In this paper, we study distributed algorithms for large-scale AUC maximization with a deep neural network as a predictive model. Although distributed learning techniques have been investigated extensively in deep learning, they are not directly applicable to stochastic AUC maximization with deep neural networks due to its striking differences from standard loss minimization problems (e.g., cross-entropy). Towards addressing this challenge, we propose and analyze a communication-efficient distributed optimization algorithm based on a non-convex concave reformulation of the AUC maximization, in which the communication of both the primal variable and the dual variable between each worker and the parameter server only occurs after multiple steps of gradient-based updates in each worker. Compared with the naive parallel version of an existing algorithm that computes stochastic gradients at individual machines and averages them for updating the model parameter, our algorithm requires a much less number of communication rounds and still achieves a linear speedup in theory. To the best of our knowledge, this is the first work that solves the non-convex concave min-max problem for AUC maximization with deep neural networks in a communication-efficient distributed manner while still maintaining the linear speedup property in theory. Our experiments on several benchmark datasets show the effectiveness of our algorithm and also confirm our theory.
overhead due to a large number of communication rounds.
In this paper, we bridge the gap between stochastic AUC maximization and distributed deep learning by proposing a communication-efficient distributed algorithm for stochastic AUC maximization with a deep neural network. The focus is to make the total number of communication rounds much less than the total number of iterative updates. We build our algorithm upon the nonconvex-concave min-max reformulation of the original problem. The key ingredient is to design a communication-efficient distributed algorithm for solving the regularized min-max subproblems using multiple machines. Specifically, we follow the proximal primal-dual algorithmic framework proposed by (Rafique et al., 2018; Liu et al., 2020b), i.e., by solving a sequence of quadratic regularized min-max saddle-point problems with periodic updated regularizers successively. The key difference is that the inner min-max problem solver is built on a distributed periodic model averaging technique, which consists of a fixed number of stochastic primal-dual updates over individual machines and a small number of averaging of model parameters from multiple machines. This mechanism can greatly reduce the communication cost, which is similar to (Zhou & Cong, 2017; Stich, 2018; Yu et al., 2019b). However, their analysis cannot be applied to our case since their analysis only works for convex or non-convex minimization problems. In contrast, our algorithm is designed for a particular non-convex concave min-max problem induced by the original AUC maximization problem. Our contributions are summarized as following:

- We propose a communication-efficient distributed stochastic algorithm named CoDA for solving a nonconvex-concave min-max reformulation of AUC maximization with deep neural networks by local primal-dual updating and periodically global variable averaging. To our knowledge, this is the first communication-efficient distributed stochastic algorithm for learning a deep neural network by AUC maximization.

- We analyze the iteration complexity and communication complexity of the proposed algorithm under the commonly used Polyak-Łojasiewicz (PL) condition as in (Liu et al., 2020b). Comparing with (Liu et al., 2020b), our theoretical result shows that the iteration complexity can be reduced by a factor of $K$ (the number of machines) in a certain region, while the communication complexity (the rounds of communication) is much less than that of a naive distributed version of the stochastic algorithm proposed in (Liu et al., 2020b). The summary of iteration and communication complexities is given in Table 1.

- We verify our theoretical claims by conducting experiments on several large-scale benchmark datasets. The experimental results show that our algorithm indeed exhibits good speedup performance in practice.

2. Related Work

Stochastic AUC Maximization. It is challenging to directly solve the stochastic AUC maximization in the online learning setting since the objective function of AUC maximization depends on a sum of pairwise losses between samples from positive and negative classes. (Zhao et al., 2011) addressed this problem by maintaining a buffer to store representative data samples, employing the reservoir sampling technique to update the buffer, calculating gradient information based on the data in the buffer, and then performing gradient-based update rule to update the classifier. (Gao et al., 2013) did not maintain a buffer, they instead maintained first-order and second-order statistics of the received data to update the classifier by gradient-based update. Both of them are infeasible in big data scenarios since (Zhao et al., 2011) suffers from a large amount of training data and (Gao et al., 2013) is not suitable for high dimensional data. Ying et al. (2016) addressed these issues by introducing a min-max reformulation of the original problem and solving it by primal-dual stochastic gradient method (Nemirovski et al., 2009), in which no buffer is needed and per-iteration complexity is the same magnitude of the dimension of the feature vector. Nøtøre et al. (2018) improved the convergence rate by adding a strongly convex regularizer upon the original formulation. Based on the same saddle point formulation as in (Ying et al., 2016), Liu et al. (2018) got an improved convergence rate by developing a multi-stage algorithm without adding the strongly convex regularizer. However, all of these studies focus on learning a linear model. Recently, (Liu et al., 2020b) considered stochastic AUC maximization for learning a deep non-linear model, in which they designed a proximal primal-dual gradient-based algorithm under the PL condition and established non-asymptotic convergence results.

