AN EXPONENTIALLY CONVERGENT ALGORITHM FOR LEARNING UNDER DISTRIBUTED FEATURES

Bicheng Ying[*†], *Kun Yuan*[*†], *and Ali H. Sayed*[†]

[*]Department of Electrical Engineering, University of California, Los Angeles
[†]School of Engineering, École Polytechnique Fédérale de Lausanne, Switzerland

ABSTRACT

This work studies the problem of learning under both large data and large feature space scenarios. The feature information is assumed to be spread across agents in a network, where each agent observes some of the features. Through local cooperation, the agents are supposed to interact with each other to solve the inference problem and converge towards the global minimizer of the empirical risk. We study this problem exclusively in the primal domain, and propose new and effective distributed solutions with guaranteed convergence to the minimizer. This is achieved by combining a dynamic diffusion construction, a pipeline strategy, and variance-reduced techniques. Simulation results illustrate the conclusions.

Index Terms— distributed features, dynamic diffusion, consensus, pipeline strategy, variance-reduced method, distributed optimization, primal solution.

1. INTRODUCTION AND PROBLEM FORMULATION

We consider empirical risks of the following form, which are common in learning and optimization formulations (see, e.g., [1–3,5,6]):

$$R(w) = \frac{1}{N}\sum_{n=1}^{N} Q\Big(h_n^{\mathsf{T}} w; \gamma_n\Big) \tag{1}$$

In (1), the unknown parameter model (or separating hyperplane) is designated by $w \in \mathbb{R}^M$, while $h_n \in \mathbb{R}^M$ denotes the n-th feature vector and γ_n the corresponding label. Moreover, the notation $Q(h^{\mathsf{T}}w;\gamma)$ refers to the loss function and is assumed to be a differentiable and convex function over w. In most problems of interest, the loss function is dependent on the inner product $h^{\mathsf{T}}w$ rather than the individual terms $\{h, w\}$.

For large data applications, where both N and M can be large, it is not uncommon for the dataset to be too large to be stored, or even processed effectively, at a single location or by a single agent. In this article, we consider the scenario in which the entries of the feature vectors are actually distributed over a collection of K networked agents, as illustrated in Fig. 1. Specifically, we partition each h_n, and similarly the weight vector w, into K sub-vectors denoted by $\{h_{n,k}, w_k\}$, where $k = 1, 2, \ldots, K$:

$$h_n \triangleq \begin{bmatrix} h_{n,1} \\ \hline h_{n,2} \\ \hline \vdots \\ \hline h_{n,K} \end{bmatrix}, \quad w \triangleq \begin{bmatrix} w_1 \\ \hline w_2 \\ \hline \vdots \\ \hline w_k \end{bmatrix}$$

Each sub-feature vector $h_{n,k}$ and sub-vector w_k are assumed to be located at agent k. In this way, the empirical risk function can be rewritten in the form

$$R(w) = \frac{1}{N}\sum_{n=1}^{N} Q\left(\sum_{k=1}^{K} h_{n,k}^{\mathsf{T}} w_k; \gamma_n\right) \tag{2}$$

Observe that in this new form, the argument of the loss function is now a sum over the inner products $h_{n,k}^{\mathsf{T}} w_k$. That is, we have a "cost-of-sum" form similar to what was discussed in [7].

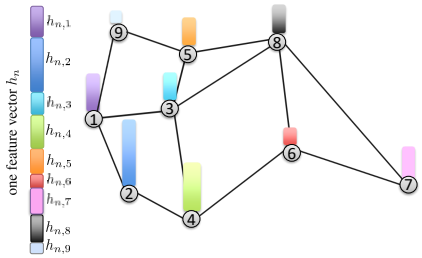

Fig. 1: Distributing the feature across the networked agents.

Our objective is to optimize (2) over the $\{w_k\}$ and to seek the optimal values in a distributed manner. Problems of this type have been pursued before in the literature by using duality arguments, such as those in [5, 7, 8]. The resulting algorithms suffer from some limitations. For example, they involve two time-scales: one scale governs the rate at which data arrives and another faster inner scale for running averaging iterations multiple times between data arrivals. This feature limits the rates at which data can arrive because the inner calculation will need to be completed before the arrival of the next datum. Another limitation is that the resulting algorithms rely on the use of conjugate functions, which are not always available in closed form; this will greatly depend on the nature of the loss function $Q(\cdot)$. Other useful approaches to solving problems of the form (2) are the Alternating Direction Method of Multipliers (ADMM) [9] and primal dual-methods [10, 11]. These techniques have good convergence properties but continue to suffer from high computational costs and two-time scale communications. Problems similar to (2) were also studied in [12] using primal methods in a deterministic setting, but the approach is not well-suited for big-data applications.

