Large Scale Asynchronous Distributed Learning based on Parameter Exchanges

Bikash Joshi · Franck Iutzeler · Massih-Reza Amini

Abstract In many distributed learning problems, the heterogeneous loading of computing machines may harm the overall performance of synchronous strategies, as each machine begins its new computations after receiving an aggregated information from a master and any delay in sending local information to the latter may be a bottleneck. In this paper, we propose an effective asynchronous distributed framework for the minimization of a sum of smooth functions, where each machine performs iterations in parallel on its local function and updates a shared parameter asynchronously. In this way, all machines can continuously work even though they do not have the latest version of the shared parameter. We prove the convergence of the consistency of this general distributed asynchronous method for gradient iterations then show its efficiency on the matrix factorization problem for recommender systems and on binary classification.

1 Introduction

With the ever growing size of available data, distributed learning strategies where training sets are stored over M connected machines have attracted much interest in both machine learning and optimization communities. In this paper, we propose a principal asynchronous way to minimize a general differentiable objective that

B. Joshi
Univ. Grenoble Alpes, CNRS, Grenoble INP, LIG
Grenoble, France
E-mail: bikash.joshi@univ-grenoble-alpes.fr

F. Iutzeler
Univ. Grenoble Alpes, CNRS, Grenoble INP, LJK
Grenoble, France
franck.iutzeler@univ-grenoble-alpes.fr

M.R. Amini
Univ. Grenoble Alpes, CNRS, Grenoble INP, LIG
Grenoble, France
massih-reza.amini@univ-grenoble-alpes.fr

can be written as:

$$\mathcal{L}(\mathbf{v}, \mathbf{w}) = \sum_{i=1}^{M} \mathcal{L}_i(\mathbf{v}_i, \mathbf{w}) \tag{1}$$

where $\mathbf{v} = (\mathbf{v}_1, .., \mathbf{v}_M)$. In this model, each loss function $\mathcal{L}_i$ depends on (i) a *local* version of parameter $\mathbf{v}$, i.e. $\mathbf{v}_i$, that does not need to be exchanged across different machines, and (ii) a *shared* parameter $\mathbf{w}$ that has to be exchanged.

Distributed Learning. This formulation covers two common situations. First, when each loss $\mathcal{L}_i$, depends only on *local* versions of parameter $\mathbf{v}$, in this scenario the learning problem reduces to, $\min_{\mathbf{v}} \sum_{i=1}^{M} \mathcal{L}_i(\mathbf{v}_i)$, which is a totally parallel problem that can be solved locally on each machine in parallel [2].

The other extreme is a more typical case where each loss $\mathcal{L}_i$, depends only on the global shared parameter $\mathbf{w}$ and the learning problem in this case reduces to, $\min_{\mathbf{w}} \sum_{i=1}^{M} \mathcal{L}_i(\mathbf{w})$. This kind of problem is extremely common in ML when one wants to find the best predictor from a dataset split in several batches. Many deterministic and stochastic synchronous distributed algorithms have been recently proposed to solve this problem [15,8,12]. In most of these methods, the next global parameter is computed using updates based on its current version. In terms of implementation, the shared parameter is sent from each machine to a master node and is then broadcasted back into the network after integrating (mostly averaging) its local copies. For these synchronous methods, the loading of machines plays a central role in the convergence time of the whole system and in the extreme case, the slowest machine may become a bottleneck. To overcome this shortcoming, recent studies have considered asynchronous framework for distributed optimization [16,3,18,9,5]. However, these approaches suffer from mainly two drawbacks. First, some of these approaches [16,3] are based on a fixed delay time for broadcasting the parameter and the automatic tuning of this hyperparameter has to take into account the dynamic load of computing nodes in a network, and is a tedious task. Whereas others [18,9,5] rely on communicating gradients after each mini-batch update, So, if the size of dataset grows large the communication cost will become huge especially for a large number of workers.

Contributions. In this paper, we propose a novel asynchronous distributed framework for the minimization of the objective (3). In this framework, each worker machine sends its updated parameter values to a master machine, or also referred as the server, after finishing an iteration over its own local subpart of the data, and, it immediately begins a new iteration using: either the updated parameter copy received from the master machine (if it had received it from master machine during previous iteration) or it continues with its local updated parameter (if no update was received). Whereas the master, aggregates the received updates with its own local update whenever it finishes its iteration and broadcasts the updated parameter to all machines. In this way, all the machines have an overall view over the complete data, and the communication cost is significantly minimized as compared to the methods which broadcast the gradients after every mini-batch iteration [18,9,5]. Thus the proposed method is totally asynchronous (non-blocking) and overcomes the bottleneck of slower machines in the distributed framework leading to a much faster convergence. We provide a proof of convergence of the updates to a (local) minimum of the overall objective function, and empirically

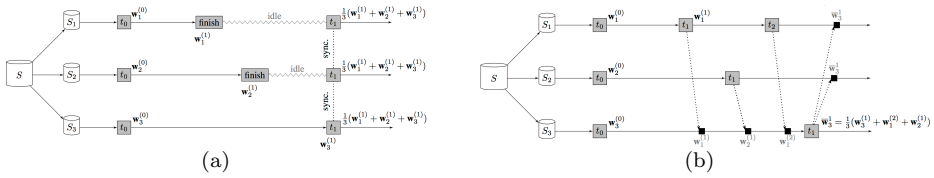


Fig. 1: Diagrams of the distributed synchronous (a) and asynchronous (b) frameworks.

show the efficiency of the proposed approach on the matrix factorization problem for recommender systems on NetFlix and MovieLens 10M datasets, as well as large-scale classification.

Paper organization. In the following section, we describe the proposed asynchronous distributed strategy, and derive two algorithms for large-scale binary classification and matrix factorization presented in Section 3. Finally, Section 4 presents experimental results corresponding to these two applications respectively.

