2016 IEEE INTERNATIONAL WORKSHOP ON MACHINE LEARNING FOR SIGNAL PROCESSING, SEPT. 13–16, 2016, SALERNO, ITALY

PARALLEL AND DISTRIBUTED TRAINING OF NEURAL NETWORKS VIA SUCCESSIVE CONVEX APPROXIMATION

Paolo Di Lorenzo

Dept. of Engineering, University of Perugia,
paolo.dilorenzo@unipg.it

Simone Scardapane

DIET Dept., Sapienza University of Rome
simone.scardapane@uniroma1.it

ABSTRACT

The aim of this paper is to develop a theoretical framework for training neural network (NN) models, when data is distributed over a set of agents that are connected to each other through a sparse network topology. The framework builds on a distributed convexification technique, while leveraging dynamic consensus to propagate the information over the network. It can be customized to work with different loss and regularization functions, typically used when training NN models, while guaranteeing provable convergence to a stationary solution under mild assumptions. Interestingly, it naturally leads to *distributed architectures* where agents solve local optimization problems exploiting *parallel* multi-core processors. Numerical results corroborate our theoretical findings, and assess the performance for parallel and distributed training of neural networks.

Index Terms— Artificial neural networks; Distributed algorithms; Parallel algorithms; Nonconvex optimization.

1. INTRODUCTION

Distributed learning is the problem of inferring a function from training data that is distributed among a network of agents having, typically, a sparse topology [1, 2]. It is a fundamental problem arising in multiple scenarios including wireless sensor networks [1], 5G mobile edge computing [3], and others. Common to these applications is the necessity of performing a completely decentralized computation/optimization. For instance, when data are stored in a distributed network, e.g., in clouds, sharing local information with a central processor is either unfeasible or not economical/efficient, owing to the large size of the network and volume of data, time-varying network topology, energy constraints, and/or privacy issues. Thus, we are interested in algorithms designed without any form of centralized coordination and/or memory sharing (e.g., a common parameter server [4]), and where communication is restricted to immediate neighbors, whose connectivity pattern can eventually vary over time. In the last years, this setting has been popularized in the field of adaptive filtering over networks [2].

Distributed optimization of *convex* cost functions has a long history in the literature of several fields as, e.g., operation research, machine learning, signal processing, communications, and automatic control. As a consequence, several models and cost functions giving rise to convex learning criteria have been explored widely in the distributed scenario, including support vector machines [5], sparse estimation [6], kernel ridge regression [1], random-weights networks [7], and many others. In most cases, these works resulted from the direct application of existing distributed optimization protocols for convex functions, e.g. [8], or from the combination of local convex solvers with in-network communication.

On the other side, solving dense *distributed nonconvex problems*, such as those encountered when training neural architectures from data distributed over a network, is still a challenging open problem. In fact, we are aware of only a few very recent works dealing with distributed algorithms for nonconvex optimization, see, e.g., [9, 10]. For this reason, up to date, research on distributed protocols for general neural network (NN) models has been scarce, mostly limited to exploiting the additivity property of the gradient when considering first-order descent procedures [11], or by resorting to sub-optimal ensembling protocols. To the best of our knowledge, there are no available algorithms that solve general (nonconvex) distributed learning problems appearing when training NNs in a distributed fashion, with provable convergence/performance guarantees. The development of such algorithmic framework would represent a fundamental tool in applications where nonconvex distributed learning is required, especially in the case of high-dimensional (big) data where "deep" NNs have recently become the state-of-the-art.

In this paper we make a step toward this exciting direction, thus developing the *first* theoretical framework for (batch) training of NN models when data is distributed over a network of agents. Our framework builds on a novel convexification-decomposition technique derived from [10], and it can handle any NN model (including architectures with multiple layers), as long as it is possible to properly define a derivative between the output and its internal weights. Addi-

The work of Paolo Di Lorenzo was supported by the "Fondazione Cassa di Risparmio di Perugia".

tionally, it can be customized to incorporate several convex cost functions (including squared errors and cross-entropy criteria), and convex regularizers. The proposed method hinges on a (primal) successive convex approximation (SCA) framework, while leveraging dynamic consensus as a mechanism to distribute the computation among the agents as well as propagate the needed information over the network. Interestingly, the method naturally leads to *distributed architectures* where agents solve local optimization problems exploiting *parallelized* multi-core processors. To the best of our knowledge, this is the first attempt to design algorithms for NN training that are *parallel* (inside each agent) and *distributed* (across the agents) at the same time.

