2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS) | 978-1-7281-7002-2/20/$31.00 ©2020 IEEE | DOI: 10.1109/ICDCS47774.2020.00072

SNAP: A Communication Efficient Distributed Machine Learning Framework for Edge Computing

Yangming Zhao[1], Jingyuan Fan[1], Lu Su[1], Tongyu Song[2], Sheng Wang[2] and Chunming Qiao[1]
[1]Department of Computer Science and Engineering, University at Buffalo
[2]School of Information and Communication Engineering
University of Electronic Science and Technology of China

***Abstract*—More and more applications learn from the data collected by the edge devices. Conventional learning methods, such as gathering all the raw data to train an ultimate model in a centralized way, or training a target model in a distributed manner under the parameter server framework, suffer a high communication cost. In this paper, we design Select Neighbors and Parameters (SNAP), a communication efficient distributed machine learning framework, to mitigate the communication cost. A distinct feature of SNAP is that the edge servers act as peers to each other. Specifically, in SNAP, every edge server hosts a copy of the global model, trains it with the local data, and periodically updates the local parameters based on the weighted sum of the parameters from its neighbors (*i.e.*, peers) only (*i.e.*, without pulling the parameters from all other edge servers). Different from most of the previous works on consensus optimization in which the weight matrix to update parameter values is predefined, we propose a scheme to optimize the weight matrix based on the network topology, and hence the convergence rate can be improved. Another key idea in SNAP is that only the parameters which have been changed significantly since the last iteration will be sent to the neighbors. Both theoretical analysis and simulations show that SNAP can achieve the same accuracy performance as the centralized training method. Compared to the state-of-the-art communication-aware distributed learning scheme TernGrad, SNAP incurs a significantly lower (99.6% lower) communication cost.**

I. Introduction

With the rapid advancement of Internet of Things (IoT) and the increasing popularity of mobile devices, an exponential growth of data are generated at the network edge [1]. Conventionally, we gather these data collected by the edge devices to a data center and train a Machine Learning (ML) model for different applications, such as computer vision [2], [3], speech recognition [4], [5] and disease diagnosis [6]. However, it has been predicted that the data generation rate will exceed the capacity of today's Internet in the near future [7]. Due to the limitation in network bandwidth and concern about data privacy, many researchers have proposed to leverage the emerging technology of Mobile Edge Computing (MEC) to deal with the data collected by the edge devices [8], [9].

In MEC systems, some edge servers are collocated with base stations to provide computation power, and each of these edge servers can host a copy of the target ML model and train it with the local data. To ensure that the target model adapts to the data collected by all the edge devices, the edge servers should exchange their local parameters. A common technique to exchange the local information is based on the parameter server model [10]. In this model, edge servers send the gradients of every parameter to one or more global parameter servers. Then, the parameter servers mix all these gradients, updates parameter values and pushes them back to the edge servers. This scheme can avoid sending a large amount of the raw data to a centralized server. However, it still introduces considerable a communication overhead, since the number of parameters can be large. For instance, when training a 3-layer neural network with hundreds of inputs, hundreds of perceptrons in the hidden layer and tens of outputs, there would be $\sim 10^5$ parameters. If each parameter is a 8-byte number and there are tens of edge servers, there would be $\sim 10^{10}$ bytes injected into the network within tens of iterations!

The above-mentioned communication overhead incurred by distributed learning could cause serious problems in edge computing scenario, especially in MEC where some of the edge servers are connected to each other using wireless links. Furthermore, when an edge server is selected as a parameter server to collect the parameter updates from other servers, the incast problem may occur. In addition, there are usually multiple physical hops from an edge server to a selected parameter server. The amount of network-wide bandwidth consumed by the distributed learning could be significant.

This work is motivated by the following question: can we design a distributed learning framework such that: 1) every edge server keeps the raw data to itself so as to preserve data privacy as much as possible; 2) there is no parameter server and the edge servers operate in a peer-to-peer manner in order to avoid the incast problem; and 3) the framework can achieve the same accuracy performance as centralized training? In this paper, we will propose Select Neighbors and Parameters (SNAP), a communication efficient distributed machine learning framework, to train the ML model based on the distributed data collected by edge servers. In SNAP, every edge server hosts a copy of the global model and trains it with its local data, and periodically exchanges the parameters with only a few other edge servers (called its neighbors). Each edge server also updates its local parameters based on the weighted sum of the parameters from its neighbors. To save the communication cost, edge servers only exchange part of the parameters but ignoring those with little change during last iteration. As far as we know, SNAP is the first Distributed Machine Learning (DML) framework with the desirable features mentioned above.

2575-8411/20/$31.00 ©2020 IEEE
DOI 10.1109/ICDCS47774.2020.00072

The main contributions of this paper can be summarized as follows:

- A communication efficient DML algorithm named SNAP
- A scheme to optimize the weight matrix used to update the parameter values in SNAP
- Implement SNAP on a small scale testbed
- Extensive simulations to show the effectiveness of SNAP

The remainder of the paper is organized as follows: in Section II, we briefly review the related works and present the system model of SNAP. We then formulate the problem to solve in SNAP in Section III. All the detailed algorithms in SNAP, such as how to exchange local parameters, how to optimize the weight matrix and how to reduce the communication cost, are discussed in Section IV. Then, extensive performance evaluations based on both testbed experiments and simulations are conducted in Section V to show the effectiveness of SNAP. We conclude this paper in Section VI.

II. Background and System Model

A. Previous Works

There are a number of related works on DML. We review the most closely related ones below.

Parameter server framework: In parameter server framework, every worker hosts a copy of the ML model, and trains its local copy of the ML model based on local data. The parameter servers collect the gradients [11], [12] or local parameters [13] derived by the workers, update the parameter values and push them back to the workers. Then, the workers continue the further iterations from the updated parameter values. All these works focus on how to speed up the algorithm convergence, and do not consider optimizing the communication cost. More seriously, they may incur incast problem in edge computing scenarios.

Consensus optimization: Another type of DML schemes are based on consensus optimization [14], [15], [16], [17], [18]. With algorithms belonging to this class, every server exchanges local parameters with other servers, till they achieve an agreement on the parameters. These schemes do not need parameter servers. Again, none of them consider how to reduce the communication overhead.

Communication overhead reduction: TernGrad [19] uses only 2 bits to encode the gradients sent in the worker-to-server direction. Unfortunately, it reduces the algorithm convergence rate, and suffers from accuracy degradation. [20] drops some of the small data when exchanging the parameters based on a heuristic method without performance guarantee.

Weight matrix optimization: In SNAP, a weight matrix is leveraged to control the parameter averaging operation. This is used in many multi-agent collaboration works [18], [21], [22]. Most of these works directly define the weight matrix without optimizing it [18], [21]. As far as we know, [22] is the only work that optimizes the weight matrix based on semidefinite programming (SDP). However, SDP cannot be used in SNAP since the feasible weight matrix is not a linear combination of semidefinte matrices.

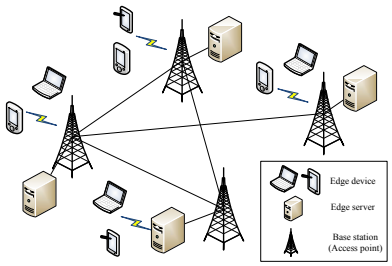


Fig. 1. System model of SNAP.

B. System Model

In this paper, we consider a system shown in Fig. 1. In SNAP, the mobile edge devices, such as tablets, cell phones and smart rings, collect data and send them to the edge servers collocated with the base stations. With these data, edge servers collaborate on training a *uniform* ML model. Different from conventional methods that deliver the raw data to a remote cloud, the data collected by each edge server will not be shared with others in SNAP, which can achieve data privacy and reduce communication overhead.