In this work, we propose a stochastic solution method to (2) that operates directly in the primal domain. We exploit the idea of dynamic consensus algorithm [13, 14], which has been adopted in the distributed optimization algorithms to track the average of gradients, see [15–17]. Meanwhile, we are interested in tracking the sum of score, $\sum_{k=1}^{K} h_{n,k}^{\mathsf{T}} w_k$, due to the different problem setting. By avoiding the dual formulation, we will arrive at a simple and effective method even for large scale applications. Even more importantly, we can show that the proposed method is able to converge at a linear rate to the exact minimizer of $R(w)$ even under constant step-size learning. The algorithm will not require two time-scales and will not

This work was supported in part by NSF grants CCF-1524250 and ECCS-1407712. Email: {ybc, kunyuan}@ucla.edu and ali.sayed@epfl.ch

necessitate the solution of auxiliary sub-optimization problems as is common in prior methods in the literature.

2. NAÏVE SOLUTION

We first propose a simple and naïve solution, which will be used to motivate the more advanced algorithms in later sections.

To begin with, let us consider the problem of minimizing (2) by means of a stochastic gradient recursion. Let $\alpha_{n,k} = h_{n,k}^{\mathsf{T}} w_k$ denote the inner product that is available at agent k at time n and define

$$z_n \triangleq \sum_{k=1}^{K} \alpha_{n,k} \tag{3}$$

If we denote the average of these local inner products by $\bar{\alpha}_n \triangleq \frac{1}{K}\sum_{k=1}^{K} \alpha_{n,k}$, then the variable z_n is a scaled multiple of $\bar{\alpha}_n$, namely, $z_n = K\bar{\alpha}_n$. Returning to problem (2), the stochastic gradient step will involve approximating the true gradient vector of $R(w)$ by the gradient vector of the loss function evaluated at some randomly selected data pair $(\boldsymbol{z}_{\boldsymbol{n}_i}, \gamma_{\boldsymbol{n}_i})$, where $\boldsymbol{n}_i$ at iteration i denotes the index of the sample pair selected uniformly at random from the index set $\{1, 2, \ldots, N\}$. In particular, the stochastic gradient descent recursion is given by

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \mu \nabla_z Q(\boldsymbol{z}_{\boldsymbol{n}_i}; \gamma_{\boldsymbol{n}_i}) h_{\boldsymbol{n}_i} \tag{4}$$

Note that $\boldsymbol{n}_i$ is independent of the iterates $\{\boldsymbol{w}_j\}_{j=0}^{i}$. Recalling that h_n and w are partitioned into K blocks, we can decompose (4) into K parallel recursions run at the local agents:

$$\boldsymbol{w}_{i+1,k} = \boldsymbol{w}_{i,k} - \mu \nabla_z Q\left(\boldsymbol{z}_{\boldsymbol{n}_i}; \gamma_{\boldsymbol{n}_i}\right) h_{\boldsymbol{n}_i,k}, \quad k = 1, \cdots, K \tag{5}$$

One main problem with (5) is that it is *not* a distributed solution because agents need to calculate $\nabla_z Q(z; \gamma)$ at z whose value depends on all sub-vectors $\{w_k\}$ from all agents and not just on w_k from agent k. This difficulty suggests one initial solution method.

Since the desired variable z_n is proportional to the average value of $\bar{\alpha}_n$, then a consensus-type construction can be used to approximate this average. To do so, neighboring agents share their $\{\alpha_{n,k}\}$ and perform consensus iterations repeatedly until they converge close enough to the desired average. This construction would take the following form. For some total number of iterations J, each agent would start from $\bar{\alpha}_{n,k}^{(0)} = \alpha_{n,k}$ and repeat the following calculations:

$$\bar{\alpha}_{n,k}^{(j)} \leftarrow \sum_{\ell \in \mathcal{N}_k} a_{\ell k} \bar{\alpha}_{n,\ell}^{(j-1)}, \quad j = 1, 2, \ldots, J \tag{6}$$

In this construction, the notation $\mathcal{N}_k$ represents the set of neighbors of agent k, while the $\{a_{\ell k}\}$ are nonnegative convex combination coefficients such that the matrix $A = [a_{\ell k}]$ is symmetric and doubly-stochastic, i.e.,

$$\sum_{\ell=1}^{N} a_{\ell k} = 1, \quad \sum_{k=1}^{N} a_{\ell k} = 1 \tag{7}$$

Also, we assume $a_{kk} > 0$ for at least one agent k. If J is large enough, it is known that $\bar{\alpha}_{n,k}$ will approach the desired average $\bar{\alpha}_n$ across all agents. However, this mode of operation requires the agents to complete J consensus updates between two data arrivals and requires a two-time scale operation: a faster time-scale for the consensus iterations and a slower time-scale for the data arrivals. One simplification is to set $J = 1$ and to have each agent perform only one single combination step to generate the variable:

$$\widehat{\boldsymbol{z}}_{\boldsymbol{n}_i,k} = \sum_{\ell \in \mathcal{N}_k} a_{\ell k} K h_{\boldsymbol{n}_i,k}^{\mathsf{T}} w_{i,k} \tag{8}$$

