A Multi-Token Coordinate Descent Method for Vertical Federated Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

Communication efficiency is a major challenge in federated learning. In client-server schemes, the server constitutes a bottleneck, and while decentralized setups spread communications, they do not reduce them. We propose a communication efficient semi-decentralized federated learning algorithm for feature-distributed data. Our multi-token method can be seen as a parallel Markov chain (block) coordinate descent algorithm. In this work, we formalize the multi-token semi-decentralized scheme, which subsumes the client-server and decentralized setups, and design a feature-distributed learning algorithm for this setup. Numerical results show the improved communication efficiency of our algorithm.

1 Introduction

Federated Learning (FL) is a machine learning paradigm where data is distributed across a set of clients who collaborate to learn a model without sharing local data [1]. Most FL literature considers data distributed by samples (horizontal FL), where each client holds all the features of a subset of the samples, yet recently there has been a growing interest on feature-distributed setups (vertical FL), where each client holds a subset of the features for all samples [2–5].

FL often deals with star-shaped topologies. However, such schemes have a single point of failure and suffer from a communication bottleneck on the server [6]. On the other hand, there is extensive literature on decentralized optimization—from earlier work motivated by applications such as wireless sensor networks and multiagent control [7–9], to recent work motivated by FL [10]—yet these algorithms converge slowly in sparse and large networks [11] and, although they spread the communication load across the network, they tend to have a poor communication efficiency [12].

We propose a Semi-Decentralized FL (SDFL) multi-token algorithm for vertical FL. By using both client-server *and* client-client communications, we reduce the communications at the server while mitigating the slow convergence of decentralized algorithms in sparse and large networks. In the communications between clients, the use of a token method [13–18], where a model-describing *token* follows a random walk over a communication graph (undergoing local updates), allows for better communication efficiency [18] than the more common consensus-based methods [7–10], where asymptotic agreement is reached through successive local averaging. Yet, the convergence rate of token methods degrades even faster for larger and sparser networks, due to a lack of parallel communications. Multi-token methods [18–20] mitigate this problem by running multiple tokens simultaneously and combining them.

As our main contributions, we:

- Formalize the multi-token semi-decentralized federated learning scheme, which is robust to server failures and flexible in the degree of dependence on the server, recovering both client-server and decentralized FL as particular cases;

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

- Design a Multi-Token Coordinate Descent (MTCD) algorithm for vertical FL with improved communication efficiency over state-of-the-art methods;
- Run numerical experiments on both synthetic and real data, for a variety of communication setups, showing the improved communication efficiency of MTCD as well as the effectiveness of resorting to multiple tokens.

Related works. Recently, SDFL approaches have been proposed to lower communication costs and deal with data heterogeneity [21, 22] and to handle intermittent connections, latency, and stragglers [23, 24]. Some SDFL works deal with (multi-layered) hierarchical networks [25, 26].

Coordinate Descent (CD) methods [27], where (blocks of) coordinates are updated sequentially, rather than simultaneously, are natural candidates for optimization in feature-distributed learning. The block selection is most often cyclic [28] or independent and identically distributed random [29, 30]. In contrast, [31] considers block selection following a Markov chain. Several extensions to CD have been proposed, such as acceleration and parallelization [32] and distributed CD methods [20, 33].

2 Single-token coordinate descent (STCD)

We now introduce a simple, particular case of our algorithm, closely related to [31] and the application mentioned therein, taken from [17]. Yet, we work in the primal domain and on a feature-distributed setting. In this section, we use the terms client and agent interchangeably.

2.1 Problem statement

Model. Our goal is to minimize the following regularized generalized linear model:

$$f(\boldsymbol{\theta}) \triangleq \sum_{n=1}^{N} \ell\left(\boldsymbol{x}_n^\top \boldsymbol{\theta}\right) + r(\boldsymbol{\theta}), \tag{1}$$

where $\boldsymbol{\theta}$ is a parameter of interest and we have N samples $\boldsymbol{x}_n$, $n \in \{1, \dots, N\}$. Our loss function ℓ is a (possibly nonconvex) smooth and block smooth loss function with a nonempty set of minimizers and r is a separable (possibly nonsmooth) closed proper function, $r(\boldsymbol{\theta}) = \sum_{k=1}^{K} r_k(\boldsymbol{\theta}_k)$, which we assume to have a simple proximal mapping.