In SNAP, each edge server hosts a copy of the uniform model, and trains the model with its local data. To achieve a consensus about the parameters, each edge server exchanges its local parameters with a predefined set of other edge servers, which are referred as its *neighbors*. In other words, each edge server sends parameters to its peers only, instead of some parameter servers as in other DML approaches. For edge server i, its neighbors can be the edge servers collocated with the base stations directly connected with i's base station through one hop wireless channel, or the base stations connected with i's base station through a persistent TCP connection in a wired network. The *node degree* of each server is defined as the number of neighbors it has. We assume that the edge servers share a global clock and exchange the parameters with its neighbors similar to how RIP [23] works for Internet routing (with the main difference being that in SNAP, only select parameters are exchanged instead of the entire distance vector).

Two main performance objectives are pursued: 1) high accuracy as the centralized training; 2) even lower communication cost than existing distributed learning approaches. In SNAP, communication cost is defined as the total traffic amount carried by the network. If a flow traverses h hops of physical links in the network, the communication cost incurred by this flow would be h times of the flow size.

III. Problem Definition

In this section, we formulate the DML problem. In Section III-A, we first introduce the key notations used in this paper; then, in Section III-B the mathematical formulation is presented.

A. Notations

In this paper, we assume there are N edge servers that are collaborating for model training. $\boldsymbol{S}$ is the set of all edge servers, and $\boldsymbol{B}_i$ is the set of edge server i's neighbors. Edge

server $i \in \boldsymbol{S}$ hosts a local copy of the model parameter x, which is noted by a column vector $x_{(i)} \in \mathbb{R}^P$. Since the edge servers need many iterations to achieve the global consensus, we assume $x_{(i)}^k$ to be the parameters hosted by edge server i after the k^{th} iteration.

To simplify the presentation, we define

$$\boldsymbol{x} \triangleq \begin{pmatrix} - & x_{(1)}^T & - \\ - & x_{(2)}^T & - \\ & \vdots & \\ - & x_{(N)}^T & - \end{pmatrix} \in \mathbb{R}^{N \times P}$$

Then, given an aggregate function $f(\boldsymbol{x}) \triangleq \sum_{i=1}^N f_i(x_{(i)})$ its gradient is defined as

$$\nabla f(\boldsymbol{x}) \triangleq \begin{pmatrix} - & \nabla^T f_1(x_{(1)}) & - \\ - & \nabla^T f_2(x_{(2)}) & - \\ & \vdots & \\ - & \nabla^T f_N(x_{(N)}) & - \end{pmatrix} \in \mathbb{R}^{N \times P}$$

When all the edge servers achieve *consensus*, all the rows in $\boldsymbol{x}$ should be identical, *i.e.*, $x_{(1)} = x_{(2)} = \cdots = x_{(N)}$.

For the self-consistence, we also list some of the algebraic definitions that will be used in this paper here. For a matrix A, we use a_{ij} to denote the element at its i^{th} row and j^{th} column. Its G-matrix norm is defined as $||A||_G \triangleq \sqrt{trace(A^T G A)}$, where G is a *symmetric and positive semidefinite* matrix. Its largest singular value is denoted as $\sigma_{max}(A)$. The largest and smallest eigenvalues of matrix A are $\lambda_{max}(A)$ and $\lambda_{min}(A)$, respectively. In addition, $\bar{\lambda}_{max}(A)$ is the largest eigenvalues of matrix A that is smaller than 1, while $\bar{\lambda}_{min}(A)$ is the smallest eigenvalues of matrix A that is larger than 0. Usually, we use S_n denote the set of all the $n \times n$ doubly stochastic matrices.

B. Problem Formulation

In our system, all the edge servers are working on the same ML model. Therefore, they adopt the same cost function $c(x_{(i)})$. Since edge servers are training the model based on different data sets, for edge server i, it should minimize

$$l_i = \mathbb{E}_{\xi \sim D_i} c(x_{(i)}; \xi)$$

where D_i is the set of data collected by edge server i. Then, SNAP is to minimize the aggregate cost function

$$L = \sum_{i=1}^N l_i = \sum_{i=1}^N \mathbb{E}_{\xi \sim D_i} c(x_{(i)}; \xi)$$

Because $c(x_{(i)}; \xi)$ $(i = 1, 2, \ldots, N)$ are calculated based on different sets of data, we can treat the cost function to different edge servers as specific ones, *i.e.*,

$$f_i(x_{(i)}) = l_i = \mathbb{E}_{\xi \sim D_i} c(x_{(i)}; \xi)$$

Then, the aggregate cost function is

$$f(\boldsymbol{x}) = \sum_{i=1}^N f_i(x_{(i)}) \tag{1}$$

Since all the edge servers are collaborating to train the uniform model based on their own local data, there will inevitably be communications among them. Say d_{ij}^k is the amount of data sent from edge server i to edge server j in iteration k, the total traffic during the training procedure is

$$C = \sum_{k=0}^K \sum_{i,j} d_{ij}^k \tag{2}$$

In addition, all the local model hosted by every edge server should converge to the same solution (a.k.a achieve consensus), *i.e.*,

$$x_{(1)} = x_{(2)} = \cdots = x_{(N)} \tag{3}$$

In SNAP, we have two objectives to pursue. First, we should minimize the loss function, which problem can be formulated as

$$\begin{aligned} \text{minimize} \quad & f(\boldsymbol{x}) = \sum_{i=1}^N f_i(z) \\ \text{subject to:} \quad & x_{(i)} = z \quad \forall i = 1, 2, \cdots, N \end{aligned} \tag{4}$$

Since we do not allow raw data transfer in SNAP to keep privacy, this problem should be solved in a distributed manner. During we solve the problem (4), the second objective, *i.e.*, minimize the communication cost to derive the optimal solution, should be pursued. This problem can be formulated as

$$\text{minimize} \quad C = \sum_{k=0}^K \sum_{i,j} d_{ij}^k \tag{5}$$

Accordingly, SNAP contains not only a distributed learning iteration algorithm to derive the optimal parameters for the machine learning model, but also a scheme to save the communication cost during each iteration. We will discuss them in detail in the following section.

IV. SNAP in Detail

Since SNAP leverages the same procedure as in EXTRA [18], we briefly introduce this consensus optimization framework in Section IV-A. In EXTRA, the weight matrix used to update the parameters significantly impacts the algorithm performance in terms of convergence rate, and hence we discuss how to optimize the weight matrix in Section IV-B. Then, we propose a scheme to reduce the communication cost during the iteration procedure in Section IV-C. At last, we address some issues for implementing SNAP in real systems in Section IV-D.

A. EXTRA Iteration

SNAP inherits the consensus optimization algorithm EXTRA [18], in which local parameters kept by the edge servers are updated according to the following matrix operation:

$$\begin{aligned} \boldsymbol{x}^1 =& W\boldsymbol{x}^0 - \alpha \nabla f(\boldsymbol{x}^{\boldsymbol{0}}) \\ \boldsymbol{x}^{k+2} =& (W + I)\boldsymbol{x}^{k+1} - \widetilde{W}\boldsymbol{x}^k \\ & - \alpha(\nabla f(\boldsymbol{x}^{k+1}) - \nabla f(\boldsymbol{x}^k)) \end{aligned} \tag{6}$$

where $\boldsymbol{x}^{\mathbf{0}}$ is the initial parameters and W is a *symmetric doubly stochastic* matrix, *i.e.*, $W = W^T$, $W\mathbf{1} = \mathbf{1}$. If $j \notin \boldsymbol{B}_i$, there should be $w_{ij} = 0$. In addition, $\widetilde{W}$ is defined as

$$\widetilde{W} = \frac{W + I}{2} \tag{7}$$

Based on the matrix operations in (6), each edge serve i performs the following operation in each iteration

$$\begin{aligned} x_{(i)}^1 =& \sum_{j=1}^{N} w_{ij} x_{(j)}^0 - \alpha \nabla f_i(x_{(i)}^0) \\ x_{(i)}^{k+2} =& x_{(i)}^{k+1} + \sum_{j=1}^{N} w_{ij} x_{(j)}^{k+1} - \sum_{j=1}^{N} \widetilde{w}_{ij} x_{(j)}^k \\ & - \alpha(\nabla f_i(x_{(i)}^{k+1}) - \nabla f_i(x_{(i)}^k)) \end{aligned} \tag{8}$$

we can see that each edge server can update its local parameters only based on the parameters hosted by its neighboring edge servers and its local data. However, it should be noted that at the $(k+2)^{th}$ iteration, the operation performed by edge server i not only depends on parameters from its neighbors at the $(k+1)^{th}$ iteration, *i.e.*, $x_{(j)}^{k+1}$, but also their values at the k^{th} iteration, *i.e.*, $x_{(j)}^k$. To implement the iteration presented above, an important issue is that how to determine the matrix W. We will discuss this in detail in Section IV-B.