Algorithm 1 (Naïve feature-distributed method for agent k)

Repeat for $i = 1, 2, \ldots$:

$$\boldsymbol{n}_i \sim \mathcal{U}[1, N] \quad \text{(uniformly sampled)} \tag{9}$$

$$\widehat{\boldsymbol{z}}_{\boldsymbol{n}_i,k} = \sum_{\ell \in \mathcal{N}_k} a_{\ell k} (K \cdot h_{\boldsymbol{n}_i,k}^{\mathsf{T}} w_{i,k}) \tag{10}$$

$$\boldsymbol{w}_{i+1,k} = \boldsymbol{w}_{i,k} - \mu \nabla_z Q\Big(\widehat{\boldsymbol{z}}_{\boldsymbol{n}_i,k}; \gamma_{\boldsymbol{n}_i}\Big) h_{\boldsymbol{n}_i,k} \tag{11}$$

End

where we are expressing the result of this single combination by $\widehat{\boldsymbol{z}}_{\boldsymbol{n}_i,k}$ to indicate that this is the estimate for $z_{\boldsymbol{n}_i}$ that is computed at agent k at iteration i.

We list the resulting algorithm in (9)–(11). Observe that this implementation requires all agents to use the same random index $\boldsymbol{n}_i$ at iteration i. Although this requirement may appear restrictive, it can still be implemented in distributed architectures. For example, each agent can be set with the same random seed so that they can generate the same index variable $\boldsymbol{n}_i$ at iteration i. Alternatively, agents can sample the data in a cyclic manner instead of uniform sampling. In this paper, we assume each agent k will sample the same index $\boldsymbol{n}_i$ at iteration i for simplicity.

2.1. Limitations

Algorithm 1 is easy to implement. However, it suffers from two major drawbacks. First, the variable $\widehat{\boldsymbol{z}}_{\boldsymbol{n}_i,k}$ generated by the combination step (10) is not generally a good approximation for the global variable $\boldsymbol{z}_{\boldsymbol{n}_i}$. This approximation error affects the stability of the algorithm and requires the use of very small step-sizes. A second drawback is that the stochastic-gradient implementation (9)–(11) will converge to a small neighborhood around the exact minimizer rather than to the exact minimizer itself [3, 18]. In the following sections, we will design a more effective solution.

3. CORRECTING THE APPROXIMATION ERROR

3.1. Dynamic Diffusion Strategy

Inspired by the dynamic average consensus method [13, 14], we will design a stochastic diffusion-based algorithm to correct the error introduced by the one-step average consensus (10). To motivate the algorithm, let us step back and assume that each agent k in the network is observing some dynamic input signal, assuming that each agent k observes the dynamic input signal $r_{i,k} \in \mathbb{R}^M$, that changes with time i. Assume we want to develop a scheme to ensure that each agent k is able to track the average of all local signals, i.e. $\bar{r}_i = \frac{1}{K}\sum_{k=1}^{K} r_{i,k}$. For that purpose, we formulate an optimization problem of the form:

$$\min_{x \in \mathbb{R}^P} \; C_i(x) = \frac{1}{K} \sum_{k=1}^{K} \frac{1}{2} \|x - r_{i,k}\|^2 \tag{12}$$

where the cost function $C_i(x)$ is changing with time i. The global minimizer of $C_i(x)$ is the desired average $\bar{r}_i$. However, we would like the agents to attain this solution in a distributed fashion. To this end, we apply the exact diffusion algorithm [19, 20] to solve (12). The algorithm simplifies to the following recursions:

$$\psi_{i+1,k} = x_{i,k} - \mu(w_{i,k} - r_{i+1,k}) \tag{13}$$

$$\phi_{i+1,k} = \psi_{i+1,k} + x_{i,k} - \psi_{i,k} \tag{14}$$

$$x_{i+1,k} = \sum_{\ell \in \mathcal{N}_k} a_{\ell k} \phi_{i+1,\ell} \tag{15}$$

Since the Lipschitz constant L of C_i is always 1, we can set $\mu = 1/L = 1$ in (13) and combine all three recursions to get

$$x_{i+1,k} = \sum_{\ell \in \mathcal{N}_k} a_{\ell k} \left(x_{i,\ell} + r_{i+1,\ell} - r_{i,\ell} \right) \tag{16}$$

with $x_{0,k} = r_{0,k}$ for any k. It can be proved that each $x_{i,k}$ will track $\bar{r} = \frac{1}{K}\sum_{k=1}^{K} r_k$ very well when $r_{i,k}$ converges to r_k as $i \to \infty$ [13, 19, 20]. We refer to (16) as the dynamic diffusion method.

