2021 60th IEEE Conference on Decision and Control (CDC)
December 13-15, 2021. Austin, Texas

L-DQN: An Asynchronous Limited-Memory Distributed Quasi-Newton Method

Bugra Can[1], Saeed Soori[2], Maryam Mehri Dehnavi[2], and Mert Gürbüzbalaban[1]

2021 60th IEEE Conference on Decision and Control (CDC) | 978-1-6654-3659-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/CDC45484.2021.9682985

***Abstract*— This work proposes a distributed algorithm for solving empirical risk minimization problems, called L-DQN, under the master/worker communication model. L-DQN is a distributed limited-memory quasi-Newton method that supports asynchronous computations among the worker nodes. Our method is efficient both in terms of storage and communication costs, i.e., in every iteration, the master node and workers communicate vectors of size $O(d)$, where d is the dimension of the decision variable, and the amount of memory required on each node is $O(md)$, where m is an adjustable parameter. To our knowledge, this is the first distributed quasi-Newton method with provable global linear convergence guarantees in the asynchronous setting where delays between nodes are present. Numerical experiments are provided to illustrate the theory and the practical performance of our method.**

I. Introduction

Due to the rapid increase in the size of datasets in the last decade, distributed algorithms that can parallelize the computations to multiple (computational units) nodes connected over a communication network became indispensable [1], [2]. A common communication model in distributed machine learning is the master/worker model in which the master keeps a copy of the global decision variable x and shares it with the workers. Each worker operates locally on its own data and then communicates the results to the master to update the decision variable in a synchronous [3], [4], [5], [6], [7], [8], [9] or asynchronous fashion [10], [11], [12], [13], [14], [15], [16]. In the synchronous setting, the master waits to receive updates from all workers before updating the decision variable, which can lead to a slow execution if the nodes and/or the network are heterogeneous [17]. In the asynchronous setting, coordination amongst workers is not needed (or is more relaxed) and the master can proceed with updates without having to wait for slow worker nodes. As a result, the asynchronous setting can be more efficient than the synchronous in heterogeneous computing environments [18].

In this paper, we consider distributed algorithms for empirical risk minimization, i.e. for solving the finite-sum problem

$$x^* := \underset{x\in\mathbb{R}^d}{\operatorname{argmin}}\ f(x) := \underset{x\in\mathbb{R}^d}{\operatorname{argmin}}\ \frac{1}{n}\sum_{i=1}^{n} f_i(x) \qquad (1)$$

where $x \in \mathbb{R}^d$ and $f_i : \mathbb{R}^d \to \mathbb{R}$ is the loss function of node $i \in \{1, ..., n\}$. We consider the master/worker communication model with asynchronous computations. With today's distributed computing environments, the cost of communicating information between nodes is significantly higher than the cost of local computations and sharing matrices of size $O(d^2)$ across nodes to be prohibitively expensive in many machine learning applications. Thus, inspired by prior work [19], [20], [21], [22], [23], [24], we focus on algorithms that communicate between nodes only vectors of size (at most) $O(d)$. There are a number of distributed algorithms for empirical risk minimization that can support asynchronous computations; the most relevant to our work are the recently proposed DAve-RPG [25] and DAve-QN algorithms [26]. DAve-RPG is a delay tolerant proximal gradient method with linear convergence guarantees that also handles a non-smooth term in the objective. However, it is a first-order method that does not estimate the second-order information of the underlying objective, therefore it can be slow for ill-conditioned problems. DAve-QN is a distributed quasi-Newton method with local superlinear convergence guarantees, however it does not admit global convergence guarantees. Furthermore, it relies on BFGS updates on each node, which requires $O(d^2)$ memory as well as $O(d^2)$ computations for updating the Hessian estimate at each node. For large d, this can be slow where DAve-QN looses its edge over first-order approaches [26]; furthermore its $O(d^2)$ memory requirement can be impractical or prohibitively expensive when d is large, say when d is on the order of ten thousands or hundred thousands.

Contributions. On this paper, we propose the L-DQN, a distributed limited-memory quasi-Newton method, which is based on limited BFGS-type updates at workers, requiring less memory and less computational work (per iteration) than DAve-QN algorithm [26]. More specifically, the per iteration and per node storage and computation of L-DQN are $O(md)$ and $O(md)$ respectively, where m is a configurable parameter and is the number of vectors stored in the memory at every iteration that contains information about the past gradients and iterates. Because of the reduced storage and computation costs, our proposed algorithm scales well for large datasets, it is communication-efficient as it exchanges vectors of size $O(d)$ at every communication. When the number of nodes is large enough, with an appropriate stepsize, L-DQN has global linear convergence guarantees for strongly convex objectives, even though the computations are done in an asynchronous manner, as opposed to the DAve-QN method which does not provide global convergence guarantees. In

[1] Department of Management Sciences and Information Systems, Rutgers Business School, Piscataway, NJ-08854, USA.
[2] Department of Computer Sciences, University of Toronto, Toronto, Canada.

*Bugra Can and Mert Gürbüzbalaban acknowledge support from the Office of Naval Research Award Number N00014-21-1-2244, and the grants National Science Foundation (NSF) CCF-1814888, NSF DMS-2053485, NSF DMS-1723085.

practice, we have also observed that L-DQN works well even if the number of nodes n is not large, for example when $n = 2$. To our knowledge, L-DQN is the first distributed quasi-Newton method with provable linear convergence guarantees, even in the presence of asynchronous computations.

Related work. The proposed method can be viewed as an asynchronous distributed variant of the traditional quasi-Newton and limited-memory BFGS methods that have been extensively studied in the optimization community ([27], [28], [29], [30]). L-DQN builds on the limited-memory BFGS method [19]. Prior work have also investigated incremental gradient ([31], [32]) and incremental aggregated gradient algorithms ([4], [5], [6], [7], [3], [8], [9], [33], [34]), which are originally developed for centralized problems. These methods update the global decision variable by processing the gradients of the component functions f_i in a deterministic fashion in a specific (e.g. cyclic) order. They are applicable to our setting in practice, however, these methods do not provide convergence guarantees in asynchronous settings. The Lazily Aggregated Gradient (LAG) [35] method, which has a convergence rate similar to batch gradient descent in strongly convex, convex, and nonconvex cases as well as its quantized version [36], is an exception, however, LAG is a first-order method that does not use second-order information. For synchronous settings, the distributed quasi-Newton algorithm proposed by [37] is globally linearly convergent and can handle non-smooth regularization terms; convergence analysis for the algorithm does not exist for asynchronous settings. In this work, we use the star network topology where the nodes follow a master/worker hierarchy. However, there is another setting known as the *decentralized setting* which does not have a master node and communication between the nodes is limited to a given fixed arbitrary network topology ([38], [15]). Amongst algorithms for this setting, [39] proposes a linearly convergent decentralized quasi-Newton method and [15] develops an asynchronous Newton-based approach that has local superlinear convergence guarantees to a neighborhood of the problem (1). There are also distributed second-order methods developed for non-convex objectives. Among these, most relevant to our paper are [16] which proposes a stochastic asynchronous-parallel L-BFGS method and the DINGO method ([40]) which admits linear convergence guarantees to a local minimum for non-convex objectives that satisfy an invexity property.

Notation. Throughout the paper, we use $\|.\|$ to denote the matrix 2-norm or the (Euclidean norm) L_2 norm depending on the context. The Frobenius norm of a matrix $A \in \mathbb{R}^{n\times m}$ is defined as $\|A\|_F^2 := \sum_{i=1}^n \sum_{j=1}^m A_{ij}^2$. The matrix I_d denotes the $d\times d$ identity matrix. A memory with capacity m, denoted as $\mathcal{M}_m$, is a set of tuples (y, q, α, β) where $y, q \in \mathbb{R}^d$ and $\alpha, \beta \in \mathbb{R}$; the size of the memory $|\mathcal{M}_m|$ satisfies $|\mathcal{M}_m| \leq m$. A function $f : \mathbb{R}^d \to \mathbb{R}$ is called L-smooth and μ strongly convex if for any vector $x, \hat{x} \in \mathbb{R}^d$, the Hessian satisfies $\mu\|x - \hat{x}\| \leq \|\nabla^2 f(x) - \nabla^2 f(\hat{x})\| \leq L\|x - \hat{x}\|$.