Setup. In this section, we do not require the existence of a server. The clients $[K] \triangleq \{1, \dots, K\}$ learn the model in a fully decentralized way, communicating through channels described by a static, undirected communication graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = [K]$ is the vertex set and $\mathcal{E}$ the edge set. We assume $\mathcal{G}$ is connected and denote the set of neighbors of agent k by $\mathcal{N}_k \triangleq \{i \colon \{i, k\} \in \mathcal{E}\}$.

Our dataset $\boldsymbol{X} \in \mathbb{R}^{N \times d}$ with samples $\boldsymbol{x}_n \in \mathbb{R}^d$, $n \in [N]$, is distributed by features across the clients. More precisely, each sample $\boldsymbol{x}_n$ is partitioned as $\boldsymbol{x}_n = (\boldsymbol{x}_{nk})_{k=1}^K \in \mathbb{R}^d$, where $\boldsymbol{x}_{nk} \in \mathbb{R}^{d_k}$ and $\sum_{k=1}^K d_k = d$. Consequently, we can equivalently define the partitioning as $\boldsymbol{X} = [\boldsymbol{X}_1 \dots \boldsymbol{X}_K]$, where $\boldsymbol{X}_k \in \mathbb{R}^{N \times d_k}$. Our parameter $\boldsymbol{\theta} \in \mathbb{R}^d$ is partitioned similarly to $\boldsymbol{x}_n$. That is, $\boldsymbol{\theta} = (\boldsymbol{\theta}_k)_{k=1}^K \in \mathbb{R}^d$ with $\boldsymbol{\theta}_k \in \mathbb{R}^{d_k}$. The labels $\boldsymbol{y} \in \mathbb{R}^N$ are part of the loss function, which we assume all agents to know.

2.2 Algorithm

We now propose a decentralized token method algorithm to minimize (1). Our *token* carries $\boldsymbol{z} \triangleq \boldsymbol{X}\boldsymbol{\theta} \in \mathbb{R}^N$ and performs a random walk over the graph, being updated and communicated by each agent after performing local computations. We denote by k^t the agent holding the token at iteration t.

The token suffices to compute partial gradients. From the definition of $\boldsymbol{z}$, we have at iteration t that $\boldsymbol{z}^t = \sum_{k=1}^K \boldsymbol{X}_k \boldsymbol{\theta}_k^t$. Letting $L(\boldsymbol{\theta}) \triangleq \sum_{n=1}^N \ell\left(\boldsymbol{x}_n^\top \boldsymbol{\theta}\right)$, we see that, given $\boldsymbol{z}^t$, agent k can compute the gradient of the loss function with respect to (w.r.t.) its local parameter $\boldsymbol{\theta}_k$:

$$\nabla_k L(\boldsymbol{\theta}) \triangleq \nabla_{\boldsymbol{\theta}_k} \sum_{n=1}^{N} \ell\left(\boldsymbol{x}_n^\top \boldsymbol{\theta}\right) = \sum_{n=1}^{N} \nabla_z \ell\left(z_n\right) \boldsymbol{x}_{nk}, \qquad z_n = \boldsymbol{x}_n^\top \boldsymbol{\theta},$$

Algorithm 1 Single-token Coordinate Descent (STCD)

Initialize $\boldsymbol{\theta}^0 = \mathbf{0}$, $\boldsymbol{z}^0 = \mathbf{0}$, arbitrary k^0, and choose step-size μ
for $t \geq 0$ **do**
  if $t \bmod Q = 0$ **then**
    Agent k^t sends $\boldsymbol{z}^{t+1}$ to agent $k^{t+1} \sim \mathcal{U}(\mathcal{N}_{k_t})$
  else
    $k^{t+1} = k^t$
  end if
  $\boldsymbol{\theta}_k^{t+1} = \begin{cases} \mathrm{prox}_{\mu r_k}(\boldsymbol{\theta}_k^t - \mu \nabla_k L(\boldsymbol{\theta}^t)) & \text{if } k = k^t \\ \boldsymbol{\theta}_k^t & \text{if } k \neq k^t \end{cases}$ $\{\boldsymbol{z}^t$ used when computing $\nabla_{k^t} L(\boldsymbol{\theta}^t)\}$
  $\boldsymbol{z}^{t+1} = \boldsymbol{z}^t + \boldsymbol{X}_{k^t}(\boldsymbol{\theta}_{k^t}^{t+1} - \boldsymbol{\theta}_{k^t}^t)$
end for