Communication Efficient Algorithms. There are multiple approaches for reducing the communication cost in distributed optimization, including skipping communication and compression techniques. Due to limit of space, we mainly review the literature on skipping communication. For compression techniques, we refer the readers to (Jiang & Agrawal, 2018; Stich et al., 2018; Basu et al., 2019; Wangni et al., 2018; Bernstein et al., 2018) and references therein. Skipping communication is realized by doing multiple local gradient-based updates in each worker before aggregating the local model parameters together. One special case is so-called one-shot averaging (Zinkevich et al., 2010; McDonald et al., 2010; Zhang et al., 2013), where each machine solves a local optimization problem and averages these solutions only at the last iterate. (Zhang et al., 2013; Shamir & Srebro, 2014; Godichon-Baggioni & Saadane, 2017; Jain
Table 1. Summary of Iteration and Communication Complexities, where $K$ is number of machines and $\mu \leq 1$. NP-PPD-SG denotes the naive parallel version of PPD-SG, which is also a special case of our algorithm, whose complexities can be derived following our analysis.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PPD-SG (Liu et al., 2020b)</td>
<td>Single</td>
<td>$O(1/(\mu^2\epsilon))$</td>
<td>-</td>
</tr>
<tr>
<td>NP-PPD-SG</td>
<td>Distributed</td>
<td>$O(1/(K\mu^2\epsilon))$</td>
<td>$O(1/(K\mu^2\epsilon))$</td>
</tr>
<tr>
<td>CoDA</td>
<td>Distributed</td>
<td>$O(1/(K\mu^2\epsilon))$</td>
<td>$O(1/(\mu^{3/2}\epsilon^{1/2}))$</td>
</tr>
</tbody>
</table>

et al., 2017; Koloskova et al., 2019; Koloskova et al., 2020) considered one-shot averaging with one-pass of the data and established statistical convergence, which is usually not able to guarantee the convergence of training error. The scheme of local SGD update in each worker with skipping communication is analyzed for convex (Stich, 2018; Jaggi et al., 2014) and nonconvex problems (Zhou & Cong, 2017; Jiang & Agrawal, 2018; Wang & Joshi, 2018b; Lin et al., 2018b; Wang & Joshi, 2018a; Yu et al., 2019b; Basu et al., 2019; Haddadpour et al., 2019). There are also several empirical studies (Povey et al., 2014; Su & Chen, 2016; McMahan et al., 2014; Su & Chen, 2015; McMahan et al., 2016; Chen & Huo, 2016; McMahan et al., 2016; Lin et al., 2018b; Kamp et al., 2018) showing that this scheme exhibits good empirical performance in distributed deep learning. However, all of these works only consider minimization problems and do not apply to the nonconvex-concave min-max formulation as considered in this paper.

Nonconvex Min-max Optimization Stochastic nonconvex min-max optimization has garnered increasing attention recently (Rafique et al., 2018; Lin et al., 2018a; Sanjabi et al., 2018; Lu et al., 2019; Lin et al., 2019; Jin et al., 2019; Liu et al., 2020a). (Rafique et al., 2018) considered the case where the objective function is weakly-convex and concave and proposed an algorithm based on the spirit of proximal point method (Rockafellar, 1976), in which a proximal subproblem with periodically updated reference points is approximately solved by an appropriate stochastic algorithm. They established the convergence to nearly stationary point for the equivalent minimization problem. Under the same setting, (Lu et al., 2019) designed a block-based algorithm and showed that it can converge to a solution with a small stationary gap, and (Lin et al., 2019) considered solving the problem using vanilla stochastic gradient descent ascent and established its convergence to a stationary point under the smoothness assumption. There are also several papers (Lin et al., 2018a; Sanjabi et al., 2018; Liu et al., 2020a) trying to solve non-convex non-concave min-max problems. (Lin et al., 2018a) proposed an inexact proximal point method for solving a class of weakly-convex weakly-concave problems, which was proven to converge to a nearly stationary point. (Sanjabi et al., 2018) exploited the PL condition for the inner maximization problem and designed a multi-step alternating optimization algorithm which was able to converge to a stationary point. (Liu et al., 2020a) considered solving a class of nonconvex-nonconcave min-max problems by designing an adaptive gradient method and established an adaptive complexity for finding a stationary point. However, none of them is particularly designed for distributed stochastic AUC maximization problem with a deep neural network.