We now apply this intermediate result to the earlier recursion (5). Recall that there we need to evaluate the variable

$$\boldsymbol{z}_{\boldsymbol{n}_i} = \sum_{k=1}^{K} h_{\boldsymbol{n}_i,k}^{\mathsf{T}} \boldsymbol{w}_{i,k} \tag{17}$$

Calculating this quantity is similar to solving problem (12). However, there is one key difference: the signal $h_{\boldsymbol{n}_i}$ is not *deterministic* but *stochastic* and it varies randomly with the data index $\boldsymbol{n}_i$. This suggests that in principle we should keep track of N variables z_n, one for each possible $n = 1, 2, \ldots, N$. This is of course not feasible for large data. Instead, we propose a more efficient solution where the data is sparsely sampled as we proceed to describe. Assume first, for the sake of argument only, that we move ahead and compute the variable z_n for every possible value for n. If we do so, we would need to repeat construction (16) a total of N times, one for each n, as follows:

$$\boldsymbol{z}_{1,k}^{i+1} = \sum_{\ell \in \mathcal{N}_k} a_{\ell k} \left(\boldsymbol{z}_{1,\ell}^{i} + K \cdot h_{1,\ell}^{\mathsf{T}} \boldsymbol{w}_{i,\ell} - K h_{1,\ell}^{\mathsf{T}} \boldsymbol{w}_{i-1,\ell} \right) \tag{18}$$

$$\vdots$$

$$\boldsymbol{z}_{N,k}^{i+1} = \sum_{\ell \in \mathcal{N}_k} a_{\ell k} \left(\boldsymbol{z}_{N,\ell}^{i} + K \cdot h_{N,\ell}^{\mathsf{T}} \boldsymbol{w}_{i,\ell} - K h_{N,\ell}^{\mathsf{T}} \boldsymbol{w}_{i-1,\ell} \right) \tag{19}$$

In this description, we are adding a superscript i to each $\boldsymbol{z}_{n,k}^{i}$ to indicate the iteration index. In this way, each $\boldsymbol{z}_{n,k}^{i}$ will be able to track the sum $\sum_{k=1}^{K} h_{n,k}^{\mathsf{T}} w_{i,k}$. However, since the data size N is usually very large, it is too expensive to communicate and update all $\{\boldsymbol{z}_{n,k}\}_{n=1}^{N}$ per iteration. We will propose a stochastic algorithm in which only *one* data $h_{\boldsymbol{n}_i,k}$ will be selected at iteration i and only the corresponding entry $z_{\boldsymbol{n}_i,k}^{i+1}$ be updated while all other $\boldsymbol{z}_{n,k}^{i+1}$ will stay unchanged for $n \neq \boldsymbol{n}_i$:

$$\begin{cases} \boldsymbol{z}_{\boldsymbol{n}_i,k}^{i+1} = \sum_{\ell \in \mathcal{N}_k} a_{\ell k} \left(\boldsymbol{z}_{\boldsymbol{n}_i,\ell}^{j} + K h_{\boldsymbol{n}_i,\ell}^{\mathsf{T}} \boldsymbol{w}_{i,\ell} - K h_{\boldsymbol{n}_i,\ell}^{\mathsf{T}} \boldsymbol{w}_{j-1,\ell} \right) \\ \boldsymbol{z}_{n,k}^{i+1} = \boldsymbol{z}_{n,k}^{i}, \qquad n \neq \boldsymbol{n}_i \end{cases} \tag{20}$$

where the index j in the first equation refers to the most recent iteration where the same index $\boldsymbol{n}_i$ was chosen the last time. Note that the value j depends on $\boldsymbol{n}_i$ and the history of sampling, and therefore we need to store the inner product value that is associated with it. To fetch $\boldsymbol{z}_{n,k} j$ and $K h_{\boldsymbol{n}_i,\ell}^{\mathsf{T}} \boldsymbol{w}_{j-1,\ell}$ easily, we introduce two auxiliary variables:

$$\boldsymbol{u}_{\boldsymbol{n}_i,k}^{i} \leftarrow \boldsymbol{z}_{\boldsymbol{n}_i,\ell}^{j}, \quad \boldsymbol{v}_{\boldsymbol{n}_i,k}^{i} \leftarrow K h_{\boldsymbol{n}_j,k}^{\mathsf{T}} \boldsymbol{w}_{j-1,k} \tag{21}$$

3.2. Variance-Reduction Algorithm

We can enhance the algorithm by accounting for the gradient noise that is present in (4): this noise is the difference between the true gradient of the cost function and the gradient of the loss function that is used in (4). It is known in [3, 18, 21] that under constant step-size adaptation, and due to the gradient noise, recursion (4) will only approach an $O(\mu)-$neighborhood around the global minimizer. We can modify the recursion to ensure convergence to the exact minimizer as follows.

