MULTI-TIER FEDERATED LEARNING FOR VERTICALLY PARTITIONED DATA

Anirban Das, Stacy Patterson

Department of Computer Science
Rensselaer Polytechnic Institute, Troy, New York, USA

ABSTRACT

We consider decentralized model training in tiered communication networks. Our network model consists of a set of silos, each holding a vertical partition of the data. Each silo contains a hub and a set of clients, with the silo's vertical data shard partitioned horizontally across its clients. We propose Tiered Decentralized Coordinate Descent (TDCD), a communication-efficient decentralized training algorithm for such two-tiered networks. To reduce communication overhead, the clients in each silo perform multiple local gradient steps before sharing updates with their hub. Each hub adjusts its coordinates by averaging its workers' updates, and then hubs exchange intermediate updates with one another. We present a theoretical analysis of our algorithm and show the dependence of the convergence rate on the number of vertical partitions, the number of local updates, and the number of clients in each hub. We further validate our approach empirically via simulation-based experiments using a variety of datasets and both convex and non-convex objectives.

***Index Terms*—** vertical machine learning, coordinate descent, federated learning, stochastic gradient descent

1. INTRODUCTION

In recent times, we have seen an exponential increase of data produced at the edge of the communication networks. In many settings, it is infeasible to transfer the entire dataset to a centralized cloud for downstream analysis, either due to practical constraints such as high communication cost or latency, or to maintain user privacy and security [1]. This has led to the deployment of distributed machine learning and deep-learning techniques where computation is performed collaboratively by set of clients, each close to its own data source.

Once scenario that arises in distributed training is when clients have different sets of features, but there is a sizable overlap in the sample ID space among their datasets [2]. For example, the training dataset may be distributed across silos in a multi-organizational context, for example in healthcare, banking, finance, retail, etc. [2, 3]. Each silo holds a distinct set of features (e.g., customer/patient list); the data within each silo may even be of a different modality, for example, one silo may have audio features, whereas another silo has image data. The paradigm of training a global model over such feature-partitioned data is called *vertical federated learning* [4, 5]. This is different from the more prevalent alternative of *horizontal learning*, where the participating clients each have the entire set of features for a subset of the sample space [6, 7, 1].

Earlier vertical learning works [8, 4, 9, 10] considered a case where each party needs to communicate in each iteration, which may be expensive communication-wise. To save communication, multiple rounds of training can be performed on a client before reconciling the local model updates into the global model. A more recent work [5] proposed an algorithm that addresses this problem by performing multiple local training iterations before reconciling the client model updates into the global model. All of these works assume that the entire dataset of a silo is contained in a single client. However, this model fails to capture the case where the dataset within a silo is horizontally partitioned across multiple clients, for example, the dataset of a bank may be distributed among its branches, or healthcare data among hospitals in a chain.

We propose a training algorithm, tiered decentralized coordinate descent (TDCD), for vertical federated learning where there are multiple clients in each silo. We consider a two tiered network architecture consisting of multiple silos. Each silo holds a vertical partitioning of the data, and internally consists of a hub and multiple clients connected to the hub. The data in a silo is further horizontally distributed among its clients. Our goal, is to jointly train a model on the features of the data contained across silos, without explicitly sharing raw data from clients, and only via passing intermediate information vectors. TDCD works by performing a non-trivial combination of parallel coordinate descent on the top tier between silos, and distributed stochastic gradient descent in the bottom tier of clients inside each silo. To reduce communication, each client performs multiple local gradient steps before sending updates to its hub. This optimization is similar to the method studied in [6, 11, 12] for horizontal learning. We note that some existing works have proposed training algorithms for hierarchical network architectures [13, 14, 15, 16], but only from the perspective of horizontal learning. Our approach is thus a novel combination of learning with both vertically and horizontally partitioned data in a multi-tiered network.

Specifically, our contributions are the following: (1) we present a system model for decentralized learning in a two-tier network, where data is both vertically and horizontally partitioned; (2) we develop a communication-efficient decentralized learning algorithm, using principles from coordinated descent and stochastic gradient descent; (3) we analyze the convergence of our proposed algorithm and show how it depends on the number of silos, the number of clients, and the number of local training rounds; (4) we validate our analysis via experiments using convex and non-convex objectives.