2 Asynchronous Distributed Strategy

In this section, we present our proposed asynchronous distributed approach by first describing the deduced learning strategy. We then provide a consistency justification in the form of a convergence proof.

2.1 Description

The main challenge of distributed learning is to effectively partition the data into computing nodes, and efficiently perform communication between them. Indeed, in the synchronous case, the slowest node becomes the bottleneck of the whole system and a potentially large amount of computational time is lost (Figure 1 (a)).

The main idea of our approach is that when a machine finishes an iteration over the subpart of the data it contains, it broadcasts its updated parameter values to the master node; which gathers the received parameter values from the workers (if any, and taking only the last one if multiple parameter values are received from one machine); and updates the parameter vector with the received updates. Then the updated parameter is broadcasted to worker nodes. In this way each computing node runs its iterations independently and gets rid of the synchronization bottleneck. Faster machines will perform their epochs faster, whereas the slower ones will be lagging on time but after finishing each epoch they will receive the most updated parameters from the master. This situation is depicted in Figure 1 (b).

The main difference with other distributed asynchronous algorithms proposed in the literature [18,9], our approach does not exchange gradients but rather parameter values updated after one complete pass over local subpart of the data. Although these quantities have the same sizes, the broadcasting of parameters

performs better in practice, since they are exchanged after each epoch, whereas gradients need to be exchanged after every mini-batch update.

2.2 Consistency justification

In the case where the training data is partitioned into M batches $\{\mathcal{S}_1, \ldots, \mathcal{S}_M\}$, one for each computing machine, in the *shared parameter* case, the objective Eq. (3) can be rewritten as

$$\mathcal{L}(\mathbf{w}) = \sum_{i=1}^{M} \mathcal{L}_i(\mathbf{w}). \tag{2}$$

Here we may take advantage of the differentiability of $(\mathcal{L}_i)_{i=1}^M$ and use a gradient algorithm to find a minimizer of the global objective, $\mathcal{L}$. With a fixed stepsize gradient as an elementary operation before exchanging, we make the following assumptions :

Assumption 1 (on the functions)

a. The objective function, $\mathcal{L}$, has a unique minimizer $\boldsymbol{w}^\star$;
b. Each $\mathcal{L}_i$ is differentiable and $\nabla\mathcal{L}_i$ is $\frac{1}{L}$-cococercive, that is $\forall \boldsymbol{w}, \boldsymbol{w}' \in \mathbb{R}^d$:

$$\langle \boldsymbol{w} - \boldsymbol{w}'; \nabla\mathcal{L}_i(\boldsymbol{w}) - \nabla\mathcal{L}_i(\boldsymbol{w}')\rangle \geq \frac{1}{L}\|\nabla\mathcal{L}_i(\boldsymbol{w}) - \nabla\mathcal{L}_i(\boldsymbol{w}')\|^2.$$

As a consequence of the Baillon-Haddad theorem (Th. 18.15 in [1]); Assumption 1 (b) is notably verified whenever all functions $\mathcal{L}_i$ are convex and L_i-smooth, that is differentiable with an L_i-Lipschitz continuous gradient with $L = \max_i L_i$. Also, if a function $\mathcal{L}_i$ is L_i-smooth but not necessarily convex, then, considering $g_i = \mathcal{L}_i + \lambda/2\|\cdot\|^2$, it comes that ∇g_i is $1/(2\lambda)$ cocoercive for $\lambda > L$ (see Prop. 2 in [19]). In our case, this means that if the (smooth) cost function is non-convex, then one can add a ℓ_2 regularization term so that the sum function verifies the sought property.

In Assumption 2, we also make the rather mild assumption that the delays are bounded, meaning that no machine is infinitely slower than the others. More precisely, we consider that the duration of its computation is bounded by D in the sense that if machine i finishes its computation at time $k+1$, then the value of the averaged parameter it used is at most D ticks old. Mathematically, denoting the computation delay for machine i at time k by d_i^k, our bounded delay assumptions means that when machine i finishes, say at time k, the (outdated) value of the averaged parameter it used is $\overline{\mathbf{w}}^{k-d_i^k}$ with $d_i^k \leq D$.

Assumption 2 (on the algorithm) *The delays are uniformly bounded, i.e. there is $D < \infty$ such that for any machine i and iteration k; the delay $d_i^k \leq D$.*

The proposed Asynchronous Distributed update rule, corresponding to Figure 1 (b), is summarized in the pseudo-code presented in Figure 2. In the **local step**, all machines including the master update their parameters; and in the **master step**, once the master finishes its update, it broadcasts the aggregated parameters (from the latest received ones) to all workers. Furthermore, using a gradient step as an elementary operation, the convergence of the algorithm can be proven with the attractive properties that the considered stepsizes can be chosen fixed, as in the standard gradient algorithm, and thus do not decay or depend on the delay; and that no assumptions are made on the distribution of the delays.

When machine i finishes computing $\nabla\mathcal{L}_i(\overline{\mathbf{w}}^{k-d_i^k})$

(**Local step**) at i: $\mathbf{w}_i^{k+1} = \overline{\mathbf{w}}^{k-d_i^k} - \gamma\nabla\mathcal{L}_i(\overline{\mathbf{w}}^{k-d_i^k})$

(**Master step**) $\overline{\mathbf{w}}^{k+1} = \frac{1}{M}\sum_{j=1}^M \mathbf{w}_j^{k+1}$

Broadcast $\overline{\mathbf{w}}^{k+1}$

Fig. 2: Asynchronous Distributed Gradient update rule

Theorem 1 (Convergence) *Suppose that Assumptions 1 and 2 hold. Let $\gamma \in]0, 2/L[$. Then the sequence $(\overline{\mathbf{w}}^k)_k$ produced by our* Asynchronous Distributed Gradient update rule *converges to $\boldsymbol{w}^\star$.*