There is a family of variance-reduction algorithms such as SVRG [22], SAGA [23], and AVRG [24] that can approach the exact solution with constant step-size. In this work, we exploit SAGA construction because the variables $\{\boldsymbol{u}_{n,k}\}$ can readily be used in that implementation. In this case, the stochastic gradient recursion(5) at each agent k will be modified to (24) with two correction terms. The resulting algorithm is summarized in Algorithm 2.

Algorithm 2 [Variance-reduced dynamic diffusion (VRD^2) for learning from distributed features]

Initialization: Set $w_{0,k} = 0$; $\boldsymbol{u}_{n,k}^{0} = 0$; $\boldsymbol{v}_{n,k}^{0} = 0$.

Repeat for $i = 1, 2, \ldots$:

$$\boldsymbol{n}_i \sim \mathcal{U}[1, N] \quad \text{(uniformly sampled)} \tag{22}$$

$$\boldsymbol{z}_{\boldsymbol{n}_i,k} = \sum_{\ell \in \mathcal{N}_k} a_{\ell k} \left(\boldsymbol{u}_{\boldsymbol{n}_i,\ell}^{i} + K h_{\boldsymbol{n}_i,\ell}^{\mathsf{T}} \boldsymbol{w}_{i,\ell} - \boldsymbol{v}_{\boldsymbol{n}_i,\ell}^{i} \right) \tag{23}$$

$$\boldsymbol{w}_{i+1,k} = \boldsymbol{w}_{i,k} - \mu \Bigg\{ \left[\nabla_z Q(\boldsymbol{z}_{\boldsymbol{n}_i,k}; \gamma_{\boldsymbol{n}_i}) - \nabla_z Q(\boldsymbol{u}_{\boldsymbol{n}_i,k}^{i}; \gamma_{\boldsymbol{n}_i}) \right] h_{\boldsymbol{n}_i,k} + \frac{1}{N} \sum_{n=1}^{N} \nabla_z Q\left(\boldsymbol{u}_{n,k}^{i}; \gamma_n \right) h_{n,k} \Bigg\} \tag{24}$$

$$\boldsymbol{u}_{n,k}^{i+1} = \begin{cases} \boldsymbol{z}_{\boldsymbol{n}_i,k}^{i+1}, & \text{if } n = \boldsymbol{n}_i \\ \boldsymbol{u}_{n,k}^{i}, & \text{otherwise} \end{cases} \tag{25}$$

$$\boldsymbol{v}_{n,k}^{i+1} = \begin{cases} K h_{\boldsymbol{n}_i,k}^{\mathsf{T}} \boldsymbol{w}_{k,i}, & \text{if } n = \boldsymbol{n}_i \\ \boldsymbol{v}_{n,k}^{i}, & \text{otherwise} \end{cases} \tag{26}$$

End

4. ACCELERATION WITH PIPELINE

Algorithm 2 can be shown to converge to the solution of problem (1) for sufficiently small step-sizes. However, it is observed in numerical experiments that the convergence rate of Algorithm 2 can be slow. One reason is that the variable $\boldsymbol{z}_{n,k}$ generated by (23) converges slowly to $\sum_{k=1}^{K} h_{n,k}^{\mathsf{T}} \boldsymbol{w}_{i,k}$. To accelerate its convergence, it is necessary to run (23) multiple times before the gradient descent step (24), which, however, will result in a two-time-scale algorithm. In this section, we propose a pipeline method that can accelerate the convergence of $\boldsymbol{z}_{n,k}$ while maintaining the one-time-scale structure.

A pipeline is a set of data processing elements connected in series, where the output of one element is the input to the next one. We assume each agent k stores J variables at iteration i:

$$[\boldsymbol{z}_{\boldsymbol{n}_i,k}^{(0)}, \boldsymbol{z}_{\boldsymbol{n}_{i-1},k}^{(1)}, \cdots \boldsymbol{z}_{\boldsymbol{n}_{i-J+1},k}^{(J-1)}] \in \mathbb{R}^{J}, \tag{27}$$

where $\boldsymbol{z}_{\boldsymbol{n}_{i-j},k}^{(j)}$ is the result after j average dynamic diffusion recursions (23). The subscript $\boldsymbol{n}_{i-j}$ indicates that $\boldsymbol{z}$ is computed from the index selected at time $i-j$ and iterates $\boldsymbol{w}_{i-j}$. Thus, $\boldsymbol{z}_{\boldsymbol{n}_{i-j-1},k}^{(j+1)}$ is *not* $\boldsymbol{z}_{\boldsymbol{n}_{i-j},k}^{(j)}$ after one dynamic diffusion step. At iteration i, we let each agent k run a *one-step* average consensus recursion:

$$[\boldsymbol{z}_{\boldsymbol{n}_i,k}^{(1)}, \boldsymbol{z}_{\boldsymbol{n}_{i-1},k}^{(2)}, \cdots \boldsymbol{z}_{\boldsymbol{n}_{i-J+1},k}^{(J)}] = \sum_{\ell \in \mathcal{N}_k} a_{\ell k} [\boldsymbol{z}_{\boldsymbol{n}_i,\ell}^{(0)}, \boldsymbol{z}_{\boldsymbol{n}_{i-1},\ell}^{(1)}, \cdots \boldsymbol{z}_{\boldsymbol{n}_{i-J+1},\ell}^{(J-1)}] \tag{28}$$