2. SYSTEM MODEL AND PROBLEM FORMULATION

In this section, we describe the system architecture, the allocation of the training data, and the loss function we seek to minimize.

2.1. System Architecture and Training Data

We consider a decentralized system consisting of N silos, shown Fig. 1. Each silo consists of a hub and multiple clients connected to

This work is supported by the Rensselaer-IBM AI Research Collaboration (http://airc.rpi.edu), part of the IBM AI Horizons Network (http://ibm.biz/AIHorizons), and by the National Science Foundation under grants CNS 1553340 and CNS 1816307.

it in a hub-and-spoke fashion. The hub network forms a complete graph. For simplicity, we assume that each silo has K clients. Our network model thus has two tiers, the top tier of hubs, shown in orange, that communicate with each other, and the bottom tier of clients in each silo, shown in gray.

The training data consists of M samples that are common across all silos. Each sample has D features. The data is partitioned vertically across the N silos so that each silo owns a disjoint set of D_j features for all of the M samples. We can express the entire training dataset by a matrix $\mathbf{X} \in \mathbb{R}^{M \times D}$. We denote set of data, i.e., the columns of $\mathbf{X}$, held in silo j by $\mathbf{X}_{(j)}$. Within each silo, its data is partitioned horizontally across its clients, so that each client holds some rows of $\mathbf{X}_{(j)}$. We denote the horizontal shard of $\mathbf{X}_{(j)}$ that is held by client k in silo j as $\mathbf{X}_{k,j}$. Lastly, we denote a sample i of the dataset (single row of $\mathbf{X}$) as $\mathbf{X}^{(i)}$, and $\mathbf{X}^{(i)}_{(j)}$ denotes the features of the ith sample corresponding to silo j. We assume that each client stores the sample labels $\mathbf{y}_{k,j}$ for its data $\mathbf{X}_{k,j}$.

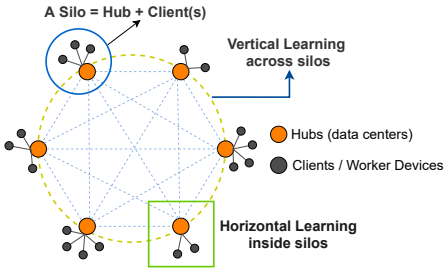


Fig. 1: System architecture.

2.2. Loss Function

The objective is to train a global model $\tilde{\boldsymbol{\theta}}$, which is a d-vector that can be decomposed as

$$\tilde{\boldsymbol{\theta}} = [\tilde{\boldsymbol{\theta}}_{(1)}^T, \ldots, \tilde{\boldsymbol{\theta}}_{(N)}^T]^T$$

where each $\tilde{\boldsymbol{\theta}}_{(j)}$ is the block of features, or coordinates, for silo j. The goal of the training algorithm is to minimize an objective function with following structure:

$$\mathcal{L}(\tilde{\boldsymbol{\theta}}, \mathbf{X}; \mathbf{y}) \stackrel{\text{def}}{=} \frac{1}{M} \sum_{i=1}^{M} f(\tilde{\boldsymbol{\theta}}_{(1)}, \ldots, \tilde{\boldsymbol{\theta}}_{(N)}; \mathbf{X}^{(i)}, \mathbf{y}^{(i)}) + \lambda \sum_{d=1}^{N} \omega(\tilde{\boldsymbol{\theta}}_{(d)})$$

where f has the partially separable form

$$f(\tilde{\boldsymbol{\theta}}, \mathbf{X}^{(i)}; \mathbf{y}^{(i)}) = f\left(\sum_{d=1}^{N} \mathbf{X}^{(i)}_{(d)} \tilde{\boldsymbol{\theta}}_{(d)} \ , \mathbf{y}^{(i)} \right).$$

