Personalized Decentralized Multi-Task Learning Over Dynamic Communication Graphs

Matin Mortaheb Sennur Ulukus
Department of Electrical and Computer Engineering
University of Maryland, College Park, MD 20742
mortaheb@umd.edu *ulukus@umd.edu*

***Abstract*—Decentralized and federated learning algorithms face data heterogeneity as one of the biggest challenges, especially when users want to learn a specific task. Even when personalized headers are used concatenated to a shared network (PF-MTL), aggregating all the networks with a decentralized algorithm can result in performance degradation as a result of heterogeneity in the data. Our algorithm uses exchanged gradients to calculate the correlations among tasks automatically, and dynamically adjusts the communication graph to connect mutually beneficial tasks and isolate those that may negatively impact each other. This algorithm improves the learning performance and leads to faster convergence compared to the case where all clients are connected to each other regardless of their correlations. We conduct experiments on a synthetic Gaussian dataset and a large-scale celebrity attributes (CelebA) dataset. The experiment with the synthetic data illustrates that our proposed method is capable of detecting tasks that are positively and negatively correlated. Moreover, the results of the experiments with CelebA demonstrate that the proposed method may produce significantly faster training results than fully-connected networks.**

I. Introduction

Decentralized learning (DL) algorithms are able to operate over arbitrary network topologies, in which participants communicate only with their immediate neighbors without a need for communication with a central server. A key challenge in DL is to deal with data heterogeneity: as each agent has its own data, local datasets typically exhibit different distributions. This is especially true for multi-task learning (MTL), where tasks are distributed across different users.

Data heterogeneity is addressed through personalization in federated learning (FL), in which the parameter server and clients train a common base model, and each client additionally trains a small header for its own specific task; shown in Fig. 1 for the DL setting. By using personalization, users can obtain essentially different learning models that are better fitted to their unique data while capturing the common knowledge distilled from other devices' data [1]–[3].

Prior to the consideration of heterogeneity as a factor in DL convergence, communication topologies have been entirely characterized by their spectral gaps [4]. The choice of topology, however, has a large impact on heterogeneous settings as observed in [5]–[7]. In the presence of data heterogeneity, choosing a good topology for DL is important. On one hand, clients (tasks) may have adverse effects on each other if the topology of the graph is chosen inappropriately. On the other hand, having a fully connected network can result in a high

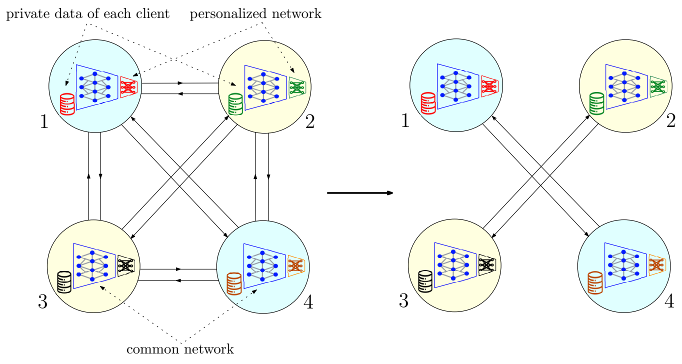

Fig. 1. Dynamic communication graph in personalized decentralized learning framework with a common network (shown in blue) and small personalized headers (shown in red, green, black, orange). Users with the same-color circles have positive correlation, and different colors mean negative correlation.

communication cost. Using a time-varying and data-aware design of the communication network, [8], [9] investigate how decentralized SGD performance can be improved in the presence of data heterogeneity. They propose an algorithm to adapt the connectivity matrix (network topology) by minimizing the relative heterogeneity in each round. However, solving an optimization problem that finds the optimal topology in each round results in high computational complexity.

In order to increase user performance, the topology should be configured to connect similar tasks and to isolate unrelated ones. To identify related tasks, we use a technique called *transference* [10]. In MTL [11], transference is a metric to quantify the positive/negative effect of a task's gradient update to the parameters of a shared encoder on another task's loss during training. In other words, transference metric of i to j, Z_{ij}, measures the loss of task j before and after applying the gradient update of task i on the shared network. A positive value of Z_{ij} indicates that the update on the shared parameters (by task i) results in a smaller loss on task j than the original parameter values. In contrast, a negative value of transference indicates the negative impact of tasks on each other.