When the above recursion finishes, agent k pops up the variable $\boldsymbol{z}_{\boldsymbol{n}_{i-J+1},k}^{(J)}$ from memory and uses it to continue the stochastic gradient descent steps. Since $\boldsymbol{z}_{n,k}^{(J)}$ is the result after J dynamic average

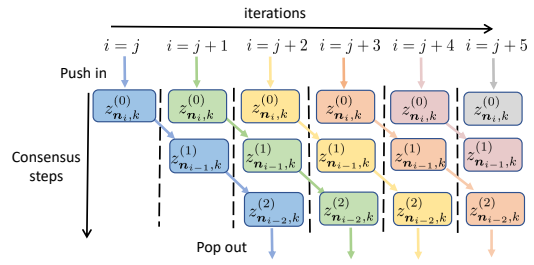

Fig. 2: Illustration of the pipeline strategy with buffer length $J = 3$.

consensus recursions, it can be viewed as a good approximation for $\sum_{k=1}^{K} h_{n,k}^{\mathsf{T}} \boldsymbol{w}_{n,k}$. At iteration $i+1$, agent k will push a new variable $\boldsymbol{z}^{0}_{\boldsymbol{n}_{i+1},k} = K \boldsymbol{h}^{\mathsf{T}}_{\boldsymbol{n}_{i+1},k} \boldsymbol{w}_{i+1,k}$ into buffer.

Recursion (28) employs the pipeline strategy. For example, variable $\boldsymbol{z}^{(1)}_{\boldsymbol{n}_i,k}$ is updated at iteration i. This new output $\boldsymbol{z}^{(1)}_{\boldsymbol{n}_i,k}$ will then be the second input at iteration i and is used to produce the output $\boldsymbol{z}^{(2)}_{\boldsymbol{n}_i,k}$. Next, the output $\boldsymbol{z}^{(2)}_{\boldsymbol{n}_i,k}$ will be the third input at iteration $i+2$ and is used to produce the output $\boldsymbol{z}^{(3)}_{\boldsymbol{n}_i,k}$. If we follow this procedure, the output $z^{(J)}_{\boldsymbol{n}_i,k}$ will be reached at iteration $i+J-1$. At that time, we can pop up $z^{(J)}_{\boldsymbol{n}_i,k}$ and use it in the stochastic gradient steps. The pipeline procedure is summarized in the "Pipeline function" shown above, and Fig. 2 also illustrates the pipeline strategy.

The pipeline strategy has two advantages. First, it is able to calculate $z^{(J)}_{n,k}$ without inner loop, which accelerates the algorithm and maintains the one-time-scale structure. Second, in one iteration, the two-time-scale solution sends a scalar J times while the pipeline solution sends J-length vector just once. Though the communication load is the same, the pipeline solution is usually faster than the two-time-scale approach in practice. That is because sending a scalar with J time needs all agents to synchronize for J times, which can take longer time than the one-time communication.

Theorem 1 *Under the assumption that the risk $R(w)$ is v-strongly convex and has Lipschitz continuous gradient with respect to z and w, Algorithm PVRD^2 converges at a linear rate for sufficiently small step-sizes μ, i.e.,*

$$\mathbb{E}\,\|\boldsymbol{w}_{i,k} - w^\star\|^2 \leq \rho^i C, \quad \forall k, i > 0 \tag{29}$$

for some constant C, where:

$$\rho = \max\left(1 - \frac{1-\lambda^J}{2N}, 1 - \mu\nu/4\right) \tag{30}$$

and λ is the magnitude of the second largest eigenvalue of A.

Pipeline function

Initialization: $\boldsymbol{z}_{\boldsymbol{n}_i,k} = 0$ for any $i \leq 0$

Function Pipeline $\left(\boldsymbol{z}^{(0)}_{\boldsymbol{n}_i,k}, \boldsymbol{v}^{i+J-1}_{\boldsymbol{n}_i,k}\right)$