2. PROBLEM FORMULATION

Let us consider a set $\mathcal{S}$ of M input-output pairs $\{\mathbf{x}_m, \mathbf{d}_m\}_{m=1}^{M}$, where $\mathbf{x}_m \in \mathbb{R}^D$ and $\mathbf{d}_m \in \mathbb{R}^O$. Denote by $f(\mathbf{w};\mathbf{x})$ the NN model, where all the adjustable parameters are concatenated in a single weight vector $\mathbf{w} \in \mathbb{R}^Q$. Let us also assume that the set $\mathcal{S}$ of data is distributed over a network of I agents, such that each agent has access only to a smaller set $\mathcal{S}_i$, with $\bigcup_{i=1}^{I} \mathcal{S}_i = \mathcal{S}$. The goal of the network is to find the vector $\mathbf{w}$, which better fits the training data, in a totally distributed fashion. A general formulation for the distributed training of neural networks can be cast as the minimization of a social cost function G plus a regularization term r, which writes as:

$$\min_{\mathbf{w}} \; U(\mathbf{w}) = G(\mathbf{w}) + r(\mathbf{w}) = \sum_{i=1}^{I} g_i(\mathbf{w}) + r(\mathbf{w}) \,, \qquad (1)$$

where $g_i(\cdot)$ is the local cost function of agent i, defined as:

$$g_i(\mathbf{w}) = \sum_{m \in \mathcal{S}_i} l(\mathbf{d}_{i,m}, f(\mathbf{w};\mathbf{x}_{i,m})) \,, \qquad (2)$$

where $l(\cdot,\cdot)$ is a (convex) loss function, and $\mathbf{x}_{i,m}$ (and $\mathbf{d}_{i,m}$) denotes the m-th data sample available to the ith agent. Due to the nonlinearity of the NN model $f(\mathbf{w};\mathbf{x})$, problem (1) is typically *nonconvex*. We consider the following assumptions on the functions involved in (1)-(2).

Assumption A [On Problem (1)]:

(A1) f is C^1, with Lipschitz continuous gradient;

(A2) l is convex and C^1, with Lipschitz continuous gradient;

(A3) r is a convex function (possibly nondifferentiable) with bounded subgradients;

(A4) U is coercive, i.e., $\lim_{\|\mathbf{w}\|\to\infty} U(\mathbf{w}) = +\infty$.

The structure of the function $l(\cdot,\cdot)$ depends on the learning task (i.e., regression, classification, etc.). Typical choices are the squared loss for regression problems, and the cross-entropy for classification tasks. The regularization function $r(\mathbf{w})$ is commonly chosen to avoid overfitted solutions and/or impose a specific structure in the solution, e.g. sparsity. Typical choices are the ℓ_2 and ℓ_1 norms.

On network topology: Time is slotted, and at any time-slot n, the network is modeled as a digraph $\mathcal{G}[n] = (\mathcal{V}, \mathcal{E}[n])$, where $\mathcal{V} = \{1, \ldots, I\}$ is the vertex set (i.e., the set of agents), and $\mathcal{E}[n]$ is the set of (possibly) time-varying directed edges. The in-neighborhood of agent i at time n (including node i) is defined as $\mathcal{N}_i^{\text{in}}[n] = \{j | (j,i) \in \mathcal{E}[n]\} \cup \{i\}$; it sets the communication pattern between single-hop neighbors: agents $j \neq i$ in $\mathcal{N}_i^{\text{in}}[n]$ can communicate with node i at time n. Associated with each graph $\mathcal{G}[n]$, we introduce (possibly) time-varying weights $c_{ij}[n]$ matching $\mathcal{G}[n]$:

$$c_{ij}[n] = \begin{cases} \theta_{ij} \in [\vartheta, 1] & \text{if } j \in \mathcal{N}_i^{\text{in}}[n]; \\ 0 & \text{otherwise,} \end{cases} \qquad (3)$$

for some $\vartheta \in (0,1)$, and define the matrix $\boldsymbol{C}[n] \triangleq (c_{ij}[n])_{i,j=1}^{I}$. These weights will be used later on in the definition of the proposed algorithm. We also make the following assumption on the network connectivity, on the sequence of weight matrices $\{\boldsymbol{C}[n]\}_n$, and on the knowledge of each agent.