II. Algorithm

A. Preliminaries

BFGS algorithm. In the following, we provide a brief summary of the BFGS algorithm, see [41] for more detail. Given a convex smooth function $f : \mathbb{R}^d \to \mathbb{R}$, the BFGS algorithm consists of iterations: $x^{t+1} = x^t - \eta_t (B^{t+1})^{-1}\nabla f(x^t)$, where η_t is a properly chosen stepsize where the matrix $H^{t+1} := (B^{t+1})^{-1}$ is an estimate of the inverse Hessian matrix at x^t and satisfies the *secant equation*:

$$H^{t+1} y^{t+1} = s^{t+1}, \tag{2}$$

where $s^{t+1} := x^t - x^{t-1}$and $y^{t+1} := \nabla f(x^t) - \nabla f(x^{t-1})$ are the differences of the iterates and the gradients respectively. By Taylor's theorem, $y^{t+1} = [\int_0^1 \nabla^2 f(x^{t-1} + \tau(x^t - x^{t-1}))d\tau]s^{t+1}$, therefore for a small enough stepsize η_t any matrix H^{t+1} solving the secant equation can be considered as an approximation to the inverse of Hessian $(\nabla^2 f(x^t))^{-1}$. In fact, the secant equation (2) has infinitely many solutions and quasi-Newton methods differ in how they choose a particular solution. BFGS chooses the matrix H^{t+1} according to

$$H^{t+1} = \left(I - \frac{s^{t+1}(y^{t+1})^\top}{(y^{t+1})^\top s^{t+1}}\right) H^t \left(I - \frac{s^{t+1}(y^{t+1})^\top}{(y^{t+1})^\top s^{t+1}}\right) + \frac{s^{t+1}(s^{t+1})^\top}{(y^{t+1})^\top s^{t+1}}.$$

The corresponding update for B^{t+1} is

$$B^{t+1} = B^t + U^{t+1} + V^{t+1},$$
$$U^{t+1} = \frac{y^{t+1}(y^{t+1})^\top}{(y^{t+1})^\top s^{t+1}}, \quad V^{t+1} = -\frac{B^t s^{t+1}(s^{t+1})^\top B^t}{(s^{t+1})^\top B^t s^{t+1}}. \tag{3}$$

If the function f is strongly convex then $(s^{t+1})^\top y^{t+1} > 0$ so that the denominator in (3) cannot be zero. Note that U^t and V^t are both rank-one therefore these updates require $\mathcal{O}(d^2)$ operations. Even though the BFGS algorithm (3) enjoys local superlinear convergence with an appropriate stepsize, its $\mathcal{O}(d^2)$ memory requirement to store the matrix B^t and $\mathcal{O}(d^2)$ computations required for the updates (3) may be impractical or prohibitively expensive for machine learning problems when d is large.

Limited-memory BFGS (L-BFGS) algorithm. Limited-memory BFGS (L-BFGS) requires less memory compared to BFGS algorithm. Instead of storing the whole B^t matrix, L-BFGS stores up to m pairs $\{s^t, y^t\}$ in memory and uses these vectors to approximate the Hessian. The parameter m is adjustable which results in a memory requirement of $\mathcal{O}(md)$. At the start of iteration t, we have access to $\{s^j, y^j\}$ for $j = t - m, t - m + 1, \ldots, t - 1$. Since the storage is full[1], the oldest pair $\{s^{t-m}, y^{t-m}\}$ is replaced by the latest pair $\{s^t, y^t\}$. The resulting L-BFGS algorithm has the updates:

$$x^{t+1} = x^t - \eta_t(\tilde{B}^{t+1})^{-1}\nabla f(x^t),$$

where the matrices $\tilde{B}^{t+1}$ are computed according to the following formula

[1]In the beginning of the iterations, when the total number of gradients computed is less than m the storage capacity is not full but the details are omitted for keeping the discussion simpler, see [41] for details.

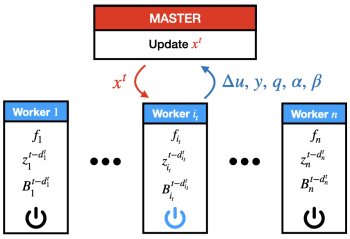


Fig. 1: Asynchronous communication scheme used by proposed algorithm.

$$\tilde{B}^{t+1} = \gamma^{t+1} I_d + \sum_{j=1}^{m} \tilde{U}^{t+2-j} + \tilde{V}^{t+2-j},$$

$$\tilde{U}^j = \frac{y^{j+1}(y^{j+1})^\top}{(y^{j+1})^\top s^{j+1}}, \quad \tilde{V}^j = -\frac{\tilde{B}^j s^{j+1}(s^{j+1})^\top \tilde{B}^j}{(s^{j+1})^\top \tilde{B}^j s^{j+1}},$$

where γ^{t+1} is a scaling factor. We note that L-BFGS requires $\mathcal{O}(md)$ memory which is significantly less compared to $\mathcal{O}(d^2)$ for BFGS for a large d.

DAve-QN algorithm. The DAve-QN algorithm [26] is an asynchronous quasi-Newton method for solving the optimization problem (1) in master/worker communication models. Let x^t be the variable that is kept at the master at time t and z_i^t be the local copy that agent i keeps after its last communication with the master. At time t, an agent i_t communicates with the master and updates its local estimate $B_{i_t}^t$ for the local Hessian $\nabla^2 f_{i_t}(x^t)$ with a BFGS update:

$$B_{i_t}^{t+1} = B_{i_t}^t + \frac{y_{i_t}^{t+1}(y_{i_t}^{t+1})^\top}{\alpha_{i_t}^{t+1}} - \frac{q_{i_t}^{t+1}(q_{i_t}^{t+1})^\top}{\beta_{i_t}^{t+1}}, \qquad (4)$$

where $q_{i_t}^{t+1} := B_{i_t}^t s_{i_t}^{t+1}$, $y_{i_t}^{t+1} := \nabla f_i(x^t) - \nabla f_i(z_{i_t}^t)$, $\alpha_{i_t}^{t+1} := (y_{i_t}^{t+1})^\top s_{i_t}^{t+1}$, and $\beta_{i_t}^{t+1} := \left(s_{i_t}^{t+1}\right)^\top q_{i_t}^{t+1}$ are computed using the local copy $z_{i_t}^t$ and the iterate x^t. Let $D_{i_t}^t$ be delay time between information received and send at agent i_t at time t, then agent i_t sends the information $(B_{i_t}^t x^t - B_{i_t}^{t-D_{i_t}^t} z_{i_t}^{t-D_{i_t}^t})$, $y_{i_t}^{t+1}$, $q_{i_t}^{t+1}$, $\alpha_{i_t}^{t+1}$ and $\beta_{i_t}^{t+1}$ to the master after making the update (4). Consequently, the master updates the global decision variable with:

$$x^t = \left(\sum_{i=1}^n B_i^t\right)^{-1} \left[\sum_{i=1}^n B_i^t z_i^t - \nabla f_i(z_i^t)\right].$$

In the next section, we introduce the L-DQN method which is a limited-memory version of the DAve-QN algorithm. L-DQN will allow us to improve performance for large dimensional problems. The basic idea is that each agent stores m-many tuples $\{y_i^j, q_i^j, \alpha_i^j, \beta_i^j\}$ requiring $O(md)$ memory instead of storing the $d \times d$ matrix B_i^t and carries out L-BFGS-type updates (4) to compute the Hessian estimate $\nabla^2 f_i(x^t)$.