From [18], we know the following results:

- Suppose the iteration of (6) converges to $\boldsymbol{x}^*$, $\boldsymbol{x}^*$ is the optimal solution of (4).
- If there exists $q^* = Up$ for some $p \in \mathbb{R}^{N \times P}$, such that

$$\begin{cases} U\boldsymbol{x}^* = 0 \\ Uq^* + \alpha \nabla f(\boldsymbol{x}^*) = 0 \end{cases}$$

  $\boldsymbol{x}^*$ is a consensual optimal solution to problem (4).
- Let $\boldsymbol{x}^*$ and q^* satisfy the conditions specified by last bullet, we introduce auxiliary sequence $q^k = \sum_{t=0}^{k} U\boldsymbol{x}^t$ and define

$$\boldsymbol{z}^k = \begin{pmatrix} q^k \\ x^k \end{pmatrix}, \quad \boldsymbol{z}^* = \begin{pmatrix} q^* \\ x^* \end{pmatrix}, \quad G = \begin{pmatrix} I & \mathbf{0} \\ \mathbf{0} & \widetilde{W} \end{pmatrix}.$$

  With an iteration step size α satisfying $0 \le \alpha < \frac{2\lambda_{min}(\widetilde{W})}{L_f}$, we have

$$||\boldsymbol{z}^k - \boldsymbol{z}^*||_G^2 - ||\boldsymbol{z}^{k+1} - \boldsymbol{z}^*||_G^2 \ge \zeta ||\boldsymbol{z}^k - \boldsymbol{z}^{k+1}||_G^2, \quad \forall k \tag{9}$$

  where $\zeta = 1 - \frac{\alpha L_f}{2\lambda_{min}(\widetilde{W})}$.

Based on above results, we can propose

Theorem 1. *If the objective function of each edge server i is convex, $\boldsymbol{z}^k$ converges to an optimal $\boldsymbol{z}^*$.*

It is worth noting that though the proof of Theorem 1 can be found in [18], it is based on an incorrect citation. Accordingly, we present our proof here.

Proof: Assume $\boldsymbol{z}^k - \boldsymbol{z}^{k+1}$ does not converge to 0, *i.e.*, there are infinite number of k such that $||\boldsymbol{z}^k - \boldsymbol{z}^{k+1}||_G^2 > \epsilon$ for a given $\epsilon > 0$ when $k \to \infty$. By adding (9) for different k, we can see that $\lim_{k\to\infty} ||\boldsymbol{z}^k - \boldsymbol{z}^*||_G^2 \to \infty$.

On the other hand, since W is doubly stochastic, we know $w_{ij} < 1$. In addition, when $\boldsymbol{z}^k$ converges, $\nabla f(x) \to 0$. Then, from the definition of $\boldsymbol{z}^k$, we know that $\boldsymbol{z}^k$ should be bounded, which contradicts with $\lim_{k\to\infty} ||\boldsymbol{z}^k - \boldsymbol{z}^*||_G^2 \to \infty$. Accordingly, there must be $\lim_{k\to\infty} ||\boldsymbol{z}^k - \boldsymbol{z}^{k+1}||_G^2 \to 0$.

Since $||\boldsymbol{z}^k - \boldsymbol{z}^{k+1}||_G^2$ converges to 0, there must be some l such that for all $k > l$, $||\boldsymbol{z}^k - \boldsymbol{z}^*||_G^2 \ge ||\boldsymbol{z}^{k+1} - \boldsymbol{z}^*||_G^2$. From the convexity of f_i, $f(\cdot)$ in (4) is also convex. In addition, Problem (4) has a nonempty optimal solution set, and hence we know $\boldsymbol{z}^*$ should be the cluster point of the iteration. Then, we conclude that $\lim_{k\to\infty} \boldsymbol{z}^k = \boldsymbol{z}^*$ ■

Based on Theorem 1, we know

Corollary 1. *If the objective function of each edge server i is convex, $\boldsymbol{x}^k$ converges to an optimal $\boldsymbol{x}^*$.*

B. Weight Matrix Optimization

Though SNAP can derive an optimal consensus solution in a distributed manner, the weight matrix in iteration (6) would significantly impact the convergence rate. However, in most of the previous works in multi-agent collaboration [18], [21], the weight matrix is predefined without any optimization. In this section, we will discuss how to design this weight matrix to improve the convergence rate. It is worth noting that if we get $w_{ij} = 0$ in the optimized weight matrix, edge server i and j does not need to exchange parameters during the training procedure, which can further reduce the communication cost.

According to the equation (3.38) in [18], by introducing

$$g(x) = f(x) + \frac{1}{4\alpha} ||x||_{\widetilde{W}-W}^2 \tag{10}$$

if $g(x)$ is strongly convex with respect to x^* with constant $\mu_g > 0$, then with proper step size $\alpha < \frac{2\mu_g \lambda_{min}(\widetilde{W})}{L_f^2}$ the convergence rate of iteration (6) is $O((1+\delta)^k)$, where

$$\begin{aligned} &\delta \le \min \\ &\{\frac{\alpha(2\mu_{\boldsymbol{g}} - \eta)\bar{\lambda}_{min}(\widetilde{W} - W)}{\theta(\sigma_{max}(I + W - 2\widetilde{W}) + \alpha L_f)^2 + \lambda_{max}(\widetilde{W})\bar{\lambda}_{min}(\widetilde{W} - W)}, \\ &\frac{(\theta - 1)(\eta\lambda_{min}(\widetilde{W}) - \alpha L_f^2)\bar{\lambda}_{min}(\widetilde{W} - W)}{\theta\eta(\sigma_{max}(\widetilde{W}) + \alpha L_f)^2}\} \end{aligned} \tag{11}$$

θ and η are two parameters satisfying $\theta > 1$ and $\eta \in (0, 2\mu_g)$. Now, we try to simplify the bound of δ based on the settings in SNAP. At first, W is doubly stochastic matrix, namely, $W\mathbf{1} = \mathbf{1}$. Accordingly, 1 is one of the eigenvalues of matrix W. In addition, W is symmetric, we know $\mathbf{1}^T W = \mathbf{1}^T$. Thus, $|\lambda_{max}(W)| = 1$. Then, we know

$$\lambda_{max}(W) = 1 \tag{12}$$

and

$$\lambda_{max}(\widetilde{W}) = (\lambda_{max}(W) + 1)/2 = 1 \tag{13}$$

In SNAP, we set $\widetilde{W} = \frac{W+I}{2}$, and thus

$$I + W - 2\widetilde{W} = 0 \tag{14}$$

and

$$\widetilde{W} - W = \frac{I - W}{2} \tag{15}$$

Based on the fact that W is symmetric, then $\widetilde{W}$ must also be symmetric. Accordingly, we know

$$\sigma_{max}(\widetilde{W}) = |\lambda_{max}(\widetilde{W})| = 1 \tag{16}$$

By substituting (12), (13), (14) and (16) into (11), it can be simplified as

$$\begin{aligned}\delta \leq \min\{&\frac{\alpha(2\mu_{\boldsymbol{g}} - \eta)\bar{\lambda}_{min}(I-W)}{2\theta\alpha^2 L_f^2 + \bar{\lambda}_{min}(I-W)}, \\ &\frac{(\theta-1)(\eta + \eta\lambda_{min}(W) - 2\alpha L_f^2)\bar{\lambda}_{min}(I-W)}{4\theta\eta(1+\alpha L_f)^2}\}\end{aligned} \tag{17}$$