Proof. From Assumption 1 (i), $\mathbf{w}^\star$ is the unique minimizer of $\mathcal{L}$ and $\nabla\mathcal{L}(\mathbf{w}^\star) = \sum_{i=1}^M \nabla\mathcal{L}_i(\mathbf{w}^\star) = 0$. Let us define for all $i = 1, .., M$ $\mathbf{w}_i^\star = \mathbf{w}^\star - \gamma\nabla\mathcal{L}_i(\mathbf{w}^\star)$. Then at time k for the updating machine i, it comes from the cocoercivity of $\nabla\mathcal{L}_i$, Assumption 1 (b); and the definition $\mathbf{w}_i^{k+1} = \overline{\mathbf{w}}^{k-d_i^k} - \gamma\nabla\mathcal{L}_i(\overline{\mathbf{w}}^{k-d_i^k})$:

$$\begin{aligned}
&\left\|\mathbf{w}_i^{k+1} - \mathbf{w}_i^\star\right\|^2 \\
&= \left\|\overline{\mathbf{w}}^{k-d_i^k} - \gamma\nabla\mathcal{L}_i(\overline{\mathbf{w}}^{k-d_i^k}) - (\mathbf{w}^\star - \gamma\nabla\mathcal{L}_i(\mathbf{w}^\star))\right\|^2 \\
&\leq \left\|\overline{\mathbf{w}}^{k-d_i^k} - \mathbf{w}^\star\right\|^2 + \gamma^2\left\|\nabla\mathcal{L}_i(\overline{\mathbf{w}}^{k-d_i^k}) - \nabla\mathcal{L}_i(\mathbf{w}^\star)\right\|^2 \\
&\quad - \frac{2\gamma}{L}\left\|\nabla\mathcal{L}_i(\overline{\mathbf{w}}^{k-d_i^k}) - \nabla\mathcal{L}_i(\mathbf{w}^\star)\right\|^2.
\end{aligned}$$

Now by setting $\delta = \gamma\left(\frac{2}{L} - \gamma\right) > 0$ we get:

$$\begin{aligned}
&\left\|\mathbf{w}_i^{k+1} - \mathbf{w}_i^\star\right\|^2 \\
&\leq \left\|\overline{\mathbf{w}}^{k-d_i^k} - \mathbf{w}^\star\right\|^2 - \delta\left\|\nabla\mathcal{L}_i(\overline{\mathbf{w}}^{k-d_i^k}) - \nabla\mathcal{L}_i(\mathbf{w}^\star)\right\|^2 \\
&= \left\|\frac{1}{M}\sum_{j=1}^M(\mathbf{w}_j^{k-d_i^k} - \mathbf{w}_j^\star)\right\|^2 - \delta\left\|\nabla\mathcal{L}_i(\overline{\mathbf{w}}^{k-d_i^k}) - \nabla\mathcal{L}_i(\mathbf{w}^\star)\right\|^2 \\
&\leq \frac{1}{M}\sum_{j=1}^M\left\|\mathbf{w}_j^{k-d_i^k} - \mathbf{w}_j^\star\right\|^2 - \delta\left\|\nabla\mathcal{L}_i(\overline{\mathbf{w}}^{k-d_i^k}) - \nabla\mathcal{L}_i(\mathbf{w}^\star)\right\|^2,
\end{aligned}$$

where we used the fact that

$$\sum_{j=1}^M \mathbf{w}_j^\star = \sum_{j=1}^M \mathbf{w}^\star - \gamma\sum_{j=1}^M \nabla\mathcal{L}_j(\mathbf{w}^\star) = M\mathbf{w}^\star.$$

As the gradient of the objective $\nabla\mathcal{L}(\mathbf{w}) = \sum_{j=1}^M \nabla\mathcal{L}_j(\mathbf{w})$ is null at $\mathbf{w}^\star$. The last inequality is due to the convexity of the squared norm. For all other $j \neq i$, $\left\|\mathbf{w}_j^{k+1} - \mathbf{w}_j^\star\right\|^2 = \left\|\mathbf{w}_j^k - \mathbf{w}_j^\star\right\|^2$.

Let $\mathbf{y}_d^k = (\left\|\mathbf{w}_i^{k-d} - \mathbf{w}_i^\star\right\|^2)_{i=1,..,M}$ be the size-M vector of the individual errors at time $k-d$; and let $\mathbf{y}^k$ be the size-$M(D+1)$ vector obtained by concatenating the $(\mathbf{y}_d^k)_{d=0,..,D}$. From $\mathbf{y}^k$ to $\mathbf{y}^{k+1}$, we have that i) the last M values, $\mathbf{y}_D^k$, are dropped as they cannot intervene as D is the maximal delay; ii) the other ones are moved M coordinates lower $\mathbf{y}_{d+1}^{k+1} = \mathbf{y}_d^k$ for $d = 0, .., D-1$; iii) for the first M coordinates, they are copied from time k, $\mathbf{y}_0^{k+1} = \mathbf{y}_0^k$, except for the i-th one which verifies $\|\mathbf{w}_i^{k+1} - \mathbf{w}_i^\star\|^2 \leq \frac{1}{M}\sum_{j=1}^M \|\mathbf{w}_j^{k-d_i^k} - \mathbf{w}_j^\star\|^2$ thus $\mathbf{y}_0^{k+1}(i) \leq \frac{1}{M}\sum_{j=1}^M \mathbf{y}_{d_i^k}^k(j)$. Thus one can write $\mathbf{y}^{k+1} \preceq A^{k+1}\mathbf{y}^k$ where '$\preceq$' indicates the elementwise inequality and A^{k+1} represents the linear (in)-equalities mentioned above. A^{k+1}, seen as a $(D+1)\times(D+1)$ block matrix has identities on its sub-diagonal, and the top left block is the identity except for line i which has $1/M$ coefficients on the M columns corresponding to d_i^k. One can notice that it is non-negative and the row sum is constant equal to 1.

