and 8. Finally, we review some related work in Section 5, and conclude with promising directions for future work in Section 6.

II. NOTATION AND DEFINITIONS

To make the article more understandable, we define in this section some of the concepts and keywords that we will be using.

- **Set of nodes**, \( N \)
- **Identifier of a node**, \( i \in N \)
- **Set of classes or labels**, \( L \)
- **Local data of node** \( i \), \( D_i \)
- **Global data**, \( D \), is the union of the local data of all participating nodes in \( N \), i.e \( D = \bigcup_{i \in N} D_i \)
- **Local label distribution** of node \( i \), described by the vector of probabilities \( P_i = [P_i;l \in L] \)
- **Local model of node** \( i \), \( \theta_i \)
- **Loss of the model of node** \( i \) on a data point \( s \), \( F(\theta_i; s) \)
- **A clique** \( C \), is a subset of the nodes \( N \), where all nodes in \( C \) are fully-connected i.e \( C \subseteq N \mid \forall i, j \in C, (i, j) \) is an edge
- **A set of distributed cliques** \( DC \), is a set of all cliques, such that all cliques are disjoint

III. PROBLEM SETTING

Our goal is to train a global model that optimizes a global training loss. Formally, we state, in the following, our assumptions and objective.\(^1\)

**Assumptions:** We consider a system of \( n \) nodes, each denoted by an integer \( i \in \{1, ..., n\} \), trying to solve in a collaborative fashion a classification task with \( L \) classes. We denote a data point in the global dataset by a tuple \((x, y)\), where \( x \) represents the feature vector and \( y \), its class or label i.e \( y \in \{1, ..., L\} \). Note that we are assuming a horizontal federated learning setting.\(^4\) This means the feature vectors \( x \) across all nodes live in the same feature space. Each node \( i \) has access to a local dataset, denoted by \( D_i \), which may differ across nodes.

**Global Objective:** Our goal is to minimize the average training loss:

\[
\min_{\theta} \frac{1}{mn} \sum_{i=1}^{n} \sum_{j=1}^{m} F(\theta_i; s_j) \tag{1}
\]

where \( m \) denotes the number of examples in each client node. Note that \(^1\) is the decentralized approximation to the population risk function:

\[
\mathbb{E}_{s \sim D} F(\theta; s) \tag{2}
\]

However, the loss \(^1\) is not an observable measure in the decentralized setting, as no central entity that has knowledge of all the system exists.

**Training algorithm: D-SGD:** To optimize the objective \(^1\) we use Decentralized Stochastic Gradient Descent or D-SGD algorithm. Intuitively, for each node, D-SGD samples a mini-batch from the local data, computes the gradient of the loss function with respect to the node’s local model parameters. Then, new model parameters are computed by taking a weighted step in the opposite direction of the gradient. The model is then updated by performing a weighted average of the local model with those of its neighbors. This weighted average is defined by a mixing matrix \( W \), where \( W_{ij} \) is the weight of the outgoing edge from node \( i \) to node \( j \) and \( W_{ij} = 0 \) for \( \{i, j\} \notin E \). \( W \) must be doubly stochastic i.e \( \sum_{j \in N} W_{ij} = 1 \) and \( \sum_{j \in N} W_{ji} = 1 \) and symmetric i.e \( W_{ji} = W_{ij} \).

Algorithm \(^5\) shows D-SGD steps run on a node.

### Algorithm 1 D-SGD, Node \( i \)

**Require:** nodes’ initial models \( \theta_i^{(0)} \), Learning rate \( \gamma \), Edge weights \( W_{ij} \), Mini-batch size \( m \), Number of steps \( K \)

1: for \( k = 1, ..., K \) do
2: \( S_i^{(k)} \leftarrow \) mini-batch of \( m \) samples from \( D_i \)
3: \( \theta_i^{(k)} \leftarrow \theta_i^{(k-1)} - \nabla F(\theta_i^{(k-1)}; s_i^{(k)}) \)
4: \( \theta_i^{(k)} \leftarrow \sum_{j \in N} W_{ji}^{(k)} \theta_j^{(k-1)} \)
5: end for

IV. D-CLIQUE

Recently, D-Clique\(^1\), a new topology for Decentralized Federated Learning was proposed. D-Clique aims to account for the data heterogeneity across client nodes by creating clusters of nodes, called *cliques* such that the nodes in a clique are in constant communication i.e all nodes in a clique are neighbors. The property that we are trying to achieve is that the label distribution inside a clique is close to the global label distribution.