Push $\left[\boldsymbol{z}^{(0)}_{\boldsymbol{n}_i,k}, \boldsymbol{v}^{i+J-1}_{\boldsymbol{n}_i,k}\right]$ into the queue (36)

$$[\boldsymbol{z}^{(1)}_{\boldsymbol{n}_i,k}, \boldsymbol{z}^{(2)}_{\boldsymbol{n}_{i-1},k}, \cdots \boldsymbol{z}^{(J)}_{\boldsymbol{n}_{i-J+1},k}] = \sum_{\ell \in \mathcal{N}_k} a_{\ell k} [\boldsymbol{z}^{(0)}_{\boldsymbol{n}_i,\ell}, \boldsymbol{z}^{(1)}_{\boldsymbol{n}_{i-1},\ell}, \cdots \boldsymbol{z}^{(J-1)}_{\boldsymbol{n}_{i-J+1},\ell}] \tag{37}$$

Pop $\left[\boldsymbol{z}^{(J)}_{\boldsymbol{n}_{i-J+1},k}, \boldsymbol{v}^{i}_{\boldsymbol{n}_{i-J+1},k}\right]$ out of the queue (38)

Return $\left[\boldsymbol{z}^{(J)}_{\boldsymbol{n}_{i-J+1},k}, \boldsymbol{v}^{i}_{\boldsymbol{n}_{i-J+1},k}\right]$

Algorithm 3 [Pipelined variance-reduced dynamic diffusion (PVRD^2) learning]

Initialization: Set $w_{0,k} = 0$; $\boldsymbol{u}^0_{n,k} = 0$; $\boldsymbol{v}^0_{n,k} = 0$.

Repeat for $i = 1, 2, \ldots$:

$$\boldsymbol{n}_i \sim \mathcal{U}[1, N] \quad \text{(uniformly sampling)} \tag{31}$$

$$\left[\boldsymbol{z}^{(J)}_{\boldsymbol{n}'_i,k}, \boldsymbol{v}^{i+1}_{\boldsymbol{n}'_i,k}\right] \quad (\text{denote } \boldsymbol{n}'_i \triangleq \boldsymbol{n}_{i-J+1}) \tag{32}$$

$$= \text{Pipeline}\left(\boldsymbol{u}^i_{\boldsymbol{n}_i,k} + K h^{\mathsf{T}}_{\boldsymbol{n}_i,k} \boldsymbol{w}_{i,k} - \boldsymbol{v}^i_{\boldsymbol{n}_i,k},\ K h^{\mathsf{T}}_{\boldsymbol{n}_i,k} \boldsymbol{w}_{i,k}\right)$$

$$\boldsymbol{w}_{i+1,k} = \boldsymbol{w}_{i,k} - \mu \Bigg\{ \left[\nabla_z Q\left(\boldsymbol{z}^{(J)}_{\boldsymbol{n}'_i,k}; \gamma_{\boldsymbol{n}'_i}\right) - \nabla_z Q\left(\boldsymbol{u}^i_{\boldsymbol{n}'_i,k}; \gamma_{\boldsymbol{n}'_i}\right)\right] h_{\boldsymbol{n}'_i,k} + \frac{1}{N} \sum_{n=1}^{N} \nabla_z Q\left(\boldsymbol{u}^i_{n,k}; \gamma_n\right) h_{n,k} \Bigg\} \tag{33}$$

$$\boldsymbol{u}^{i+1}_{n,k} = \begin{cases} \boldsymbol{z}^{(J)}_{\boldsymbol{n}'_i,k}, & \text{if } n = \boldsymbol{n}'_i \\ \boldsymbol{u}^i_{n,k}, & \text{otherwise} \end{cases} \tag{34}$$

$$\boldsymbol{v}^{i+1}_{n,k} = \begin{cases} \boldsymbol{v}^{i+1}_{\boldsymbol{n}'_i,k}, & \text{if } n = \boldsymbol{n}'_i \\ \boldsymbol{v}^i_{n,k}, & \text{otherwise} \end{cases} \tag{35}$$

End

5. SIMULATION

We exam on the MNIST dataset, which consists of 50000 28×28 handwritten digits. In the simulation, we consider the classification task of predicting digit 0 and digit 1. We separate the features into 8 networked agents. From top two subplots in Fig. 3, we see that each agent is in charge of part of w, and each converges to its corresponding part of $w^\star$. Next, we compare our algorithm to the method proposed in [12] with some modification, which can be viewed as the deterministic full-gradient version of our algorithm without pipeline. To make a fair comparison, we plot the convergence curve based on the count of gradients calculated in bottom of Fig. 3. The curve shows that the larger J we set, the faster the algorithm converges.

