Communication-Efficient Asynchronous Stochastic Frank-Wolfe over Nuclear-norm Balls

Jiacheng Zhuo UT Austin **Qi Lei** UT Austin **Alexandros Dimakis** UT Austin **Constantine Caramanis** UT Austin

Abstract

Large-scale machine learning training suffers from two prior challenges, specifically for nuclear-norm constrained problems with distributed systems: the synchronization slowdown due to the straggling workers, and high communication costs. In this work, we propose an asynchronous Stochastic Frank Wolfe (SFW-asyn) method, which, for the first time, solves the two problems simultaneously, while successfully maintaining the same convergence rate as the vanilla SFW. We implement our algorithm in python (with MPI) to run on Amazon EC2, and demonstrate that SFW-asyn yields speed-ups almost linear to the number of machines compared to the vanilla SFW.

1 Introduction

We consider the problem of minimizing a convex and smooth matrix function subject to a nuclear norm constraint:

$$\min_{\|\boldsymbol{X}\|_* \leq \theta} F(\boldsymbol{X}) = \frac{1}{N}\sum_{i}^{N} f_i(\boldsymbol{X}), \tag{1}$$

where F maps from $\mathbb{R}^{D_1 \times D_2}$ to $\mathbb{R}$, and $\|\boldsymbol{X}\|_*$ denotes the nuclear norm of $\boldsymbol{X}$, i.e., the sum of its singular values. We view the objective F as a summation of N sub-problems $f_1, f_2, \cdots f_N$. This formulation covers various important machine learning applications, including matrix completion, matrix sensing, multi-class and multi-label classification, affine rank minimization, phase retrieval problems, and many more (see e.g. Candes et al. (2015); Allen-Zhu et al. (2017); Lei et al. (2019) and references therein). We are especially interested in the large sample setting, i.e., when the total number of subproblems (data examples) N and model size D_1, D_2 are large.

The Frank-Wolfe methods are frequently used for solving problem 1. As first proposed by Frank and Wolfe (1956), one can compute and update the parameters along the gradient descent direction while remaining within the constraint set. This is thus called the Frank-Wolfe (FW) algorithm or conditional gradient descent. The nuclear norm constraint requires the computation of the leading left and right singular vectors of the gradient matrix, with a complexity of $\mathcal{O}(D_1 D_2)$. A natural alternative among first-order algorithms is the Projected Gradient Descent (PGD). The projection step, however, requires a full SVD per iteration, with much higher complexity: $\mathcal{O}(D_1 D_2 \cdot \min(D_1, D_2))$. This makes PGD computationally expensive for large-scale datasets.

Classical FW runs on a single machine and passes the whole dataset in each iteration. As datasets increase and Moore's Law is slowing down (Simonite, 2016), the move towards stochastic variants of FW has become imperative (Hazan and Luo, 2016). In the meantime, it is also crucial to study distributed FW implementations, where gradient computation and aggregation is parallelized across multiple worker nodes (Zheng et al., 2018; Bellet et al., 2015). Although distributed computation boosts the amount of data that can be processed per iteration, it may introduce prohibitive communication overhead, particularly in the big data setting with high-dimensional parameters. Specifically, there are two challenges: (1) the stragglers dominate each iteration due to synchronization (Liu et al., 2015; Hsieh et al., 2015; Reddi et al., 2015; Recht et al., 2011; Tandon et al., 2017); (2) there is a large communication cost per iteration ($\mathcal{O}(D_1 D_2)$) due to gradient and parameter sharing. For problem (2), (stochastic) FW is natural and appealing: the low-rank up-

Proceedings of the 23[rd]International Conference on Artificial Intelligence and Statistics (AISTATS) 2020, Palermo, Italy. PMLR: Volume 108. Copyright 2020 by the author(s). Correspondence to: `jzhuo@utexas.dot.edu`

dates can be represented as a few vectors and they require fewer bits to transmit than the partial gradients. However, naively aggregating the low-rank updates from the workers does not yield an algorithm that converges, as the Singular Vector Averaging algorithm in the work of Zheng et al. (2018). While natural and promising, there are several significant technical obstacles towards designing an efficient and provably convergent distributed FW algorithm. To the best of our knowledge, no prior work has successfully addressed this.

We thus raise and answer the following two questions in this work:

(1) Can FW type algorithms ***provably*** *run in an asynchronous manner, so that the iteration time is not dominated by the slowest worker?*

(2) Can we improve the communication cost of running FW in the distributed setting?

In this work, we are able to answer both of these questions affirmatively, by proposing and analyzing an asynchronous version of Stochastic Frank-Wolfe algorithm, which we call **SFW-asyn**.

Our contributions.

We propose a Stochastic Frank-Wolfe algorithm (SFW-asyn) that runs asynchronously and is communication-efficient. We establish that SFW-asyn (Algorithm 3) enjoys a $\mathcal{O}(1/k)$ convergence rate, which is in line with the standard SFW. Apart from the asynchronous nature, SFW-asyn reduces the communication cost from $\mathcal{O}(D_1 D_2)$ as in most prior work, to $\mathcal{O}(D_1 + D_2)$.

Our theoretical analysis framework is general for other Frank-Wolfe type method. We specifically provide the extension for Stochastic Variance Reduced Frank-Wolfe (SVRF) (Hazan and Luo, 2016).

Finally, we use Amazon EC2 instances to provide extensive performance evaluation of our algorithm, and comparisons to vanilla SFW and distributed SFW on two tasks, matrix sensing and learning polynomial neural networks. Our results show that SFW-asyn outperforms all the benchmarks and achieves speed-ups almost linear in the number of distributed machines.

Related work.

We mainly overview two different lines of work that are respectively related to the distributed Frank-Wolfe algorithms, and the asynchronous manner in distributed learning.

-*Distributed Frank-Wolfe Algorithms.* For general network topology, Bellet et al. (2015) proposes a distributed Frank-Wolfe algorithm for ℓ_1 and simplex constraints, but not for the nuclearn-norm constraint.