3. Preliminaries and Notations
The area under the ROC curve (AUC) on a population level for a scoring function $h: \mathcal{X} \to \mathbb{R}$ is defined as

$$\text{AUC}(h) = \Pr(h(x) \geq h(x')) | y = 1, y' = -1, \quad (1)$$

where $z = (x, y)$ and $z' = (x', y')$ are drawn independently from $\mathcal{P}$. By employing the squared loss as the surrogate for the indicator function which is commonly used by previous studies (Gao et al., 2013; Ying et al., 2016; Liu et al., 2018; 2020b), the deep AUC maximization problem can be formulated as

$$\min_{w \in \mathbb{R}^d} \mathbb{E}_{x, x'} \left[ \left(1 - h(w; x) + h(w; x') \right) | y = 1, y' = -1 \right], \quad (2)$$

where $h(w; x)$ denotes the prediction score for a data sample $x$ made by a deep neural network parameterized by $w$. It was shown in (Ying et al., 2016) that the above problem is equivalent to the following min-max problem:

$$\min_{w \in \mathbb{R}^d} \max_{(a, b, \alpha) \in \mathbb{R}} f(w, a, b, \alpha) = \mathbb{E}_z \left[ F(w, a, b, \alpha, z) \right], \quad (2)$$

where

$$F(w, a, b, \alpha; z) = (1 - p)(h(w; x) - a)^2 I[y = 1] + p(h(w; x) - b)^2 I[y = -1] + 2(1 + \alpha)(ph(w; x)I[y = 1] - (1 - p)h(w; x)I[y = -1]) - (1 - p)h(w; x)I[y = -1],$$

where $p = \Pr(y = 1)$ denotes the prior probability that an example belongs to the positive class and $I$ denotes an indicator function. The above min-max reformulation allows us to decompose the expectation over all data into the expectation over data on individual machines.

In this paper, we consider the following distributed AUC maximization problem:

$$\min_{w \in \mathbb{R}^d} \max_{(a, b, \alpha) \in \mathbb{R}} \frac{1}{K} \sum_{k=1}^{K} f_k(w, a, b, \alpha), \quad (3)$$

where $K$ is the total number of machines, $f_k(w, a, b, \alpha) = \mathbb{E}_{z^k} \left[ F_k(w, a, b, \alpha; z^k) \right]$, $z^k = (x^k, y^k) \sim \mathbb{P}_k$, $\mathbb{P}_k$ is the
data distribution on machine $k$, and $F_k(w, a, b; z^k) = F(w, a, b; z^k)$. Our goal is to utilize $K$ machines to jointly solve the optimization problem (3). We emphasize that the $k$-th machine can only access data $z^k \sim P_k$ of its own. It is notable that our formulation includes both the batch-learning setting and the online learning setting. For the batch-learning setting, $P_k$ represents the empirical distribution of data on the $k$-th machine and $P$ denotes the empirical positive ratio for all data. For the online learning setting, $P_k = P, \forall k$ represents the same population distribution of data and $p$ denotes the positive ratio in the population level.

**Notations.** We define the following notations:

$$v = (w^T, a, b)^T, \quad \phi(v) = \max_{\alpha} f(v, \alpha),$$

$$\phi_s(v) = \phi(v) + \frac{1}{2\gamma} \|v - v_{s-1}\|^2,$$

$$v^*_s = \arg \min_v \phi(v), \quad v^*_s = \arg \min_v \phi_s(v).$$

We make the following assumption throughout this paper.

**Assumption 1**

(i) There exist $\nu_0, \Delta_0 > 0$ such that $\phi(v_0) - \phi(v^*_0) \leq \Delta_0$.

(ii) For any $x$, $\|\nabla h(w; x)\| \leq G_h$.

(iii) $\phi(v)$ satisfies the $\mu$-PL condition, i.e., $\mu(\phi(v) - \phi(v^*)) \leq \frac{1}{2}\|\nabla \phi(v)\|^2$; $\phi(v)$ is $L_1$-smooth, i.e., $\|\phi(v_1) - \phi(v_2)\| \leq L_1 \|v_1 - v_2\|$.

(iv) For any $x$, $h(w; x)$ is $L_h$-smooth, and $h(w; x) \in [0, 1]$.

**Remark:** Assumptions (i), (ii), (iii) and (iv) are also assumed in (Liu et al., 2020b), which have been justified as well. $L$-smoothness of function $h$ is a standard assumption in the optimization literature. Finally, it should be noted that $\mu$ is usually much smaller than 1 (Yuan et al., 2019). This is important for us to understand our theoretical result later.

### 4. Main Result and Theoretical Analysis

In this section, we first describe our algorithm, and then present its convergence result followed by its analysis. For simplicity, we assume that the ratio $p$ of data with positive label is known. For the batch learning setting, $p$ is indeed the empirical ratio of positive examples. For the online learning setting with an unknown distribution, we can follow the online estimation technique in (Liu et al., 2020b) to do the parameter update.