Taking the ℓ_∞-norm, we have,
$\|\mathbf{y}^{k+1}\|_\infty \leq \|A^{k+1}\mathbf{y}^k\|_\infty \leq \|A^{k+1}\|_\infty\|\mathbf{y}^k\|_\infty \leq \|\mathbf{y}^k\|_\infty$ as the ℓ_∞-induced matrix $\|\cdot\|_\infty$ is the maximal row sum and all rows of non-negative matrix A^{k+1} have unit sum. This means that $(\|\mathbf{y}^k\|_\infty)_k$ is a converging sequence, say to some value α. Now, suppose that there is some coordinate that is strictly lower than α, then it cannot be equal to α or greater anymore due to the above inequality; this means, that as the communication time is bounded, any coordinate holding the value α will have to (strictly) decrease due to the averaging with the strictly lower coordinate, which contradicts α being the limit of sequence $(\|\mathbf{y}^k\|_\infty)_k$. Thus, all errors converge to the same value which means that $\|\nabla\mathcal{L}_i(\overline{\mathbf{w}}^{k-d_i^k}) - \nabla\mathcal{L}_i(\mathbf{w}^\star)\|^2 \to 0$, implying that all $\mathbf{w}_i^k$ and thus $\overline{\mathbf{w}}^k$ converge. Furthermore, all limits points of $\overline{\mathbf{w}}^k$ null the gradient of $\mathcal{L}$; $\mathbf{w}^\star$ being unique (Assumption 1 (i)), the convergence ensues. □

One can notice that using this asynchronous framework, the machines local parameters all converge to different values while their sum converge to the sought minimizer. As this sum is received after each iteration, the agents also have individual knowledge of the full minimizer. Finally, the tools used in this proof make it adaptable to a wide range of elementary operations verifying cocoercive contraction properties. For instance, if the loss has a smooth and a non-smooth part, the gradient step can be replaced by a proximal gradient step. Other possible extensions here include the Alternating Direction Method of Multipliers (ADMM) and Primal-Dual algorithms.

3 Applications

In the following sections we present two algorithms for the estimation of parameters on each machine, corresponding to the **local step** of the proposed Asynchronous Distributed Gradient update rule, for large-scale binary classification (Section 3.1) and matrix factorization for recommender systems (Section 3.2).

3.1 Asynchronous Distributed Gradient for Binary Classification (ADG$_{BC}$)

For the classification problem, we consider the following convex loss function :

$$\mathcal{L}(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \ell(\mathbf{w}, \mathbf{x}_i, y_i), \tag{3}$$

defined over a training set of size n, $S = \{(\mathbf{x}_i, y_i); i \in \{1, \ldots, n\}\} \in (\mathbb{R}^d \times \{-1, +1\})^n$, where the instantaneous loss associated to example $(\mathbf{x}_i, y_i) \in S$, $\ell(\mathbf{w}, \mathbf{x}_i, y_i)$ is the ℓ_2-regularized logistic surrogate :

$$\ell(\mathbf{w}, \mathbf{x}_i, y_i) = \log(1 + \exp(-y_i \mathbf{w}^\top \mathbf{x}_i)) + \frac{\lambda}{2n}||\mathbf{w}||^2, \tag{4}$$

where $\lambda \geq 0$ is a regularization parameter. In order to have an accelerated update of the parameters on a given machine, we rely on a *variance reduced* variant of the Stochastic Gradient Descent (SGD) algorithm. Different such variants proposed recently, like SVRG [10] or SAG/SAGA [14,6] reduce the variance caused through random-sampling in SGD by occasionally computing full-gradients. As a result, this reduction in variance contributes to better convergence properties when using fixed learning rates.

The distributed memory algorithm, corresponding to the **local step** in a computing machine $j \in \{1, \ldots, m\}$ is shown in Algorithm 1. Let $\widetilde{\mathbf{w}}$ be the last received aggregated parameter from the master, or the last updated parameter estimated locally if the computation finished before a new aggregated parameter has been received. A local average gradient is then estimated using the local subpart of the data stored in machine j; $\widetilde{\mu}_j = \bar{\nabla}\mathcal{L}_j(\widetilde{\mathbf{w}})$. Considering a mini-batch I_j^t at the inner iteration t of the computing machine j, the current parameter $\mathbf{w}^t$ is then updated as:

$$\mathbf{w}^{t+1} \leftarrow \mathbf{w}^t - \frac{\gamma}{|I_j^t|} \sum_{(\mathbf{x}_i, y_i) \in I_j^t} \Big(\nabla \ell(\mathbf{w}^t, \mathbf{x}_i, y_i) - \nabla \ell(\widetilde{\mathbf{w}}, \mathbf{x}_i, y_i) + \widetilde{\mu}_j \Big), \tag{5}$$

where γ is the learning rate. This modification in update rule of SGD is similar to the one of SVRG [10] with the difference that the local average gradient here is computed over the aggregated parameter sent by the master using the local subpart of the data, rather than it would be estimated over the whole data as in SVRG. The rational of using this slightly different version of SVRG, is that in the standard case it has been shown that SVRG reduces the variance of the algorithm near the convergence point, and it has a linear convergence rate. Each machine performs parameter update on their local data and after each iteration the computing machines send the updated parameter to the master which directly responds by sending the averaged common parameter using the last gathered updates (**Master step**). In this way, all the machines have an overall view of the parameter updates from whole data, while only working with their local data.

Algorithm 1: $\texttt{ADG}_{\texttt{BC}}$, **local step** in the computing machine $j \in \{1, \ldots, m\}$

Input: Maximum number of iterations T, batch size B and learning rate γ
Initialize: Receive parameter $\widetilde{\mathbf{w}} \in \mathbb{R}^d$ from the master, or use the last parameter estimation happened before a new reception ;
$\mathbf{w}^0 \leftarrow \widetilde{\mathbf{w}}$;
Compute $\tilde{\mu}_j \leftarrow \bar{\nabla}\mathcal{L}_j(\widetilde{\mathbf{w}})$;
for $t = 0, .., T-1$ **do**
  Randomly pick a mini-batch I_j^t of size B in the subpart of the data stored in machine j;
  $\mathbf{w}^{t+1} \leftarrow \mathbf{w}^t - \frac{\gamma}{|I_j^t|}\sum_{(\mathbf{x}_i, y_i) \in I_j^t}(\nabla\ell(\mathbf{w}^t, \mathbf{x}_i, y_i) - \nabla\ell(\widetilde{\mathbf{w}}, \mathbf{x}_i, y_i) + \widetilde{\mu}_j)$;
end
$\widetilde{\mathbf{w}} \leftarrow \mathbf{w}^T$ and send $\mathbf{w}^T$ to the master.