On a master-slave computational model, Zheng et al. (2018) proposes to distribute the exact gradient computation among workers and aggregate the gradient by performing $\mathcal{O}(t)$ distributed power iterations at iteration t. This involves even more frequent synchronizations than the vanilla distributed Frank-Wolfe method and will be hindered heavily by staleness. Specifically, to run T iterations, the distributed FW algorithm proposed by Zheng et al. (2018) requires a total communication cost of $\mathcal{O}(T^2(D_1 + D_2)$, while SFW-asyn only requires $\mathcal{O}(T(D_1 + D_2))$. Furthermore, as we consider the case when data samples are very large, stochastic optimization is more efficient than exact gradient computation, and the whole sample set might not fit into memory. Therefore the method proposed by Zheng et al. (2018) and other full-batch FW methods are beyond the interests of our paper.

Quantization techniques have also been used to reduce the communication cost for SFW, like Zhang et al. (2019), who proposed a novel gradient encoding scheme (s-partition) to reduce iteration communication cost. However their results are for ℓ_1 constrained problem, and are thus not directly comparable to our results.

-*Asynchronous Optimization.* There is a rich previous effort in the research on asynchronous optimization, such as asynchronous Stochastic Gradient Descend (SGD) (Recht et al., 2011), asynchronous Coordinate Descend (CD) (Liu et al., 2015; Hsieh et al., 2015), and asynchronous Stochastic Variance Reduced Gradient (SVRG) (Reddi et al., 2015). Mania et al. (2015) proposes a general analysis framework, which however, is not applicable for SFW. Wang et al. (2014) proposes Asynchronous Block Coordinate FW and achieves sub-linear rate. They assume the parameters are separable by coordinates, and hence different from our problem setting.

2 Preliminary

2.1 Frank-Wolfe method

We start by reviewing the classical (Stochastic) Frank-Wolfe (FW/SFW) algorithms. Consider a general constrained optimization problem $\min_{\boldsymbol{X}\in\Omega}\left\{F(\boldsymbol{X}) := \frac{1}{N}\sum_i^N f_i(\boldsymbol{X})\right\}$. FW proceeds in each iteration by computing:

$$\boldsymbol{U}_k = \underset{\boldsymbol{U}\in\Omega}{\arg\min} \quad \langle \nabla F(\boldsymbol{X}_{k-1}), \boldsymbol{U}\rangle, \tag{2}$$

$$\boldsymbol{X}_k = (1-\eta_k)\boldsymbol{X}_{k-1} + \eta_k \boldsymbol{U}_k. \tag{3}$$

and SFW replaces the full batch gradient by a mini-batch gradient

$$\boldsymbol{U}_k = \arg\min_{\boldsymbol{U}\in\Omega} \left\langle \frac{1}{|\mathcal{S}_k|}\sum_{i\in\mathcal{S}_k} \nabla f_i(\boldsymbol{X}), \boldsymbol{U} \right\rangle, \quad (4)$$

$$\boldsymbol{X}_k = (1-\eta_k)\boldsymbol{X}_{k-1} + \eta_k \boldsymbol{U}_k. \quad (5)$$

where $\mathcal{S}_k$ is a randomly selected index set. We will refer to Eqns (2) and (4) as linear optimization steps.

The complexity of the Frank-Wolfe type methods depends on that of the linear optimization steps. Fortunately, for a wide class of constraints Ω, the linear optimization, with the projection-free nature, is of relatively low computational complexity. For instance when Ω is $\|\boldsymbol{X}\|_* \leq 1$, the linear optimization step is to compute the leading left singular vector $\boldsymbol{u}$ and the right singular vector $\boldsymbol{v}$ of the negative gradient (or mini-batch gradient), and return $\boldsymbol{u}\boldsymbol{v}^T$.

2.2 Computational Model

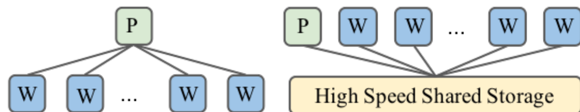

Figure 1: *Illustration on different distributed frameworks.* 'P' stands for Parameter-server, and 'W' denotes workers. We focus on the master-slave paradigm (left), where all the workers are directly connected to the master node. Our algorithms and analysis can be easily modified for a shared-memory model (right), where all the machines have high speed access to one storage device, instead of being connected by network.

In this work we focus on the master-slave computational model, which is commonly used for distributed applications (Li et al., 2014a). In the master-slave model, we have W worker (slave) nodes and one Master node (parameter server). Each worker has access to all the data and therefore is able to compute $f_i(\boldsymbol{X})$ and $\nabla f_i(\boldsymbol{X})$ when $\boldsymbol{X}$ is given. The Master node has direct connection to each worker node and serves as the central coordinator by maintaining the model parameters.

Although synchronous framework is commonly used for iterative algorithms as they require minimal modification of the original algorithms, the running time of each iteration is confined by the slowest worker (Li et al., 2014b; Dean et al., 2012). This is why *asynchronous distributed algorithms* are desirable: each machine just works on their assigned computations, and communicates by sending signals or data asynchronously (Recht et al., 2011; Liu et al., 2015; Hsieh et al., 2015). The benefits of asynchronous algorithms are obvious: the computational power for each machine is fully utilized, since no waiting for the straggling workers is necessary. Unfortunately, the asynchronous version of an iterative algorithm may suffer from the effect of staleness.

Definition 1. *In the asynchronous computational model, workers may return stale gradients that were evaluated at an older version of the model* $\boldsymbol{X}$*; we call that there is a staleness of* τ *if* $\boldsymbol{X}_{t+1}$ *is updated according to* $\boldsymbol{X}_{t-\tau}$.

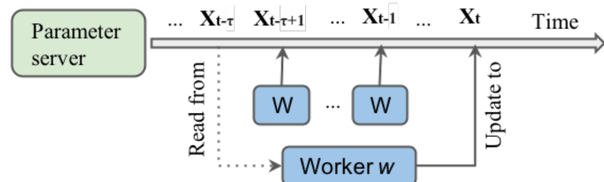

Figure 2: Delay (staleness) for a worker w is the number of updates by other workers between two updates of w.

As shown in Figure 2, staleness happens when a worker tries to update the parameter server with out-dated information. In other words, when each worker deals with its own computation, other workers may have updated the model of the parameter server already. Staleness on a worker w during t execution cycle is a series of t random variables, and we call this *staleness process* for the worker w. We require mutual independence on the staleness process, and the independence between the staleness process among workers and the sampling process of a worker.