Algorithm 1 describes the proposed algorithm CoDA for optimizing AUC in a communication-efficient distributed manner. CoDA shares the same algorithmic framework as proposed in (Liu et al., 2020b). In particular, we employ a proximal-point algorithmic scheme that successively solves the following convex-concave problems approximately:

$$\min_{v} \max_{\alpha} f(v, \alpha) + \frac{1}{2\gamma} \|v - v_0\|^2, \quad (4)$$

where $\gamma$ is an appropriate regularization parameter to make sure that the regularized function is strongly-convex and strongly-concave. The reference point $v_0$ is periodically updated after a number of iterations. At the $s$-th stage our algorithm invokes a communication-efficient algorithm for solving the above strongly-convex and strongly-concave subproblems. After obtaining a primal solution $v_s$ at the $s$-th stage, we sample some data from individual machines to obtain an estimate of corresponding dual variable $\alpha_s$.

Our new contribution is the communication-efficient distributed algorithm for solving the above strongly-convex and strongly-concave subproblems. The algorithm referred to as DSG is presented in Algorithm 2. Each machine makes a stochastic proximal-gradient update on the primal variable and a stochastic gradient update on the dual variable at each iteration. After every $I$ iterations, all the $K$ machines communicate to compute an average of local primal solutions $v^k_I$ and local dual solutions $\alpha^k_I$. It is not difficult to show that when $I = 1$, our algorithm reduces to the naive parallel version of the PPD-SG algorithm proposed in (Liu et al., 2020b), i.e.,, by averaging individual primal and dual gradients and then updating the primal-dual variables according to the averaged gradient. Our novel analysis allows us to use $I > 1$ to skip communications, leading to a much less number of communications. The intuition behind this is that, as long as the step size $\eta_s$ is sufficiently small we can control the distance between individual solutions $(v^k_I, \alpha^k_I)$ to their global averages, which allows us to control the error term that is caused by the discrepancy between individual

---

1 A tiny difference is that we use a proximal gradient update to handle the regularizer $\frac{1}{2\gamma} \|v - v_0\|^2$, while they directly use the gradient update. Using the proximal gradient update allows us to remove the assumption that $\|v^k_0 - v_0\|$ is upper bounded.
Communication-Efficient Distributed Stochastic AUC Maximization with Deep Neural Networks

Algorithm 2 DSG($v_0, \alpha_0, \eta, T, I, \gamma$)

Each machine does initialization: $v_0^k = v_0, \alpha_0^k = \alpha_0$.

for $t = 0, 1, ..., T - 1$ do

Each machine $k$ updates its local solution in parallel:

$$v_{t+1}^k = \arg\min_v \left[\nabla_v F_k(v_t^k, \alpha_t^k; z_t^k)^T v + \frac{1}{2\gamma} \|v - v_t^k\|^2 + \frac{1}{2\gamma} \|v - v_0\|^2\right],$$

$$\alpha_{t+1}^k = \alpha_t^k + \eta \nabla_{\alpha} F_k(v_t^k, \alpha_t^k; z_t^k).$$

if $t + 1 \mod I = 0$ then

$$v_{t+1}^k = \frac{1}{K} \sum_{k=1}^{K} v_{t+1}^k,$$  \circ communicate

$$\alpha_{t+1}^k = \frac{1}{K} \sum_{k=1}^{K} \alpha_{t+1}^k,$$  \circ communicate

end if

end for

Return $v = \frac{1}{K} \sum_{k=1}^{K} \frac{1}{T} \sum_{t=1}^{T} v_t^k$.

machines. We will provide more explanations as we present the analysis.

Below, we present the main theoretical result of CoDA. Note that in the following presentation, $L_v, H, B, \sigma_v, \sigma_\alpha$ are appropriate constants, whose values are given in the proofs of Lemma 1 and Lemma 2 in the supplement.

Theorem 1 Set $\gamma = \frac{1}{2T_v}, c = \frac{\mu}{L_v}$, $\eta_s = \eta_0 K \exp(-s - 1)c \leq O(1)$, $T_v = \max(8, 8G_2^2) \exp((s - 1)c)$, $I_s = \max(1, 1/\sqrt{K \eta_s})$

and $m_s = \max \left(\frac{1+C}{\eta_s + 1 + \frac{1}{2}\log^2(K)}, \log(K)\right)$, where

$$C = \frac{3g m_{\text{max}}}{2m_{\text{max}}} \text{ and } \hat{p} = \max(p, 1 - p).$$