The functions $\omega(\cdot)$ constitute a regularizer, and λ is a hyperparameter. A concrete example of the loss function is an L_2 regularized square loss function for empirical risk minimization:

$$\mathcal{L}(\tilde{\boldsymbol{\theta}}, \mathbf{X}; \mathbf{y}) = \frac{1}{2M} ||\mathbf{X}\tilde{\boldsymbol{\theta}} - \mathbf{y}||_2^2 + \frac{\|\tilde{\boldsymbol{\theta}}\|_2^2}{2}.$$

3. PROPOSED ALGORITHM

In this section, we present our Tiered Decentralized Coordinate Descent algorithm (TDCD). The pseudocode is given in Algorithm 1. We first note that the hubs update their own corresponding blocks of coordinates of $\tilde{\boldsymbol{\theta}}_{(j)}$ in parallel; no hub has the entire $\tilde{\boldsymbol{\theta}}$. We define $\boldsymbol{\theta}^t_{k,j} \in \mathbb{R}^{D_j}$ as the local version of the coordinates of the weight vector $\tilde{\boldsymbol{\theta}}^t_{(j)}$ that each client updates. These local versions are initialized by the clients at iteration $t = 0$.

In iteration 0, and every Qth iteration thereafter, the hubs first average the models from the clients, where hub j, updates the jth block coordinates of global weight $\tilde{\boldsymbol{\theta}}^t$ as $\tilde{\boldsymbol{\theta}}^t_{(j)} = \frac{1}{K} \sum_{k=1}^{K} [\boldsymbol{\theta}^t_{k,j}]$. This step is similar to horizontal federated learning. The hubs then agree on Q minibatches $\{\zeta^\tau\}_{\tau=t}^{t+Q-1}$, each containing B samples randomly drawn from the global dataset $\mathbf{X}$. The hubs communicate the aggregated model and the minibatch information to their clients. The clients, in turn, reply with the intermediate information for the samples IDs in those Q minibatches using the newest aggregated model. It is necessary to propagate this intermediate information to allow clients in other hubs to calculate partial derivatives during training. We define the *intermediate information* for the jth coordinate block for a single sample p as $\Phi^{(p)}_{(j)} = \mathbf{X}^{(p)}_{(j)} \tilde{\boldsymbol{\theta}}^t_{(j)}$. For a single minibatch ζ, each client computes a set of information $\Phi^{\zeta}_{k,j} = \{\Phi^{(p)}_{(j)}\}_{p \in \zeta}$. Each client then sends Q such sets of intermediate information to its hub corresponding to the Q minibatches. The hub then stacks the set of updates $\{\Phi^{\zeta}_{k,j}\}$ from each of its clients to form Φ^j_t. Each hub j then broadcast Φ^j_t to other hubs to propagate this information. For hub j, we denote the intermediate information obtained from other hubs by $\Phi_{-j} = \sum_{l=1, l \neq j}^{N} \Phi_j$. Once this is done, the hub then applies a projection function for each client k to send the subset of information from Φ_{-j} relevant to client $k's$ samples to that client. Alternatively a hub can send the entire Φ_{-j} to the client and the client can do the projection itself to extract the rows corresponding to its own samples. We define a projection function $\pi_{k,j}$ such that $\pi_{k,j}(\Phi_{-j}) = \Phi_{-k,j}$, where $\Phi_{-k,j}$ is the extracted relevant information for client k of silo j.

After receiving this intermediate information, at each iteration t each client k of silo j can now calculate its own local partial derivatives of $\mathcal{L}$ with respect to coordinate block j. This is denoted by $g_{k,j}$ and is a function of $\Phi_{-k,j}$, the part of $\mathbf{X}_{k,j}$ in minibatch ζ^t, and the local set of weights $\boldsymbol{\theta}_{k,j}$. Each client executes Q local stochastic gradient steps, on the features for their respective silos, using a different minibatch in each iteration:

$$\boldsymbol{\theta}^{t+1}_{k,j} = \boldsymbol{\theta}^t_{k,j} - \eta g_{k,j}(\Phi^{t_0}_{-k,j}, \boldsymbol{\theta}^t_{k,j}; \zeta^\tau). \tag{1}$$

η is the step size (learning rate), and t_0 represents the most recent iteration $t_0 < t$ in which the client received intermediate information from its hub. The entire process is repeated until convergence.