Assumption 1. *The staleness process of worker* w_i *and the staleness process of worker* w_j *are independent, for all* $i \neq j$.

Assumption 2. *The staleness process of worker* w *is independent with the sampling process (for the stochastic gradient computation) on the worker* w.

These two assumptions are standard in the analysis of asynchronous algorithms (Mitliagkas et al., 2016).

Although we focus on the master-slave computational model, our algorithm and analysis can be easily modified for the shared-memory computational model (Recht et al., 2011), where all the workers have high speed access to one shared storage device. A typical example is a server with multiple CPUs, and these CPUs have access to a same piece of memory. In the shared-memory model, communication is no longer an issue. Our proposal mostly benefits from the lock-free nature under this setting.

2.3 Notation

We use bold upper case letter for matrices, e.g. $\boldsymbol{X}$, and bold lower case letter for vectors, e.g. $\boldsymbol{x}$. We define the matrix inner product as $\langle \boldsymbol{X}, \boldsymbol{Y}\rangle =$

Algorithm 1 Distributed Stochastic Frank-Wolfe (SFW-dist) (A baseline method)

1: **Input:** Max iteration count T; Step size η_t and batch size m_t for iteration t.
2: **Initialization:** Random X_0 s.t. $\|\boldsymbol{X}_0\|_* = 1$.
3: **for** iteration $k = 1, 2, \cdots, T$ **do**
4: Broadcast $\boldsymbol{X}_{k-1}$ to all workers.
5: **for** each worker $w = 1, 2, \cdots, W$ **do**
6: Randomly sample an index set S_w where $|S_w| = m_k/W$
7: Compute and send $\sum_{i \in S_w} \nabla f_i(\boldsymbol{X}_{k-1})$ to the master
8: **end for**
9: $\nabla_k = \sum_{w=1}^{W} \sum_{i \in S_w} \nabla f_i(\boldsymbol{X}_{k-1})$
10: $\boldsymbol{U}_k \leftarrow \mathrm{argmin}_{\|\boldsymbol{U}\|_* \leq \theta} \langle \nabla_k, \boldsymbol{U} \rangle$
11: $\boldsymbol{X}_k \leftarrow \eta_k \boldsymbol{U}_k + (1 - \eta_k) \boldsymbol{X}_{k-1}$
12: **end for**

trace$(\boldsymbol{X}^T \boldsymbol{Y})$. A function F is convex on set Ω if

$$F(\boldsymbol{Y}) \geq F(\boldsymbol{X}) + \langle \nabla F(\boldsymbol{X}), \boldsymbol{Y} - \boldsymbol{X} \rangle, \forall \boldsymbol{X}, \boldsymbol{Y} \in \Omega.$$

Similarly, F is L-smooth on set Ω when

$$\|\nabla F(\boldsymbol{X}) - \nabla F(\boldsymbol{Y})\| \leq L \|\boldsymbol{X} - \boldsymbol{Y}\|, \forall \boldsymbol{X}, \boldsymbol{Y} \in \Omega.$$

We use F^* to denote $\min_{\|\boldsymbol{X}\|_* \leq \theta} F(\boldsymbol{X})$, namely the minimal value that F can achieve under the constraints. Note there could be multiple optimal $\boldsymbol{X}$, and hence we use $\boldsymbol{X}^*$ to denote one of them, unless otherwise specified. Specifically for Problem 1, define D as the diameter of the constraint set: $D = \max_{\|\boldsymbol{X}\|_* \leq \theta, \|\boldsymbol{Y}\|_* \leq \theta} \|X - Y\|_F$, and define G as the variance of the stochastic gradient: $G^2 \leq \mathbb{E} \|\nabla f_i(\boldsymbol{X}) - \nabla F(\boldsymbol{X})\|_F^2$, for all $\|\boldsymbol{X}\|_* \leq \theta$.

3 Methodology

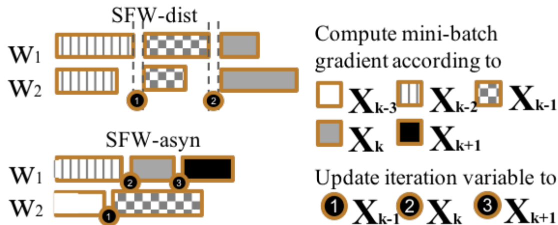

Figure 3: *Illustration of simplified asynchronous computation scheme.* 'W' denotes workers, and 'X' denotes the iteration variable. All the subscript of 'X' refer to the iteration number at the master node. In SFW-asyn, the update '1' updates the model to X_{k-1} with gradient computed according to X_{k-3}, and hence the delay is 1. The delay is 1 and 0 for the update '2' and '3' respectively.

A natural way to deploy Stochastic Frank-Wolfe (SFW) on a master-slave distributed paradigm is explained as follows in four steps. For each iteration, we first have each worker $w = 1...W$ to compute $1/W$ portion of the mini-batch gradients. The master node next collects the mini-batch gradients from all the workers. It then computes the update for $\boldsymbol{X}$, as in Eqn.(4), Eqn.(5). Finally the master node broadcasts the updated $\boldsymbol{X}$ to all the workers. We call this approach **SFW-dist** and describe it in Algorithm 1. SFW-dist and its full batch variant serve as competitive baselines for previous state-of-the-arts (Zheng et al., 2018).