To return $v$ such that $E[\phi(v_s) - \phi(v_0)] \leq \epsilon$, it suffices to choose $S \geq \frac{5L_v + \mu}{\mu} \max \left\{ \log \left(\frac{2\Delta_1}{\mu}\right), \log S + \log \left(\frac{2\eta_0}{\epsilon} \frac{6HB^2 + 12(\sigma_v^2 + \sigma_\alpha^2)}{5}\right) \right\}$. As a result, the number of iterations is at most $T = \tilde{O} \left(\max \left(\frac{\Delta_1^{1/2}}{\eta_0}, \frac{L_v}{\mu} K\right)\right)$ and the number of communications is at most $\tilde{O} \left(K/\mu + \frac{\Delta_1^{1/2}}{\eta_0}, \frac{L_v}{\mu} K + \frac{L_v^2}{\mu^{1/2} \frac{1}{2\gamma}}\right)$, where $\tilde{O}$ suppresses logarithmic factors, and $H, B, \sigma_v, \sigma_\alpha$ are appropriate constants.

We have the following remarks about Theorem 1.

- First, we can see that the step size $\eta_s$ is reduced geometrically in a stagewise manner. This is due to the PL condition. We note that a stagewise geometrically decreasing step size is usually used in practice in deep learning (Yuan et al., 2019). Second, by setting $\eta_0 = O(1/K)$ we have $I_s = \Theta\left(\frac{1}{\sqrt{K}} \exp((s - 1)c)/2\right)$. It means two things: (i) the larger the number of machines the smaller value of $I_s$, i.e., more frequently the machines need to communicate. This is reasonable since more machines will create larger discrepancy between data among different machines; (ii) the value $I_s$ can be increased geometrically across stages. This is because the step size $\eta_s$ is reduced geometrically, which causes one step of primal and dual updates on individual machines diverging less from their averaged solutions. As a result, more communications can be skipped.

- Second, we can see that when $K \leq \Theta(1/\mu)$, we have the total iteration complexity given by $\tilde{O}\left(\frac{1}{\mu K C}\right)$. Compared with the iteration complexity of the PPD-SG algorithm proposed in (Liu et al., 2020b) that is $\tilde{O}\left(\frac{1}{\mu K C}\right)$, the proposed algorithm CoDA enjoys an iteration complexity that is reduced by a factor of $K$. This means that up to a certain large threshold $\Theta(1/\mu)$ for the number $K$ of machines, CoDA enjoys a linear speedup.

- Finally, let us compare CoDA with the naive parallel version of PPD-SG, which is CoDA by setting $I = 1$. In fact, our analysis of the iteration complexity for this case is still applicable, and it is not difficult to show that the iteration complexity of the naive parallel version of PPD-SG is given by $O\left(\frac{1}{\mu K C}\right)$ when $K \leq 1/\mu$. As a result, its communication complexity is also $\tilde{O}\left(\frac{1}{\mu K C}\right)$. In contrast, CoDA’s communication complexity is $O\left(\frac{1}{\mu^{1/2} K C}\right)$ when $K \leq 1/\mu \leq \frac{1}{\mu^{1/2} K C} \leq \frac{1}{\epsilon}$ according to Theorem 1 ². Hence, our algorithm is more communication efficient, i.e., $O\left(\frac{1}{\mu^{1/2} K C}\right) \leq \tilde{O}\left(\frac{1}{\mu K C}\right)$ when $K \leq \frac{1}{\mu}$. This means that up to a certain large threshold $\Theta(1/\mu)$ for the number $K$ of machines, CoDA has a smaller communication complexity than the naive parallel version of PPD-SG.

4.1. Analysis

Below, we present a sketch of the proof of Theorem 1 by providing some key lemmas. We first derive some useful properties regarding the random function $F_k(v, \alpha, z)$.

Lemma 1 Suppose that Assumption 1 holds and $\eta \leq \min\left(\frac{1}{2p(1-p)}, \frac{1}{2p(1-p)} \frac{1}{1-p} \frac{1}{2p}\right)$. Then there exist some constants $L_2, B_\alpha, B_\sigma, \sigma_\alpha, \sigma_\alpha$ such that

$$||\nabla_v F_k(v_1, \alpha; z) - \nabla_v F_k(v_2, \alpha; z)|| \leq L_2 ||v_1 - v_2||,$$

$$||\nabla_\alpha F_k(v, \alpha; z)||^2 \leq B_\alpha^2,||\nabla_\alpha F_k(v, \alpha; z)||^2 \leq B_\alpha^2,$$

$$\mathbb{E}[||\nabla_v F_k(v, \alpha; z) - \nabla_v F_k(v, \alpha; z)||^2] \leq \sigma_\alpha^2,$$

$$\mathbb{E}[||\nabla_\alpha F_k(v, \alpha; z) - \nabla_\alpha F_k(v, \alpha; z)||^2] \leq \sigma_\alpha^2.$$

²Assume $\epsilon$ is set to be small than $\mu$.