Assumption B [On the network topology/knowledge]:

(B1) The sequence of graphs $\mathcal{G}[n]$ is B-strongly connected, i.e., there exists an integer $B > 0$ such that the graph $\mathcal{G}[k] = (\mathcal{V}, \mathcal{E}_B[k])$, with $\mathcal{E}_B[k] = \bigcup_{n=kB}^{(k+1)B-1} \mathcal{E}[n]$ is strongly connected, for all $k \geq 0$;

(B2) Every weight matrix $\boldsymbol{C}[n]$ in (3) is doubly stochastic, i.e. it satisfies

$$\boldsymbol{C}[n]\,\mathbf{1} = \mathbf{1} \quad \text{and} \quad \mathbf{1}^T \boldsymbol{C}[n] = \mathbf{1}^T \quad \forall n. \qquad (4)$$

(B3) Each agent i knows only its own cost function g_i (but not the entire G), and the common function r.

Assumption B1 states that it exists an integer $B > 0$ such that the union graph over a time window of B instants is strongly connected; B1 allows strong connectivity to occur over a long time period and in arbitrary order. Note also from B2 that $\boldsymbol{C}[n]$ (and thus the network topology) can be time-varying and need not be symmetric (but doubly stochastic). Our aim in the sequel is to design a class of algorithms for the solution of general NN training problems in (1), while being implementable in the above setting (Assumptions A and B).

3. DISTRIBUTED SCA FRAMEWORK

Devising parallel and distributed algorithms for Problem (1) faces two main challenges, namely: the nonconvexity of the U in (1) and the lack of global information at each agent side. To cope with these issues, we exploit the framework for distributed nonconvex optimization recently proposed in [10], which combines SCA techniques (Step 1) with dynamic consensus mechanisms (Steps 2 and 3), as described next.

Step 1 (local SCA optimization): Each agent i maintains a local estimate $\mathbf{w}_i[n]$ of the optimization variable $\mathbf{w}$ that is iteratively updated. Solving directly Problem (1) may be too costly (due to the nonconvexity of G) and is not even feasible in a distributed setting. One may then prefer to approximate Problem (1), in some suitable sense, in order to permit each agent to compute *locally* and *efficiently* the new iteration. Proceeding as in [10], writing $G(\mathbf{w}_i) = g_i(\mathbf{w}_i) + \sum_{j\neq i} g_j(\mathbf{w}_i)$, we consider a convexification of G having the following form: i) at every iteration n, the (possibly) nonconvex $g_i(\mathbf{w}_i)$ is replaced by a strongly convex surrogate, say $\widetilde{g}_i(\bullet;\mathbf{w}_i[n]) : \mathbb{R}^Q \to \mathbb{R}$, which may depend on the current iterate $\mathbf{w}_i[n]$; and ii) $\sum_{j\neq i} g_j(\mathbf{w}_i)$ is linearized around $\mathbf{w}_i[n]$. More formally, the proposed updating scheme reads: at every iteration n, given the local estimate $\mathbf{w}_i[n]$, each agent i solves the *strongly convex* optimization problem:

$$\begin{aligned}\widetilde{\mathbf{w}}_i[n] &= \underset{\mathbf{w}_i}{\arg\min}\ \widetilde{U}_i\left(\mathbf{w}_i;\mathbf{w}_i[n],\boldsymbol{\pi}_i[n]\right) \qquad (5)\\ &= \underset{\mathbf{w}_i}{\arg\min}\ \widetilde{g}_i(\mathbf{w}_i;\mathbf{w}_i[n]) + \boldsymbol{\pi}_i[n]^T(\mathbf{w}_i - \mathbf{w}_i[n]) + r(\mathbf{w}_i).\end{aligned}$$

where

$$\boldsymbol{\pi}_i[n] \triangleq \sum_{j\neq i} \nabla_{\mathbf{w}}\, g_j(\mathbf{w}_i[n]). \qquad (6)$$

The evaluation of (6) would require the knowledge of all $\nabla g_j(\mathbf{w}_i[n])$, $j \neq i$ at node i. This information is not directly available at node i (see assumption B3); we will cope with this local lack of global knowledge later on in step 3. Once the surrogate problem (5) is solved, each agent computes an auxiliary variable, say $\mathbf{z}_i[n]$, as the convex combination:

$$\mathbf{z}_i[n] = \mathbf{w}_i[n] + \alpha[n]\left(\widetilde{\mathbf{w}}_i[n] - \mathbf{w}_i[n]\right)\,, \qquad (7)$$

where $\alpha[n]$ is a possibly time-varying step-size sequence. This concludes the optimization phase of the algorithm.