A clique in a D-Clique can be seen as a proxy node, with a close distribution to the global distribution as this is not achievable at the node level.

The D-clique paper discusses also several methods to construct cliques. Formally, the resulting graph should have the following properties: (1) the overall topology is sparse i.e the number of edges \( |E| << |N|^2 \) and (2) the distribution of all cliques is close to the global label distribution.

In practical implementations, the construction of clique\(^4\) is done in the following steps:

1) Select the number of cliques and their node membership
2) Assign edges between all nodes inside a clique
3) Assign inter-clique edges
4) Compute and assign weights to edges

Note that steps 1, 2 and 4 are subject to design choices. First, the node assignment to cliques can be done according to a predefined criterion or randomly. Second, the inter-clique edges can be made sparse (e.g ring topology) or dense i.e fully-connected. For the choice of edge weights, the D-Clique paper uses the Metropolis-Hasting weights\(^1\).

One effective way to construct cliques is Greedy Swap.\(^1\) Greedy Swap perform node swaps between cliques by comparing label distribution inside cliques with the global distribution. More on how Greedy Swap can be found in Section VII-C.

However, in Greedy Swap, clique distributions need to be exchanged. This is revealing to the labels represented inside cliques, and with small cliques, this can reveal information about individual nodes.

\(^1\)We refer for the Cliques construction to the report written by Paulette Vazquez and Alexandra Senderovich. The performances of different clique construction algorithms, both centralized and decentralized, are compared
We propose a dynamic approach to construct the cliques. In particular, we construct the cliques in parallel with the training using the losses of nodes’ models. As we will show in Section VIII, this is equivalent to directly optimize the label skew.

A. Intuition

The steps done in our algorithm are similar to the ones described by Greedy Swap i.e at each step, sample two cliques, find a pair of nodes to exchange and exchange them. Greedy Swap uses the skew metric to evaluate a potential swap. We use instead the model loss.

Our work comes from the intuition that a client node with a highly skewed data with respect to the global distribution, would train a model e.g a classifier that would perform particularly poorly when presented to a client node with a different highly skewed data. This is because the trained model is biased towards over-represented classes and hence, overfits to its local distribution and fails to generalize when used for inference on data that comes from a different distribution.[6]

A clique would choose a node that performs poorly on the clique data, because this means the node were trained using a different distribution than the clique distribution.

B. DC-SGD: Pairing training with D-cliques construction

We propose DC-SGD, an extension to D-SGD. Our approach consists in optimizing the cliques, randomly constructed prior to training, concurrently with the training.

The algorithm works as follows:

1) sampling a mini batch from the local data
2) Update the local model parameters by taking a step in the gradient of the loss computed on the batch from step 1
3) Sample two cliques $C_1$ and $C_2$ at random
4) Iterate over all pairs of nodes $(i,j) \in C_1 \times C_2$
5) For each node in the pair, compute the gain of loss that the other clique would have if the the pair is swapped
6) Perform the swap with the best gain
7) Update the local model by taking a weighted average of the neighbors’ models (after swap)

Algorithm 2 is the pseudo-code of our approach. We call it DC-SGD. Note that we use a procedure Cliquify(.) that takes the set of cliques and computes the new mixing weights matrix $W$. Also, Line 14 uses the threshold $T$. This is to avoid performing insignificant swaps. In our evaluations, we are using $T = 0$.

An illustration of the inter-clique communication that should be performed in DC-SGD is shown in Figure 1.

Our first implementation of the algorithm performs node swaps after the model averaging step i.e updating the local model with the average of neighboring models. The results of the experiments have shown poor performance in decreasing the clique skew. The problem is illustrated in Figure 2. In fact, the models diverge from each other after the first update i.e local gradient step. However, averaging the local models with the neighbor nodes’ models brings the models close together. Therefore, by performing model averaging, we lose the information about the local data.