³Indeed, $K$ can be as large as $\frac{1}{\mu^{1/2} K C}$ for CoDA to be more communication-efficient.
Remark: We include the proofs of these properties in the Appendix. In the following, we will denote $B^2 = \max(B_2^2, B_0^2)$ and $L_v = \max(L_1, L_2)$.

Next, we introduce a key lemma, which is of vital importance to establish the upper bound of the objective gap of the regularized subproblem.

**Lemma 2** (One call of Algorithm 2) Let $\psi(v) = \max f(v, \alpha) + \frac{1}{2 \nu} \|v - v_0\|^2$, $\hat{v}$ be the output of Algorithm 2 and $\alpha^*(\hat{v}) = \arg \max \alpha f(\hat{v}, \alpha) + \frac{1}{2 \nu} \|\hat{v} - v_0\|^2$. By running Algorithm 2 with given input $v_0, \alpha_0$ for $T$ iterations, $\gamma = \frac{1}{2 \nu},$ and $\eta \leq \min \{\frac{1}{L_v + 3G^2/\mu_\alpha}, \frac{1}{L_v + 3G^2/\nu}, \frac{3}{2 \mu_\alpha}, \frac{1}{2(1-p)}, \frac{1}{2p}\}$, we have

$$E[\psi(\hat{v}) - \min_\nu \psi(v)] \leq \frac{\|v_0 - v_*\|^2}{\eta} + \frac{E[(\psi_0 - \alpha^*(\hat{v}))^2]}{\eta^2} + H \frac{\|v_0 - v_*\|^2}{\eta},$$

where $\mu_\alpha = 2p(1-p), L_\gamma = 2p(1-p), G_\alpha = 2\max(p, 1-p)G_h,$ and $H = \frac{(6G^2)}{\nu} + 6L_v + \frac{6G^2}{\nu} + \frac{6\nu^2}{\mu_\alpha}.$

Remark: The above result is similar to Lemma 2 in (Liu et al., 2020b). The key difference lies in the second and third terms in the upper bound. The second term arises because of discrepancy of updates between individual machines. The third term is due to the variance reduction by using multiple machines, which is the key to establish the linear speed-up. It is easy to see that by setting $I = \frac{1}{\sqrt{KH}},$ the second term and the third term have the same order. With above lemma, the proof of Theorem 1 follows similar analysis to in (Liu et al., 2020b).

**Sketch of the Proof of Lemma 2.** Below, we present a roadmap for the proof of the key Lemma 2. The main idea is to first bound the objective gap of the subproblem in Lemma 3. Then we further bound every term in the RHS in Lemma 3 appropriately, which are realized by Lemma 4, Lemma 5 and Lemma 6. All the detailed proofs of Lemmas can be found in Appendix.

**Lemma 3** Define $v_t = \frac{1}{K} \sum_{k=1}^{K} v_{t,k}, \alpha_t = \frac{1}{K} \sum_{k=1}^{K} \alpha_{t,k}$.

Suppose Assumption 1 holds and by running Algorithm 2, we have

$$\psi(\hat{v}) - \min_\nu \psi(v) \leq \frac{1}{T} \sum_{t=1}^{T} \left( \sum_{k=1}^{K} \|v_{t,k} - v_{t-1,k}\|_2^2 \right) + 2L_v \sum_{t=1}^{T} \left( \sum_{k=1}^{K} \|v_{t,k} - v_{t-1,k}\|_2^2 \right) + \frac{L_\alpha + 3G^2/\nu}{2} \|v_t - v_{t-1}\|^2 + \frac{L_\alpha + 3G^2/L_v}{2} (\alpha_t - \alpha_{t-1})^2 + \frac{L_\alpha + 3G^2/\mu_\alpha}{2} \|v_t - v_{t-1}\|^2 + \frac{L_\alpha + 3G^2/L_v}{2} (\alpha_t - \alpha_{t-1})^2 \frac{A_3}{3} + \frac{2L_v}{3} \|v_{t-1} - v^*\|^2 - L_v \|v_t - v^*\|^2 - \frac{\mu_\alpha}{3} (\alpha_t - \alpha^*)^2 \right].$$

Next, we will bound $A_1, A_2$ in Lemma 4 and Lemma 5. $A_3$ can be cancelled with similar terms in the following two lemmas. The remaining terms will be left to form a telescoping sum with other similar terms in the following two lemmas.