allowing agent k to (locally) perform a proximal CD step w.r.t. $\boldsymbol{\theta}_k$, as long as it also holds $\boldsymbol{\theta}_k^t$:

$$\boldsymbol{\theta}_k^{t+1} = \mathrm{prox}_{\mu r_k}(\boldsymbol{\theta}_k^t - \mu \nabla_k L(\boldsymbol{\theta}^t)), \quad \mathrm{prox}_g(\cdot) \triangleq \mathrm{argmin}_u \, g(u) + \frac{1}{2}\|u - \cdot\|^2.$$

Given that we only update $\boldsymbol{\theta}_k$ at agent k, k^t always knows $\boldsymbol{\theta}_{k^t}^t$.

We now describe STCD, summarized in Algorithm 1, where $\mathcal{U}$ denotes the uniform distribution.

Initialization. To start STCD at a given client k^0, we need to ensure that $\boldsymbol{z}^0$ which, as we saw, contains all the information needed to do a (proximal) CD step, and $\boldsymbol{\theta}^0$ verify $\boldsymbol{z}^0 = \boldsymbol{X}\boldsymbol{\theta}^0$. To do this without requiring any information from the other clients, we must initialize $\boldsymbol{\theta}^0 = \mathbf{0}$ and $\boldsymbol{z}^0 = \mathbf{0}$.

Updating and communicating the token. After performing a local (proximal) coordinate descent step, which gives agent k^t the iterate $\boldsymbol{\theta}_{k^t}^{t+1}$, we need to update the token to ensure that $\boldsymbol{z}^{t+1} = \boldsymbol{X}\boldsymbol{\theta}^{t+1}$. Observing that $\boldsymbol{z}^t$ allows for a local update $\boldsymbol{z}^{t+1} = \boldsymbol{z}^t + \boldsymbol{X}_{k^t}(\boldsymbol{\theta}_{k^t}^{t+1} - \boldsymbol{\theta}_{k^t}^t)$ whose result can then be sent to agent k^{t+1}, we see that, by induction, $\boldsymbol{z}^t$ can be kept up-to-date throughout our algorithm. Additionally, this also means that an agent can actually perform Q multiple coordinate descent steps w.r.t. its local parameters, $\boldsymbol{\theta}_k$.

In essence, STCD is a technique allowing for Markov Chain Coordinate Descent [31] to be performed in feature-distributed setups. In terms of the progress made in the parameter space, Algorithm 1 differs from [31] only in that it requires initializing $\boldsymbol{\theta}^0 = \mathbf{0}$ and in that we allow for $Q > 1$.

Let f be a function that is L_g-smooth and block L-smooth for all blocks k, with a nonempty set of minimizers, and let r be a separable closed proper function. It is shown in [31] that, for $Q = 1$, Algorithm 1 converges almost surely to a solution of (1).

2.3 Limitations

The decentralized token algorithm in this section has an appealing simplicity. However, while it outperforms state-of-the-art FD learning algorithms in a variety of setups, as we will see in Section 4, its performance deteriorate faster with network connectivity than these decentralized consensus-based algorithms. Yet, this simple algorithm acts as a stepping stone to the more general multi-token algorithm we now present in Section 3, which mitigates this problem.

3 Multi-token coordinate descent (MTCD)

In this section, instead of a fully decentralized setup, we deal with SDFL and generalize the algorithm used in the previous section in a direction allowing us to leverage the existence of a server.