To efficiently design the communication graph, we calculate an approximate transference metric using the gradient updates exchanged among the neighbors. Based on the approximated transference, we dynamically adjust the connectivity matrix W so that the positively correlated tasks are connected together and the negatively correlated tasks are disconnected from each other, thereby, preventing performance degradation. We generate the connectivity matrix by applying the spectral clustering technique [12] to the approximate transference matrix. Our experiments demonstrate that our framework is capable of achieving better performance faster than a fully-connected network in which all users are connected to each other.

The main contributions of our paper are as follows: 1) We propose a novel algorithm, which dynamically changes the network topology of the users according to their positive/negative correlations. 2) We conduct several experiments on a synthetic Gaussian dataset and a large-scale celebrity attributes (CelebA) dataset [13]. By deliberately designing the covariance matrix for the synthetic Gaussian datasets, we show that the framework can detect and cluster correlated tasks correctly. In addition, the experiments with CelebA dataset exhibit faster convergence and improved performance in our framework compared to the fully-connected network.

II. System Model and Problem Formulation

We consider a personalized distributed learning (PDL) setting [1], [3], with N clients which are connected via a directed graph. The network of clients aim to solve the following optimization problem associated with an ML problem,

$$\min_{\{\theta_{s,i}\}_{i=1}^N,\{\theta_i\}_{i=1}^N} \left\{ F(\{\theta_{s,i}\},\{\theta_i\}) \triangleq \frac{1}{N}\sum_{i=1}^N \mathcal{L}_i(\mathcal{X}_i^t,\theta_{s,i}^t,\theta_i^t) \right\} \tag{1}$$

where $\mathcal{X}_i^t$ is the ith user's data at time t, and $\mathcal{L}_i(\mathcal{X}_i^t,\theta_{s,i}^t,\theta_i^t)$ is the task-specific loss obtained by using $\mathcal{X}_i^t$ as an input over the shared network and private head of task i. Each client contains a shared network and a private network. The shared network parameters are represented by $\theta_{s,i}^t$ and the private head parameters are represented by θ_i^t, for user i at time t.

The matrix W represents the graph topology among the nodes: $W_{ij} > 0$ means that agent i can communicate with agent j, while $W_{ij} = 0$ means that agent i cannot communicate with agent j. The matrix W is a mixing matrix, which is composed of values between 0 and 1, and is doubly stochastic, i.e., $1^T W = 1$ and $W1 = 1$, where 1 is a vector of all ones. Mixing property of the topology matrix W is needed to ensure convergence of an iterative algorithm to solve problem (1).

Similar to the centralized case, the problem in (1) can be solved through an alternating minimization approach in the decentralized case, using decentralized SGD. First, client i performs K_1 local gradient based updates to optimize θ_i^t, while the global network parameters at client i, i.e., $\theta_{s,i}^t$, are frozen,

$$\theta_i^{t+1} = \theta_i^t - \eta\nabla_{\theta_i}\mathcal{L}_i(\mathcal{X}_i^t,\theta_{s,i}^t,\theta_i^t) \tag{2}$$

Then, by keeping the parameters corresponding to the client-specific head frozen, each node sends the gradient update corresponding to the shared network to its neighbors based on the network topology at time t and aggregates the received updates again based on the network topology specified by the connectivity matrix W, K_2 times as follows,

$$\theta_{s,i}^{t+1} = \sum_{j=1}^N W_{ij}^t \left(\theta_{s,j}^t - \eta\nabla_{\theta_{s,j}}\mathcal{L}_j(\mathcal{X}_j^t,\theta_{s,j}^t,\theta_j^t)\right) \tag{3}$$