3.2 Asynchronous Distributed SGD for Matrix Factorization ($\texttt{ADG}_{\texttt{MF}}$)

The problem of matrix factorization for collaborative filtering captured much attention, especially after the Netflix prize [11]. The premise behind this approach is to approximate a large rating matrix R with the multiplication of two low-dimensional factor matrices P and Q, i.e. $R \approx \hat{R} = PQ^\top$ that model respectively users and items in the same latent space. For a pair of user and item (u, i) for which a rating r_{ui} exists, the corresponding instantaneous loss is defined as ℓ_2-regularized quadratic error:

$$\ell(P, Q, u, i) = \left(r_{ui} - q_i^\top p_u\right)^2 + \lambda(||p_u||^2 + ||q_i||^2), \tag{6}$$

where p_u (resp. q_i) is u-th line of P (resp. i-th line of Q) and $\lambda \geq 0$ is a regularization parameter. The global objective is hence :

$$\mathcal{L}(P, Q) = \sum_{(u,i):r_{ui}\text{exists}} \ell(P, Q, u, i). \tag{7}$$

Note that instantaneous error $\ell(P, Q, u, i)$ depends only on P and Q through p_u and q_i; however, item i may also be rated by user u' so that the optimal factor q_i depends on both p_u and $p_{u'}$.

For this problem, SGD was found to offer a high prediction accuracy on different recommender system datasets. In this case, the approach proceeds as follows: at each iteration k, i) select a user/item pair (u^k, i^k) for which a rating exists; ii) perform a gradient step on $\ell(P, Q, u^k, i^k)$. Here stochasticity is used in the sense that the gradient on $\ell(P, Q, u^k, i^k)$ can be seen as an approximation of the gradient on an underlying global model but the choice of the considered users/items may or may not be random depending on the algorithm.

Despite its simplicity, there are several computational challenges associated with this problem. As previously, performing SGD sequentially on a single machine takes unacceptably large amount of time to converge for common rating matrices of several million ratings. So, there is a need to perform SGD in an efficient distributed manner for such large datasets. However, parallelizing SGD is not trivial. A drawback of a straightforward implementation is that updates on factor matrices might not be independent. For example, for training points that lie on same rows (i.e. ratings corresponding to the same users), an SGD step modifies the same

corresponding rows in factor matrix P; thus, these points cannot be learnt over in parallel and efficient communication between the computing nodes is necessary to synchronize the updates on factor matrices.

A popular approach in this case is to divide the rating matrix into several blocks and run gradient on each of the blocks on distinct machines. From the decomposition $\hat{R} = PQ^\top$, one can see that if the rating matrix is divided by row-blocks, $\hat{R}_b = P_b Q^\top$, that is; the block b of $\hat{R}$ depends only on the block b of P then, the block-split problem writes:

$$\min_{P,Q} \sum_{\text{blocks } b} \left[\sum_{(u,i):r_{b_{ui}} \text{exists}} \ell(P_b, Q, u, i) \right]. \tag{8}$$

Factor matrices are thus updated independently on each machine for the corresponding ratings. Even though the rating matrix parts on each machine are different, the factor matrix updates are not independent. So, after each epoch the factor matrices present in each machine are synchronized. We refer to this approach as Synchronous `SGD`, as all machines synchronize their updates after every epoch. One example of such algorithm is `ASGD` proposed in [13].

Another popular approach, referred to as Distributed `SGD` (`DSGD`) [7], divides the rating matrix into set of disjoint blocks with non-overlapping rows and columns. A set of such disjoint blocks is named stratum, and the number of stratums in the rating matrix is fixed to the number of machines to be used in parallel. These mutually independent sub-blocks in a stratum are processed in parallel and the updated parameters are synchronized after each stratum is processed (i.e. a subepoch). So, this method requires several synchronizations within an epoch which may hurt the computational performance.

The main challenge of these distributed approaches is to effectively partition the data into computing nodes, and efficiently perform communication between them. Indeed, in the situation above, the slowest node becomes the bottleneck of the whole system.

In order to apply the asynchronous distributed strategy to this problem (referred to as $\texttt{ADG}_{\texttt{MF}}$ in the following), we split the rating matrix in row-wise manner. In this case, we only need to communicate the matrices Q between machines, whereas the matrices P are updated locally, corresponding to each sub-part, and are later concatenated at the end of the operation. Due to the shared variable, the **local step** of the algorithm has to be slightly adapted as shown in Algorithm 2. As previously, **the master step** remains the same.

Algorithm 2: $\texttt{ADG}_{\texttt{MF}}$ **local step** in the computing machine $j \in \{1, \ldots, m\}$

Parameters: learning rate γ
Initialize: P_j
Receive matrix Q from the master;
From the subpart of the data stored in machine j, **pick** randomly (u, i) for which r_{ui} exists ;
$(P_j, Q_j) \leftarrow (P_j, Q) - \gamma \nabla \ell(P_j, Q, u, i)$;
Send Q_j to the master;

4 Experimental Results

We conducted a number of experiments aimed at testing the behaviour of the proposed $\texttt{ADG}_{\texttt{BC}}$ and $\texttt{ADG}_{\texttt{MF}}$ on large scale classification and matrix factorization for recommender systems by comparing them to the state-of-the-art distributed approaches

4.1 Experimental Results for Binary Classification

In the first set of experiments we study the convergence and the communication overhead of the proposed $\texttt{ADG}_{\texttt{BC}}$ algorithm.