Informally, each silo effectively takes an approximate (stochastic) gradient step towards the minimizer of $\mathcal{L}(\tilde{\boldsymbol{\theta}}^t)$ along the direction of the its coordinates every Q iterations.

In TDCD, clients only communicate their local model and intermediate information every Q iterations. This is in contrast to distributed SGD algorithms, where the clients need to sync with a coordinating hub in each iteration. This allows TDCD to save bandwidth by increasing Q, especially when the the size of the model is large. Hubs still need to exchange intermediate information for all Q minibatches, in between local training rounds. However, sending all information at the beginning of Q iterations, rather than in every iteration, potentially saves network latency and overhead. The significant bandwidth savings comes in the silos themselves, since each hub and its clients only share the models every Q iterations. As a rough estimate, the intermediate information for a sample ranges from a simple scalar value to a small vector of very few dimensions.

Algorithm 1 Tiered Decentralized Coordinate Descent (TDCD)

Initialize $\boldsymbol{\theta}_{k,j}^t = \boldsymbol{\theta}_{k,j}^{init} \in \mathbb{R}^{D_j}$, $\forall k, j$
for $t = 0, \ldots, \infty$ **do**
  if t $(\text{mod } Q)$=0 **then**
    for $j = 1, \ldots, N$ silos *in parallel* **do**
      Hub j computes $\tilde{\boldsymbol{\theta}}_{(j)}^t = \frac{1}{K} \sum_{k=1}^{K} \boldsymbol{\theta}_{k,j}^t$
      Randomly sample Q minibatches $\{\zeta^\tau\}_{\tau=t+1}^{t+Q}$
      for $k = 1, \ldots, K$ clients *in parallel* **do**
        Set $\boldsymbol{\theta}_{k,j}^t = \tilde{\boldsymbol{\theta}}_{(j)}^t$
        Send $\Phi_{k,j}^{\zeta}$ to hub j, for each $\zeta \in \{\zeta^\tau\}_{\tau=t+1}^{t+Q}$
      end for
      Hub j stack $\{\Phi_{k,j}^{\zeta}\}, \zeta \in \{\zeta^\tau\}_{\tau=t+1}^{T+Q}$ to form Φ_j^t
      All hubs exchange $\Phi_j^t, \forall j = 1, \ldots, N$
      Hub j calculate $\Phi_{-j}^t = \sum \Phi_p^t, \forall p \neq j$
      In parallel set $\Phi_{-k,j}^{t_0} = \pi_{k,j}(\Phi_{-j}^t)$ in K clients.
    end for
  end if
  for $j = 1, \ldots, N$ silos *in parallel* **do**
    for $k = 1, \ldots, K$ clients *in parallel* **do**
      $\boldsymbol{\theta}_{k,j}^{t+1} = \boldsymbol{\theta}_{k,j}^t - \eta g_{k,j}(\Phi_{-k,j}^{t_0}, \boldsymbol{\theta}_{k,j}^t; \zeta^\tau)$
    end for
  end for
end for

Therefore while training models in deep learning, the intermediate information of B minibatches with M samples each would be of the order of a few megabytes or less. Compared to this, the size of the actual model can be in the order of gigabytes. We explore how Q impacts the convergence of TDCD in the next section.

We note that at any step of training hubs can communicate their slice of the global model with each other to form the entire global model for use in inference purposes.

4. CONVERGENCE ANALYSIS

In this section, we provide the convergence analysis of the TDCD algorithm. Our analysis is based on the evolution of the global model $\tilde{\boldsymbol{\theta}} \in \mathbb{R}^D$ following Algorithm 1. It to be noted that the components of $\tilde{\boldsymbol{\theta}}$, $\tilde{\boldsymbol{\theta}}_{(j)}$ are realized every Q iterations, but we will study the evolution of a virtual $\tilde{\boldsymbol{\theta}}_{(j)}$ at each iteration, $\tilde{\boldsymbol{\theta}}_{(j)}^t = \frac{1}{K} \sum_{k=1}^{K} \boldsymbol{\theta}_{k,j}^t$.