**Lemma 4** Define $\bar{v}_t = \arg \min_\nu \left( \frac{1}{K} \sum_{k=1}^{K} \sum_{k=1}^{K} \|v^* - v_{t,k}\|^2 + \frac{2L_v}{2} \frac{1}{2} \|v_{t-1} - v_{t-1,k}\|^2 + \frac{2L_v}{2} \|v_t - v_{t,k}\|^2 \right).$ We have

$$A_1 \leq \frac{3G^2}{2L_v} \frac{1}{K} \sum_{k=1}^{K} \left( \|v_{t-1,k} - \alpha_{t-1,k}\|^2 + 3L_v \frac{1}{K} \sum_{k=1}^{K} \|v_{t-1,k} - v_{t-1,k}\|^2 \right) + \frac{\eta}{2} \left( 1 - \frac{1}{\sqrt{K}} \right) \sum_{k=1}^{K} \left( \sum_{k=1}^{K} \sum_{k=1}^{K} \|v_{t-1,k} - v_{t-1,k}\|^2 \right) + \frac{1}{2\eta} \|v_{t-1,k} - v_{t-1,k}\|^2 - \|v_t - v_{t,k}\|^2 - \|v_t - v^*\|^2) + \frac{2L_v}{3} \|v_t - v^*\|^2.$$

**Lemma 5** Define $\tilde{\alpha}_t = \tilde{\alpha}_{t-1} + \frac{\eta}{2} \sum_{k=1}^{K} \sum_{k=1}^{K} \|v_{t,k} - v_{t,k}\|^2 + \frac{L_\alpha}{2} \frac{1}{K} \sum_{k=1}^{K} \left( \|v_{t,k} - v_{t,k}\|^2 \right).$ We have

$$A_2 \leq \frac{3G^2}{2L_v} \frac{1}{K} \sum_{k=1}^{K} \sum_{k=1}^{K} \|v_{t-1,k} - v_{t-1,k}\|^2 + \frac{3\eta}{2} \left( \frac{1}{K} \sum_{k=1}^{K} \sum_{k=1}^{K} \|v_{t,k} - v_{t,k}\|^2 \right) + \frac{1}{K} \sum_{k=1}^{K} \sum_{k=1}^{K} \|v_{t,k} - v_{t,k}\|^2 \right) + \frac{1}{2} \left( \|v_{t,0} - v_{t,1}\|^2 - \|v_{t,0} - v_{t,1}\|^2 \right) + \frac{\mu_\alpha}{3} (\alpha_t - \alpha^*)^2 + \frac{1}{2\eta} (\alpha^* - \alpha_t)^2 - (\alpha^* - \alpha_t)^2.$$

The first two terms in the upper bounds of $A_1, A_2$ are the differences between individual solutions and their averages, the third term is the variance of stochastic gradient, and the expectation of the fourth term will diminish. The lemma below will bound the difference between the averaged solution and the individual solutions.

**Lemma 6** If $K$ machines communicate every $I$ iterations, and update with step size $\eta$, then
\[
\frac{1}{K} \sum_{k=1}^{K} \mathbb{E}[\|v_t - v_k^t\|^2] \leq 4\eta^2 I^2 B_1^2 \mathbb{I}_{t>1}
\]
\[
\frac{1}{K} \sum_{k=1}^{K} \mathbb{E}[\|\alpha_t - \alpha_k^t\|^2] \leq 4\eta^2 I^2 B_2^2 \mathbb{I}_{t>1}.
\]

Combining the results in Lemma 3, Lemma 4, Lemma 5 and Lemma 6, we can prove the key Lemma 2.

5. Experiments

In this section, we conduct some experiments to verify our theory. In our experiments, one “machine” corresponds to one GPU. We use a cluster of 4 computing nodes with each computer node having 4 GPUs, which gives a total of 16 “machines”. We would like to emphasize that even 4 GPUs sit on one computing node, they only access to different parts of the data. For the experiment with \( K = 1 \) GPU, We run one computing node by using one GPU. For experiments with \( K = 4 \) GPUs, we run one computer node by using all four GPUs, and for those experiments with \( K = 16 \) GPUs, we use four computing nodes by using all GPUs. We notice that the communication costs among GPUs on one computing node might be less than that among GPUs on different computing nodes. Hence, it should be kept in mind that when comparing with \( K = 4 \) GPUs on different computer nodes, the margin of using \( K = 16 \) GPUs over using \( K = 4 \) GPUs should be larger than what we will see in our experimental results. All algorithms are implemented using PyTorch (Paszke et al., 2019).