Datasets: We performed our experiments on two popular large-scale binary classification datasets: `Epsilon` and `RCV1`[1]. The various characteristics of the datasets are presented in Table 3.

Table 1: Characteristics of Datasets used in our experiments.

Dataset	Training Size	Test Size	Feature Dimension	$\#nonzeros$
`Epsilon`	400000	100000	2000	10^9
`RCV1`	558112	139529	47236	51,055,210

Baselines: We compare our approach with the following methods which also consider totally distributed scenario without shared memory.

- The proposed approach $\texttt{ADG}_{\texttt{BC}}$ (Section 3.1),
- `Sync-SVRG`, SVRG based method [10] with synchronization of gradients after every mini-batch update.
- `Async-SVRG`: Distributed architecture proposed in [9], which asynchronously communicate gradients after every mini-batch updates.

Since the asynchronous methods were quite sensitive to initial point, we performed a synchronized gradient step during the first pass over the data. This gave a stable start for all the algorithms.

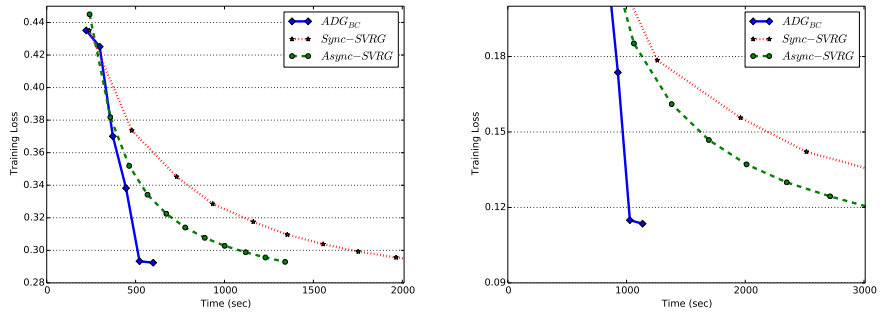
Platform: Experiments were conducted in a platform with 7 disparate servers without shared memory. The code was implemented using a python module `mpi4py` using OpenMPI[2] as the MPI library.

Hyper-parameters: In all the experiments, we used a fixed regularization rate, $\lambda = \frac{1}{n}$, where n is the size of the initial training set. The fixed learning rates were chosen from a set of values in range $\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$ and the reported performance were the best obtained with one of those stepsizes. The mini-batch size for `Epsilon` and `RCV1` datasets were respectively fixed to 10 and 20.

Evaluation Measures: Convergence result was evaluated in terms of minimization of objective function over time. The communication overhead incurred by each algorithm in the network as well as the communication time are shown in terms of the total number of send/receive calls.

[1] `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`

[2] `https://www.open-mpi.org/`



(a) `Epsilon` dataset (b) `RCV1` dataset

Fig. 3: Training Loss Vs Time Plot for (a) `Epsilon` and (b) `RCV1` Datasets

4.1.1 Evaluation of Convergence Time.

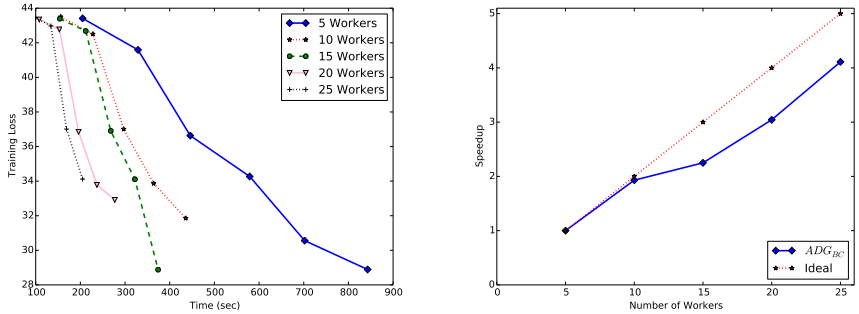
Figure 3 compares the convergence results for the three methods on all datasets. The convergence results are presented in terms of minimization of the objective function in the training sub-part of the data on the master machine. As It can be observed the proposed method $\texttt{ADG}_{\texttt{BC}}$ converges much faster than the other two methods. It can be seen that this behavior becomes more noticeable for larger datasets. For example on the `RCV1` collection, $\texttt{ADG}_{\texttt{BC}}$ converges three times faster than the other methods. Also it is to be noted that the difference in the convergence speed can become even larger if some of the machines are extremely overloaded, which is generally the case in the cluster environments.

4.1.2 Communication Overhead

Methods	`Epsilon`		`RCV1`	
	Number of Calls	time (sec)	Number of Calls	time (sec)
`Sync-SVRG`	108009	589.9	83711	4756.25
`Async-SVRG`	108000	110.03	30701	733.47
$\texttt{ADG}_{\texttt{BC}}$	**12004**	**29.6**	**8380**	**631.92**

Table 2: Comparison of the communication overhead for baselines

We also present the communication overhead incurred by each of the methods. The total communication cost for each algorithm is compared in terms of the total number of communication calls (send, receive, broadcast, gather), as well as the time spent in those calls. Since for `Sync-SVRG` and `Async-SVRG` methods the convergence is very slow near the tail, we compare the communication cost till the iteration when all methods achieve the same minimization of the objective function. Table 2 shows the detailed results obtained for each algorithm on all datasets. It can be observed that the $\texttt{ADG}_{\texttt{BC}}$ incurs the minimum communication overhead as the number of communication between the machines is very low. Most



(a) Loss Function Vs. Time (sec) (b) Speedup Vs. Number of Workers

Fig. 3: Convergence Speedup Result for `Epsilon` Dataset

of the calls shown for $\texttt{ADG}_{\texttt{BC}}$ are made during the first epoch where the gradients are synchronized. Whereas `Sync-SVRG` and `Async-SVRG` methods have to communicate large number of times in order to broadcast their local gradients to the master and receive the updated parameters from the master machine.