To facilitate the analysis, we first define the notion of an auxiliary local vector, which represents the local view of the global model at each client. Let $y_{k,j}^t$ denote the auxiliary weight vector used by client k in hub j to calculate the partial derivative $g_{k,j}(\boldsymbol{y}_{k,j}^t)$,

$$\boldsymbol{y}_{k,j}^t = [\boldsymbol{\theta}_{-j}^{t_0}, \boldsymbol{\theta}_{k,j}^t] \tag{2}$$

where, $\boldsymbol{\theta}_{-j}^{t_0}$ denotes the vector of all coordinates of $\tilde{\boldsymbol{\theta}}$ excluding block j at iteration t, where t_0 is the iteration when the client k last updated the value of $\boldsymbol{\theta}_{-j}^{t_0}$ from its hub. Therefore, when a client takes multiple local steps to update $\boldsymbol{\theta}_{k,j}$, it uses a stale value of the elements in the other coordinates of $\boldsymbol{y}_{k,j}$.

We further define the following two quantities

$$\boldsymbol{G}^t = [(\boldsymbol{G}_{(1)}^t)^T, \ldots, (\boldsymbol{G}_{(1)}^t)^T]^T \, , \boldsymbol{G}_{(j)}^t = \frac{1}{K} \sum_{k=1}^{K} g_{k,j}(\boldsymbol{y}_{k,j}) \tag{3}$$

We can then write the evolution of the global model as follows,

$$\tilde{\boldsymbol{\theta}}^{t+1} = \tilde{\boldsymbol{\theta}}^t - \eta \boldsymbol{G}^t \tag{4}$$

We make the following assumptions about the loss function $\mathcal{L}$ and the gradients $g_{k,j}$ at each client.

Assumption 1. *The gradient of the loss function is Lipschitz continuous with constant L; further, the partial derivative of $\mathcal{L}$ with respect to each coordinate block j is Lipschitz continuous with constant L_j, i.e., for all $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2 \in \mathbb{R}^D$*

$$\|\nabla \mathcal{L}(\boldsymbol{\theta}_1) - \nabla \mathcal{L}(\boldsymbol{\theta}_2) \| \leq L \| \boldsymbol{\theta}_1 - \boldsymbol{\theta}_2 \| \tag{5}$$

$$\|\nabla_{(j)} \mathcal{L}(\boldsymbol{\theta}_1) - \nabla_{(j)} \mathcal{L}(\boldsymbol{\theta}_2) \| \leq L_j \| \boldsymbol{\theta}_1 - \boldsymbol{\theta}_2 \|. \tag{6}$$

Assumption 2. *The function $\mathcal{L}$ is lower bounded so that for all $\boldsymbol{\theta} \in \mathbb{R}^D$, $\mathcal{L}(\boldsymbol{\theta}) \geq \mathcal{L}_{inf}$.*

Assumption 3. *Let ζ be a mini-batch drawn uniformly at random from all samples. We assume that the data is distributed so that, for all $\boldsymbol{\theta} \in \mathbb{R}^D$*

$$\mathbb{E}_{\zeta|\boldsymbol{\theta}} \left[g_{k,j}(\boldsymbol{\theta}) \right] = \nabla_{(j)} \mathcal{L}(\boldsymbol{\theta}) \tag{7}$$

$$\mathbb{E}_{\zeta|\boldsymbol{\theta}} \left[\| g_{k,j}(\boldsymbol{\theta}) - \nabla_{(j)} \mathcal{L}(\boldsymbol{\theta}) \|^2 \right] \leq \sigma_j^2. \tag{8}$$

We also use the following definitions:

$$L_{max} = \max_{1 \leq j \leq N} L_j \, , \, \sigma_{max} = \max_{1 \leq j \leq N} \sigma_j$$

We now provide the main theoretical result of the paper. The proof is deferred to a technical report available in [17].