To speed up the convergence, we should maximize δ. To this end, we study the two terms in the "min" operator, separately. For the first term, it can be reformulated as

$$\begin{aligned}&\frac{\alpha(2\mu_{\boldsymbol{g}} - \eta)\bar{\lambda}_{min}(I-W)}{2\theta\alpha^2 L_f^2 + \bar{\lambda}_{min}(I-W)} \\ =&\frac{\alpha(2\mu_{\boldsymbol{g}} - \eta)}{\frac{2\theta\alpha^2 L_f^2}{\bar{\lambda}_{min}(I-W)} + 1}\end{aligned} \tag{18}$$

To maximize (18), we have to maximize $\bar{\lambda}_{min}(I-W)$. $I-W$ is positive definite, thus, we know $\lambda_{min}(I-W) \geq 0$. In addition, $\lambda_{max}(W) = 1$, hereby, we have $\bar{\lambda}_{min}(I-W) = 1 - \bar{\lambda}_{max}(W)$. Based on above discussions, we can simplify the problem to optimize the weight matrix as

$$\text{minimize} \qquad \bar{\lambda}_{max}(W) \tag{19}$$

Consider the second term, we are actually to maximize $(\eta - 2\alpha L_f^2 + \eta\lambda_{min}(W))\bar{\lambda}_{min}(I-W)$, which indicates that we should pursue following two objectives

$$\begin{aligned}\text{minimize} \quad & \bar{\lambda}_{max}(W) \\ \text{maximize} \quad & \lambda_{min}(W)\end{aligned} \tag{20}$$

Comparing (19) and (20), we find that (19) is only a subproblem in (20), and hence we only consider (20) hereafter. Since it is difficult to find a weight matrix that can optimize both objectives in (20), to solve this problem, we consider them separately and derive two weight matrices. Then, we implement the solution that can result in the larger convergence rate in SNAP. Following the thought of this line, we solve following two optimization problems:

$$\begin{aligned}\text{minimize} \quad & \bar{\lambda}_{max}(W) \\ \text{subject to} \quad & W \in S_N \\ & w_{ij} = 0 \quad \forall j \notin \boldsymbol{B}_i\end{aligned} \tag{21}$$

and

$$\begin{aligned}\text{maximize} \quad & \lambda_{min}(W) \\ \text{subject to} \quad & W \in S_N \\ & w_{ij} = 0 \quad \forall j \notin \boldsymbol{B}_i\end{aligned} \tag{22}$$

Since $\lambda_{max}(W) = 1$, we can modify problem (21) to be

$$\begin{aligned}\text{minimize} \quad & \lambda_{max}(W) + \bar{\lambda}_{max}(W) \\ \text{subject to} \quad & W \in S_N \\ & w_{ij} = 0 \quad \forall j \notin \boldsymbol{B}_i\end{aligned} \tag{23}$$

According to [24], we know that the objective function of problem (22) is a concave function on W, while the objective function of (23) is a convex function on W [25]. In addition, we can prove following theorem:

Theorem 2. *The feasible set of problems* (22) *and* (23) *is a convex set.*

Proof: Suppose W_1 and W_2 are two different feasible solutions to problems (22) and (23), μ is a parameter on $[0, 1]$, and $W = \mu W_1 + (1-\mu)W_2$. Then, we know

$$W^T = \mu W_1^T + (1-\mu)W_2^T = \mu W_1 + (1-\mu)W_2 = W$$

In addition,

$$W\mathbf{1} = \mu W_1\mathbf{1} + (1-\mu)W_2\mathbf{1} = \mu\mathbf{1} + (1-\mu)\mathbf{1} = \mathbf{1}$$

Accordingly, W is also double stochastic, *i.e.* $W \in S_N$.

When $j \notin \boldsymbol{B}_i$, we know $\{w_1\}_{ij} = 0$ and $\{w_2\}_{ij} = 0$. Therefore, $w_{ij} = \mu\{w_1\}_{ij} + (1-\mu)\{w_2\}_{ij} = 0$. So, we can conclude that W is also in the feasible set of problems (22) and (23), *i.e.*, The feasible set of problems (22) and (23) is a convex set. ■

Based on the convexity of objective functions of problems (22) and (23), and Theorem 2, we know

Theorem 3. *Both Problems* (22) *and* (23) *are convex optimization problems.*

According to Theorem 3, we can leverage interior-point method [26] to solve Problems (22) and (23). The only issue to implement interior-point method is how to set the initial point, which must be a feasible solution to both problems, to start the iteration. To address this issue, we construct an auxiliary graph on which each vertex presents an edge server, and there is an edge between vertex i and j if and only if $j \in \boldsymbol{B}_i$. Then, we can initialize the weight matrix according to

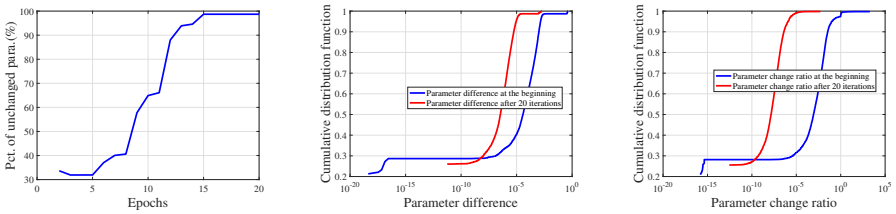
$$w_{ij} = \begin{cases} \frac{1}{max\{deg(i), deg(j)\}+\epsilon}, & \text{if } j \in \boldsymbol{B}_i \\ 0, & \text{if } j \notin \boldsymbol{B}_i \text{ and } i \neq j \\ 1 - \sum_{k \in \boldsymbol{S}/j} w_{ik}, & \text{if } i = j \end{cases} \tag{24}$$

for some small positive ϵ, where $deg(i)$ is the node degree of server i. It is easy to check that the weight matrix defined by (24) is symmetric and doubly stochastic.

C. Communication Cost Reduction

With the iteration presented in (6), it may introduce significant communication cost into the system. For example, even in a very small scale neural network, there would be millions of parameters, each of which may be presented by a 8-byte number. If there are tens of edge servers collaborating for the learning; each edge server has 4 neighbors on average; and the algorithm converges in 100 iterations; then, there would be tens of gigabytes data injected into the network. Accordingly, we have to reduce the communication cost.

To this end, we first implement a DML system, in which each server performs the operation in (8). Based on this system, we first study how the parameters hosted by each edge server evolve, and then propose an algorithm to reduce the communication cost.



(a) Percentage of unchanged parameters. (b) Log-CDF of parameter difference. (c) Log-CDF of parameter change ratio.

Fig. 2. How weights change during the iteration.

1) Parameter Evolution Characteristics: If there are some parameters without any changes or with only slight changes during one iteration, these parameters do not need to be sent to its neighbors. This should be a good way to save the communication cost. Intuitively, when the iteration process is close to convergence, the parameters on each edge server would not change a lot. Accordingly, we can reduce communication cost during the last few iterations.

However, it may not be enough only saving the communication at the last few iterations. Therefore, we are also interested in if we can save the communication cost at the early stage of the learning procedure. To find the potential to save communication cost at the early stages, we implement the iteration (8) in a toy network with 3 servers to train a conventional 3-layer full mesh neural network with 30 perceptrons in the hidden layer. We use the MNIST [27] data set as input. To emulate the distributed data collection, we randomly allocate each training sample to one of these 3 servers. We investigate how the variables evolve in such a small scale network.