Now, the objective is to automatically capture the correlations among the users and to dynamically modify the mixing matrix W so that the positively correlated tasks are connected and uncorrelated tasks are disconnected to prevent negative transference. To that end, let us define the quantity $\theta_{s,j|i}^{t+1}$ to represent the updated jth shared parameters after a gradient step with respect to loss of task i,

$$\theta_{s,j|i}^{t+1} = \theta_{s,j}^t - \eta\nabla_{\theta_{s,j}}\mathcal{L}_i(\mathcal{X}^t,\theta_{s,j}^t,\theta_i^t) \tag{4}$$

where X^t is a shared data that each client has for calculating the transference value. As a result, the transference of task i on a single task j at time t can be calculated as,

$$Z_{i\to j}^t = 1 - \frac{\mathcal{L}_j(\mathcal{X}^t,\theta_{s,j|i}^{t+1},\theta_j^{t+1})}{\mathcal{L}_j(\mathcal{X}^t,\theta_{s,j}^t,\theta_j^t)} \tag{5}$$

A positive value for $Z_{i\to j}^t$ indicates that using task i loss values to update the shared parameters would result in a lower loss for task j. Thus, a more positive value for $Z_{i\to j}^t$ means more correlation among these two tasks. Conversely, having a negative value means that learning task i and j simultaneously would deteriorate the performance of both of the tasks.

Instead of calculating $\theta_{s,j|i}^{t+1}$ for each user at each time to obtain transference, we can use its first order Taylor series expansion to obtain transference directly from the exchanged gradients among the clients. Let us define the gradient update for the shared network of user i at time t as $g_{s,i}^t = \nabla_{\theta_{s,i}}\mathcal{L}_i(\mathcal{X}^t,\theta_{s,i}^t,\theta_i^t)$. Then, the first order Taylor series expansion for the transference can be written as,

$$\begin{aligned} Z_{i\to j}^t &= 1 - \frac{\mathcal{L}_j(\mathcal{X}^t,\theta_{s,j|i}^{t+1},\theta_j^{t+1})}{\mathcal{L}_j(\mathcal{X}^t,\theta_{s,j}^t,\theta_j^t)} && (6)\\ &\approx 1 - \frac{\mathcal{L}_j(\mathcal{X}^t,\theta_{s,j}^t,\theta_j^t) - \langle g_{s,j}^t, g_{s,i}^t\rangle}{\mathcal{L}_j(\mathcal{X}^t,\theta_{s,j}^t,\theta_j^t)} && (7)\\ &= \frac{\langle g_{s,j}^t, g_{s,i}^t\rangle}{\mathcal{L}_j(\mathcal{X}^t,\theta_{s,j}^t,\theta_j^t)} && (8) \end{aligned}$$

Therefore, instead of calculating $\theta_{s,j|i}^{t+1}$ in each round to obtain $Z_{i\to j}^t$ value, users can exchange their gradient updates obtained via the shared data X^t to more efficiently compute $Z_{i\to j}^t$. After calculating $Z^t \in \mathbb{R}^{N\times N}$ matrix, we must convert it to a doubly stochastic mixing matrix W^t to ensure the convergence of the algorithm. We convert Z^t to a doubly stochastic matrix by clustering the Z^t matrix.

We use the *spectral clustering* technique [12] to perform clustering over a communication graph expressed by matrix Z^t. In spectral clustering, we use the eigenvalues of the graph Laplacian to find the appropriate clusters. To calculate the graph Laplacian, let us define matrix D as the degree matrix, which is a diagonal matrix where the (i,i)th entry indicates the degree of node i (the number of edges connected to node i).

Then, the graph Laplacian can be calculated as $L = D - Z$. In the graph Laplacian matrix, diagonal entries are the degrees of the nodes, and off-diagonal entries are the negative edge weights. Finally, we calculate the eigenvalues of the graph Laplacian matrix L.

By sorting the eigenvalues, we see that the number of 0 eigenvalues corresponds to the number of connected components in the graph. Also, an eigenvalue with a small value, e.g., eig$(L) \leq 1$, indicates that there is almost a separation of the two components. Therefore, we can determine the number of clusters by calculating the number of graph Laplacian eigenvalues which have values less than 1. The vectors associated with those eigenvalues contain information as to how to segment the network. Finally, we perform k-means on those vectors in order to obtain the labels for the nodes. The spectral clustering algorithm is given in Algorithm 1.