B. A Limited-Memory Distributed Quasi-Newton Method (L-DQN)

In this section, we introduce the L-DQN algorithm on a master/worker communication setting that consists of n workers that are connected to one master with a star topology (see Figure 1). Let d_i^t be the delay in communication at time t with the i-th worker and the master and D_i^t denote the (penultimate) double delay in communication, i.e. the last exchange between the master and the worker i was at time $t - d_i^t$, and before that the communication took place at $t - D_i^t$ where $D_i^t = d_i^t + d_i^{t-d_i^t-1} + 1$. For example, if the node i communicated with master at times $t = 1, t = 5$ and $t = 7$, we have $d_i^4 = 3, d_i^5 = 0, d_i^6 = 1, d_i^7 = 0, d_i^8 = 1$ and $D_i^6 = 6$, $D_i^7 = 2$ and $D_i^8 = 3$.

Let us introduce the historical time $T_i(t) = t - d_i^t$ with the convention $T_i^0(t) = t$ and $T_i^n(t) = T_i(T_i^{n-1}(t))$. We introduce the notation $\tilde{q}_{i_t}^{t+1} := \tilde{B}_{i_t}^t s_{i_t}^{t+1}$, $\tilde{\beta}_{i_t}^{t+1} := \left(s_{i_t}^{t+1}\right)^\top \tilde{q}_{i_t}^{t+1}$, and explain the L-DQN updates on worker and master in detail:

Worker Updates: Each agent i keeps m-many tuples $\{y_i^j, \tilde{q}_i^j, \alpha_i^j, \tilde{\beta}_i^j\}$ at their local memory $\mathcal{M}_m(i,t)$ at time t and at the end of the m-th iteration, the worker replaces the oldest tuple $\{y_i^{T_i^m(t)}, \tilde{q}_{i_t}^{T_i^m(t)}, \alpha_i^{T_i^m(t)}, \tilde{\beta}_i^{T_i^m(t)}\}$ with the new one $\{y_i^{T_i(t)}, \tilde{q}_i^{T_i(t)}, \alpha_i^{T_i(t)}, \tilde{\beta}_i^{T_i(t)}\}$. Suppose master communicates with worker i_t at the moment $t - d_{i_t}^t$ and sends the copy $x^{t-d_{i_t}^t}$; then upon receiving $x^{t-d_{i_t}^t}$, the worker i_t computes $\tilde{B}_{i_t}^{t+1} x^{t-d_{i_t}^t}$ where $\tilde{B}_{i_t}^{t+1}$ is computed according to

$$\tilde{B}_{i_t}^{t+1} = \gamma_{i_t}^{t+1} I_d + \sum_{j=1}^{m} \tilde{U}_{i_t}^{T_{i_t}^j(t+1)} + \tilde{V}_{i_t}^{T_{i_t}^j(t+1)},$$

$$\tilde{U}_{i_t}^j = \frac{y_{i_t}^j (y_{i_t}^j)^\top}{(y_{i_t}^j)^\top s_{i_t}^j}, \quad \tilde{V}_{i_t}^j = -\frac{\tilde{q}_{i_t}^j \left(\tilde{q}_{i_t}^j\right)^\top}{(s_{i_t}^j)^\top \tilde{B}^j s_{i_t}^j}, \qquad (5)$$

and the scaling factor is chosen as $\gamma_{i_t}^{t+1} = \frac{\|y_{i_t}^{t+1}\|^2}{(y_{i_t}^{t+1})^\top s_{i_t}^{t+1}}$.

A number of choices for γ_i^t are proposed in the literature [41]. γ_i^t given above (which is also considered at [20]) is an estimate for the largest eigenvalue of Hessian $\nabla^2 f_i(x^{t-d_i^t})$ and works well in practice, therefore our algorithm analysis is based on given γ_i^t. However, our analysis on the linear convergence of L-DQN can be extended to different choice of γ_i^t's as well. Worker i_t calls Algorithm 1 to perform

Algorithm 1: Compute u given memory $\mathcal{M}_m := \{y^i, \tilde{q}^i, \alpha^i, \tilde{\beta}^i\}_{i=1}^m$

Function: u = LBFGS $(\gamma, \mathcal{M}_m, x)$
Set $u = \gamma x$
for *i=1,..,m* **do**
  Retract $y^i, \tilde{q}^i, \alpha^i, \tilde{\beta}^i$ from $\mathcal{M}_m$
  Set $c_1 = \frac{(y^i)^\top x}{\alpha^i}$ and $c_2 = \frac{(\tilde{q}^i)^\top x}{\tilde{\beta}^i}$
  $u = u + c_1 y^i - c_2 \tilde{q}^i$
end
Return u.

the update (5) locally based on its memory $\mathcal{M}_m(i,t)$. Then, the worker sends $\Delta u_{i_t}^{t+1} := \tilde{B}_{i_t}^{t+1} x^t - \tilde{B}_{i_t}^{t-d_{i_t}^t} z_{i_t}^{t-d_{i_t}^t}$, $y_{i_t}^{t-d_{i_t}^t}$, $\tilde{q}_{i_t}^{t-d_{i_t}^t}$, $\alpha_{i_t}^{t-d_{i_t}^t}$ and $\tilde{\beta}_{i_t}^{t-d_{i_t}^t}$ to the master.

Master Updates: Following its communication with the worker, the master receives the vectors Δu_{i_t}, y_{i_t}, $\tilde{q}_{i_t}$, the scalars α_{i_t}, $\tilde{\beta}_{i_t}$ and computes

$$x^{t+1} = \left(\tilde{B}^t\right)^{-1} \left[\sum_{i=1}^n \tilde{B}_i^t z_i^t - \eta_t \sum_{i=1}^n \nabla f_i(z_i^t)\right], \qquad (6)$$

where $\tilde{B}^t := \sum_{i=1}^n \tilde{B}_i^t = \sum_{i=1}^n \tilde{B}_i^{t-d_i^t}$ and stepsize η_t determined by the master. Soori et al. have shown in [26] that

the computation of $\tilde{B}^t$ and $(\tilde{B}^t)^{-1}$ can be done at master locally by using only vectors send by workers. In particular, if we define $u^t := \sum_{i=1}^n \tilde{B}_i^t z_i^t = \sum_{i=1}^n \tilde{B}_i^{t-d_i^t} z_i^{t-d_i^t}$ and $g^t := \sum_{i=1}^n \nabla f_i(z_i^t) = \sum_{i=1}^n \nabla f_i(z_i^{t-d_i^t})$, then the updates at the master follow the below rules: $\tilde{B}^{t+1} = \tilde{B}^t + (\tilde{B}_{i_t}^t - \tilde{B}_{i_t}^{t-d_i^t})$, $u^{t+1} = u^t + \left(\tilde{B}_{i_t}^{t+1} x^{t-d_{i_t}^t} - \tilde{B}_{i_t}^{t-d_{i_t}^t} x^{t-D_{i_t}^t}\right)$, and $g^{t+1} = g^t + \left(\nabla f_{i_t}(x^{t-d_{i_t}^t}) - \nabla f_{i_t}(x^{t-D_{i_t}^t})\right)$. Hence the master only requires $\tilde{B}_{i_t}^{t+1}$ and $\nabla f_{i_t}(z_{i_t}^{t+1}) = \nabla f_{i_t}(x^{t-d_{i_t}^t})$ to proceed to $t+1$. Let

$$U^{t+1} := (\tilde{B}^t)^{-1} - \frac{(\tilde{B}^t)^{-1} y_{i_t}^{t+1} (y_{i_t}^{t+1})^\top (\tilde{B}^t)^{-1}}{(y_{i_t}^{t+1})^\top s_{i_t}^{t+1} + (y_{i_t}^{t+1})^\top (\tilde{B}^t)^{-1} y_{i_t}^{t+1}}, \quad (7)$$

then Sherman-Morrison-Woodbury formula implies

$$\left(\tilde{B}^{t+1}\right)^{-1} = U^{t+1} + \frac{U^{t+1}(\tilde{B}_{i_t}^{t-d_{i_t}^t} s_{i_t}^{t+1})(\tilde{B}_{i_t}^{t-d_{i_t}^t} s_{i_t}^{t+1})^\top U^{t+1}}{(s_{i_t}^{t+1})^\top \tilde{B}_{i_t}^{t-d_{i_t}^t} s_{i_t}^{t+1} - (\tilde{B}_{i_t}^{t-d_i^t} s_{i_t}^{t+1})^\top U^{t+1} (\tilde{B}_{i_t}^{t-d_i^t} s_{i_t}^{t+1})}. \quad (8)$$

Thus, if the master already has $(\tilde{B}^t)^{-1}$, then $(\tilde{B}^{t+1})^{-1}$ is computed using the vectors $y_{i_t}^{t+1}$ and $\tilde{w}^{t+1} := U^{t+1}\tilde{q}_{i_t}^{t+1}$.