During the iterations, we record 3 criteria: 1) the number of parameters that have not changed at all; 2) the *parameter difference*, which is defined as $D(x^k) = |x^{k+1} - x^k|$ and 3) the *parameter change ratio* which is defined as $R(x^k) = \frac{|x^{k+1} - x^k|}{|x^k|}$ during each iteration. We show how these criteria evolve during the iteration in Fig. 2. For clear presentation, we only show the CDFs of parameter difference and parameter change ratio in the first iteration and those after 20 iterations. From this figure, we can see that 1) there are more than 30% of the parameters remaining the same in an iteration even at the beginning of the training process, and this percentile increases to 50% after 10 iterations and 98% after 15 iterations; 2) even in the first iteration, more than 90% of the parameter differences are less than 10^{-3} (Fig. 2(b)); and 3) more than 94% of the parameters change less than 10% (Fig. 2(c)) during one iteration. Accordingly, if each edge server maintains the parameter values from neighbors in previous iteration, the parameters with little or no changes do not need to be sent out once more, such that the communication cost can be reduced.

In addition, fewer and fewer parameters change significantly with the training process. After 20 iterations, more than 98% of the parameter differences are smaller than 10^{-4} during each iteration and almost all the parameter change ratios are smaller than 0.1. This demonstrates the large potential to reduce communication cost. Based on above observations, we can get two takeaways: 1) even at the early stage of the algorithm iteration, there are lots of local parameters that do not need to be sent to the neighbors; 2) as the algorithm goes through more and more iterations, fewer and fewer parameters need to be sent to the neighbors with iterations. In the following, we will discuss how to determine which parameters should be sent to neighbors in order to save the communication cost.

Frame structure 1

# of var keep the same	Index of variables that keep the same	Value of variables that need update

Frame structure 2

Index	Value	Index	Value	·····	Index	Value

Fig. 3. Candidate frame structures in SNAP.

2) Algorithm to Reduce Communication Cost: Based on above discussions, the key idea to save communication cost in SNAP is to reduce the number of local parameters sent by each edge server to its neighbors. More specifically, we will set a parameter change threshold such that parameters whose changes in value fall below the threshold will not be sent while those whose changes in value exceed the threshold will be sent. Without receiving some of the updated parameters from a neighbor j, an edge server will use the latest values of those parameters from edge server j for its next iteration. Following this idea, we should focus on two questions: 1) when an edge server receives parameter update from one of its neighbors, how to find out which parameters are updated, *i.e.*, the communication protocol issue; 2) how an edge server determines which parameters do not need to be sent to its neighbors, without significantly hurting the learning performance.

Transmission Protocol: In SNAP, two candidate frame structures as shown in Fig. 3 can be used for the parameter transmission. In one frame structure, we first send the number of unchanged parameters along with their indexes (or identifiers), and then, the remaining parameters need to be updated. In the other frame structure, every updated parameter is sent following its index (or identifier). For a server hosting N parameters and M of which will not be sent to its neighbors, $4 + 8N - 4M$ bytes are in the first type of frame, while $12(N - M)$ bytes in a frame if the second type of frame is adopted (4 bytes for an integer number and 8 bytes for a double number). Thus, if $N > 2M + 1$, the first type of frame should be adopted. Otherwise, SNAP uses the second type of frame.

Sending Parameter Selection for Communication Reduction: In order to determine which parameters need not to

be sent, even when they have minor changes during an iteration, we first estimate the *Accumulated Parameter Error* (APE), resulted from missing the updated parameters from its neighbors. To understand this concept, we first define *l-iteration debt parameter error* during iteration k, denoted by $y_{(i)}^{k,l}$, where $l < k$. Assuming that after iteration $k-l$, every edge server i sends $x_{(i)}^{k-l} + \Delta x_{(i)}^{k-l}$ rather than $x_{(i)}^{k-l}$; after iteration k, the parameter for every edge server i becomes $\hat{x}_{(i)}^{k}$ instead of $x_{(i)}^{k}$; then we have $y_{(i)}^{k,l} = \hat{x}_{(i)}^{k} - x_{(i)}^{k}$. If edge servers do not send updated parameter value to their neighbors prior to iteration k, the APE for edge server i after iteration k could be as large as $APE_{(i)}^{k} = \sum_{l=1}^{k-1} y_{(i)}^{k,l}$.

In fact, we can decompose the operation (6) as $\boldsymbol{x}^{k+1} = \widetilde{W}\boldsymbol{x}^{k} - \alpha\nabla f(\boldsymbol{x}^{k})$ and $\boldsymbol{x}^{k+2} = W\boldsymbol{x}^{k+1} - \alpha\nabla f(\boldsymbol{x}^{k+1})$. Accordingly, when every edge server j sends $x_{(j)}^{k} + \Delta x_{(j)}^{k}$ to its neighbor rather than $x_{(j)}^{k}$, we have

$$\hat{x}_{(i)}^{k+1} = \sum_{j=1}^{N} w_{ij}(x_{(j)}^{k} + \Delta x_{(j)}^{k}) - \alpha\nabla f_i(x_{(i)}^{k} + \Delta x_{(i)}^{k})$$

or

$$\hat{x}_{(i)}^{k+1} = \sum_{j=1}^{N} \widetilde{w}_{ij}(x_{(j)}^{k} + \Delta x_{(j)}^{k}) - \alpha\nabla f_i(x_{(i)}^{k} + \Delta x_{(i)}^{k})$$

Compared with $\boldsymbol{x}^{k+1} = \widetilde{W}\boldsymbol{x}^{k} - \alpha\nabla f(\boldsymbol{x}^{k})$, we know

$$y_{(i)}^{k+1,1} = \sum_{j=1}^{N} w_{ij}\Delta x_{(j)}^{k} - \alpha[\nabla f_i(x_{(j)}^{k} + \Delta x_{(i)}^{k}) - \nabla f_i(x_{(i)}^{k})]$$

or

$$y_{(i)}^{k+1,1} = \sum_{j=1}^{N} \widetilde{w}_{ij}\Delta x_{(j)}^{k} - \alpha[\nabla f_i(x_{(i)}^{k} + \Delta x_{(i)}^{k}) - \nabla f_i(x_{(i)}^{k})]$$

When $\Delta x_{(j)}^{k}$ is small enough, according to the definition of derivative, above equation can be approximated by

$$y_{(i)}^{k+1,1} = \sum_{j=1}^{N} w_{ij}\Delta x_{(j)}^{k} - \alpha\nabla^2 f_i(x_{(i)}^{k})\Delta x_{(i)}^{k}$$

or

$$y_{(i)}^{k+1,1} = \sum_{j=1}^{N} \widetilde{w}_{ij}\Delta x_{(j)}^{k} - \alpha\nabla^2 f_i(x_{(i)}^{k})\Delta x_{(i)}^{k}$$

Assume that the second order gradient of $f_i(x)$ is bounded, *i.e.*, $|\nabla^2 f_i(x)| \leq G$, and notice $W\mathbf{1} = \widetilde{W}\mathbf{1} = \mathbf{1}$, we know, in either case,

$$|y_{(i)}^{k+1,1}| \leq (1 + \alpha G)\max_j |\Delta x_{(j)}^{k}| \tag{25}$$

Extending (25) to the case every edge server j sends $x_{(j)}^{k-l} + \Delta x_{(j)}^{k-l}$ to its neighbor rather than $x_{(j)}^{k-l}$, we know

$$|y_{(i)}^{k,l}| \leq (1 + \alpha G)^{l}\max_j |\Delta x_{(j)}^{k-l}| \tag{26}$$

Accordingly, we can bound the maximum APE due to missing updates from each and every iteration so far as follows:

$$|APE_{(i)}^{k}| \leq \sum_{l=1}^{k-1}(1 + \alpha G)^{l}\max_j |\Delta x_{(j)}^{k-l}| \tag{27}$$

Algorithm 1: Communication cost reduction in SNAP.