Algorithm 1 Unnormalized spectral clustering [12]

```
1: **Input:** Similarity matrix $Z$, number $k$ of clusters to construct.
2: Compute the unnormalized Laplacian $L$.
3: Compute the first $k$ eigenvectors $u_1, \dots, u_k$ of $L$.
4: Let $U \in \mathcal{R}^{n \times k}$ be the matrix containing the vectors $u_1, \dots, u_k$ as columns.
5: For $i = 1, \dots, n$, let $y_i \in \mathbb{R}^k$ be the vector corresponding to the $i$th row of $U$.
6: Cluster the points $\{y_i\}_{i=1,\dots,n}$ in $\mathbb{R}^k$ with the $k$-means algorithm into clusters $C_1, \dots, C_k$.
7: **Output:** Clusters $A_1, \dots, A_k$ with $A_i = \{j | y_j \in C_i\}$.
```

Let D_l be the number of nodes in cluster A_l for $l \in 1, \dots, k$. Then, W^t can be calculated as,

$$W^t(i,j) = \begin{cases} \frac{1}{d_l}, & \text{if } i, j \in A_l \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

This method ensures that W^t remains doubly-stochastic. To prevent error due to the tolerance of calculating transference, the Z^t matrix is averaged over each H epochs and then W^t is calculated by using spectral clustering.

Finally, in each training step, we perform a decentralized SGD step to exchange and update the shared network across users connected via an edge in the derived topology at time t, W^t. The overall algorithm is summarized in Algorithm 2.

III. Convergence Analysis

Our convergence analysis follows [14], [15]. We assume that each worker's objective function $f_i : \mathbb{R}^{d+d_i} \to \mathbb{R}$ for all i is L-smooth and μ-strongly convex and that the variance on each worker is bounded. We also assume that the connectivity matrix (mixing matrix) is doubly-stochastic. Therefore, the convergence analysis follows from [14, Thm. 4]. Further, [15] uses much weaker assumptions to prove convergence.

Algorithm 2 Training with our proposed algorithm

```
1: **Input:** step sizes $\eta$, initialization $\theta_s$, $\{\theta_i | i \in N\}$, $H$, $K_2$.
2: **for** t = 1, ..., $T$ **do**
3:     **for each** $i \in N$ (in parallel) **do**
4:         Compute task-loss $\mathcal{L}_i(\mathcal{X}_i^t, \theta_{s,i}^t, \theta_i^t)$
5:         $\theta_i^{t+1} \leftarrow \theta_i^t - \eta \nabla_{\theta_i} \mathcal{L}_i(\mathcal{X}_i^t, \theta_{s,i}^t, \theta_i^t)$
6:     **end for**
7:     **for each** $i, j \in N$ (in parallel) **do**
8:         Calculating $Z_{i \to j}^t$ using (6)
9:     **end for**
10:    Calculating mixing matrix $V^t$ from $Z^t$ using (9)
11:    $W_{temp}^t = W_{temp}^t + V^t$
12:    **if** $t = H$ **then**
13:        $W^t = W_{temp}^t / H$
14:        $W_{temp} = 0$
15:    **end if**
16:    **for** $k$= 1, ..., $K_2$ **do**
17:        **for each** $i \in N$ (in parallel) **do**
18:            $\theta_{s,i}^{t+\frac{1}{2}} \leftarrow \theta_{s,i}^t - \eta \nabla_{\theta_s} \mathcal{L}_i(\mathcal{X}_i^t, \theta_{s,i}^t, \theta_i^t)$
19:            $\theta_{s,i}^{t+1} \leftarrow \sum_{j=1}^N W_{ij}^t \theta_{s,j}^{t+\frac{1}{2}}$
20:        **end for**
21:    **end for**
22: **end for**
```

IV. Experimental Results

We compare the task losses achieved by naive fully connected network and our proposed dynamic communication graph algorithm.