Theorem 4.1. *Under Assumptions 1, 2, and 3, when the step size η satisfies the following condition:*

$$1 - \eta L - \eta^2 L_{max}^2 Q^2 \geq 0 \tag{9}$$

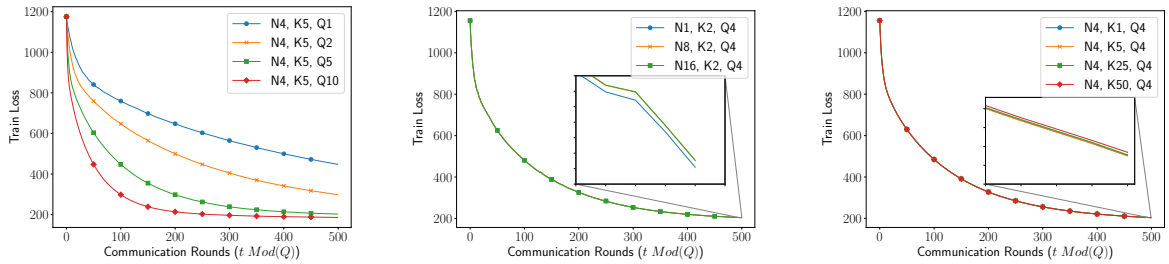
then, for $T > 0$, the expected squared norm of the gradient of $\mathcal{L}$ averaged over all T iterations satisfies the following bound:

$$\mathbb{E} \left[\frac{1}{T} \sum_{t=0}^{T-1} \|\nabla \mathcal{L}(\tilde{\boldsymbol{\theta}}^t)\|^2 \right] \leq \frac{2 \left(\mathcal{L}(\tilde{\boldsymbol{\theta}}^0) - \mathcal{L}_{inf} \right)}{\eta T} + \frac{\eta L N \sigma_{max}^2}{K} + L_{max}^2 \eta^2 \sigma_{max}^2 Q^2 N^2 \tag{10}$$

We note that the bound in Theorem 4.1 converges to a non-zero value as $T \to \infty$. The convergence error results from the parallel updates on the coordinate blocks (on N) , staleness due to multiple local iterations (on Q) and due to parallel updates based on horizontal partitioning (on K) as well. With an increase in the number of vertical partitions, the error term increases quadratically. The error also depends quadratically on Q, however, in practice, if Q is offset by a suitable learning rate η, then we can leverage multiple local iterations to achieve faster convergence as we will show in Sec. 5. However, choosing a very small η will decrease the convergence error, but it will but increase the first term on the right hand side of (10), leading to slower convergence.

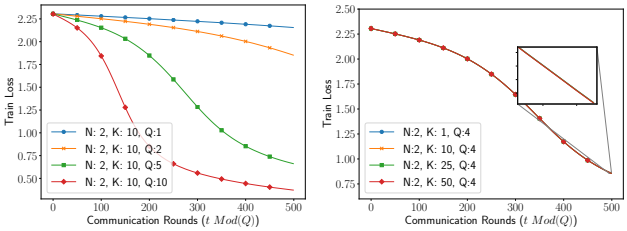
5. EXPERIMENTAL RESULTS

We verify the convergence properties of TDCD with respect to the different algorithm parameters of the system via a simulation. In our experiments, each client has the same number of samples = $\frac{M}{K}$.



(a) Variation with Q (b) Variation with N (c) Variation with K

Fig. 2: Ridge Regression Convex Objective. Training loss vs communication rounds for variations of Q, N and K.



(a) Variation with Q (b) Variation with K

Fig. 3: CNN Multi-class classification with Non-Convex Objective. Training loss vs communication rounds for variations of Q and K.

5.1. Datasets

We first briefly discuss the two datasets used in this study.

Superconductivity (Convex Objective: Ridge Regression): For the first experiments, we use the Superconductivity dataset [18], which consists of numerical values in all coordinates. The goal is to predict the critical temperature of superconducting materials. We standardized the dataset before using it by normalizing each coordinate to have zero mean and unit variance. We use $20,000$ samples from the original dataset for training. We use all 81 coordinates and include add one for bias.