3.1 Problem statement

In this section, the set of clients and the data and model partitioning is similar to Section 2. However, we now consider a server-enhanced setup (that is, SDFL), illustrated in Figure 1, where we also take

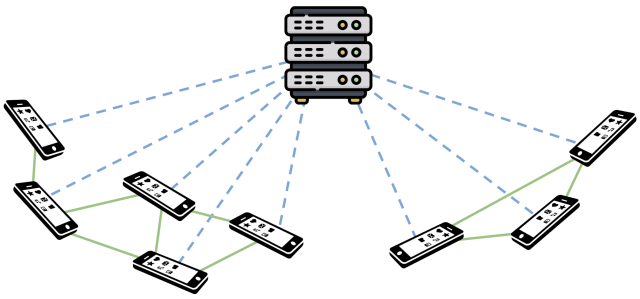

Figure 1: Semi-decentralized setup with $K = 9$. Client-server communications are represented by the dashed blue lines and client-client communications are represented by the unbroken green lines.

into account a central server with links to all the clients. In this section, we do not require the graph describing the communications between the clients to be connected.

Considering the same model (1) as in the previous section, we now develop an algorithm to leverage the existence of this server, exploiting it to mitigate the deterioration of the performance of Algorithm 1 in poorly connected networks.

3.2 Algorithm

The algorithm introduced in this section can be seen as having two parts. The roaming part is similar to the procedure in Algorithm 1, except for the fact that we run multiple tokens in parallel, each with an associated model. The novelty lies in the averaging part, where we exploit the access to the server by periodically averaging these tokens there.

More precisely, we use Γ tokens $\boldsymbol{z}_\gamma \in \mathbb{R}^N$, $\gamma \in [\Gamma]$, each with an associated model instance $\boldsymbol{\theta}_\gamma$ and $\boldsymbol{z}_\gamma \triangleq \boldsymbol{X}\boldsymbol{\theta}_\gamma \in \mathbb{R}^N$. Similarly to what we had in Algorithm 1, we initialize $\boldsymbol{\theta}_\gamma^0 = \mathbf{0}$ and $\boldsymbol{z}_\gamma^0 = \mathbf{0}$, $\gamma \in [\Gamma]$, and for $S \times Q$ iterations, each such token undergoes a process similar to the one in Algorithm 1. The difference lies in the fact that now, every S hops of the tokens, between each of which each agent performs Q local steps, the Γ tokens are averaged at the server and the Γ models are averaged locally, at the clients. Crucially, these averaging operations allow us to preserve the relationship defining $\boldsymbol{Z}_\gamma^t$. (This relationship is also preserved during the roaming part, as explained in the previous section.)

This method, a parallel Markov Chain Coordinate Descent for feature-distributed SDFL setups, is summarized in Algorithm 2, where k_γ^t denotes the agent holding token γ at time t.

Recovering client-server and decentralized setups. If no client-server communications are available ($S \to \infty$) our algorithm is reduced to a decentralized one. In this setting, even if at a lower rate, our algorithm will still converge, as long as $\mathcal{G}$ is connected, being reduced to Γ simultaneous runs of Algorithm 1. While a decentralized asynchronous token averaging is possible (by exploring the random intersections of multiple tokens are at the same client), this is significantly less effective than server averaging in terms of mitigating the performance deterioration caused by poor connectivity. In contrast, if the set of edges $\mathcal{E}$ is empty and we assign a token per agent, we recover the client-server setting.[1]

Extension to locally nonlinear models. We can generalize our model while keeping an additive structure w.r.t. $\{\boldsymbol{X}_k, \boldsymbol{\theta}_k\}$, capturing nonlinearities between $\boldsymbol{X}_k$ and $\boldsymbol{\theta}_k$. Letting $\boldsymbol{h}_k \colon \mathbb{R}^{|\boldsymbol{\theta}_k| \times d_k} \to \mathbb{R}^E$, we would minimize:

$$f(\boldsymbol{\theta}) \triangleq \sum_{n=1}^{N} \ell\left(\sum_{k=1}^{K} \boldsymbol{h}_k(\boldsymbol{\theta}_k, \boldsymbol{x}_{nk})\right) + r(\boldsymbol{\theta}),$$

which has generalized linear models as the particular case $\boldsymbol{h}_k(\boldsymbol{\theta}_k, \boldsymbol{x}_{nk}) = \langle \boldsymbol{\theta}_k, \boldsymbol{x}_{nk} \rangle$, where $E = 1$ and $|\boldsymbol{\theta}_k| = |\boldsymbol{x}_{nk}| = d_k$. However, this would require additional communications, since a separate averaging of $\boldsymbol{\theta}^t$ and $\boldsymbol{z}^t$ would no longer preserve their relation, leading to a need to recompute