A. Dataset Specifications

We use the following two datasets for our experiments:

Synthetic Gaussian vector dataset that contains 30,000 training data, 10,000 test data, and 10,000 shared data. Each data point in this dataset is a Gaussian vector of size 10, i.e., $x_i \in \mathbb{R}^{10}$, with mean of $\mu \in \mathbb{R}^{10}$ and covariance of $\Sigma \in \mathbb{R}^{10 \times 10}$. Same as Fig. 1, the covariance matrix is purposefully designed in such a way that the first and fourth coordinates are positively correlated, and they are negatively correlated with the second and third coordinates. Conversely, the second and third coordinates have a positive correlation, while they both have negative correlations with the first and forth tasks. We consider four attributes for each sample in our experiments. The first four coordinates with purposefully assigned correlations are selected as those four attributes. For each sample, the attributes are 1 if the corresponding coordinate is greater than the assigned mean and 0 otherwise. This synthetic dataset is designed to test the effectiveness of our proposed algorithm to connect the tasks who have positive correlation among themselves and disconnect those who have negative correlations. The training data points are distributed among clients, but each client has access only to one of the attributes (coordinates).

CelebA dataset [13] that contains 200,000 images, where each image contains 40 attributes. In our experiment, we

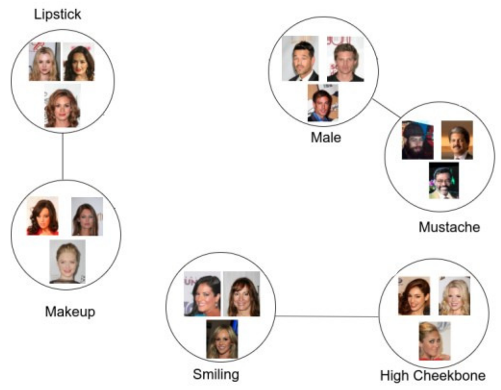

Fig. 2. An expected communication graph (correlation) among the chosen attributes from CelebA dataset.

only use 6 of the attributes, namely, male, mustache, high-cheekbones, smiling, heavy-makeup, and wearing lipsticks. Therefore, the dataset is divided into 6 parts, and each user has access to a single attribute of the given images. The chosen attributes and the expected correlation among attributes is shown in Fig. 2.

B. Hyperparameters and Model Specifications

We use Adam optimizer with learning rate $\eta = 2 \times 10^{-5}$ to train the shared and personalized networks for both datasets. In the synthetic Gaussian dataset, we use a shared encoder as explained in Table I, and each client also has a simple linear layer as a personalized network that maps the shared network's output to the corresponding prediction value. We use cross-entropy as a loss function for the classification task.

SHARED NETWORK
FC(10, 64)
FC(64, 128)
FC(128, 256)
FC(256, 512)
FC(512, 256)
FC(256, 128)

TABLE I
SHARED NETWORK MODEL.

In CelebA dataset, we use resnet-18 as a shared network for all the users, and a simple 2-layer network for each user to map the output of the shared network to the corresponding classification task. In both experiments, we perform $K_2 = 2$ decentralized-SGD protocols in each epoch. Also, we average the transference metric through 5 epochs before modifying the connectivity matrix, i.e., $H = 5$.

C. Results and Analysis

We begin with the synthetic dataset. As shown in Fig. 3, by capturing the true correlation among users with the transference method at epoch 15, the connectivity matrix clusters users properly based on their correlation, which results in improved performance as compared to the case where all users are connected regardless of their correlation to the other tasks. For tasks 2-4, our proposed method converges approximately at epoch 30, while the naive fully-connected method converges at epoch 60. A faster convergence is more evident in task 1 where our method converges after 45 epochs, while a naive fully-connected network does not converge satisfactorily even after 80 epochs.

The heatmap of the dynamic topology is shown in Fig. 4. According to our proposed method, the connectivity matrix initially starts with a fully-connected network, and the topology changes every five epochs based on the calculated transference. In epoch 15, the topology finally captures the correct transference as purposefully designed as in Fig. 1. Ultimately, the weights are uniformly distributed among tasks 1, 4 and tasks 2, 3 that have positive correlations between them.

Next we examine the proposed method on a larger and more realistic dataset, CelebA. As shown in Fig. 5, the results demonstrate the superiority of our method compared to a naive fully-connected network. As part of this experiment, we stopped changing the connectivity matrix at epoch 21 to observe the effect of changing the topology on the initial training phase compared with the whole training process. Our results indicate that the training speed increases when the topology is not modified after sufficiently capturing the correlation of tasks during the initial training phase. Therefore, changing the topology until the end of the training may result in drifting from the optimal performance and slower convergence. However, in the majority of our experiments, our proposed method converged faster in both early stopping and permanent topology change cases.