VI. EXPERIMENTAL SETUP

A. Datasets

We experiment with 3 datasets: MNIST handwritten database [7], CIFAR [8] and FashionMNIST. The three of them have 10 classes.
B. Models

We train and assess three models depending on the dataset used, and therefore the problem that we are trying to optimize. For MNIST, we use a simple multi-class logistic regression. For fashionMNIST, we use a Multi-layer perceptron. For Cifar10, we use a variation of LeNet[9] based on Group Normalization[10], called Group-Normalized LeNet or GN-LeNet.

These models are shown to be performant on these datasets. We train and test the three models on our datasets. The test accuracies are shown in Figure 3.

C. Data partition

We partitioned the data in the way described in Communication-Efficient Learning of Deep Networks from Decentralized Data [11] paper by Google. To partition the data, we first sort the data by class, divided it into shards of equal size and assign shards to client nodes. A shard is the basic unit of data in our data partition scheme. Therefore, a shard is homogeneous i.e has examples of only one class. In contrast with Google’s paper, we assign one shard to each node. Therefore, we are assuming a high level of data heterogeneity between client nodes.

VII. Baseline Algorithms

To evaluate the performance of our algorithm to achieve a minimal skew and an optimal convergence speed, we choose to use fully-connected and random clique constructions as benchmarks.

A. Random clique construction

Our optimization is expected to have a better performance than changing the clique topology at random at each training step. We expect this to have a high variance and high skew since the construction of the cliques is not optimized over a specified metric.

B. fully-connected topology

The optimal solution to the problem that minimizes the skew and has faster convergence speed is the one-clique solution, where all nodes are put in the same unique clique. This is equivalent to a fully connected setting with all nodes have knowledge of all other nodes. However, this implies a big overhead of communication as nodes need to send their models to all other nodes in the network. Note that this is the ideal baseline representing the fastest convergence speed achievable if topology had no impact. In our experiments, we will compare, when possible, our models convergence speeds to this optimal setting.

C. Greedy Swap

Greedy Swap is a greedy algorithm that optimizes constructed cliques. It works over random cliques. It receives the cliques constructed by random and performs a fixed number of iterations that depend on the data distribution. For this, the algorithm minimizes the label skew in the cliques. We define in the following the skew of a clique C

**Definition 1.** The skew of a clique C with label distribution \( p_{C,l} \), for \( l \in \{1, ..., L\} \) is given by \( \text{skew}(C) = \sum_l | p_{C,l} - p_l | \)

Greedy Swap works in iterative fashion. For each iteration

1) Randomly sample two cliques
2) Compute the skew of the two cliques, the sum of these skews represent the baseline score
3) Take one node from each selected clique, swap them, calculate the difference between skew from the updated cliques and the baseline from Step 1
4) Take a pair of node such that swapping them improves the skew

The D-clique paper [1] has proved that Greedy Swap improves cliques constructed randomly at an affordable cost. In particular, Greedy Swap is efficient in constructing cliques with low skew. Also, Cliques built with Greedy Swap converge faster than random cliques.

One problem with Greedy Swap is that, in exchanging the nodes between the two cliques, a clique needs to communicate information about the skew of its data. This can be revealing to the distribution of the clique, and with small cliques, this can reveal information about individual nodes.
VIII. Evaluation

The problem we are trying to optimize has multiple parameters that need to be taken into account. In our experimental evaluation, we try to fix some of these parameters and use them as environment parameters, while changing the remaining parameters to evaluate their effect.

One of the things that we need to account for, in a decentralized setting, is the number of messages that need to be exchanged. With the graph formulation of our problem, this is equivalent to the number of edges in the network graph.

A. Skew convergence

Our first intuition is that using the client nodes’ models’ performances to optimize the clique construction is equivalent to using the skew metric. In this section, we evaluate our approach by assessing the skew convergence in the clique construction. We compare the robustness of our algorithm to Greedy Swap.