[1] To be precise, we would have to change $k_\gamma^{t+1} \sim \mathcal{U}(\mathcal{N}_{k_\gamma^t})$ to $k_\gamma^{t+1} \sim \mathcal{U}(\bar{\mathcal{N}}_{k_\gamma^t})$, where $\bar{\mathcal{N}}_{k_\gamma^t} \triangleq \mathcal{N}_{k_\gamma^t} \cup k_\gamma^t$.

Algorithm 2 Multi-token Coordinate Descent (MTCD)

Initialize $\boldsymbol{\theta}_\gamma^0 = \mathbf{0}$, $\boldsymbol{z}_\gamma^0 = \mathbf{0}$, and arbitrary k_γ^0, for $\gamma \in [\Gamma]$, and choose step-size μ
for $t \geq 0$ **do**
  if $t \bmod Q = 0$ and $\lfloor t/Q \rfloor \bmod S = 0$ **then**
    clients send $\boldsymbol{z}_\gamma^t$ to the server
    server computes $\boldsymbol{z}_1^t, \ldots, \boldsymbol{z}_\Gamma^t = \frac{1}{\Gamma}\sum_{\gamma=1}^{\Gamma} \boldsymbol{z}_\gamma^t$
    server sends updated $\boldsymbol{z}_\gamma^t$ to the clients
    for $k \in [K]$ **in parallel do**
      $\boldsymbol{\theta}_{1k}, \ldots, \boldsymbol{\theta}_{\Gamma k} = \frac{1}{\Gamma}\sum_{\gamma=1}^{\Gamma} \boldsymbol{\theta}_{\gamma k}$
    end for
  end if
  if $t \bmod Q = 0$ **then**
    for $\gamma \in [\Gamma]$ **in parallel do**
      Agent k_γ^t sends $\boldsymbol{z}_\gamma^{t+1}$ to agent $k_\gamma^{t+1} \sim \mathcal{U}(\mathcal{N}_{k_\gamma^t})$
    end for
  else
    $k_\gamma^{t+1} = k_\gamma^t,\ \gamma \in [\Gamma]$
  end if
  for $\gamma \in [\Gamma]$ **in parallel do**
    $$\boldsymbol{\theta}_{\gamma k}^{t+1} = \begin{cases} \operatorname{prox}_{\mu r_k}(\boldsymbol{\theta}_{\gamma k}^t - \mu \nabla_k L(\boldsymbol{\theta}_\gamma^t)) & \text{if } k = k_\gamma^t \\ \boldsymbol{\theta}_{\gamma k}^t & \text{if } k \neq k_\gamma^t \end{cases} \quad \{\boldsymbol{z}_\gamma^t \text{ used when computing } \nabla_{k_\gamma^t} L(\boldsymbol{\theta}_\gamma^t)\}$$
    $$\boldsymbol{z}_\gamma^{t+1} = \boldsymbol{z}_\gamma^t + \boldsymbol{X}_k(\boldsymbol{\theta}_{\gamma k}^{t+1} - \boldsymbol{\theta}_{\gamma k}^t),\ k = k_\gamma^t$$
  end for
end for

the tokens, instead of simply updating them in an online manner. This would require all clients to communicate with the server.

4 Experiments

In this section, we compare STCD to Dual Consensus Proximal Algorithm (DCPA) [4], a state-of-the-art decentralized method used as a baseline, and to MTCD. We use CVXPY [34] to obtain $f^\star$, which we use to compute the suboptimality gap $f(\boldsymbol{\theta}^t) - f^\star$ throughout our experiments.