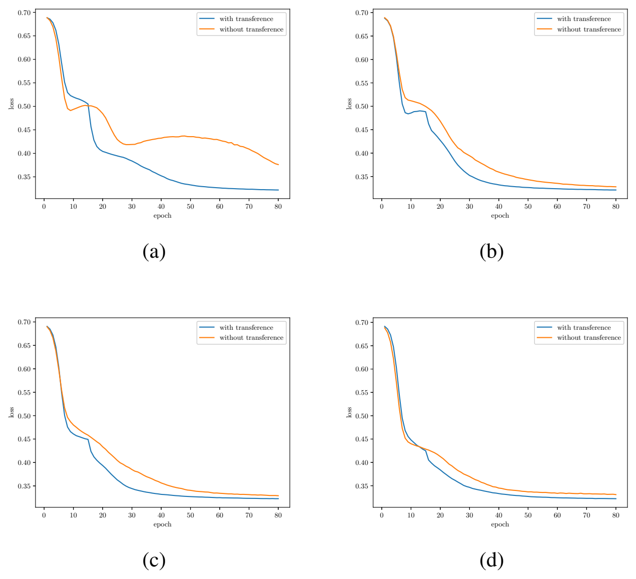

Fig. 3. Comparison of task losses achieved via dynamic communication graph using transference and naive fully-connected case for synthetic dataset.

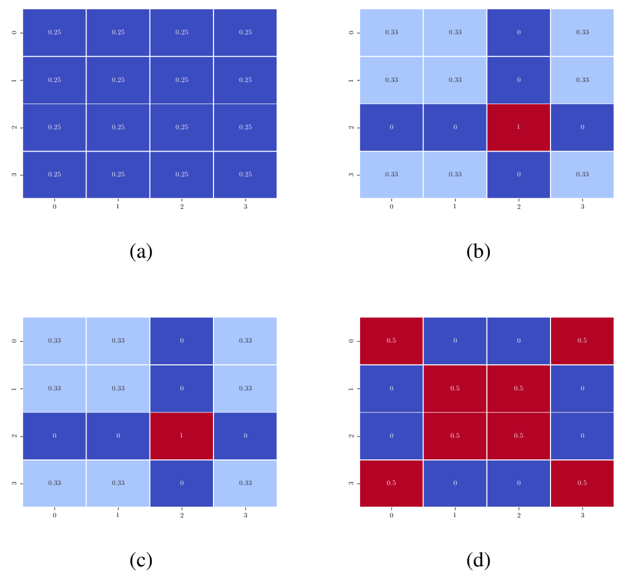

Fig. 4. Heatmap of the connectivity matrix achieved by the transference method over different epochs, (a) epoch 1, (b) epoch 6, (c) epoch 11, (d) epoch 16.