Algorithm 2 Naive Asynchronous Stochastic Frank-Wolfe Method (Only for analysis)

1: // The Master Node
2: **Input:** Max delay tolerance τ; Max iteration count T; Step size η_t and batch size m_t for iteration t.
3: **Initialization:** Random X_0 s.t. $\|\boldsymbol{X}_0\|_* = 1$; broadcast X_0 to all the workers; the iteration count at the master node $t_m = 0$.
4: **while** $t_m < T$ **do**
5: Wait until $\{\boldsymbol{U}_w, t_w\}$ is received from a worker w.
6: // $\boldsymbol{U}_w$ is computed by worker w according to $\boldsymbol{X}_{t_w}$
7: If $t_m - t_w > \tau$, abandon $\boldsymbol{U}_w$ and continue.
8: $t_m = t_m + 1$
9: $\boldsymbol{X}_{t_m} \leftarrow \eta_{t_m} \boldsymbol{U}_w + (1 - \eta_{t_m}) \boldsymbol{X}_{t_m - 1}$
10: // Update $\boldsymbol{X}$ in the master node with $\boldsymbol{U}_{t_w}$.
11: Broadcast $\boldsymbol{X}_{t_m}$ to all the workers
12: **end while**
13: // For each worker $w = 1, 2, \cdots, W$
14: **while** No Stop Signal **do**
15: Receive from the Master $\boldsymbol{X}_{t_m}$.
16: // t_m is the iteration count at the master node.
17: $t_w = t_m$, $\boldsymbol{X}_w = \boldsymbol{X}_m$.
18: // Update the local copy of $\boldsymbol{X}$ and iteration count.
19: Randomly sample an index set S where $|S| = m_{t_w}$
20: $\boldsymbol{U}_w \leftarrow \mathrm{argmin}_{\|\boldsymbol{U}\|_* \leq \theta} \langle \sum_{i \in S} \nabla f_i(\boldsymbol{X}_w), \boldsymbol{U} \rangle$
21: Send $\{\boldsymbol{U}_w, t_w\}$ to the Master node.
22: **end while**

However, SFW-dist has two obvious drawbacks: **high communication cost** and **synchronization slow down.** Specifically, the master has to collect a gradient estimation from each worker for each iteration, while the size of the gradient is $\mathcal{O}(D_1 \cdot D2)$. When D_1 and D_2 are large, this communication cost is unaffordable (Zheng et al., 2018). On the other hand, the master has to wait to receive the gradient matrix from all workers in order to compute each update, and therefore the time cost per iteration is dominated by the slowest worker.

We thus propose Asynchronous Stochastic Frank-Wolfe method (SFW-asyn), which simultaneously resolves the above two challenges.

For the ease of understanding, we gradually build up to our methodology and first introduce a naive version of asynchronous SFW in Algorithm 2. It only bares the benefit of asynchronous nature but no efficient communication yet. Algorithm 2 is for illustration and analysis only, and not for implementation.

Algorithm 3 Asynchronous Stochastic Frank-Wolfe Method (SFW-asyn)

1: // The Master Node
2: **Input:** Max delay tolerance τ; Max iteration count T; Step size η_t; Batch size m_t for iteration t;
3: **Initialization:** Randomly initialize $\boldsymbol{X}_0 = \boldsymbol{u}_0\boldsymbol{v}_0^T$ s.t. $\|\boldsymbol{X}_0\|_* = 1$ and broadcast $\{\boldsymbol{u}_0, \boldsymbol{v}_0\}$ to all the workers; The iteration count at the master node $t_m = 0$.
4: **while** $t_m < T$ **do**
5: Wait until $\{\boldsymbol{u}_w, \boldsymbol{v}_w, t_w\}$ is received from a worker w. // $\boldsymbol{u}_w, \boldsymbol{v}_w$ are computed according to $\boldsymbol{X}_{t_w}$.
6: **if** $t_m - t_w > \tau$ **then**
7: Send $(\boldsymbol{u}_{t_m}, \boldsymbol{v}_{t_m}), ...(\boldsymbol{u}_{t_w+1}, \boldsymbol{v}_{t_w+1})$ to node w.
8: **continue**.
9: **end if**
10: $t_m = t_m + 1$ and store $\{\boldsymbol{u}_w, \boldsymbol{v}_w\}$ as $\boldsymbol{u}_{t_m}$ and $\boldsymbol{v}_{t_m}$
11: Send $(\boldsymbol{u}_{t_m}, \boldsymbol{v}_{t_m}), ...(\boldsymbol{u}_{t_w+1}, \boldsymbol{v}_{t_w+1})$ to node w.
12: $X_{t_m} \leftarrow \eta_{t_m} u_{t_m} v_{t_m}^T + (1 - \eta_{t_m}) X_{t_m - 1}$ // Not in real time; maintain a copy of $\boldsymbol{X}_k$ for output only.
13: **end while**
14: // For each worker $w = 1, 2, \cdots, W$
15: **while** No Stop Signal **do**
16: Obtain $(\boldsymbol{u}_{t_m}, \boldsymbol{v}_{t_m}), ...(\boldsymbol{u}_{t_w+1}, \boldsymbol{v}_{t_w+1})$ from the master.
17: Update the local copy of $\boldsymbol{X}_{t_w}$ to $\boldsymbol{X}_{t_m}$ (Eqn.(6));
18: Update the local iteration count $t_w = t_m$.
19: Randomly sample an index set S where $|S| = m_{t_w}$
20: $\boldsymbol{u}_w\boldsymbol{v}_w^T \leftarrow \text{argmin}_{\|\boldsymbol{U}\|_* \le \theta} \langle \sum_{i\in S} \nabla f_i(\boldsymbol{X}_{t_w}), \boldsymbol{U}\rangle$
21: send $\{\boldsymbol{u}_w, \boldsymbol{v}_w, t_w\}$ to the Master node.
22: **end while**

Then we will explain how to reduce the iteration complexity to $\mathcal{O}(D_1 + D_2)$ by making use of the low rank update nature of SFW. The whole algorithm, SFW-asyn, is described in Algorithm 3.

As described in Algorithm 2, each worker computes a mini-batch gradient according to the latest $\boldsymbol{X}$ it has access to, and sends the updates to the master. We denote by t_w as the iteration count of $\boldsymbol{X}$ that the worker w is using to compute the updates. When the update from the worker w reaches the master node, the $\boldsymbol{X}$ in the master node reaches t_m already, due to updates from the other workers. If the delay, defined as $t_m - t_w$, exceeds the maximum delay tolerance τ, the master node will abondon this update. Otherwise, the master node updates the $\boldsymbol{X}$ accordingly and broadcasts the new $\boldsymbol{X}$ to all the workers. Algorithm 2 does not have to wait for one particular slow worker in order to proceed, and therefore is resilient to the staleness of workers.