MNIST (Non-Convex Objective: CNN): We train a CNN model on the MNIST dataset [19]. MNIST is a set of 28×28 pixels hand-written digits images with 60,000 digits in the training set and 10,000 digits in the test set. We use $N = 2$ for all the experiments and divide each MNIST image vertically into two parts (28×14). Each client trains a local CNN model with a shared linear classifier layer at the top that uses cross-entropy loss. The local CNNs have two *conv* layers followed by a 256 dimension embedding layer which is fed into the final classifier layer. The two feature representations of $\mathbb{R}^{256}$ are inputs to the classifier layer with $\mathbb{R}^{512}$ input and $\mathbb{R}^{10}$ output. We thus train the weights of the final layer via TDCD while also updating the local CNNs in each iteration.

5.2. Results

In all figures N represents the number of silos (vertical partitions of the dataset), and K represents the number of clients in each silo. In each of the experiments, The training loss is calculated using the global model $\tilde{\boldsymbol{\theta}}$ and the full training data matrix every Q iterations. We call every Qth iteration a *communication round* because it is when communication between clients and hubs occur.

We first study the performance of TDCD on the convex case of ridge regression in Fig. 2. We start with the impact of varying the number of local iterations Q on the convergence rate. We fix the network configuration to N=4 silos and K=5 clients per silo, with a minibatch size of $B = 100$ and learning rate $\eta = 0.001$. The results are shown in Fig. 2a. We observe that with increasing values of Q, the convergence rate improves. This is intuitive as the clients can train more with a larger number of local rounds between communications, however, as stated in Theorem 4.1, this can result in a larger convergence error. This implies that by increasing the number of local iterations at clients, we can improve the overall communication efficiency by reducing the total number of communication rounds required for a given loss.

In Fig. 2b, we show the impact of varying the number of vertical partitions on the convergence rate. To observe results at higher granularity, we use a subset of 2000 samples from the original training dataset. We fix K=2, Q=4, and B=20 for this experiment. We observe that the effect of increasing N is observable but not very strong. The inset figure shows the last five communication rounds, and we observe that the convergence rate improves with lower value of N, which is as per Theorem 4.1.

We next study how the number of workers in a silo effects the convergence rate. The results are shown in Fig. 2c. We fix N=4, and Q=4 and $B = 500$. Further, we use the same 2000 data points as in the previous experiment. The inset figure here also shows the last five communication rounds of training. We observe that variation of convergence rate is low with varying K. This shows that K does not play a large role as Q in its effect on the convergence rate or convergence error.

Finally, we study the performance of TDCD with the non-convex objective. We fix the number of silos at N=2 and the learning rate $\eta = 0.001$ for all experiments. We first investigate the impact of Q on the convergence rate and error. The results are shown in Fig. 3a. Here, K=10 and $B = 640$. We observe that the convergence rate improves radically for larger values of Q. This result is similar to what we obtained from the convex case. Hence, by choosing Q carefully it is possible to significantly decrease the communication cost without losing performance. Lastly, in Fig. 3b, we explore the effect of varying the number of clients at each silo. We fix the product of K and B to 1250 across the experiments, so that each silo effectively trains on the same number of samples in each experiment. Similar to the convex case, we again observe that the effect of K is very mild. Overall, we we observe that TDCD performs well with both convex and non-convex objectives.

6. CONCLUSION

We have introduced TDCD, a communication efficient decentralized algorithm for a multi tier network model with both horizontally and vertically partitioned data. We provided theoretical analysis of the algorithm convergence and its dependence on the number of verti-

cal partitions, the number of clients in each hub, and the number of local iterations. Finally, we presented experimental results to show convergence of our algorithm in practice. In future work, we plan to explore the possibility of hubs communicating with each other asynchronously to share information.

7. REFERENCES

[1] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al., "Advances and open problems in federated learning," *arXiv preprint arXiv:1912.04977*, 2019.