The steps for the master and worker nodes are provided in Algorithm 2. After receiving x^t from the master, worker i computes its estimate $\tilde{B}_i^{t+1}$ using the vector $x^{t-d_i^t}$ received from the master, then updates its memory $\mathcal{M}_m(i,t)$ and sends the vectors $\Delta u_i, y_i, \tilde{q}_i$ together with the scalars $\alpha_i, \tilde{\beta}_i$ back to the master. Based on (7) and (8), the master computes x^{t+1} using the vectors received from worker i. We define the *epochs* $\{E_m\}_{m\in\mathbb{N}^+}$ recursively as follows: We set $E_1 = 0$ and define $E_{m+1} = \min\{t : t - D_i^t \geq E_m \text{ for all } i = 1, ..., n\}$. In other words, E_{m+1} is the first time t such that each machine makes at least 2 updates on the interval $[E_m, t]$. Epochs as defined above satisfy the properties:

- For any $t \in [E_{m+1}, E_{m+2})$ and any $i = 1, 2, .., N$ one has $t - D_i^t \in [E_m, t)$
- If delays are uniformly bounded, i.e. there exists a constant $d_i^t \leq d$ for all i and t, then for all m we have $E_{m+1} - E_m \leq D := 2d + 1$ and $E_m \leq Dm$.
- If we define average delays as $\bar{d}^t := \frac{1}{N}\sum_{i\equiv 1}^n d_i^t$, then $\bar{d}^t \geq (n-1)/2$. Moreover, assuming that $\bar{d}^t \leq (n-1)/2 + \bar{d}$ for all t, we get $E_m \leq 4n(d+1)m$.

Notice that convergence to optimum x^* is not possible without visiting every function f_i, so measuring performance using epochs where every node has communicated with the master at least once is more convenient than the number of communications, t, for comparison.

III. Convergence Analysis

In this section, we study theoretical results for linear convergence of L-DQN algorithm with a constant stepsize $\eta_t = \eta$. Firstly, we assumed that the functions f_i's and the matrices $\tilde{B}_i^t$'s satisfy the following conditions:

Assumption 1: The component functions f_i are L- smooth and μ-strongly convex, i.e. there exist positive constants $0 < \mu < L$ such that, for all i and $x, \hat{x} \in \mathbb{R}^p$,

$$\mu\|x - \hat{x}\|^2 \leq (\nabla f_i(x) - \nabla f_i(\hat{x}))^\top (x - \hat{x}) \leq L\|x - \hat{x}\|^2.$$

Assumption 2: There exist constants $0 < \epsilon_d < \epsilon_u$ such that the following bounds are satisfied for all $i = 1, ..., n$ and $x \in \mathbb{R}^d$ at any $t > 0$:

$$\epsilon_d I_d \preceq (\tilde{B}_i^t)^{-1/2} \nabla^2 f_i(x^t) (\tilde{B}_i^t)^{-1/2} \preceq \epsilon_u I_d. \quad (9)$$

Assumption 2 says that $\tilde{B}_i^t$ approximates the Hessian $\nabla^2 f_i(x^t)$ up to a constant factor.For example, if the objective is a quadratic function of the form $f_i(x) = \frac{1}{2}(x - x_*)^\top Q_i (x - x_*)$ and $\frac{1}{1+c} Q_i \preceq \tilde{B}_i^t \preceq (1+c) Q_i$ for some constant $c > 0$ then we would have $\epsilon_d = \frac{1}{1+c}, \epsilon_u = (1+c)$ and the ratio $\epsilon = \epsilon_u/\epsilon_d$ satisfies $\epsilon = (1+c)^2 \geq 1$. In fact, this ratio can be thought as a measure of the accuracy of the Hessian approximation. In the special case when the Hessian approximations are accurate (when $\tilde{B}_i^t = Q_i$), we have $c = 0$ and $\epsilon = 1$. Otherwise, we have $\epsilon > 1$.

In particular, if the eigenvalues of $\tilde{B}_i^t$ stay in the interval $[\lambda_d, \lambda_u]$, then Assumption (2) holds with $\epsilon_d = \frac{\mu}{\lambda_u}$ and $\epsilon_u = \frac{L}{\lambda_d}$. We note that in the literature, there exist estimates for λ_u and λ_d [41], [20]. For example, it is known that if we choose $\gamma_i^t = \frac{\|y_i^t\|^2}{(y_i^t)^\top s_i^t}$ then we can take $\lambda_u = (m+d)L$, and $\lambda_d = \frac{(\mu)^{m+d}}{((m+d)L)^{m+d-1}}$ for memory/storage capacity m (see [20] for details). A shortcoming of these existing bounds [42], [43] is that they are not tight, i.e. with an increasing memory capacity m, the bounds get worse.In our experiments, we have also observed in real datasets that Assumption 2 holds where we estimated the constants ϵ_d and ϵ_u (see the supplementary file for details). These numerical results show that Assumption 2 is a reasonable assumption to make in practice for analyzing L-DQN methods.

Before we provide a convergence result for L-DQN, we observe that the iterates $\{x^t\}_{t\in\mathbb{N}^+}$ of L-DQN provided in (6) satisfy the property

$$x^{t+1} - x^* = \left(\tilde{B}^t\right)^{-1} \left[\sum_{i=1}^n \tilde{B}_i^t (z_i^t - x^*) - \eta_t \sum_{i=1}^n \nabla f_i(z_i^t) - \nabla f_i(x^*)\right]. \quad (10)$$

The next theorem uses bounds (9) together with equality (10) to find the condition on fixed step size $\eta := \eta_t$ such that the L-DQN algorithm is linearly convergent on epochs $[E_m, E_{m+1})$ for $m \in \mathbb{N}^+$. The proof can be found in the appendix.

Theorem 1: Suppose Assumptions 1-2 hold and accuracy $\epsilon := \epsilon_u/\epsilon_d$ satisfies

$$\epsilon < \frac{1}{2}\left[1 + \frac{1}{\kappa} + \sqrt{\left(1 + \frac{1}{\kappa}\right)^2 + \frac{4}{\kappa}}\right], \quad (11)$$

where $\kappa = \frac{L}{\mu}$ is the condition number of f. Let stepsize $\eta \in (\frac{1}{\epsilon_d}[1 - \frac{1}{\epsilon\kappa}], \frac{2}{\epsilon_d + \epsilon_u})$, then for each $t \in [E_{m-1}, E_m)$, the iterates x^t generated by L-DQN algorithm (2) converge linearly on epochs $[E_m, E_{m+1})$, i.e. there exists $\rho < 1$ (with $\rho = \mathcal{O}(\kappa)$) such that $\|x^{t+1} - x_*\| \leq \rho^m \|x^0 - x_*\|$ for all $t \in [E_m, E_{m+1})$.