While Algorithm 2 addresses the straggler problem, the communication cost is still $\mathcal{O}(D_1D_2)$. The intuition of reducing the communication cost is that, all the updates matrix $\boldsymbol{U}$ are rank one matrices, and can be perfectly represented by the outer products of two vectors. A natural idea is thus to transfer and store the vectors for potential updates, instead of the whole gradient matrices. We ask the workers to transfer, instead of $\boldsymbol{U}$, two vectors $\boldsymbol{u}, \boldsymbol{v}$ with $\boldsymbol{u}\boldsymbol{v}^T = \boldsymbol{U}$, to the master. The master transfers back $\{\boldsymbol{u}_{t_w}, \boldsymbol{v}_{t_w}\}, ..., \{\boldsymbol{u}_{t_m+1}, \boldsymbol{v}_{t_m+1}\}$, instead of $\boldsymbol{X}_{t_m}$, to the worker w, and let the worker w to update its own copy of $\boldsymbol{X}_{t_w}$ to $\boldsymbol{X}_{t_m}$ by recursively computing

$$\boldsymbol{X}_k = (1 - \eta_k)\boldsymbol{X}_{k-1} + \eta_k \boldsymbol{u}_k \boldsymbol{v}_k^T. \quad (6)$$

From this practice, we reduce the communication cost to $\mathcal{O}(D_1 + D_2)$.

Although we introduce a few extra operations that require (D_1D_2) computation as in the lines 3 of Algorithm 3, we lifted the need to aggregate gradients from workers as in SFW-dist, which also requires $\mathcal{O}(D_1D_2)$ computation. Since the linear optimization (in this case, 1-SVD) requires $\mathcal{O}(D_1D_2)$ computation anyways, these operations do no change the asymptotic computation time.

Following the similar spirit, we propose the asynchronous Stochastic Variance Reduced Frank-Wolfe method (SVRF-asyn). We defer the full version (both asynchronous and communication efficient) to the Appendix in Algorithm 4 (not communication efficient, and only for analysis instead of implementation) and Algorithm 5.

Communication Cost of SFW-asyn For each iteration, one worker is communicating with the master. The worker sends a $\{\boldsymbol{u}_{t_w}, \boldsymbol{v}_{t_w}, t_w\}$ to the master, and the master sends back an update sequence $\{\boldsymbol{u}_{t_m}, \boldsymbol{v}_{t_m}\}, ..., \{\boldsymbol{u}_{t_w+1}, \boldsymbol{v}_{t_w+1}\}$, to the worker. Consider the following amortized analysis. From iteration 1 to iteration T, a worker w received at most T pairs of $\{\boldsymbol{u}, \boldsymbol{v}\}$ from the parameter server so that to keep its local copy of $\boldsymbol{X}$ up-to-date. For all the W workers, the total amount of message they send to the parameter server is T pairs of $\{\boldsymbol{u}, \boldsymbol{v}\}$. Therefore, from iteration 1 to iteration T, for all the W workers, messages sent from the parameter server to the workers are at most with the size of $(TW(D_1+D_2))$, and the messages sent from workers to the parameter server are at most with the size of $(T(D_1+D_2))$. For each iteration, on average, the communication cost is $\mathcal{O}(W(D_1 + D_2))$. Since in the master-slave computational model, each worker has a direct connection channel with the parameter server, the communication cost along each channel is $\mathcal{O}(D_1 + D_2)$ per iteration.

4 Theoretical Analysis

The analysis shown in this Section are developed under Assumption 1 and 2. We first present the convergence results for SFW-asyn, and the discuss the convergence behavior in a more practical scenario

where the batch size is fixed (or capped).

Theorem 1. ***(The $\mathcal{O}(1/k)$ convergence rate of SFW-asyn).*** *Consider problem 1, where $F(\boldsymbol{X})$ is convex and L-smooth over the nuclear-norm ball. Denote $h_k = F(\boldsymbol{X}_k) - F^*$, where $\boldsymbol{X}_k$ is the output of the kth iteration of SFW-asyn. If for all $i < k$, $m_i = \frac{G^2(i+1)^2}{\tau^2 L^2 D^2}$, $\eta_i = \frac{2}{i+1}$, $\tau < T/2$,*

$$\mathbb{E}\left[h_k\right] \leq \frac{(3\tau+1)\cdot 4LD^2}{k+2} \tag{7}$$

We defer the proof to the appendix. The $\mathcal{O}(1/k)$ convergence rate of SFW-asyn matches the convergence rate of the original SFW (Hazan and Luo, 2016). While there is a $\mathcal{O}(\tau)$ slowdown in the convergence rate comparatively, we only require a $\mathcal{O}(i^2/\tau^2)$ batch size for each iteration i, instead of $\mathcal{O}(i^2)$ as in the original SFW.

The requirement of increasing batch size for SFW is that, there is no self-tuning gradient variance, and decreasing step size is not enough to control the error introduced by stochastic gradient.

The intuition of why SFW-asyn requires a small batch size for each iteration is that, while the error of asynchronous update already dominates the error for each iteration (by a factor of τ), it will not change the order-wise behavior by having the gradient variance to be in the same scale as the error introduced by asynchronous update.

Our analysis, detailed in the appendix, is based on perturbed iterate analysis. This analysis is generalizable: with minor modification, we can show that SVRF-asyn converges, as in the below theorem.

Theorem 2. ***(The convergence rate of SVRF-asyn).*** *Consider problem 1, where $F(\boldsymbol{X})$ is convex and L-smooth over the nuclear-norm ball. Denote $h_k = F(\boldsymbol{W}_k) - F^*$, where $\boldsymbol{W}_k$ is the output of the SVRF-asyn at its k outer iteration. If $m_k = \frac{96(k+1)}{\tau}$, $\eta_k = \frac{2}{k+1}$, $N_t = 2^{t+3} - 2$, the maximum delay is τ, then*

$$\mathbb{E}\left[h_k\right] \leq \frac{(3\tau+12)\cdot 4LD^2}{2^{k+1}} \tag{8}$$

We defer the proof to the appendix. This convergence rate is the same as in original SVRF (Hazan and Luo, 2016).