1) Balanced data, perfect cliques:

We start by defining a perfect clique

**Definition 2.** A clique $C$ with label distribution $p_{C,l}$, for $l \in \{1, ..., L\}$ is called perfect if $\text{skew}(C) = \sum_l |p_{C,l} - p_l| = 0$

In a setting of perfectly balanced dataset i.e equal label distribution, and cliques of size equal to the number of classes, perfect cliques are theoretically achieved.

We choose to test this approach starting from simple settings to more complex ones. We simulate the setting with the following properties:

- Clique size equal to number of classes $|L|$
- number of nodes $|N|$ is divisible by the number of cliques
- perfectly balanced data i.e same number of examples per class, or equivalently $\rho = 1$

In general, DC-SGD successfully optimize the cliques randomly constructed through swaps. The results of our experiments are shown in the figures 4, 5 and 6. In almost all the settings, DC-SGD achieves the zero skew scenario. In this case, either no swaps are further done or only swaps between nodes with same labels are performed.

One thing to notice though that DC-SGD performance depends on the underlying model. As we can already see in the simple scenarios, the MLP classifier for FashionMNIST achieved the perfect clique, whereas the simple Linear model for MNIST could not converge in Figure 6a.

The results for running DCSGD with Cifar10 with 100 nodes and 10 cliques are shown in Figure 7.

2) Unbalanced data, perfect cliques:

Our previous experiments with balanced data and an appropriate clique size achieved perfect cliques. However, we can achieve perfect cliques with unbalanced data. This proves that our algorithm is not just trying to create diversity inside a clique.

For the purpose of comparing experimental results, we formulate a metric that we use to quantify the balance of data. The ratio $\rho$ in Eq. 3 will be used to indicate the maximum between-class imbalance level.

$$\rho = \frac{\max_i |C_i|}{\min_i |C_i|}$$

Figure 4: Skew convergence in balanced data, 3 classes, 12 nodes and cliques of size 3. Perfect cliques are achieved

Figure 5: Skew convergence in balanced data with 5 classes, 20 nodes and cliques of size 5. Perfect cliques are achieved

Figure 6: Skew convergence in balanced data, 3 classes, 12 nodes and cliques of size 3. Perfect cliques are achieved

Figure 7: Skew convergence in balanced data with 5 classes, 20 nodes and cliques of size 5. Perfect cliques are achieved
Figure 6: Skew convergence in balanced data with 6 classes, 30 nodes and cliques of size 6. We achieve perfect cliques for FashionMNIST. The skew fails to converge in the first 150 steps for MNIST.

Figure 7: Cifar10: skew convergence using DC-SGD and Greedy Swap in balanced data with 100 heterogeneous nodes and cliques of size 10.

$C_i$ is a set of examples in class $i$, and $max_i |C_i|$ and $min_i |C_i|$ return the maximum and minimum class size over all $i$ classes, respectively. A larger value of $\rho$ indicates a large label skew or data imbalance.

We show in Figure 8 the test accuracy of training the MLP classifier for Fashion-MNIST on an unbalanced data. The performance is affected by the data balance. As our algorithm depends on models’ performance, it is convenient to test the robustness of DC-SGD.

We simulate the setting with the following properties:

- number of nodes $|N|$ is divisible by the number of cliques
- one over-represented class in the data, with different degrees of balance $\rho > 1$
- all under-represented classes are balanced among each other, i.e all of them have fixed number $M$ of examples
- Clique size equal to $\sum_i M_i/M$, $M_i$ denotes the number of examples in class $i$

Note that the last property insures that inside a clique, we can perfectly mimic the global distribution. This means that, if in the global data, we have twice as much data in class $l_1$ than other classes, then we need to have two nodes inside the clique representing $l_1$. The results of our experiments are shown in Figure 10.

3) heterogeneous client nodes: Until now, we are assuming complete homogeneity in the nodes level i.e a node has only examples from one class.

In the following, we relax this assumption and we evaluate scenarios where nodes can hold up 2 different labels. We assign 2 shards to each node randomly, which allows a node to be heterogeneous.

Note that in this case, we can not ensure that perfect cliques is theoretically achievable, as this depends on the data assigned to each node.