Theorem 1 says that if the Hessian approximations are good enough, then ε is small enough and L-DQN will admit a linear convergence rate. Even though the condition (11) seems conservative on the accuracy of the Hessian estimates, to our knowledge, there exists no other linear convergence

Algorithm 2: L-DQN

Worker i:

Initialize $x_i = x^0, y_i = x^0, \tilde{B}_i, u_{-1} = 0$ and memory $\mathcal{M}_m = \{\}$ with capacity m.
while *not interrupted by master* **do**
  Receive x from master
  $s_i = x - z_i$, $y_i = \nabla f_i(x) - \nabla f_i(z_i)$, $\gamma_i = \frac{y_i^\top y_i}{s_i^\top y_i}$
  Compute $\tilde{q}_i$=LBFGS $(\gamma_i, \mathcal{M}_m, s_i)$
  $\alpha_i = y_i^\top s_i$, $\tilde{\beta}_i = s_i^\top \tilde{q}_i$
  if $|\mathcal{M}_m| < m$ **then**
    Add $\{y_i, \tilde{q}_i, \alpha_i, \tilde{\beta}_i\}$ to $\mathcal{M}_m$
  else if $|\mathcal{M}_m| = m$ **then**
    Replace the oldest tuple with $(y_i, \tilde{q}_i, \alpha_i, \tilde{\beta}_i)$ at $\mathcal{M}_m$
  Compute u=LBFGS $(\gamma_i, \mathcal{M}_m, x)$
  $\Delta u = u - u_{-1}$
  $u_{-1} = u$, $z_i = x$
  Send $\Delta u, y_i, \tilde{q}_i, \alpha_i, \tilde{\beta}_i$ to the master

Master:

Initialize x, η, $\tilde{B}_i$, $g = \sum_{i=1}^n \nabla f_i(x)$, $\tilde{B}^{-1} = (\sum_{i=1}^n \tilde{B}_i)^{-1}$, $u = \sum_{i=1}^n \tilde{B}_i x$.
for $t = 1, ..., T$ **do**
  If a worker sends an update
  Receive $\Delta u, y, \tilde{q}, \alpha, \tilde{\beta}$ from worker
  $u = u + \Delta u$, $g = g + y$, $v = \tilde{B}^{-1} y$
  $U = \tilde{B}^{-1} - \frac{vv^\top}{\alpha + v^\top y}$
  $w = U\tilde{q}$, $\tilde{B}^{-1} = U + \frac{ww^\top}{\tilde{\beta} - \tilde{q}^\top w}$
  $x = \tilde{B}^{-1}(u - \eta g)$
  Send x to the worker in return
Interrupt all the workers.
Output x^T

result that supports global convergence of asynchronous distributed second-order methods for distributed empirical risk minimization. Also, on real datasets, we observe that L-DQN algorithm performs well even though limited memory updates fail to satisfy the condition (11) on the accuracy.

IV. Numerical Experiments

We tested our algorithm on the multi-class logistic regression problem with L_2 regularization where the objective is

$$f(x) = \frac{1}{n}\sum_{i=1}^{n} \log(1 + \exp(-b_i a_i^T x)) + \frac{\lambda}{2}\|x\|^2. \qquad (12)$$

and $\lambda > 0$ is the regularization parameter, $a_i \in \mathbb{R}^d$ is a feature vector, and b_i is the corresponding label. We worked with five datasets (SVHN, mnist8m, covtype, cifar10, rcv1) from the LIBSVM repository [44] where the covtype dataset is expanded based on the approach in [45] for large-scale experiments.

We compare L-DQN with the following other recent distributed optimization algorithms:

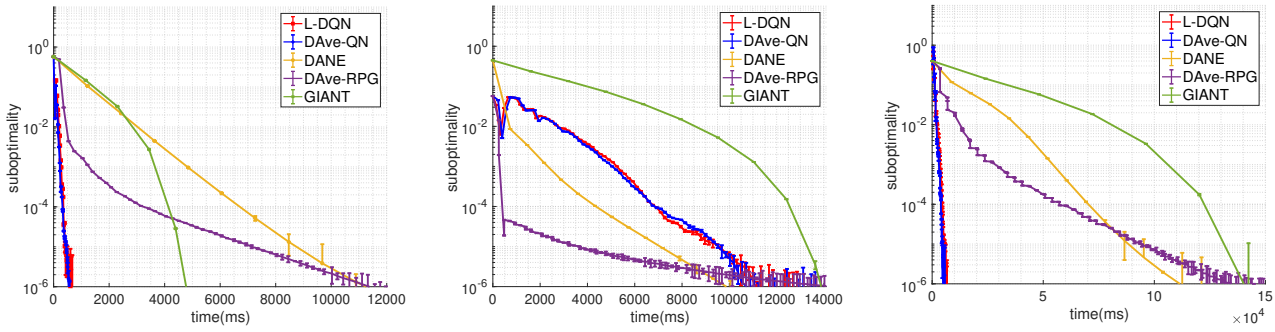
- **DAve-QN** [26]: An asynchronous distributed quasi-Newton method.
- **Distributed Average Repeated Proximal Gradient (DAve-RPG)**[46]: A first-order asynchronous method that performs better compared to incremental aggregated gradient [3] and synchronous proximal gradient methods.
- **Globally Improved Approximate Newton (GIANT)**[47]: A synchronous communication-efficient approximate Newton method, for which the numerical experiments in [47] demonstrate it outperforms DANE [48] and the Accelerated Gradient Descent [49].
- **Distributed Approximate Newton (DANE)**[48]: A well-known second-order method that requires sychronization step among all workers.

All experiments are conducted on the XSEDE Comet resources [50] with 24 workers (on Intel Xeon E5-2680v3 2.5 GHz architectures) and with 120 GB Random Access Memory (RAM.) For L-DQN, DAve-RPG, and DAve-QN, we use 17 processes where one as a master and the 16 processes dedicated as workers. DANE and GIANT do not have a master, thus, we use 16 processes are workers with no master. All datasets are normalized to [0,1] and randomly distributed so the load is roughly balanced among workers. We use Intel MKL 11.1.2 and MVAPICH2/2.1 for BLAS (sparse/dense) operations and MPI programming compiled with mpicc 14.0.2 for optimized communication. We run gradient descent until norm of gradient is below 10^{-8} to compute suboptimality, $f(x_k) - f(x_*)$ for the iterations generated by the algorithms.Each experiment is repeated five times and the average and standard deviation is reported as error bars in our results.

Parameters: The recommended parameters for each method is used. λ is tuned to ensure convergence for all methods. We use $\lambda = 1$ for mnist8m, and $\lambda = 0.1$ for SVHN, cifar10 and covtype. Other choices of λ show similar performances. For DANE, SVRG [51] is used as a local solver; parameters are selected based on experiments in [52]. DANE has two parameters η and μ which are set to 1 and 3λ respectively based on the recommendation of the authors in [52]. For DAve-RPG, the number of passes on local data is set to 5 ($p = 5$) and its stepsize is selected using a standard backtracking line algorithm [53]. For L-DQN, the memory capacity is set as $m = 20$ for covtype, mnist8m and cifar10 where stepsize is $\eta = 0.8$,$\eta = 0.8$ and $\eta = 0.6$ respectively. On SVHN, the parameters of L-DQN are chosen as $m = 25$ and $\eta = 0.9$.

Figure 2 shows the average suboptimality versus time for the datasets mnist8m, SVHN and covtype. We observe that L-DQN converges with a similar rate compared to DAve-QN while it uses less memory. For larger datasets (such as the rcv1 with $d = 47,000$ and $n = 697,000$ at Figure 3), DAve-QN was not able to run due to its memory requirement whereas the other methods run successfully. DAve-RPG demonstrates good performance at the beginning for SVHN compared to other methods due to its cheaper iteration complexity. However, L-DQN becomes faster eventually and outperforms DAve-RPG.