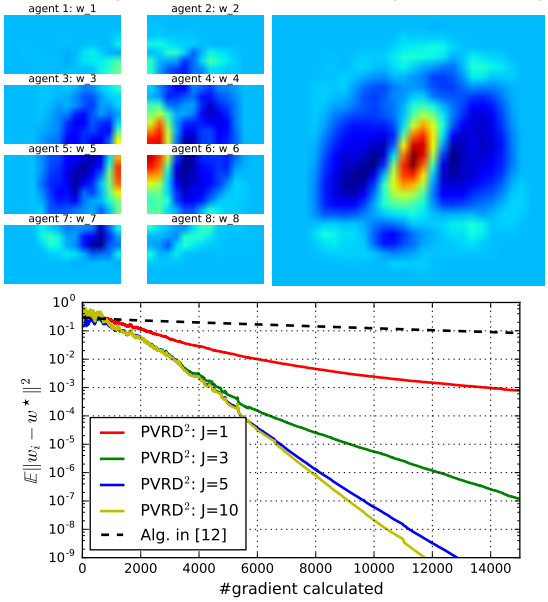

Fig. 3: Top left: Visualization of $\boldsymbol{w}_{i,k}$; Top right: Visualization of $w^\star$. Bottom: Illustration of the convergence behavior.

6. REFERENCES

[1] C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006.

[2] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, Springer, 2009.

[3] A. H. Sayed, "Adaptation, learning, and optimization over networks," *Foundations and Trends in Machine Learning*, vol. 7, no. 4–5, pp. 311–801, 2014.

[4] S. Kar and J. M. F. Moura, "Distributed consensus algorithms in sensor networks with imperfect communication: Link failures and channel noise," *IEEE Transactions on Signal Processing*, vol. 57, no. 1, pp. 355–369, 2009.

[5] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*, Prentice Hall, NJ, 1989.

[6] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[7] J. Chen, Z. J. Towfic, and A. H. Sayed, "Dictionary learning over distributed models," *IEEE Transactions on Signal Processing*, vol. 63, no. 4, pp. 1001–1016, 2015.

[8] B. Ying and A. H. Sayed, "Diffusion gradient boosting for networked learning," in *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2017, pp. 2512–2516.

[9] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.

[10] T.-H. Chang, A. Nedic, and A. Scaglione, "Distributed constrained optimization by consensus-based primal-dual perturbation method," *IEEE Transactions on Automatic Control*, vol. 59, no. 6, pp. 1524–1538, 2014.

[11] J. F. Mota, J. M. Xavier, P. M. Aguiar, and M. Puschel, "Distributed basis pursuit," *IEEE Transactions on Signal Processing*, vol. 60, no. 4, pp. 1942–1956, 2012.

[12] S. Sundhar, A. Nedić, and V. V. Veeravalli, "A new class of distributed optimization algorithms: Application to regression of distributed data," *Optimization Methods and Software*, vol. 27, no. 1, pp. 71–88, 2012.

[13] M. Zhu and S. Martinez, "Discrete-time dynamic average consensus," *Automatica*, vol. 46, no. 2, pp. 322–329, 2010.

[14] R. A. Freeman, P. Yang, and K. M. Lynch, "Stability and convergence properties of dynamic average consensus estimators," in *45th IEEE Conference on Decision and Control*, 2006, pp. 338–343.

[15] A. Nedic, A. Olshevsky, and W. Shi, "Achieving geometric convergence for distributed optimization over time-varying graphs," *SIAM Journal on Optimization*, vol. 27, no. 4, pp. 2597–2633, 2017.

[16] S. Pu, W. Shi, J. Xu, and A. Nedich, "A push-pull gradient method for distributed optimization in networks," *Avaible at arXiv:1803.07588*, 2018.

[17] R. Xin and U. A. Khan, "A linear algorithm for optimization over directed graphs with geometric convergence," *Avaible at arXiv:1803.02503*, 2018.

[18] K. Yuan, B. Ying, S. Vlaski, and A. H. Sayed, "Stochastic gradient descent with finite samples sizes," in *Proc. IEEE International Workshop on Machine Learning for Signal Processing*, Salerno, Italy, 2016, pp. 1–6.

[19] K. Yuan, B. Ying, X. Zhao, and A. H. Sayed, "Exact dffusion for distri- buted optimization and learning – Part I: Algorithm development," *submitted for publication* and available as *arXiv:1702.05122*, Feb. 2017.

[20] K. Yuan, B. Ying, X. Zhao, and A. H. Sayed, "Exact dffusion for distri- buted optimization and learning – Part II: Convergence analysis," *submitted for publication* and available as *arXiv:1702.05122*, Feb. 2017.

[21] B. T. Polyak, *Introduction to Optimization*, Optimization Software, NY, 1987.

[22] R. Johnson and T. Zhang, "Accelerating stochastic gradient descent using predictive variance reduction," in *Proc. Advances in Neural Information Processing Systems* (NIPS), Lake Tahoe, Navada, 2013, pp. 315–323.

[23] A. Defazio, F. Bach, and S. Lacoste-Julien, "SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives," in *Proc. Advances in Neural Information Processing Systems* (NIPS), Montreal, Canada, 2014, pp. 1646–1654.

[24] B. Ying, K. Yuan, and A. H. Sayed, "Variance-reduced stochastic learning under random reshuffling," submitted for publication. *Also available at arXiv:1708.01383*, Aug. 2017.