[2] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong, "Federated machine learning: Concept and applications," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 2, Jan. 2019.

[3] C Sun, L Ippel, J van Soest, B Wouters, A Malic, O Adekunle, B van den Berg, O Mussmann, A Koster, C van der Kallen, et al., "A privacy-preserving infrastructure for analyzing personal health data in a vertically partitioned scenario.," *Studies in health technology and informatics*, vol. 264, pp. 373, 2019.

[4] Shengwen Yang, Bing Ren, Xuhui Zhou, and Liping Liu, "Parallel distributed logistic regression for vertical federated learning without third-party coordinator," *arXiv preprint arXiv:1911.09824*, 2019.

[5] Yang Liu, Yan Kang, Xinwei Zhang, Liping Li, Yong Cheng, Tianjian Chen, Mingyi Hong, and Qiang Yang, "A communication-efficient collaborative learning framework for distributed features," *arXiv preprint arXiv:1912.11187*, 2019, Presented in Workshop on Federated Learning for Data Privacy and Confidentiality, NeuRIPS 2019.

[6] H Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, et al., "Communication-efficient learning of deep networks from decentralized data," *arXiv preprint arXiv:1602.05629*, 2016.

[7] Jakub Konečnỳ, H Brendan McMahan, Daniel Ramage, and Peter Richtárik, "Federated optimization: Distributed machine learning for on-device intelligence," *arXiv preprint arXiv:1610.02527*, 2016.

[8] Stephen Hardy, Wilko Henecka, Hamish Ivey-Law, Richard Nock, Giorgio Patrini, Guillaume Smith, and Brian Thorne, "Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption," *arXiv preprint arXiv:1711.10677*, 2017.

[9] Siwei Feng and Han Yu, "Multi-participant multi-class vertical federated learning," *arXiv preprint arXiv:2001.11154*, 2020.

[10] Tianyi Chen, Xiao Jin, Yuejiao Sun, and Wotao Yin, "Vafl: a method of vertical asynchronous federated learning," *arXiv preprint arXiv:2007.06081*, 2020.

[11] Sebastian U Stich, "Local sgd converges fast and communicates little," *arXiv preprint arXiv:1805.09767*, 2018.

[12] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith, "Federated optimization in heterogeneous networks," in *Proceedings of Machine Learning and Systems 2020, MLSys 2020, Austin, TX, USA, March 2-4, 2020*, Inderjit S. Dhillon, Dimitris S. Papailiopoulos, and Vivienne Sze, Eds. 2020, mlsys.org.

[13] Jianyu Wang and Gauri Joshi, "Cooperative sgd: A unified framework for the design and analysis of communication-efficient sgd algorithms," *arXiv preprint arXiv:1808.07576*, 2018.

[14] M. S. H. Abad, E. Ozfatura, D. GUndUz, and O. Ercetin, "Hierarchical federated learning across heterogeneous cellular networks," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 8866–8870.

[15] Timothy Castiglia, Anirban Das, and Stacy Patterson, "Multi-level local SGD: Distributed SGD for heterogeneous hierarchical networks," in *International Conference on Learning Representations*, 2021.

[16] L. Liu, J. Zhang, S. H. Song, and K. B. Letaief, "Client-edge-cloud hierarchical federated learning," in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, 2020, pp. 1–6.

[17] Anirban Das and Stacy Patterson, "Multi-tier federated learning for vertically partitioned data," *arXiv preprint arXiv:2102.03620*, 2021.

[18] Kam Hamidieh, "A data-driven statistical model for predicting the critical temperature of a superconductor," *Computational Materials Science*, vol. 154, pp. 346–354, 2018.

[19] L. Bottou, C. Cortes, J. Denker, H. Drucker, I. Guyon, L. Jackel, Y. LeCun, U. Muller, E. Sackinger, P. Simard, et al., "Comparison of classifier methods: a case study in handwritten digit recognition," in *Proceedings of the 12th IAPR International Conference on Pattern Recognition*. IEEE, 1994, vol. 2, pp. 77–82.