The right panel of Figure 3 shows the suboptimality versus time for the dataset cifar10 where we choose the parameter $\lambda = 0.1$, $m = 20$ and $\eta = 0.6$ for cifar10. DAve-RPG is the fastest on this dataset whereas L-DQN is competitive with DAve-QN with less memory requirements. We conclude that



(a) covtype$(54, \approx 2.9M)$ (b) SVHN$(3072, \approx 73K)$ (c) mnist8m$(784, 8.1M)$

Fig. 2: Expected suboptimality versus time. The first and second numbers adjacent to the dataset names are variables d and n respectively.

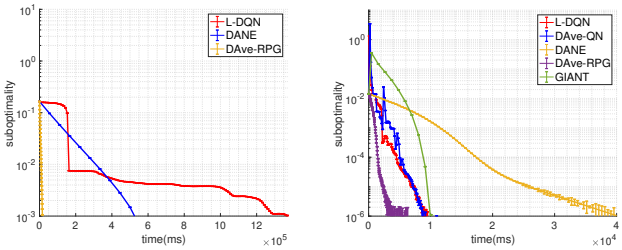


Fig. 3: Expected suboptimality versus time on rcv1 (left) and cifar10 (right)

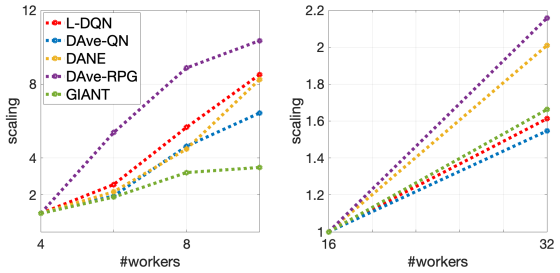


Fig. 5: Scaling comparisons of the algorithms on `covtype`(left) and `mnist8m`(right) datasets

when the underlying optimization problem is ill-conditioned (such as the case of `mnist8m` dataset), L-DQN improves performance with respect to other methods while being scalable to large datasets. In case of less ill-conditioned problems (such as `SVHN` and `cifar10`), first-order methods such as DAve-RPG are efficient where second-order methods may not be necessary.

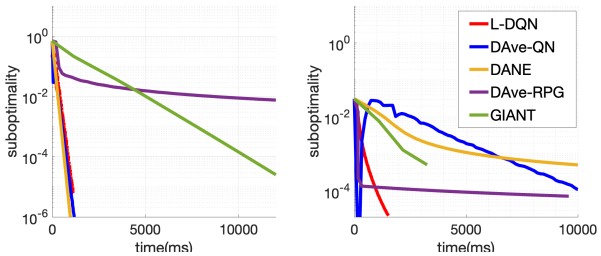


Fig. 4: Suboptimality comparison without strong convexity assumption on datasets `covtype`(left) and `cifar10`(right).

Figure 4 exhibits the suboptimality results of the algorithms on `cifar10` and `covtype` without regularization parameter which makes the problems more ill-conditioned. Due to its less memory requirement, we can see that the performance of L-DQN algorithm on `cifar10` is significantly better than other distributed algorithms. L-DQN is competitive with DAve-QN and DANE on `covtype` as well. We also compare the strong scaling of the distributed algorithms on different number of workers for `mnist8m` and `covtype` in Figure 5. In particular, we look at the improvement in time to achieve the same suboptimality as we increase the number of workers. We see that L-DQN shows a nearly linear speedup and a slightly better scaling compared to DAve-QN. DAve-RPG scales better but considering the total runtime, it is slower than L-DQN.

In addition to suboptimality and scaling, we also compared the performance of these algorithms for different sparsity of the datasets. For the problem of interest (logistic regression), computing the gradient takes $O(nd)$ for dense and $O(n.\texttt{nnz})$ for sparse datasets where `nnz` is the number of non-zeros in the dataset. Therefore, L-DQN has $O(n.\texttt{nnz} + md)$ while DAve-QN has a iteration complexity of $O(nd^2.\texttt{nnz})$. Similarly, DAve-RPG has a complexity of $O(pn.\texttt{nnz} + pd)$ where p is number of passes on local data. We observe that L-DQN has a cheaper iteration complexity compared to DAve-QN while in case of very sparse datasets, DAve-RPG has a cheaper iteration complexity compared to L-DQN.This is illustrated over the dataset `rcv1` on the left panel of Figure 3. The dataset `rcv1` is quite sparse with $\approx 1\%$ non-zeros. We use the parameters $\lambda = 0.01$, $m = 10$, $\eta = 0.95$. For this dataset, DAve-QN fails as it requires more memory than the resources available. GIANT requires each worker to have $|S| > d$ where $|S|$ is the number of local data points on a worker. Hence, GIANT diverges with 16 workers. We observe that DAve-RPG converges faster than DANE and L-DQN because of its cheap iteration complexity.

In order to show the effect of sparsity on performance, we design a synthetic dataset based on a similar approach taken in [52]. First we generate N i.i.d input samples $x \sim \mathcal{N}(0, \Sigma)$ where $x \in \mathbb{R}^{2000}$ and the covariance matrix Σ is diagonal with $\Sigma_{ii} = i^{-1.2}$. Then, we randomly choose some entries of all samples and make them zero to add sparsity. We set $z = \langle x, w^* \rangle + \xi$, $\xi \sim \mathcal{N}(0, 0.09)$ and w^* is the vector of all ones. Finally, labels $y \in \{0, 1\}$ are generated based on the probabilities $p = S(z)$ where $S(z) = 1/(1 + exp(-z))$

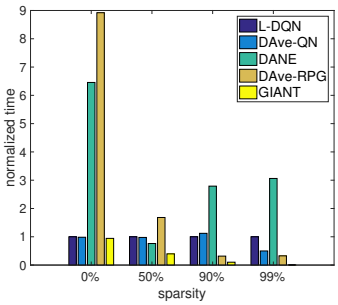


Fig. 6: The effect of dataset sparsity on the performance of distributed optimization methods.

is the logistic function. The parameters $\lambda = 0.01$ and $N = 32000$ are chosen for the objective function and for this experiment we have the following $m = 20$ and $\eta = 0.9$. Time to the accuracy of $1e-4$ for all methods is measured and normalized based on L-DQN timing. The results are shown in Figure 6. DAve-RPG and DANE performs poorly for fully dense datasets (sparsity = 0%), however, DAve-RPG and GIANT perform better compared to L-DQN as the dataset sparsity increases. We observe that when above %90 of the data is sparse, DAve-RPG is the most efficient method; whereas for denser datasets GIANT and L-DQN are more efficient on the synthetic data.

V. Conclusion

We proposed the L-DQN method which is an asynchronous limited-memory BFGS method. We showed that under some assumptions, L-DQN admits linear convergence over epochs despite asynchronous computations. Our numerical experiments show that L-DQN can lead to significant performance improvements in practice in terms of both memory requirements and running time.

APPENDIX

VI. Proof of Theorem 1

Recall the definition of the average Hessian $\bar{G}_i^t = \int_0^1 \nabla^2 f_i(x^* + \tau(z_i^t - x^*))d\tau$ satisfies the equality $\nabla f_i(z_i^t) - \nabla f_i(x^*) = \bar{G}_i^t(z_i^t - x^*)$. Hence the equation (10) implies

$$\|x^t - x^*\|^2 \leq \sum_{i=1} \|\Gamma^t(\tilde{B}_i^t - \eta\bar{G}_i^t)\|_2^2 \max_{i=1,..,n} \|z_i^t - x^*\|^2 \quad (13)$$

where $\Gamma^t = (\tilde{B}^t)^{-1}$ and $\|\cdot\|_2$ denotes the 2-norm of a matrix. Notice that by its definition and from (9), it can be found that Γ^t has the bounds $\frac{1}{n\lambda_u} I_d \preceq \Gamma^t \preceq \frac{1}{n\lambda_d} I_d$ and hence $(\Gamma^t)^2$ is positive definite. So the function $\Psi(A) := A(\Gamma^t)^2 A$ defined from the set of symmetric positive-definite matrices S^n to itself is a matrix convex function [54, see E.7.a], that is for any $A, B \in S^n$ and $\alpha \in (0,1)$ inequality $\Psi(\alpha A + (1-\alpha)B) \preceq \alpha\Psi(A) + (1-\alpha)\Psi(B)$ holds. In particular, if $L_A \preceq A \preceq U_A$ for some positive-definite matrices L_A and U_A, by matrix convexity we have