Data. We conduct experiments on 3 datasets: Cifar10, Cifar100 and ImageNet. For Cifar10, we split the original training data into two classes, i.e., positive class contains 5 original classes and negative class are composed of the other 5 classes. Cifar100 dataset is split in a similar way, i.e., positive class contains 50 original classes and negative class are composed of the other 50 classes. Testing data for Cifar10 and Cifar100 are the same as the original dataset. For ImageNet dataset, we sample 1% of the original training data as testing data and use the remaining data as the training data. The training data is split in a similar way as Cifar10 and Cifar100, i.e., positive class contains 500 original classes and negative class are composed of the other 500 classes. For each dataset, we create two versions of training data with different positive ratio. By keeping all examples in the positive and negative class, we have \( p = 50\% \) for all three datasets. In order to create imbalanced data, we drop some proportion of the negative data for each dataset and keep all the positive examples. In particular, by keeping all the positive data and 40% of the negative data we construct three datasets with positive ratio \( p = 71\% \). Training data are shuffled and evenly divided to each GPU, i.e., each GPU has access to \( 1/K \) of the training data, where \( K \) is the number of GPUs. For all data, We use ResNet50 as our neural network (He et al., 2016) and initialize the model as the pretrained model from PyTorch. Due to limite of space, we only report the results on datasets with \( p = 71\% \) positive ratio, and other results are included in the supplement.

Baselines and Parameter Setting. For baselines, we compare with the single-machine algorithm PPD-SG as proposed in (Liu et al., 2020b), which is represented by \( K = 1 \) in our results, and the naive parallel version of PPD-SG, which is denoted by \( K = X, I = 1 \) in our results. For all algorithms, we set \( T_s = T_0 t^{3K}, \eta_s = \eta_0 t^{3K}. T_0 \) and \( \eta_0 \) are tuned for PPD-SG and set to the same for all other algorithms for fair comparison. \( T_0 \) is tuned in \([2000, 5000, 10000]\), and \( \eta_0 \) is tuned in \([0.1, 0.01, 0.001]\). We fix the batch size for each GPU as 32. For simplicity, in our experiments we use a fixed value of \( I \) in order to see its tradeoff with the number of machines \( K \).

Results. We plot the curve of testing AUC versus the number of iterations and versus running time. We notice that evaluating the training objective function value on all examples is very expensive, hence we use the testing AUC as our evaluation metric. It might cause some gap between our results and the theory, however, the trend should be enough for our purpose to verify that our distributed algorithms can enjoy faster convergence in both the number of iterations and running time. We have the following observations.

- Varying \( K \). By varying \( K \) and fixing the value of \( I \), we aim to verify the parallel speedup. The results are shown in Figures 1(a), Figures 2(a) and Figures 8(a). They show that when \( K \) becomes larger, then our algorithm requires less number of iterations to converge to the target AUC, which is consistent with parallel speedup result as indicated by Theorem 1. In addition, CoDA with \( K = 16 \) machines is also the most time-efficient algorithm among all settings.

- Varying \( I \). By varying \( I \) and fixing the value of \( K \), we aim to verify that skipping communications up to a certain number of iterations of CoDA does not hurt the iteration complexity but can dramatically reduce the total communication costs. In particular, we fix \( K = 16 \) and vary \( I \) in the range \( \{1, 8, 64, 512, 1024\} \). The results are shown in Figures 1(b), Figures 2(b) and Figures 8(b). They exhibit that even when \( I \) becomes moderately large, our algorithm is still able to deliver comparable performance in terms of the number of iterations compared with the case when \( I = 1 \). The largest value of \( I \) that does not cause dramatic performance drop compared with \( I = 1 \) is \( I = 1024, I = 64, I = 64 \) on ImageNet, CIFAR100 and CIFAR10, respectively. However, up to these thresholds the running time of CoDA can be dramatically reduced than the naive parallel version with \( I = 1 \).

- Trade-off between \( I \) and \( K \). Finally, we verify the trade-off between \( I \) and \( K \) as indicated in Theorem 1.
To this end, we conduct experiments by fixing $K = 4$ GPUs and varying the value $I$, and comparing the limits of $I$ for $K = 4$ and $K = 16$. The results of using $K = 4$ on CIFAR100 and CIFAR10 are reported in Figure 4 and Figure 5. We can observe that when $K = 4$ the upper limit of $I$ that does not cause dramatic performance drop compared with $I = 1$ is $I = 512$ for the two datasets, which is larger than the upper limit of $I = 64$ for $K = 16$. This is consistent with our Theorem 1.

6. Conclusion
In this paper, we have designed a communication-efficient distributed stochastic deep AUC maximization algorithm, in which each machine is able to do multiple iterations of local updates before communicating with the central node. We have proved the linear speedup property and showed that the communication complexity can be dramatically reduced for multiple machines up to a large threshold number. Our empirical studies verify the theory and also demonstrate the effectiveness of the proposed distributed algorithm on benchmark datasets.
Acknowledgements

This work is partially supported by National Science Foundation CAREER Award 1844403.

References


Communication-Efficient Distributed Stochastic AUC Maximization with Deep Neural Networks