Models and datasets. We perform ridge regression on a dataset generated following the same process as [4],[2] but 10 times larger both in number of samples and in dimensionality ($N = 1000$ and $d = 2000$). We have the following objective function, with $\alpha = 10$:

$$f(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y}\|_2^2 + \frac{\alpha}{2}\|\boldsymbol{\theta}\|_2^2.$$

We also perform binary classification on the Gisette Digit recognition (training) dataset [35], where $N = 6000$ and $d = 5000$. Let $s(z) \triangleq (1 + e^{-z})^{-1}$, we do sparse logistic regression, with $\beta = 1$:

$$f(\boldsymbol{\theta}) = -\sum_{n=1}^{N} \left[y_n \log s(\boldsymbol{x}_n^\top \boldsymbol{\theta}) + (1 - y_n)\log(1 - s(\boldsymbol{x}_n^\top \boldsymbol{\theta})) \right] + \beta\|\boldsymbol{\theta}\|_1, \qquad y_n \in \{0, 1\}.$$

Metrics. In the previous sections, we considered each local CD step as an STCD iteration, to avoid the visual clutter of having a double counter (for hops and local CD updates). In this section, to allow for a fairer comparison with DCPA, where only communication rounds count as a round (despite the fact that we need to compute a proximal operator with a gradient descent subroutine), we count as an STCD iteration each hop in the network. We use the same definition of iteration for MTCD.

[2] Each entry of $\boldsymbol{X}$ drawn uniformly at random from $\{0, 1\}$ and each entry of $\boldsymbol{y}$ drawn from a standard normal distribution (in both cases the entries are independent).

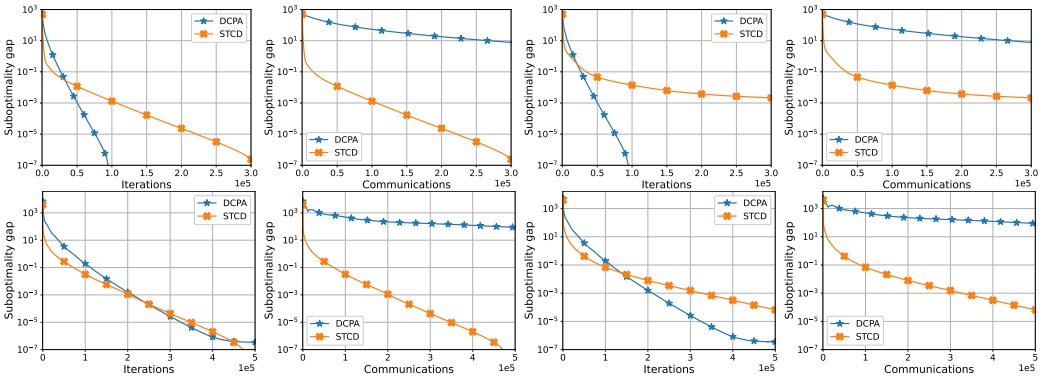

Figure 2: Suboptimality with respect to iterations and communications for a random network with $p = 0.4$ (four figures on the left) and a chain network (four figures on the right), both with $K = 20$. The top row concerns ridge regression and the bottom row concerns sparse logistic regression. Each communication is size N: STCD has one communication per iteration and DCPA has $2|\mathcal{E}|$.

Comparison with state-of-the-art. For ridge regression, we randomly generate a graph as in [36], with connectivity ratio $p = 0.4$, and a chain graph, both with $K = 20$ nodes and $d_k = 100$ for all k. For STCD, we use $\mu = 10^{-5}$ and $Q = 20$. For DCPA, we use $\mu_w = 0.01, \mu_y = 0.0003, \mu_x = 0.03$.

For sparse logistic regression, we similarly generate a random graph and chain graph, both with $K = 20$ nodes (now $d_k = 250$ for all k). For STCD, we use $\mu = 10^{-4}$ and $Q = 30$. For DCPA, we use $\mu_w = 0.001, \mu_y = 0.00003, \mu_x = 0.003$.

In Figure 2, we see that, while STCD does not improve upon DCPA in terms of progress per iteration, it significantly outperforms it in terms of communication efficiency. Yet, we can also see that STCD is particularly vulnerable to poorly connected networks, as evidenced by a performance deterioration when going from a random network to a chain network.