$$\sup_{L_A \preceq A \preceq U_A} \Psi(A) \preceq \max(\Psi(L_A), \Psi(U_A)). \quad (14)$$

where the maximum on the right-hand side is in the sense of Loewner ordering, i.e. $\max\{A, B\} = A$ if $B \preceq A$ and equals to B otherwise. From the bounds (9), we have $\left(1 - \eta\frac{L}{\lambda_d}\right)\tilde{B}_i^t \preceq \tilde{B}_i^t - \eta\bar{G}_i^t \preceq \left(1 - \eta\frac{\mu}{\lambda_u}\right)\tilde{B}_i^t$, for each $i = 1, .., n$. On the other hand, $[\tilde{B}_i^t(\Gamma^t)^2\tilde{B}_i^t]^{-1} = \left(I_d + \sum_{j\neq i}(\tilde{B}_i^t)^{-1}\tilde{B}_j\right)\left(I_d + \sum_{j\neq i}\tilde{B}_j(\tilde{B}_i^t)^{-1}\right)$ together with (9) imply that $\lambda_{\min}([\tilde{B}_i^t(\Gamma^t)^2\tilde{B}_i^t]^{-1}) \geq \left(1 + (n-1)\frac{\lambda_d}{\lambda_u}\right)^2$, where $\lambda_{\min}$ is the smallest eigenvalue. This yields to

$$\lambda_{\max}([\tilde{B}_i^t(\Gamma^t)^2\tilde{B}_i^t]) \leq \frac{\lambda_u^2}{(\lambda_u + (n-1)\lambda_d)^2}, \quad (15)$$

where $\lambda_{\max}$ denotes the largest eigenvalue. Applying matrix convexity property with $A = (\tilde{B}_i^t - \eta\bar{G}_i^t)$ with $L_A = \left(1 - \eta\frac{L}{\lambda_d}\right)\tilde{B}_i^t$ and $U_A = \left(1 - \eta\frac{\mu}{\lambda_u}\right)\tilde{B}_i^t$ and using (15), we obtain $\lambda_{\max}\left((\tilde{B}_i^t - \eta\bar{G}_i^t)\Gamma^2(\tilde{B}_i^t - \eta\bar{G}_i^t)\right) \leq \frac{\lambda_u^2}{(\lambda_u+(n-1)\lambda_d)^2}\max\left\{\left(1 - \eta\frac{L}{\lambda_d}\right)^2, \left(1 - \eta\frac{\mu}{\lambda_u}\right)^2\right\}$ for all $i = 1, .., n$. Hence,

$$\sum_{i=1}^n \|\Gamma^t(\tilde{B}_i - \eta\bar{G}_i^t)\|_2^2 \leq \frac{n\tilde{\kappa}^2}{(\tilde{\kappa}+n-1)^2}\max\left\{\left(1 - \eta\frac{L}{\lambda_d}\right)^2, \left(1 - \eta\frac{\mu}{\lambda_d}\right)^2\right\}. \quad (16)$$

Choosing $\rho^2 = \frac{n\tilde{\kappa}^2}{(\tilde{\kappa}+n-1)^2}\max\left\{\left(1 - \eta\frac{L}{\lambda_d}\right)^2, \left(1 - \eta\frac{\mu}{\lambda_d}\right)^2\right\}$ together with condition on η imply that $\|x^t - x^*\| \leq \rho\max_{i=1,..,n}\|z_i^t - x^*\|$ where $\rho < 1$. Next, we will prove convergence by induction on epoch times E_m. Notice that if $t \in [E_j, E_{j+1})$, it holds that $t - D_i^t \in [E_{j-1}, t)$ for any $j \geq 1$, therefore the inequality (16) implies $\|x^t - x^*\| \leq \rho\max_{i=1,..,n}\|z_i^0 - x^*\| \leq \rho\|x^0 - x^*\|$ for $t \in [E_0, E_1)$. Suppose for all $0 \leq j \leq m$ the inequality $\|x^t - x^*\| \leq \rho^j\|x^0 - x^*\|$ holds for $t \in [E_j, E_{j+1})$, then (13) and (16) imply $\|x^t - x^*\| \leq \rho\max_{i=1,..,n}\|z_i^{t-D_i^t} - x^*\| \leq \rho^m\|x^0 - x^*\|$. This completes the proof.

References

[1] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and distributed computation: Numerical methods*. Prentice-Hall, Inc., 1989.

[2] B. Recht, C. Re, S. Wright, and F. Niu, "Hogwild: A lock-free approach to parallelizing stochastic gradient descent," in *Advances in Neural Information Processing Systems*, 2011, pp. 693–701.

[3] M. Gürbüzbalaban, A. Ozdaglar, and P. A. Parrilo, "On the convergence rate of incremental aggregated gradient algorithms," *SIAM Journal on Optimization*, vol. 27, no. 2, pp. 1035–1048, 2017.

[4] N. L. Roux, M. Schmidt, and F. R. Bach, "A stochastic gradient method with an exponential convergence rate for finite training sets," in *Advances in Neural Information Processing Systems*, 2012, pp. 2663–2671.

[5] A. Defazio, F. Bach, and S. Lacoste-Julien, "Saga: A fast incremental gradient method with support for non-strongly convex composite objectives," in *Advances in Neural Information Processing Systems*, 2014, pp. 1646–1654.

[6] A. Defazio, J. Domke, and T. Caetano, "Finito: A faster, permutable incremental gradient method for big data problems," in *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, 2014, pp. 1125–1133.

[7] J. Mairal, "Incremental majorization-minimization optimization with application to large-scale machine learning," *SIAM Journal on Optimization*, vol. 25, no. 2, pp. 829–855, 2015.

[8] A. Mokhtari, M. Gürbüzbalaban, and A. Ribeiro, "Surpassing gradient descent provably: A cyclic incremental method with linear convergence rate," *SIAM Journal on Optimization*, vol. 28, no. 2, pp. 1420–1447, 2018.

[9] N. D. Vanli, M. Gurbuzbalaban, and A. Ozdaglar, "Global convergence rate of proximal incremental aggregated gradient methods," *SIAM Journal on Optimization*, vol. 28, no. 2, pp. 1282–1300, 2018.

[10] L. Xiao, A. W. Yu, Q. Lin, and W. Chen, "DSCVR: Randomized primal-dual block coordinate algorithms for asynchronous distributed optimization," *Journal of Machine Learning Research*, vol. 20, no. 43, pp. 1–58, 2019.

[11] R. Leblond, F. Pedregosa, and S. Lacoste-Julien, "ASAGA: Asynchronous parallel SAGA," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, A. Singh and J. Zhu, Eds., vol. 54. Fort Lauderdale, FL, USA: PMLR, 20–22 Apr 2017, pp. 46–54. [Online]. Available: http://proceedings.mlr.press/v54/leblond17a.html

[12] Z. Peng, Y. Xu, M. Yan, and W. Yin, "Arock: An algorithmic framework for asynchronous parallel coordinate updates," *SIAM Journal on Scientific Computing*, vol. 38, no. 5, pp. A2851–A2879, 2016. [Online]. Available: https://doi.org/10.1137/15M1024950

[13] P. Bianchi, W. Hachem, and F. Iutzeler, "A coordinate descent primal-dual algorithm and application to distributed asynchronous optimization," *IEEE Transactions on Automatic Control*, vol. 61, no. 10, pp. 2947–2957, 2015.

[14] R. Zhang and J. Kwok, "Asynchronous distributed ADMM for consensus optimization," in *International Conference on Machine Learning*, 2014, pp. 1701–1709.