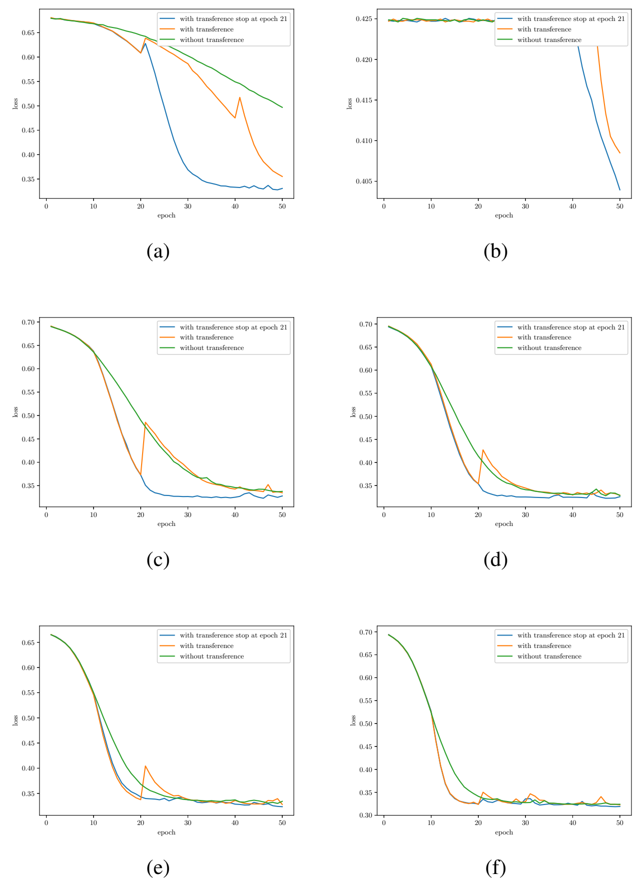

Fig. 5. Comparison of task losses achieved via dynamic communication graph using transference and naive fully-connected case for celebA dataset.

As shown in Fig. 5, as a result of our proposed method, the loss of task 1 drops with a higher slope after epoch 20, while further topology change can reduce the slope of dropping losses due to the sudden changes that may occur in the topology of the graph. Hence, our method with early stopping topology change converges at approximately epoch 40, while the transference method with a permanent topology change converges at epoch 50, and the naive fully-connected network does not converge satisfactorily even at epoch 50. The second task (learning mustache) converges quickly even from the very first epochs since it is a simple task to learn. Nevertheless, after around epoch 40, when the other correlated task converges appropriately, the loss decreases due to the changed communication topology. As a result of our proposed method, we have observed that task 3-6 have converged faster at around epoch 10, while the losses for the fully-connected tasks have started to decline much more slowly. The permanent modification of the topology matrix, however, may result in sudden divergence from the convergence and may delay the pace of convergence.

References

[1] M. G. Arivazhagan, V. Aggarwal, A. K. Singh, and S. Choudhary. Federated learning with personalization layers. Available online at arXiv:1912.00818.

[2] L. Collins, H. Hassani, A. Mokhtari, and S. Shakkottai. Exploiting shared representations for personalized federated learning. In *ICML*, July 2021.

[3] M. Mortaheb, C. Vahapoglu, and S. Ulukus. FedGradNorm: Personalized federated gradient-normalized multi-task learning. In *IEEE SPAWC*, July 2022.

[4] J. Wang, A. K. Sahu, Z. Yang, G. Joshi, and S. Kar. Matcha: Speeding up decentralized sgd via matching decomposition sampling. In *Sixth Indian Control Conference*, December 2019.

[5] A. Bellet, A. M. Kermarrec, and E. Lavoie. D-cliques: Compensating noniidness in decentralized federated learning with topology. Available online at arXiv:2104.07365.

[6] G. Neglia, C. Xu, D. Towsley, and G. Calbi. Decentralized gradient methods: does topology matter? In *AISTATS*, August 2020.

[7] T. Vogels, H. Hendrikx, and M. Jaggi. Beyond spectral gap: The role of the topology in decentralized learning. Available online at arXiv:2206.03093.

[8] B. Le Bars, A. Bellet, M. Tommasi, E. Lavoie, and A. M. Kermarrec. Refined convergence and topology learning for decentralized optimization with heterogeneous data. Available online at arXiv:2204.04452.

[9] Y. Dandi, A. Koloskova, M. Jaggi, and S. U. Stich. Data-heterogeneity-aware mixing for decentralized learning. Available online at arXiv:2204.06477.

[10] C. Fifty, E. Amid, Z. Zhao, T. Yu, R. Anil, and C. Finn. Efficiently identifying task groupings for multi-task learning. In *NeurIPS*, December 2021.

[11] Y. Zhang and Q. Yang. A survey on multi-task learning. Available online at arXiv:1707.08114.

[12] U. V. Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, August 2007.

[13] Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In *ICCV*, December 2015.

[14] A. Koloskova, S. Stich, and M. Jaggi. Decentralized stochastic optimization and gossip algorithms with compressed communication. In *ICML*, June 2019.

[15] A. Koloskova, N. Loizou, S. Boreiri, M. Jaggi, and S. Stich. A unified theory of decentralized sgd with changing topology and local updates. In *ICML*, July 2020.