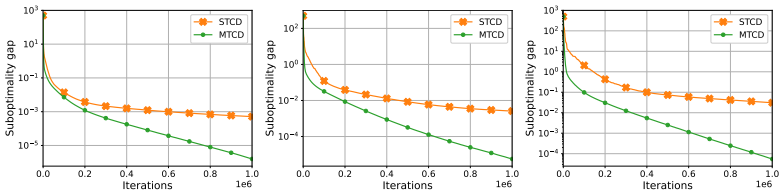

Figure 3: Suboptimality with respect to iterations for $K = 20, 40, 80$ chain networks (K increasing from left to right). All three plots concern the ridge regression model.

Comparing STCD with MTCD. We again deal with the ridge regression problem above and resort to the same dataset. However, we now focus on chain graphs (for MTCD, complemented by client-server communications), as they are poorly connected setups where the drawback of STCD is evident. We use $\mu = 10^{-5}$ and $Q = 20$ for both STCD and MTCD and use $\Gamma = 10, S = 10$ for the latter throughout all three experiments, varying only the number of nodes $K \in \{20, 40, 80\}$.

In Figure 3, we see that, while both algorithms see a drop in performance for graphs with a lower connectivity, MTCD mitigates this effect, consistently outperforming STCD.

5 Conclusions

We formalize the multi-token SDFL scheme and propose a communication efficient SDFL algorithm for feature-distributed data. Numerical results show the improved communication efficiency of our algorithm as well as the power of endowing decentralized methods with periodical client-server communications. A natural extension to this work is a detailed study of how the number of tokens and the frequency of their averaging influences the convergence rate, both empirically and analytically.

References

[1] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[2] Lie He, An Bian, and Martin Jaggi. COLA: Decentralized linear learning. *Advances in Neural Information Processing Systems*, 31, 2018.

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

[4] Sulaiman A. Alghunaim, Qi Lyu, Ming Yan, and Ali H. Sayed. Dual consensus proximal algorithm for multi-agent sharing problems. *IEEE Transactions on Signal Processing*, 69:5568–5579, 2021.

[5] Timothy J Castiglia, Anirban Das, Shiqiang Wang, and Stacy Patterson. Compressed-VFL: Communication-efficient learning with vertically partitioned data. In *International Conference on Machine Learning*, pages 2738–2766. PMLR, 2022.

[6] Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu. Can decentralized algorithms outperform centralized algorithms? A case study for decentralized parallel stochastic gradient descent. *Advances in Neural Information Processing Systems*, 30, 2017.

[7] Angelia Nedic and Asuman Ozdaglar. Distributed subgradient methods for multi-agent optimization. *IEEE Transactions on Automatic Control*, 54(1):48–61, 2009.

[8] J. C. Duchi, A. Agarwal, and M. J. Wainwright. Dual averaging for distributed optimization: Convergence analysis and network scaling. *IEEE Transactions on Automatic Control*, 57(3):592–606, 2012.

[9] Guannan Qu and Na Li. Harnessing smoothness to accelerate distributed optimization. *IEEE Transactions on Control of Network Systems*, 5(3):1245–1260, 2018.

[10] Anastasia Koloskova, Nicolas Loizou, Sadra Boreiri, Martin Jaggi, and Sebastian Stich. A unified theory of decentralized SGD with changing topology and local updates. In *International Conference on Machine Learning*, pages 5381–5393. PMLR, 2020.

[11] Angelia Nedic, Alex Olshevsky, and Michael G. Rabbat. Network topology and communication-computation tradeoffs in decentralized optimization. *Proceedings of the IEEE*, 106(5):953–976, 2018.

[12] Haoyu Zhao, Boyue Li, Zhize Li, Peter Richtárik, and Yuejie Chi. BEER: Fast $O(1/T)$ Rate for Decentralized Nonconvex Optimization with Communication Compression. *arXiv:2201.13320*, 2022.

[13] Dimitri P Bertsekas. A new class of incremental gradient methods for least squares problems. *SIAM Journal on Optimization*, 7(4):913–926, 1997.

[14] Angelia Nedic and Dimitri P Bertsekas. Incremental subgradient methods for nondifferentiable optimization. *SIAM Journal on Optimization*, 12(1):109–138, 2001.

[15] S. Sundhar Ram, A. Nedić, and V. V. Veeravalli. Incremental stochastic subgradient algorithms for convex optimization. *SIAM Journal on Optimization*, 20(2):691–717, 2009.