[15] F. Mansoori and E. Wei, "Superlinearly convergent asynchronous distributed network Newton method," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE, 2017, pp. 2874–2879.

[16] U. Şimşekli, Ç. Yıldız, T. H. Nguyen, G. Richard, and A. T. Cemgil, "Asynchronous stochastic quasi-Newton MCMC for non-convex optimization," *arXiv preprint arXiv:1806.02617*, 2018.

[17] S. Kanrar and M. Siraj, "Performance measurement of the heterogeneous network," *arXiv preprint arXiv:1110.3597*, 2011.

[18] A. Wongpanich, Y. You, and J. Demmel, "Rethinking the value of asynchronous solvers for distributed deep learning," in *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region*, 2020, pp. 52–60.

[19] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Mathematical Programming*, vol. 45, no. 1-3, pp. 503–528, 1989.

[20] A. Mokhtari and A. Ribeiro, "Global convergence of online limited memory BFGS," *The Journal of Machine Learning Research*, vol. 16, no. 1, pp. 3151–3181, 2015.

[21] S. G. Nash and J. Nocedal, "A numerical study of the limited memory BFGS method and the truncated-Newton method for large scale optimization," *SIAM Journal on Optimization*, vol. 1, no. 3, pp. 358–372, 1991.

[22] A. Skajaa, "Limited memory BFGS for nonsmooth optimization," *Master's thesis*, 2010.

[23] R. Bollapragada, D. Mudigere, J. Nocedal, H.-J. M. Shi, and P. T. P. Tang, "A progressive batching L-BFGS method for machine learning," *arXiv preprint arXiv:1802.05374*, 2018.

[24] A. S. Berahas, M. Jahani, and M. Takáč, "Quasi-Newton methods for deep learning: Forget the past, just sample," *arXiv preprint arXiv:1901.09997*, 2019.

[25] K. Mishchenko, F. Iutzeler, J. Malick, and M.-R. Amini, "A delay-tolerant proximal-gradient algorithm for distributed learning," in *Accepted to the 35th International Conference on Machine Learning, ICML, Stockhom, Sweden*, 2018.

[26] S. Soori, K. Mischenko, A. Mokhtari, M. M. Dehnavi, and M. Gürbüzbalaban, "DAve-QN: A distributed averaged quasi-Newton method with local superlinear convergence rate," *arXiv preprint arXiv:1906.00506*, 2019.

[27] D. Goldfarb, "A family of variable-metric methods derived by variational means," *Mathematics of Computation*, vol. 24, no. 109, pp. 23–26, 1970.

[28] C. G. Broyden, J. Dennis Jr, and J. J. Moré, "On the local and superlinear convergence of quasi-Newton methods," *IMA Journal of Applied Mathematics*, vol. 12, no. 3, pp. 223–245, 1973.

[29] J. E. Dennis and J. J. Moré, "A characterization of superlinear convergence and its application to quasi-Newton methods," *Mathematics of Computation*, vol. 28, no. 126, pp. 549–560, 1974.

[30] M. J. Powell, "Some global convergence properties of a variable metric algorithm for minimization without exact line searches," *Nonlinear Programming*, vol. 9, no. 1, pp. 53–72, 1976.

[31] M. Gürbüzbalaban, A. Ozdaglar, and P. Parrilo, "Convergence rate of incremental gradient and incremental Newton methods," *SIAM Journal on Optimization*, vol. 29, no. 4, pp. 2542–2565, 2019.

[32] ——, "A globally convergent incremental Newton method," *Mathematical Programming*, vol. 151, no. 1, pp. 283–313, 2015.

[33] A. Mokhtari, M. Eisen, and A. Ribeiro, "IQN: An incremental quasi-Newton method with local superlinear convergence rate," *SIAM Journal on Optimization*, vol. 28, no. 2, pp. 1670–1698, 2018.

[34] D. Blatt, A. O. Hero, and H. Gauchman, "A convergent incremental gradient method with a constant step size," *SIAM Journal on Optimization*, vol. 18, no. 1, pp. 29–51, 2007.

[35] T. Chen, G. Giannakis, T. Sun, and W. Yin, "LAG: Lazily aggregated gradient for communication-efficient distributed learning," in *Advances in Neural Information Processing Systems*, 2018, pp. 5050–5060.

[36] J. Sun, T. Chen, G. Giannakis, and Z. Yang, "Communication-efficient distributed learning via lazily aggregated quantized gradients," in *Advances in Neural Information Processing Systems*, 2019, pp. 3365–3375.

[37] C.-p. Lee, C. H. Lim, and S. J. Wright, "A distributed quasi-Newton algorithm for empirical risk minimization with nonsmooth regularization," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 1646–1655.

[38] A. Nedic and A. Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *IEEE Transactions on Automatic Control*, vol. 54, no. 1, p. 48, 2009.

[39] M. Eisen, A. Mokhtari, and A. Ribeiro, "Decentralized quasi-Newton methods," *IEEE Transactions on Signal Processing*, vol. 65, no. 10, pp. 2613–2628, 2017.

[40] R. Crane and F. Roosta, "Dingo: Distributed Newton-type method for gradient-norm optimization," *arXiv preprint arXiv:1901.05134*, 2019.

[41] J. Nocedal and S. Wright, *Numerical Optimization*. Springer Science & Business Media, 2006.

[42] J. B. Erway and R. F. Marcia, "On efficiently computing the eigenvalues of limited-memory quasi-Newton matrices," *SIAM Journal on Matrix Analysis and Applications*, vol. 36, no. 3, pp. 1338–1359, 2015.

[43] M. Apostolopoulou, D. Sotiropoulos, C. Botsaris, and P. Pintelas, "A practical method for solving large-scale TRS," *Optimization Letters*, vol. 5, no. 2, pp. 207–227, 2011.

[44] C.-C. Chang and C.-J. Lin, "Libsvm: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 2, no. 3, pp. 1–27, 2011.

[45] S. Wang, F. Roosta, P. Xu, and M. W. Mahoney, "Giant: Globally improved approximate Newton method for distributed optimization," in *Advances in Neural Information Processing Systems*, 2018, pp. 2332–2342.

[46] K. Mishchenko, F. Iutzeler, J. Malick, and M.-R. Amini, "A delay-tolerant proximal-gradient algorithm for distributed learning," in *International Conference on Machine Learning*, 2018, pp. 3584–3592.

[47] S. Wang, F. Roosta-Khorasani, P. Xu, and M. W. Mahoney, "GIANT: Globally Improved Approximate Newton Method for Distributed Optimization," *ArXiv e-prints*, Sep. 2017.

[48] O. Shamir, N. Srebro, and T. Zhang, "Communication-efficient distributed optimization using an approximate Newton-type method," in *International Conference on Machine Learning*, 2014, pp. 1000–1008.

[49] Y. Nesterov, *Introductory Lectures on Convex Optimization: A basic course*. Springer Science & Business Media, 2013, vol. 87.

[50] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G. D. Peterson *et al.*, "XSEDE: Accelerating scientific discovery computing in science & engineering, 16 (5): 62–74, sep 2014," *URL https://doi. org/10.1109/mcse*, 2014.

[51] R. Johnson and T. Zhang, "Accelerating stochastic gradient descent using predictive variance reduction," in *Advances in Neural Information Processing Systems*, 2013, pp. 315–323.

[52] O. Shamir, N. Srebro, and T. Zhang, "Communication-efficient distributed optimization using an approximate Newton-type method," in *International Conference on Machine Learning*, 2014, pp. 1000–1008.

[53] M. Schmidt, R. Babanezhad, M. Ahmed, A. Defazio, A. Clifton, and A. Sarkar, "Non-uniform stochastic average gradient method for training conditional random fields," in *Artificial Intelligence and Statistics*, 2015, pp. 819–828.

[54] A. W. Marshall, I. Olkin, and B. C. Arnold, *Inequalities: Theory of Majorization and Its Applications*. Springer, 1979, vol. 143.