The results of simple settings with these assumptions are shown in Figure 11.

B. Training speed convergence

As the results of Greedy Swap shows good training convergence, we experiment the effect of DC-SGD clique construction on the training process. We use as a metric the test accuracy per epoch.

We simulate the following experiments using a system of 100 nodes, 10 classes and cliques of size 10. The results for MNIST, FashionMNIST and Cifar10 are shown in Figure 12, 13 and 15.

For MNIST, The convergence speed of DC-SGD is better than random cliques but Greedy Swap and fully-connected converges faster.

For FashionMNIST, the DC-SGD outperforms Greedy Swap and fully-connected but performs worse than random cliques. The test accuracy convergence is, however, not equivalent to the behavior of the validation accuracy.
IX. CONCLUSION AND FUTURE WORK

Our empirical study showed that DC-SGD optimization method is similar but not completely equivalent to skew optimization. In simple settings i.e few nodes and few classes, DC-SGD decreases the skew similarly to Greedy Swap. With large networks and cliques, we noticed a divergent aspect of DC-SGD in some experiments, in particular with MNIST (cliques optimized with DC-SGD are not better than random cliques). On the other hand, in large scale i.e 100 nodes and 10 cliques, DC-SGD skew convergence is similar to Greedy Swap for Cifar10 dataset. This implies that DC-SGD is probably dependent on the loss function form and the hyperparameters used. The reasons of these different behaviors are still to be investigated. Also, a proper choice of the threshold $T$ should be studied.

Comparing the training convergence, DC-SGD is converging slower than Greedy Swap but, in most cases, faster than random cliques. For MNIST dataset, although, in this case, the cliques are not optimized skew-wise i.e skew diverges, the convergence speed outperforms random cliques. Therefore, the effect of decreasing the cliques skew on the convergence speed should be tested statistically to show its significance. On the other hand, with Cifar10 dataset, on the first 100 epochs of the training,
Random cliques seems to be converging faster. This is a counter-intuitive result and should be investigated further.

Our work is an empirical investigation of the effect of optimizing using models’ losses on the clique skews. However, future work can be more theoretically focused on the properties of the loss function that enable such use.

X. LIMITATIONS

By design, cliques with single nodes cannot be fed to DC-SGD. This is not a huge restriction because single-node cliques cannot be optimized and, in practice, they should be avoided in the D-Clique setting.

One further limitation to DC-SGD is the time complexity of loss computations. In fact, each step of DC-SGD takes time proportional to the number of model parameters. For more complex models like GN-LeNet, with dense computation in the forward pass, the method does not scale. To show this, we computed statistics about the time needed by a step of clique optimization in both DC-SGD and Greedy Swap for the sake of comparison. The table shows the average time needed.

Also, the exchange of models between cliques is not completely privacy-preserving. In some simple parametric models like linear models, the local private data are more likely to be leaked or reverse engineered. This is because the communicated gradients of the parameters of linear models are proportional to the local input data. Therefore, additional privacy-enhancing methods like homomorphic encryption (HE)
are required to be used[14].

<table>
<thead>
<tr>
<th>Experiment settings</th>
<th>DC-SGD</th>
<th>Greedy Swap</th>
</tr>
</thead>
<tbody>
<tr>
<td>12N, C-3</td>
<td>593.182</td>
<td>0.329</td>
</tr>
<tr>
<td>20N, C-5</td>
<td>3261</td>
<td>0.996</td>
</tr>
<tr>
<td>30N, C-6</td>
<td>6001</td>
<td>1.620</td>
</tr>
<tr>
<td>100N, C-10</td>
<td>15013</td>
<td>5.759</td>
</tr>
</tbody>
</table>

Table I: Average time taken by a step of best-swap search of DC-SGD and Greedy Swap, measured in milliseconds (ms). The experiment settings in given in the format xN, C-y, where x denotes the number of nodes and y the clique size. The average is taken on 10 runs using FashionMNIST dataset.

ACKNOWLEDGEMENTS

I would like to thank Dr Erick Lavoie for his continuous guidance throughout the project.

REFERENCES