4.1 Constant Batch Size

As discuss in the previous sub-section, the necessity of increasing batch size is not brought up by our asynchronous modification, but in the original Stochastic Frank-Wolfe method (Hazan and Luo, 2016). However as the size of the mini-batch increases, it may get close to the size of the entire dataset, which often violates the interests of practical implementation. Below we briefly discuss two side-steps: (1) When the problem has the *shrinking gradient variance*, the requirement of increasing batch size can be lifted (2) when only an approximately good result is needed (which is often the case when FW method is used), a fixed batch size will lead to a convergence to the optimal neighbourhood, with the same iteration convergence rate, for both the original SFW and our proposed SFW-asyn.

We show first that as long as the variance of the stochastic gradient is diminishing as the algorithm proceeds, the requirement of increasing batch size can be lifted.

Definition 2. ***(shrinking gradient variance.)*** *A function $F(\boldsymbol{X}) = \frac{1}{n} f_i(\boldsymbol{X})$ has shrinking gradient variance if the following holds:*

$$\mathbb{E}\left[\|\nabla f_i(\boldsymbol{X}_k) - \nabla F(\boldsymbol{X}_k)\|_F^2\right] \leq \frac{G^2}{(k+2)^2} \tag{9}$$

as $\mathbb{E}\left[\|\boldsymbol{X}_k - \boldsymbol{X}^*\|_F\right] \leq \frac{c}{k+2}$ for some constant c and G, and for an optimal solution $\boldsymbol{X}^*$.

For example, it is easy to verify that the noiseless matrix completion problem and the noiseless matrix sensing problem have the property of shrinking gradient variance.

It is easy to check that, under the *shrinking gradient variance* condition, one can change the batch size requirement from $m_i = \frac{G^2(i+1)^2}{L^2D^2}$ to $m = \frac{G^2(c)^2}{L^2D^2}$ for SFW and from $m_i = \frac{G^2(i+1)^2}{\tau^2 L^2 D^2}$ to $m = \frac{G^2(c)^2}{\tau^2 L^2 D^2}$ for SFW-asyn, and maintain the same convergence rate as in Theorem 3 in Hazan and Luo (2016) and Theorem 1 in the previous subsection.

While the shrinking gradient variance property is not presented, having a constant batch size will have SFW and SFW-asyn converges to a local neighbourhood of the optimal value:

Theorem 3. ***(SFW converges to a neighbour of the optimal with constant batch size).*** *Consider problem 1, where $F(\boldsymbol{X})$ is convex and L-smooth over the nuclear-norm ball. Denote $h_k = F(\boldsymbol{X}_k) - F^*$, where $\boldsymbol{X}_k$ is the output of the kth iteration of SFW. If for all $i < k$, $m_i = \frac{G^2c^2}{L^2D^2}$ for a constant c, $\eta_i = \frac{2}{i+1}$, then*

$$\mathbb{E}\left[h_k\right] \leq \frac{4LD^2}{k+2} + \frac{1}{c}LD^2 \tag{10}$$

The first term, $\frac{4LD^2}{k+2}$ is inline with the convergence rate of the original SFW (Hazan and Luo, 2016). Having a fixed batch size will incur a residual error $\frac{1}{c}LD^2$ controlled by the batch size c. Similar con-

vergence result holds for SFW-asyn:

Theorem 4. ***(SFW-asyn converges to a neighbour of the optimal with constant batch size).*** *Consider problem 1, where $F(\boldsymbol{X})$ is convex and L-smooth over the nuclear-norm ball. Denote $h_k = F(\boldsymbol{X}_k) - F^*$, where $\boldsymbol{X}_k$ is the output of the kth iteration of SFW-asyn. If for all $i < k$, $m_i = \frac{G^2c^2}{\tau^2L^2D^2}$ for a constant c, $\eta_i = \frac{2}{i+1}$, $\tau < T/2$,*

$$\mathbb{E}\left[h_k\right] \leq \frac{(4\tau+1)\cdot 2LD^2}{k+2} + \frac{\tau}{c}LD^2 \tag{11}$$

Note that the batch size in Theorem 4 is τ^2 times smaller than the batch size in Theorem 3. The proofs of the above two theorems are in the appendix.

Corollary 1. ***(Complexity to reach ϵ accuracy with fixed batch size)*** *For Algorithm 3 to achieve $F(\boldsymbol{X}) - F^* \leq \epsilon$, we need $\mathcal{O}\left(\frac{\tau}{\epsilon-\tau/c}\right)$ iterations in the master node. It means, among all the machines, in total, we need $\mathcal{O}\left(\frac{c^2}{\tau\epsilon-\tau^2/c}\right)$ stochastic gradient evaluations, and $\mathcal{O}\left(\frac{\tau}{\epsilon-\tau/c}\right)$ times linear optimization (1-SVD).*

We defer the proof to the Appendix.

	# Sto. Grad.	**# Lin. Opt.**
SFW-asyn	$\mathcal{O}\left(\frac{c^2}{\tau\epsilon-\tau^2/c}\right)$	$\mathcal{O}\left(\frac{\tau}{\epsilon-\tau/c}\right)$
SFW	$\mathcal{O}\left(\frac{c^2}{\epsilon-1/c}\right)$	$\mathcal{O}\left(\frac{1}{\epsilon-1/c}\right)$

Table 1: Complexity comparison between SFW-asyn and SFW (Hazan and Luo, 2016) with fixed batch size (defined by the constant c, see Theorem 3 and 4). **# Linear Opt.** is the abbreviation of **the number of linear optimization** and **# Sto. Grad.** is the abbreviation of **the number of stochastic gradient evaluation**.

How good are these complexity results? To interpret Table 1 in a simple way, one could consider the case using a very large batch size c, and therefore SFW-asyn roughly reduces the stochastic gradient evaluations to $\frac{1}{\tau}$ portion of SFW, and requires τ times of linear optimizations. This is a good trade-off between the two processes considering we tackle the problems with very large-scale data-set where the stochastic gradient evaluation will dominate the computation for each iteration. In this sense, by simply viewing the number of operations required in each model update, SFW-asyn is already on par with or better than vanilla SFW, not including the speed-ups due to distributed computations with multiple workers.

5 Empirical Results

In this section we empirically show the convergence performance of SFW-asyn, as supported by our theorems. We also show the speedup of SFW-asyn over SFW-dist on AWS EC2.