4.1.3 Speedup Result with Increasing Number of Workers

Finally, we evaluate the speedup in convergence (in terms of training loss and test accuracy) varying the number of workers from 5 to 25. Results shown in Figure 3 suggest that as the number of workers increases the $\texttt{ADG}_{\texttt{BC}}$ algorithm is able to achieve a near linear speedup, which is mainly due to the fact that, it relies on very low communication between the workers which is is also shown in Table 2. However, as the number of workers increases the performance of the algorithm slightly deteriorates.

4.2 Experimental Results for Matrix Factorization

We also conducted a number of experiments to empirically validate the proposed asynchronous framework on matrix factorization for recommendation where the recommendation matrix is split into M rows as in Problem (8).
Datasets: We performed experiments on Movielens-10M (ML-10M)[3] and the Netflix Collection[4] that are two popular corpora in collaborative filtering.
Baselines: To validate the asynchronous distributed algorithm described in the previous section, we compare the following four strategies:

- The proposed approach $\texttt{ADG}_{\texttt{MF}}$ (Section 3.2),
- The asynchronous distributed ADMM approach (`AD-ADMM`) [3],
- Two distributed algorithms specifically proposed for matrix factorization `ASGD` [13] and `DSGD` [7] (Section 3.2).

[3] http://grouplens.org/datasets/movielens/
[4] http://www.netflixprize.com/

Platform: The distributed framework we considered was implemented using PySpark version 1.5.1. by connecting 7 servers with different computational power.
Hyper-Parameters: Various free parameters of `SGD` such as learning rate (γ), regularization parameter (λ) and number of latent factors (K) were set following [4], [17]. These values as well as the datasets characteristics are listed in Table 3.

Table 3: Characteristics of Datasets used in our experiments. $|\mathcal{U}|$ and $|\mathcal{I}|$ denote respectively the number of users and items.

Dataset	$\|\mathcal{U}\|$	$\|\mathcal{I}\|$	γ	λ	K	training size	test size	sparsity
ML-10M	71567	10681	0.005	0.05	100	9301274	698780	98.7 %
NetFlix (NF)	480189	17770	0.005	0.05	40	99072112	1408395	99.8 %
NF-Subset	28978	1821	0.005	0.05	40	3255352	100478	93.7 %

4.2.1 Evaluation of Convergence Time

We begin our experiments by comparing the evolution of the loss function of Eq. (7) with respect to time until convergence. The convergence points are shown as names of the algorithms vertically (we stopped ASGD after 20 hours on the NF dataset). Figure 4 (top) depicts this evolution for ML-10M and NF datasets using 10 and 15 cores respectively. Synchronization based approaches (`ASGD` and `DSGD`) aggregate all the information at each epoch and thus begin to converge more sharply at the beginning. However, with these approaches, when the fastest machines finish their computations, they have to wait for slower machines; thus, they require much more time to converge than the asynchronous methods (`AD-ADMM` and $\texttt{ADG}_{\texttt{MF}}$). Finally, it comes out that $\texttt{ADG}_{\texttt{MF}}$ converges faster than the other algorithms on both datasets. This is mainly due to the fact that $\texttt{ADG}_{\texttt{MF}}$ does not obey to any delay mechanism as in `AD-ADMM` for instance.

4.2.2 Computation and Communication Trade-off

We performed another set of experiments aimed at measuring the effect of number of cores on performance of the proposed approach and the baselines. Figure 4 (bottom) depicts this effect by showing the evolution of time per epoch of the SGD method used in $\texttt{ADG}_{\texttt{MF}}$, `ASGD`, `DSGD` and `AD-ADMM` with respect to increasing number of machines. From these experiments, it comes out that for all approaches the time per epoch of the SGD method decreases as the number of machines increases.

But after a certain number of machines (10 in both experiments), the time per epoch of some approaches begin to be affected as the communication cost takes over the computation time. The approach that is the most affected by this is `DSGD`, as synchronizations in this case are done after each sub-epoch. We can also see that even though the per epoch speedup is best for `ASGD`, it requires a much higher number of epochs to converge as compared to $\texttt{ADG}_{\texttt{MF}}$ and `DSGD`.