Input: Second order gradient bound G

1: Choose an iteration step size α, e.g.$\alpha = \frac{1}{100G}$
2: Initialize APE threshold T_k and the target iteration times I_k, $k \leftarrow 0$
3: **while** $T_k > \epsilon$ **do**
4: Set $\max_j |\Delta x_j| = \frac{T_k}{I_k(1+\alpha G)^{I_k}}$
5: Run iteration (6) till APE exceeds T_k. In each iteration, every edge server i sends as less parameters as possible under the constraint $|\Delta x_i| < \max_j |\Delta x_j|$
6: Update T_k and I_k, $k \leftarrow k + 1$
7: **end while**

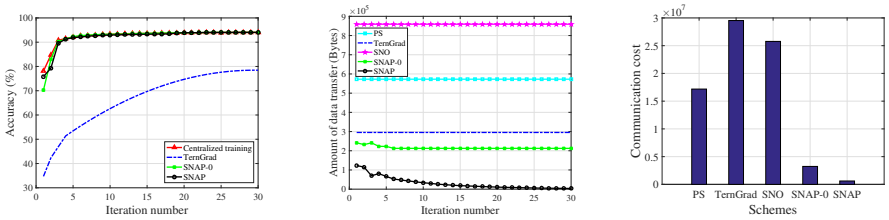
Assuming that we have set an APE threshold T, (27) can be used to determine which parameters will be sent to the neighbors and which will not. Specifically, in order to limit the maximum APE to be less than a threshold T after k iterations, all the parameters whose value changes less than $\frac{T}{\sum_{l=1}^{k-1}(1+\alpha G)^{l}}$ do not need to be sent out.

Consider that the parameter differences reduce quickly during the iteration process, we can divide the iterations into multiple stages with different APE threshold. For example, in the first 10 iterations, we would like to set the APE threshold to be 3, *i.e.*, $|APE_{(i)}^{10}| \leq 3$. Assume, $1 + \alpha G = 1.01$, then, we can easily set $\max_j |\Delta x_{(j)}| \leq 0.28$, *i.e.*, all the parameters changing less than 0.28 during one iteration will not be sent to the neighbors. After that, we restart the iteration from the solution derived by the first 10 iterations, and reduce the APE threshold to be 1, namely, set $|APE_{(i)}^{10}| \leq 1$. At this time, we should set $\max_j |\Delta x_{(j)}| \leq 0.1$. Since the parameter differences have already reduced during the first 10 iterations, there should be more parameters that do not need to be sent out in the later 10 iterations. This algorithm ends when the APE threshold reduces to 0. *It is worth noting that since the convergence and optimality of iteration* (6) *has nothing to do with the initial parameter values, by introducing the communication reduction scheme into* SNAP*, we can still derive the optimal solution when the APE threshold approaches 0.* However, we usually allow a small APE threshold to avoid the communication incurred by the iteration collision (*i.e.*, parameters still have some slight changes when the iteration converges).

Based on above discussions, the algorithm to reduce the communication cost in SNAP in Algorithm 1. It should be noted that Algorithm 1 is run by each edge server in a distributed manner.

D. Discussions

Synchronization: In SNAP, edge servers exchange the parameters with its neighbors similar to how RIP [23] works for Internet routing (with the main difference being that in SNAP, only selected parameters are exchanged instead of the entire distance vector). All the edge servers share a global clock and define a timer to exchange the parameters. The timer can be set up based on network characteristics (*e.g.*, link bandwidth)



(a) How model accuracy changes. (b) Communication cost in each iteration. (c) Total communication cost under different schemes.

Fig. 4. Testbed experiment results.

and model features (*e.g.*, scale of the model and amount of the training data).

Stragglers: Due to some reasons, such as link congestion, server shut down, *etc.*, some edge servers cannot get parameter updates from all their neighbors. In this case, these edge servers simply ignores corresponding parameter updates and leverage the latest parameter updates from those edge servers to continue the iteration. Intuitively, this would be like the dropout process [28], which does not impact system performance to much. In Section V, we will demonstrate that SNAP works well even if some of the servers cannot get all the updates from their neighbors.

Neighbor Sets Planning: In our optimization, we assume the neighbor set of each edge server is known in advance. Usually, this information can be obtained based on the locations of edge servers or the connection status among them. If this information is not available, we can assume that every edge server is neighboring with all other edge servers and optimize the weight matrix. If the weight between two edge servers are less than a predefined threshold, we can remove them from each other's neighbor set. It is worth noting that it also benefits reducing the communication cost by dismissing the neighbor relationship between two edge servers if the weight between them is not significant.

V. Performance Evaluation

In this section, we evaluate the performance of SNAP through both small scale testbed experiments and large scale simulations. The experiments are to train a neural network model based on the data set of MNIST [27]. In this data set, there are 50,000 samples for training and 10,000 samples for testing. The input of each of the samples is a picture with 28×28 pixes, while the output is a number from 0 to 9. Accordingly, we train neural networks with 784 input perceptrons and 10 output perceptrons. For simplicity, we only train a 3-layer fully connected conventional neural network with 30 hidden perceptrons.

Due to the computation complexity, in the simulations, we leverage the data set of "default of credit card clients" [29] to train an SVM model. In this data set, there are 30,000 samples and 24 features in each sample. Accordingly, there are only 24 parameters in each SVM model, and hence, we can perform the simulation in a large scale network on a single desktop. To emulate the distributed data scenario, we randomly distribute the training samples among the edge servers. When we test the model accuracy, we use the entire testing set.

When implementing SNAP, in addition to all edge servers perform the calculation (8), we initialize the APE threshold to be 10% of the mean value of all the parameters and ensure the APE threshold will effect in at least 10 iterations. When the APE meets the threshold, edge server reduces it by 10%, until the APE threshold reduces below a predefined parameter ϵ. It should be noted that this threshold is calculated by every edge server itself, rather than doing it in a centralized manner.

Comparisons: We compare following schemes with SNAP.

- **Centralized training**. This is the baseline to evaluate the accuracy of each scheme
- **Parameter server scheme (PS)**. We leverage the algorithm in [10] as the representative of PS scheme. It should be noted that in a general edge computing system, we randomly select the parameter server, and send all the data through the least hop path to minimize the network-wide data transmission.
- **Gradient compression (TernGrad)** [19]. This is the state-of-the-art communication cost optimization scheme in distributed learning. We deal with the parameter server selection problem as in the PS scheme.
- **SNAP-0**. In this scheme, edge servers send out *all the changed parameters* (*i.e.*, set the APE threshold as 0). This is to evaluate how the performance of SNAP is impacted when some of the parameter changes are ignored.
- **Select Neighbor Only (SNO)**. In this scheme, all parameters, regardless of whether they are modified or not, get exchanged. This can evaluate the function of communication reduction schemes in SNAP. Obviously, SNO derives the same accuracy and convergence performance as SNAP-0, and hence we only study the communication cost of SNO.

A. Testbed Experiment

We implement SNAP on a testbed with 3 servers. Each server is connected with the other two through links of 1 Gbps. Each server contains about 17,000 training samples. The experiment results are shown in Fig. 4.

Fig. 4(a) shows how the model accuracy changes with the iteration under different training schemes. Consider the topology of our testbed, the accuracy changing process under PS scheme should be the same as the SNAP-0 scheme. Hence, we do not test the PS scheme in this figure. During the first few iterations, the centralized training scheme may achieve the best performance as it aggregates all the training data. However, SNAP quickly catches up with the performance derived by

the centralized training, since we conduct the experiment on a small scale testbed and the information may spread around the entire network quickly.

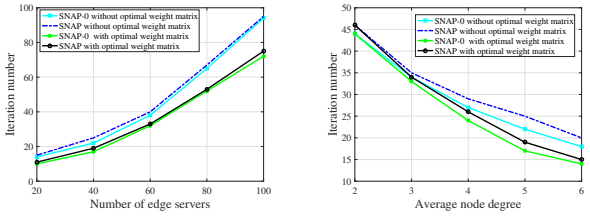
Though TernGrad is the state-of-the-art technology to reduce the communication cost for distributed learning, it achieves the reduction in communication cost at a large expense in the convergence rate. In our experiment, TernGrad can yield the accuracy of only 78% after 20 iterations. Even after 100 iterations, the accuracy derived by TernGrad is only 93.17%, which is not as accurate as that should be achieved by the centralized training scheme.