5.1 Setup

First let's review two machine learning applications: **Matrix sensing**. Matrix sensing problem is to estimate a low rank matrix $\boldsymbol{X}^*$, with observations of sensing matrices $\boldsymbol{A}_i$ and sensing response $y_i = \langle \boldsymbol{A}_i, \boldsymbol{X}^* \rangle$, for $i = 1...N$. It is an important problem that appears in various fields such as quantun computing, image processing, system design, and so on (see Park et al. (2016) and the references there in). Its connection to neural network is also an active research topic (Li et al., 2017).
Polynomial Neural Network (PNN). PNN is the neural network with polynomial activation function. PNN has been shown to have universal representation power just as neural networks with sigmoid and ReLU activation function (Livni et al., 2014).

We test the performance of SFW-asyn on minimizing the empirical risk of the matrix sensing problem with synthesized data: (1) generate the ground truth matrix $\boldsymbol{X}^* = \boldsymbol{U}\boldsymbol{V}^T/\|\boldsymbol{U}\boldsymbol{V}^T\|_*$ where $\boldsymbol{U}, \boldsymbol{V} \in \mathbb{R}^{30\times 3}$ are generated by uniformly sampling from 0 to 1 for each entry; (2) generate $N = 90000$ sensing matrices $\boldsymbol{A}_i$ by sampling each entry from a standard normal distribution (3) compute response $y_i =< \boldsymbol{A}_i, \boldsymbol{X}^* > +\epsilon$ where ϵ is sampled from a Gaussian distribution with mean 0 and standard deviation 0.1. The empirical risk minimization for this task is

$$\min_{\|\boldsymbol{X}\|_* \leq 1} F(\boldsymbol{X}) = \frac{1}{N}\sum_i^N (\langle \boldsymbol{A}_i, \boldsymbol{X}\rangle - y_i)^2.$$

We also test SFW-asyn on minimizing the training loss of a Two layers PNN with quadratic activation function and smooth hinge loss to classify MNIST dataset of handwritten digits. In MNIST we have 60000 training data $(\boldsymbol{a}_i, y_i)$, where $\boldsymbol{a}_i$ are vectorized $28 * 28$ pixels images, and y_i are the labels. We set $y_i = -1$ if the label is $0, 1, 2, 3, 4$ or $y_i = 1$ otherwise. We divide the data $\boldsymbol{a}_i$ by 255 and therefore they are within zero and one. Hence the training objective of this neural network is

$$\min_{\|\boldsymbol{X}\|_* \leq \theta} F(\boldsymbol{X}) = \frac{1}{N}\sum_i^N \text{s-hinge}(y_i, \boldsymbol{a}_i^T \boldsymbol{X} \boldsymbol{a}_i),$$

where the smooth hinge loss function s-hinge(y, t) equals $0.5 - ty$ if $ty \leq 0$, equals $(0.5 \cdot (1 - ty))^2$ if $0 \leq ty \leq 1$, and equals 0 otherwise. While one can tune θ for good classification performance, we only

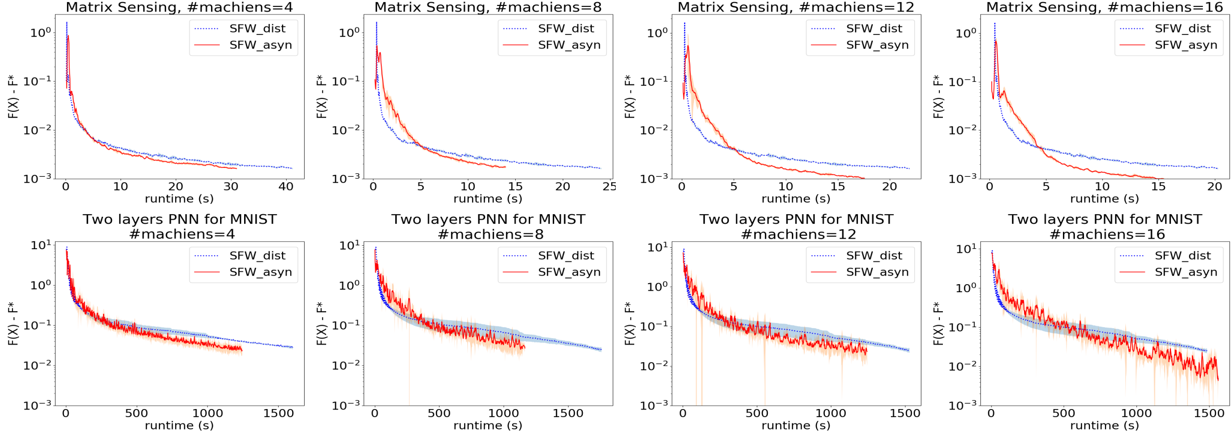

Figure 4: Convergence of the relative loss vs runtime on AWS EC2 cluster. The first row shows the results for Matrix Sensing problem on synthesized data, and the second row shows that results for PNN on MNIST classification. The averages of 10 runs are shown, and the range of one standard deviation is shown as a shadow overlay.

show the results for $\theta = 1$ as we are only interested in minimizing the objective value.

The hyper-parameters are mostly chosen as indicated by our analysis in Section 4. We set the maximum batch-size to be 10000 for the matrix sensing, and 3000 for the PNN, such that the gradient computation time dominates the 1-SVD computation.

The distributed computation environment is set-up on AWS EC2. We leave the details in the Appendix.

5.2 Result analysis

We show the convergence results against the wall clock time on AWS EC2 in Figure 4, and show the speedup against single worker in Figure 5.

Both SFW-dist and SFW-asyn obtain a better speedup result for the Matrix Sensing problem than PNN. SFW-dist has up to five times speedup for the Matrix Sensing problem, but only has marginal speedup on PNN problem, because the variable matrix size is much larger in the PNN problem than the Matrix Sensing problem. The variable matrix size in the Matrix Sensing problem is $30 \times 30 = 900$, while in PNN it is $784 \times 784 \approx 640k$. And hence the speedup is quickly counteracted by the communication overhead. Although the communication cost of SFW-asyn is not as sensitive as SFW-dist to the increase of the matrix size, a larger variable matrix does increase the computation cost of the linear optimization. As in our Corollary 1, SFW-asyn might comparatively perform more linear optimization, and hence SFW-asyn compromises in terms of the overall speedup ratio.