[16] Björn Johansson, Maben Rabi, and Mikael Johansson. A randomized incremental subgradient method for distributed optimization in networked systems. *SIAM Journal on Optimization*, 20(3):1157–1170, 2010.

[17] Xianghui Mao, Kun Yuan, Yubin Hu, Yuantao Gu, Ali H. Sayed, and Wotao Yin. Walkman: A communication-efficient random-walk algorithm for decentralized optimization. *IEEE Transactions on Signal Processing*, 68:2513–2528, 2020.

[18] Hadrien Hendrikx. A principled framework for the design and analysis of token algorithms. *arXiv:2205.15015*, 2022.

[19] Yu Ye, Hao Chen, Zheng Ma, and Ming Xiao. Decentralized consensus optimization based on parallel random walk. *IEEE Communications Letters*, 24(2):391–395, 2020.

[20] Hao Chen, Yu Ye, Ming Xiao, and Mikael Skoglund. Asynchronous parallel incremental block-coordinate descent for decentralized machine learning. *arXiv:2202.03263*, 2022.

[21] Frank Po-Chen Lin, Seyyedali Hosseinalipour, Sheikh Shams Azam, Christopher G. Brinton, and Nicolo Michelusi. Semi-decentralized federated learning with cooperative D2D local model aggregations. *IEEE Journal on Selected Areas in Communications*, 39(12):3851–3869, 2021.

[22] Yuanxiong Guo, Ying Sun, Rui Hu, and Yanmin Gong. Hybrid local SGD for federated learning with heterogeneous communications. In *International Conference on Learning Representations*, 2021.

[23] Jieming Bian and Jie Xu. Mobility improves the convergence of asynchronous federated learning. *arXiv:2206.04742*, 2022.

[24] Michal Yemini, Rajarshi Saha, Emre Ozfatura, Deniz Gunduz, and Andrea J. Goldsmith. Semi-decentralized federated learning with collaborative relaying. In *2022 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2022.

[25] Xueqing Zhang, Yanwei Liu, Jinxia Liu, Antonios Argyriou, and Yanni Han. D2D-assisted federated learning in mobile edge computing networks. In *2021 IEEE Wireless Communications and Networking Conference (WCNC)*, 2021.

[26] Seyyedali Hosseinalipour, Sheikh Shams Azam, Christopher G. Brinton, Nicolo Michelusi, Vaneet Aggarwal, David J. Love, and Huaiyu Dai. Multi-stage hybrid federated learning over large-scale D2D-enabled fog networks. *IEEE/ACM Transactions on Networking*, 30(4):1569–1584, 2022.

[27] Stephen J. Wright. Coordinate descent algorithms. *Mathematical Programming*, 151(1):3–34, 2015.

[28] Amir Beck and Luba Tetruashvili. On the convergence of block coordinate descent type methods. *SIAM Journal on Optimization*, 23(4):2037–2060, 2013.

[29] Yu. Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362, 2012.

[30] Peter Richtárik and Martin Takáč. Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. *Mathematical Programming*, 144(1-2):1–38, 2012.

[31] Tao Sun, Yuejiao Sun, Yangyang Xu, and Wotao Yin. Markov chain block coordinate descent. *Computational Optimization and Applications*, 75(1):35–61, 2019.

[32] Olivier Fercoq and Peter Richtárik. Accelerated, parallel, and proximal coordinate descent. *SIAM Journal on Optimization*, 25(4), 2015.

[33] Yang Liu, Xinwei Zhang, Yan Kang, Liping Li, Tianjian Chen, Mingyi Hong, and Qiang Yang. FedBCD: A communication-efficient collaborative learning framework for distributed features. *IEEE Transactions on Signal Processing*, 70:4277–4290, 2022.

[34] Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[35] Isabelle Guyon, Steve Gunn, Asa Ben-Hur, and Gideon Dror. Result analysis of the NIPS 2003 feature selection challenge. *Advances in neural information processing systems*, 17, 2004.

[36] Wei Shi, Qing Ling, Kun Yuan, Gang Wu, and Wotao Yin. On the linear convergence of the ADMM in decentralized consensus optimization. *IEEE Transactions on Signal Processing*, 62(7):1750–1761, 2014.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] The code and data will be made available in a later submission.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [No]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [No]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]