Fig. 4(b) & 4(c) show the communication cost during the iterations. For simplicity, without measuring the exactly data amount transmitted by developing a network interface driver, we only record the number of bytes written into the socket under each scheme. Fig. 4(b) shows how the communication cost changes with the iterations. Apparently, the communication costs during each iteration under PS, SNO and TernGrad schemes are not changing with the iteration number. In every iteration, SNO incurs more communication cost than PS as each server under SNO sends data to two other servers while it sends to only one under PS. TernGrad sends least data in one iteration among these three schemes since it uses only 2 bits to encode the worker-to-server gradient. As mentioned earlier, although TernGrad optimizes the work-to-server traffic, it does not derive an acceptable accuracy after 20 iterations when all other schemes converge to a good solution. Actually, TernGrad needs more than 100 iterations to converge. In addition, SNAP transfers fewer and fewer data with the iterations. When the algorithm converges, the amount of data sent by SNAP approaches 0. Accordingly, SNAP results in a lower communication cost. It should be noted that if edge servers send out all the unchanged parameters (SNAP-0), the number of sent bytes does not approach 0. It is because that many of the parameters have slight changes even when the iteration almost converges. In SNAP, these changes will be ignored, and hence it results in a low communication cost.

From Fig. 4(c), we can see that SNAP incurs only 3.56% of the communication cost compared with the PS scheme which sends the gradients of all the local parameters. Even when compared with SNAP-0, SNAP can save about 80% of the communication cost. On the other hand, SNO needs 1.5x of communication cost when compared to PS scheme since every server sends data to multiple neighbors instead of just one PS servers. However, we will soon see in large scale simulations that in the edge computing scenario where the network is not highly connected, SNO incurs a lower communication cost than PS. We can also observe that TernGrad in fact needs more data transmission than conventional PS scheme to achieve an acceptable performance, since more iterations are required. (more than 100 iterations with TernGrad vs. about 20 iterations with SNAP in our experiments).

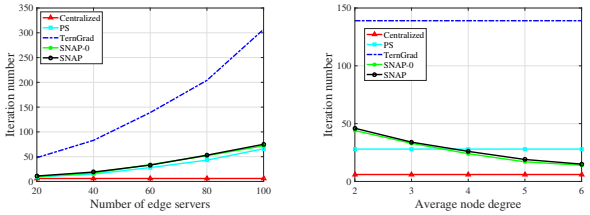
B. Large Scale Simulations

In this section, we conduct large scale simulations to study how the performance of SNAP is impacted by the



(a) Convergence rate vs. network scale. (b) Convergence rate vs. average node degree.

Fig. 5. The performance of weight matrix optimization.



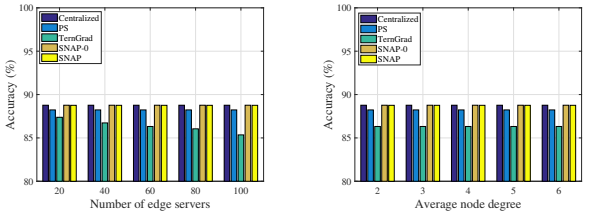
(a) Convergence rate vs. network scale. (b) Convergence rate vs. average node degree.

Fig. 6. How the required iteration number impacted by the network characteristics.

network scale. To this end, we randomly generate networks with various typologies and average node degrees (*i.e.*, the average neighbor set size). For simplicity, we assume that a network so generated is a peer-to-peer network where each node represents an edge server and each edge is a *one-hop* connection. Since it is difficult for us to train many neural networks on a single server for simulation purpose, we train SVM models in our simulations.

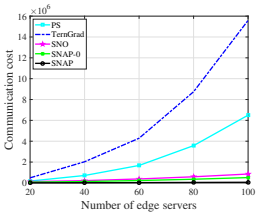
Performance of weight matrix optimization: Before studying the performance of SNAP, we first investigate the how the weight matrix optimization improves the algorithm convergence. To this end, we randomly generate the neighbor set for each server, and investigate the number of iterations required by SNAP and SNAP-0 with/without the weight matrix optimization under different settings. As a baseline, the weight matrix is computed according to (24) under the assumption that each edger server is connected with all the edge servers in its neighbor set. The default number of edge servers is 60, and the default average size of neighbor set is 3. The simulation results are shown in Fig. 5.

From this figure, we can observe that the weight matrix optimization can reduce the number of iteration required, regardless of SNAP or SNAP-0 is adopted. With the increase of network scale (as shown in Fig. 5(a)), *i.e.*, the number of

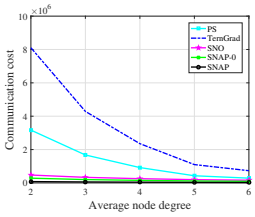


(a) Model accuracy vs. network scale. (b) Model accuracy vs. average node degree.

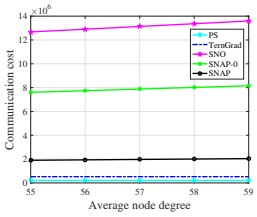
Fig. 7. How the model accuracy impacted by the network characteristics.



(a) Network scale.



(b) Small average node degree.



(c) Large average node degree.

Fig. 8. How the total communication cost impacted by the network characteristics.

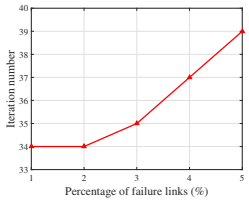


Fig. 9. Impact of stragglers.

edge servers, the number of iterations reduced by the weight matrix optimization increases. This is due to two reasons. First, in large scale networks, more iterations are required. Improving the convergence rate can result in more iteration number reduction. Second, there are more weights, namely more variables, in a large scale network, which brings larger optimization space.

Fig. 5(b) shows how the iteration number changes with the average size of neighbor set. We can see that the larger average size of neighbor set is, the larger performance improvement brought by the weight matrix optimization will be. If there are only 2 neighbors for each edge server, the weight matrix optimization cannot lead to any iteration number reduction. This is again because there are more weights and hence larger optimization space sought by the weight matrix design method. Hereafter, when we mention SNAP or SNAP-0, it denotes the version with optimized weight matrix since the weight matrix optimization is a part of the entire system.

Convergence rate: The first metric that would be impacted by the network scale is the algorithm convergence rate. To study how this metric is impacted by the network scale, we change the number of edge servers or the average node degree, and investigate how many iterations are required by each algorithm to converge. The default number of edge servers is 60, and the default average node degree is 3. The simulation results are shown in Fig. 6. Generally speaking, every training method will converge regardless of the network characteristics in our simulation.

Fig. 6(a) presents how many iterations different schemes require to converge. Since we randomly distribute the sample data among all the edge servers, intuitively, the more edge servers in the network, the more iterations they require, as there are fewer data on each edge server. Another observation is that though we ignore some small parameter changes in SNAP, it does not significantly impact the algorithm convergence; even in a network with 100 edge servers, SNAP only needs 3-4 more iterations to converge than that required in SNAP-0. On the other hand, the convergence rate of TernGrad is dramatically impacted by the network scale, *i.e.*, the number of sample data on each edge server. It may be because TernGrad introduces too much noise with fewer bits for quantification so that the algorithm fails to identify the steepest descent direction.

Fig. 6(b) depicts how the convergence rate changes with the network average node degree. Since the gradient exchanging framework would not be changed under PS scheme and TernGrad, the average node degree does not impact their algorithm convergence rate. However, in SNAP, a larger average node degree helps spread the local data information among the network faster (as to be seen later, although this would increase the communication cost, the amount of data to be transferred is still extremely small, thanks to the fact that only select parameters are exchanged), and hence it can reduce the number of iterations required for the algorithm to converge.