Nevertheless, the performance of SFW-asyn consistently outperforms SFW-dist.

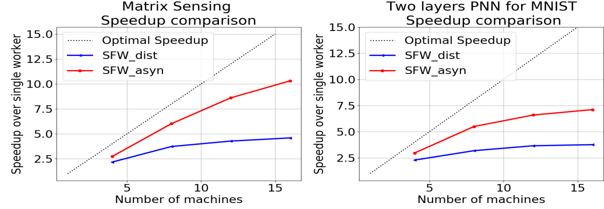

Figure 5: Comparing the time needed to achieve the same relative error (0.02 for PNN and 0.001 for Matrix Sensing) against single worker.

6 Conclusion

In this work we propose Asynchronous Stochastic Frank-Wolfe method, which simultaneously address the synchronization slow-down problem, and reduce the communication cost from $\mathcal{O}(D_1 D_2)$ to $\mathcal{O}(D_1 + D_2)$. We establish the convergence guarantee for SFW-asyn that matches vanilla SFW, and show by simulations that SFW-asyn achieves significant speedup over the baseline approach.

Acknowledgement

This research has been supported by NSF Grants 1618689, 1609279, 1646522, 1704778, DMS 1723052, CCF 1763702, AF 1901292 and research gifts by Google, Western Digital and NVIDIA.

References

Emmanuel J Candes, Yonina C Eldar, Thomas Strohmer, and Vladislav Voroninski. Phase retrieval via matrix completion. *SIAM review*, 2015.

Zeyuan Allen-Zhu, Elad Hazan, Wei Hu, and Yuanzhi Li. Linear convergence of a frank-wolfe type algorithm over trace-norm balls. In *Advances in Neural Information Processing Systems*, pages 6191–6200, 2017.

Qi Lei, Jiacheng Zhuo, Constantine Caramanis, Inderjit S Dhillon, and Alexandros G Dimakis. Primal-dual block generalized frank-wolfe. In *Advances in Neural Information Processing Systems*, pages 13866–13875, 2019.

Marguerite Frank and Philip Wolfe. An algorithm for quadratic programming. *Naval research logistics quarterly*, 3(1-2):95–110, 1956.

Tom Simonite. Moore's law is dead. now what, 2016.

Elad Hazan and Haipeng Luo. Variance-reduced and projection-free stochastic optimization. In *International Conference on Machine Learning*, pages 1263–1271, 2016.

Wenjie Zheng, Aurélien Bellet, and Patrick Gallinari. A distributed frank–wolfe framework for learning low-rank matrices with the trace norm. *Machine Learning*, 107(8-10):1457–1475, 2018.

Aurélien Bellet, Yingyu Liang, Alireza Bagheri Garakani, Maria-Florina Balcan, and Fei Sha. A distributed frank-wolfe algorithm for communication-efficient sparse learning. In *Proceedings of the 2015 SIAM International Conference on Data Mining*, pages 478–486. SIAM, 2015.

Ji Liu, Stephen J Wright, Christopher Ré, Victor Bittorf, and Srikrishna Sridhar. An asynchronous parallel stochastic coordinate descent algorithm. *The Journal of Machine Learning Research*, 16(1): 285–322, 2015.

Cho-Jui Hsieh, Hsiang-Fu Yu, and Inderjit S Dhillon. Passcode: Parallel asynchronous stochastic dual co-ordinate descent. In *ICML*, volume 15, pages 2370–2379, 2015.

Sashank J Reddi, Ahmed Hefny, Suvrit Sra, Barnabas Poczos, and Alexander J Smola. On variance reduction in stochastic gradient descent and its asynchronous variants. In *Advances in Neural Information Processing Systems*, pages 2647–2655, 2015.

Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in neural information processing systems*, pages 693–701, 2011.

Rashish Tandon, Qi Lei, Alexandros G Dimakis, and Nikos Karampatziakis. Gradient coding: Avoiding stragglers in distributed learning. In *International Conference on Machine Learning*, pages 3368–3376, 2017.

Mingrui Zhang, Lin Chen, Aryan Mokhtari, Hamed Hassani, and Amin Karbasi. Quantized frank-wolfe: Communication-efficient distributed optimization. *arXiv preprint arXiv:1902.06332*, 2019.

Horia Mania, Xinghao Pan, Dimitris Papailiopoulos, Benjamin Recht, Kannan Ramchandran, and Michael I Jordan. Perturbed iterate analysis for asynchronous stochastic optimization. *arXiv preprint arXiv:1507.06970*, 2015.

Yu-Xiang Wang, Veeranjaneyulu Sadhanala, Wei Dai, Willie Neiswanger, Suvrit Sra, and Eric P Xing. Asynchronous parallel block-coordinate frank-wolfe. *stat*, 1050:22, 2014.

Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, pages 583–598, 2014a.

Mu Li, David G Andersen, Alexander J Smola, and Kai Yu. Communication efficient distributed machine learning with the parameter server. In *Advances in Neural Information Processing Systems*, pages 19–27, 2014b.

Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231, 2012.

Ioannis Mitliagkas, Ce Zhang, Stefan Hadjis, and Christopher Ré. Asynchrony begets momentum, with an application to deep learning. In *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 997–1004. IEEE, 2016.

Dohyung Park, Anastasios Kyrillidis, Constantine Caramanis, and Sujay Sanghavi. Non-square matrix sensing without spurious local minima via the burer-monteiro approach. *arXiv preprint arXiv:1609.03240*, 2016.

Yuanzhi Li, Tengyu Ma, and Hongyang Zhang. Algorithmic regularization in over-parameterized matrix sensing and neural networks with quadratic activations. *arXiv preprint arXiv:1712.09203*, 2017.

Roi Livni, Shai Shalev-Shwartz, and Ohad Shamir. On the computational efficiency of training neural networks. In *Advances in neural information processing systems*, pages 855–863, 2014.

Lisandro Dalcín, Rodrigo Paz, Mario Storti, and Jorge D'Elía. Mpi for python: Performance improvements and mpi-2 extensions. *Journal of Parallel and Distributed Computing*, 68(5):655–662, 2008.