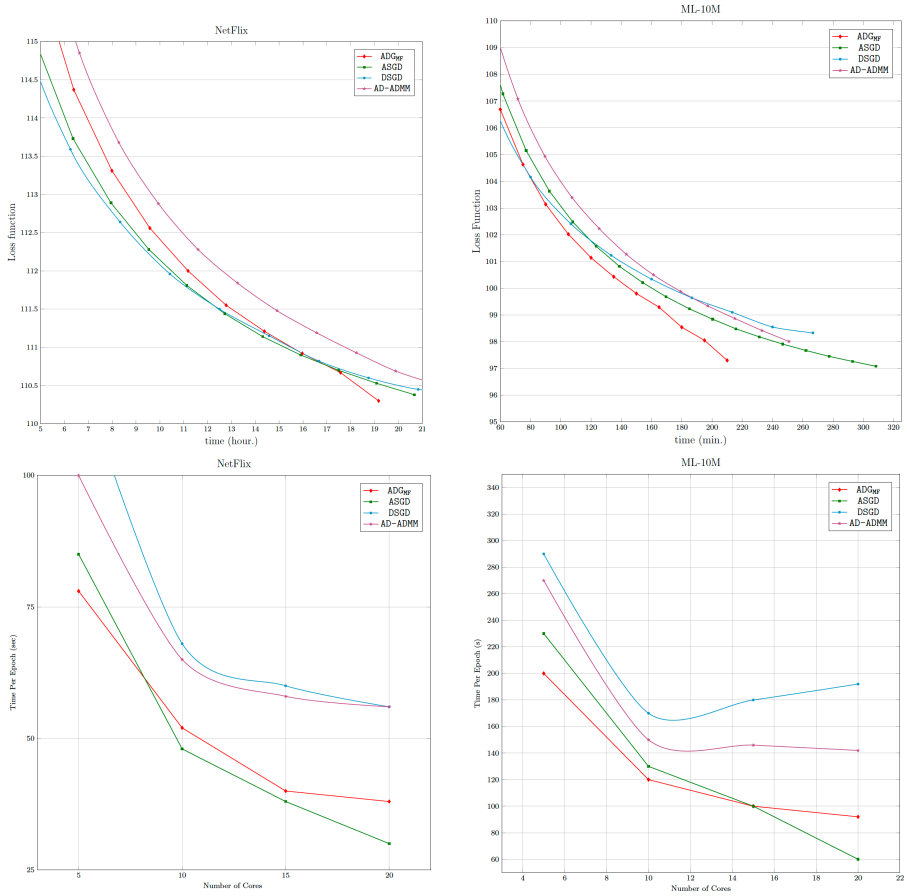


Fig. 4: **Top**: Test RMSE curves with respect to time for $\texttt{ADG}_{\texttt{MF}}$, `AD-ADMM`, `ASGD`, and `DSGD` on NetFlix (left), and ML-10M (right) Datasets. **Bottom**: Total Convergence Time Vs. Number of Cores curves for $\texttt{ADG}_{\texttt{MF}}$, `ASGD`, `DSGD` and `AD-ADMM` on the NetFlix (left), and ML-10M (right) Datasets.

5 Conclusion

In this paper we proposed a novel asynchronous distributed framework for the minimization of general smooth objective functions that write as a sum of instantaneous loss functions, where parameters are exchanged rather than gradients which is the case for almost the majority of distributed learning algorithms. We proved the consistency of this approach when the elementary operation at each node is a gradient descent. Then, we built upon this framework to propose two asynchronous distributed algorithms for: matrix factorization for recommender systems and large scale binary classification. Then we empirically validated effectiveness of the two proposed algorithms in corresponding application domains. As a perspective, we aim at extending this work by considering additional proximal

operations in order to deal with non-smooth convex functions as well as broad regularization terms.

References

1. Bauschke, H.H., Combettes, P.L.: Convex analysis and monotone operator theory in Hilbert spaces. Springer Science & Business Media (2011)
2. Bertsekas, D.P., Tsitsiklis, J.N.: Parallel and Distributed Computation: Numerical Methods. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1999)
3. Chang, T., Hong, M., Liao, W., Wang, X.: Asynchronous distributed ADMM for large-scale optimization- part I: algorithm and convergence analysis. ArXiv e-prints **1509.02597** (2015). URL `http://arxiv.org/abs/1509.02597`
4. Chin, W.S., Zhuang, Y., Juan, Y.C., Lin, C.J.: A learning-rate schedule for stochastic gradient methods to matrix factorization. In: Advances in Knowledge Discovery and Data Mining, pp. 442–455. Springer (2015)
5. Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Senior, A., Tucker, P., Yang, K., Le, Q.V., et al.: Large scale distributed deep networks. In: Advances in neural information processing systems, pp. 1223–1231 (2012)
6. Defazio, A., Bach, F., Lacoste-Julien, S.: Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In: Advances in Neural Information Processing Systems, pp. 1646–1654 (2014)
7. Gemulla, R., Nijkamp, E., Haas, P.J., Sismanis, Y.: Large-scale matrix factorization with distributed stochastic gradient descent. In: Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 69–77. ACM (2011)
8. Ho, Q., Cipar, J., Cui, H., Lee, S., Kim, J.K., Gibbons, P.B., Gibson, G.A., Ganger, G., Xing, E.P.: More effective distributed ml via a stale synchronous parallel parameter server. In: Advances in Neural Information Processing Systems 26, pp. 1223–1231 (2013)
9. Huo, Z., Huang, H.: Asynchronous stochastic gradient descent with variance reduction for non-convex optimization. arXiv preprint arXiv:1604.03584 (2016)
10. Johnson, R., Zhang, T.: Accelerating stochastic gradient descent using predictive variance reduction. In: Advances in Neural Information Processing Systems, pp. 315–323 (2013)
11. Koren, Y., Bell, R., Volinsky, C.: Matrix factorization techniques for recommender systems. Computer **42**(8), 30–37 (2009)
12. Mairal, J.: Incremental majorization-minimization optimization with application to large-scale machine learning. SIAM Journal on Optimization **25**(2), 829–855 (2015). DOI 10.1137/140957639. URL `http://dx.doi.org/10.1137/140957639`
13. Makari, F., Teflioudi, C., Gemulla, R., Haas, P., Sismanis, Y.: Shared-memory and shared-nothing stochastic gradient descent algorithms for matrix completion. Knowledge and Information Systems **42**(3), 493–523 (2015)
14. Roux, N.L., Schmidt, M., Bach, F.R.: A stochastic gradient method with an exponential convergence _rate for finite training sets. In: Advances in Neural Information Processing Systems, pp. 2663–2671 (2012)
15. Sra, S.: Scalable nonconvex inexact proximal splitting. In: Advances in Neural Information Processing Systems 25, pp. 530–538 (2012)
16. Xu, Y., Yin, W.: A globally convergent algorithm for nonconvex optimization based on block coordinate update. ArXiv e-prints:1410.1386 (2014)
17. Yu, Z.Q., Shi, X.J., Yan, L., Li, W.J.: Distributed stochastic admm for matrix factorization. In: Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, pp. 1259–1268. ACM (2014)
18. Zhang, R., Zheng, S., Kwok, J.T.: Fast distributed asynchronous sgd with variance reduction. CoRR, abs/1508.01633 (2015)
19. Zhu, D.L., Marcotte, P.: Co-coercivity and its role in the convergence of iterative schemes for solving variational inequalities. SIAM Journal on Optimization **6**(3), 714–726 (1996)