Accuracy: As an ML model, the inference accuracy is one of the most important metrics. Fig. 7 shows how the model accuracy changes with the network characteristics. Since SNAP-0 adopts the same iteration method as EXTRA [18], it can achieve the optimal solution regardless of the network characteristics. From this figure, we can again see that though SNAP ignores the parameters with small changes during the entire training procedure, it can converge to the same performance as the centralized training. On the other hand, both the PS scheme and TernGrad would result in some accuracy degradation since they cannot ensure the solution converges to an optimal one.

The most important observation is that TernGrad incurs larger accuracy degradation with the increase of network size. When there are 100 edge servers in the network, the accuracy degradation is up to 3.5%, which is unacceptable in many accuracy sensitive applications.

Communication cost: Reducing communication cost is the main objective of SNAP. Fig. 8 shows how the communication cost under different schemes changes with the network characteristics.

Fig. 8(a) compares different schemes in the total amount of data transferred before algorithm converges when the number of edge servers increases. Apparently, more edge servers in the network incur larger communication cost. This should be due to two reasons. Firstly, all the schemes need more iterations to converge in larger scale networks. Secondly, under PS scheme and TernGrad, data are transmitted through more hops in larger networks. Since the second problem does not exist in SNAP, the amount of data that should be transmitted to achieve convergence increases much slower than that in PS and TernGrad. When there are 100 edge servers in the network, the communication cost under SNAP is only 0.4% of that under TernGrad and 0.96% of that under PS scheme.

Fig. 8(b) shows in a sparsely connected network, with the increase in node degree, the total communication cost is reduced. Even SNO needs much less communication cost than PS and TernGrad. This is because larger node degree means a smaller network diameter and more neighbors to share the

temporary parameter information. Though this leads to more data transmission in each iteration, it leads to fewer iterations and reduces the communication cost. However, this is not the case in a densely connected network, as shown in Fig. 8(c). In such a network, the total communication cost increases with the node degree. The communication cost incurred by SNAP is even larger than PS and TernGrad. In this case, increase the network connectivity cannot significantly speed up the convergence, but edge servers need to send more data to their neighbors. Accordingly, the communication cost increase. It means that in a densely connected network, edge servers do not need to maintain the neighbor relationship with many other servers in order to reduce the communication cost.

Impact of Stragglers: When some of the links are temporarily unavailable due to failure or congestion, SNAP ignores the parameter updates that have not been received. Fig. 9 presents how the number of iterations needed by SNAP to achieve convergence is impacted by the percentage of unavailable links. It is obvious that more unavailable links would result in more iterations. However, when there are 1% of the links unavailable, the convergence rate of SNAP is not impacted. Even when the unavailable link percentage achieves 5%, only 11.8% more iterations are required to derive a convergent solution. This shows the robustness of SNAP.

VI. Conclusions

This paper has proposed SNAP, a communication efficient distributed machine learning framework, to train the machine learning model with distributed data collected by edge devices. The two key ideas are 1) edge servers in SNAP only periodically exchange the parameters of their local models with their neighbors; and 2) the parameters with little change are not exchanged. In addition, different from previous consensus optimization works, in which each node updates its local parameters based on a non-optimized weight matrix, SNAP proposes a scheme to optimize the weight matrix used for parameter update, which can speed up the convergence. Both testbed experiments and large scale simulations have also demonstrated that SNAP can achieve the optimal solution with a reasonable convergence time and resilience performance, but at very little communication cost.

Acknowledgment

This research is partially supported by NSF grants CNS-1626374, EFMA-1441284 and CNS-1737590, NSFC grants 61671130 and 61671124. The corresponding author of this paper is Prof. Sheng Wang (wsh_keylab@uestc.edu.cn).

References

[1] M. Chiang and T. Zhang, "Fog and iot: An overview of research opportunities," *IEEE Internet of Things Journal*, 2016.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. of NIPS*, 2012.

[3] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proc. of Multimedia*, 2014.

[4] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, 2012.

[5] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proc. of ICML*, 2008.

[6] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun, "Dermatologist-level classification of skin cancer with deep neural networks," *Nature*, 2017.

[7] M. Chiang and T. Zhang, "Fog and iot: An overview of research opportunities," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 854–864, Dec 2016.

[8] S. Davy, J. Famaey, J. Serrat, J. L. Gorricho, A. Miron, M. Dramitinos, P. M. Neves, S. Latre, and E. Goshen, "Challenges to support edge-as-a-service," *IEEE Communications Magazine*, vol. 52, no. 1, pp. 132–139, 2014.

[9] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the MCC*, 2012, pp. 13–16.

[10] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "When edge meets learning: Adaptive control for resource-constrained distributed machine learning," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, April 2018, pp. 63–71.

[11] S. Ghadimi and G. Lan, "Stochastic first- and zeroth-order methods for nonconvex stochastic programming," *SIAM Journal on Optimization*, vol. 23, no. 4, pp. 2341–2368, 2013.

[12] R. Johnson and T. Zhang, "Accelerating stochastic gradient descent using predictive variance reduction," in *Proc. of NIPS*, 2013, pp. 315–323.

[13] S. Sun, W. Chen, J. Bian, X. Liu, and T.-Y. Liu, "Ensemble-compression: A new method for parallel training of deep neural networks," in *Proc. of ECML-PKDD*, 2017.

[14] H. Tang, X. Lian, M. Yan, C. Zhang, and J. Liu, "D_2: Decentralized training over decentralized data," in *Proc. of ICML*, 2018.

[15] A. Mokhtari and A. Ribeiro, "Decentralized double stochastic averaging gradient," in *49th Asilomar Conference on Signals, Systems and Computers*, Nov 2015, pp. 406–410.

[16] A. T. Suresh, F. X. Yu, S. Kumar, and H. B. McMahan, "Distributed mean estimation with limited communication," in *Proc. of ICML*, vol. 70, 2017, pp. 3329–3337.

[17] J. Lavaei and R. M. Murray, "Quantized consensus by means of gossip algorithm," *IEEE Transactions on Automatic Control*, vol. 57, no. 1, pp. 19–32, Jan 2012.

[18] W. Shi, Q. Ling, G. Wu, and W. Yin, "Extra: An exact first-order algorithm for decentralized consensus optimization," *SIAM J. on Optimization*, vol. 25, no. 2, pp. 944–966, May 2015.

[19] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li, "Terngrad: Ternary gradients to reduce communication in distributed deep learning," in *Proc. of NIPS*, 2017, pp. 1508–1518.

[20] A. F. Aji and K. Heafield, "Sparse communication for distributed gradient descent," *CoRR*, vol. abs/1704.05021, 2017. [Online]. Available: http://arxiv.org/abs/1704.05021

[21] Y. Cao, W. Yu, W. Ren, and G. Chen, "An overview of recent progress in the study of distributed multi-agent coordination," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 1, pp. 427–438, Feb 2013.

[22] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized gossip algorithms," *IEEE Transactions on Information Theory*, vol. 52, no. 6, pp. 2508–2530, June 2006.

[23] G. S. Malkin, *RIP: An Intra-domain Routing Protocol*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2000.

[24] J. C. Allwright, "On maximizing the minimum eigenvalues of a linear combination of symmetric matrices," *SIAM J. Matrix Anal. Appl.*, vol. 10, no. 3, pp. 347–382, Jul. 1989.

[25] B. N. Parlett, *The Symmetric Eigenvalue Problem*. Prentice-Hall, Inc., 1998.

[26] R. Olfati-Saber and R. M. Murray, "Consensus problems in networks of agents with switching topology and time-delays," *IEEE Transactions on Automatic Control*, vol. 49, no. 9, pp. 1520–1533, Sep. 2004.

[27] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[28] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[29] I. C. Yeh and C. H. Lien, "The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients," *Expert Systems with Applications*, vol. 36, no. 2, pp. 2473–2480, 2009.