An appropriate choice of the surrogate function $\widetilde{g}_i(\bullet;\mathbf{w}_i[n])$ guarantees the coincidence between the fixed-points of $\widetilde{\mathbf{w}}_i[n]$ and the stationary solutions of Problem (1). The main results are given in the following proposition; the proof follows the same steps as [12, Prop. 8(b)] and thus is omitted.

Proposition 1. *Given Problem (1) under A1-A4, suppose that $\widetilde{g}_i$ satisfies the following conditions:*

(F1) $\widetilde{g}_i(\bullet;\mathbf{w})$ *is uniformly strongly convex with* $\tau_i > 0$*;*
(F2) $\nabla\widetilde{g}_i(\mathbf{w};\mathbf{w}) = \nabla g_i(\mathbf{w})$ *for all* $\mathbf{w}$*;*
(F3) $\nabla\widetilde{g}_i(\mathbf{w};\bullet)$ *is uniformly Lipschitz continuous.*

Then, the set of fixed-point of $\widetilde{\mathbf{w}}_i[n]$ in (5) coincides with that of the stationary solutions of (1).

Conditions F1-F3 state that $\widetilde{g}_i$ should be regarded as a convex approximation of g_i at the point $\mathbf{w}$, which preserves the first order properties of g_i. Several feasible choices are possible for a given g_i; the appropriate one depends on computational and communication requirements. In the sequel, we will illustrate some possible choices for the local cost (2).

Step 2 (agreement update): To force the asymptotic agreement among the $\mathbf{w}_i$'s, a consensus-based step is employed on the auxiliary variables $\mathbf{z}_i[n]$'s. Each agent i updates its local variable $\mathbf{w}_i[n]$ as:

$$\mathbf{w}_i[n+1] = \sum_{j\in\mathcal{N}_i^{\text{in}}[n]} c_{ij}[n]\,\mathbf{z}_i[n], \qquad (8)$$

where $(c_{ij}[n])_{ij}$ satisfy Assumption B2. Since the weights are constrained by the network topology, (8) can be implemented via local message exchanges: agent i updates its estimate $\mathbf{w}_i$ by averaging over the current solutions $\mathbf{z}_j[n]$ received from its neighbors. The rationale behind the proposed iterates (5)-(8) is to compute stationary solutions of Problem (1), while reaching asymptotic agreement on $\{\mathbf{w}_i\}_{i=1}^I$.

Step 3 (diffusion of information over the network): The computation of $\widetilde{\mathbf{w}}_i[n]$ in (5) is not fully distributed yet, because the evaluation of $\boldsymbol{\pi}_i[n]$ in (6) would require the knowledge of all $\nabla g_j(\mathbf{w}_i[n])$, $j \neq i$, which is a global information that is not available locally at node i (see B3). To cope with this issue, as proposed in [10], we replace $\boldsymbol{\pi}_i[n]$ in (5) with a *local* estimate, say $\widetilde{\boldsymbol{\pi}}_i[n]$, asymptotically converging to $\boldsymbol{\pi}_i[n]$. Following [10], we can update the local estimate $\widetilde{\boldsymbol{\pi}}_i[n]$ in a totally distributed manner as:

$$\widetilde{\boldsymbol{\pi}}_i[n] \triangleq I\cdot\mathbf{y}_i[n] - \nabla g_i(\mathbf{w}_i[n]), \qquad (9)$$

where $\mathbf{y}_i[n]$ is a local auxiliary variable (controlled by user i) that aims to asymptotically track the average of the gradients. Leveraging *dynamic* consensus methods [13], this can be done updating $\mathbf{y}_i[n]$ according to the following recursion:

$$\mathbf{y}_i[n+1] \triangleq \sum_{j=1}^{I} c_{ij}[n]\mathbf{y}_j[n] + \left(\nabla g_i(\mathbf{w}_i[n+1]) - \nabla g_i(\mathbf{w}_i[n])\right) \qquad (10)$$

with $\mathbf{y}_i[0] \triangleq \nabla_{\mathbf{w}_i} g_i(\mathbf{w}_i[0])$. Note that the update of $\mathbf{y}_i[n]$ and thus $\widetilde{\boldsymbol{\pi}}_i[n]$ can be now performed locally with message exchanges with the agents in the neighborhood.

The final algorithm is summarized in Algorithm 1, and builds on the iterates (5), (8), and (9)-(10). Note that we used the following simplified notation: $\nabla g_i(\mathbf{w}_i[n])$ is denoted in Algorithm 1 as $\nabla g_i[n]$. The convergence properties of Algorithm 1 are given in Theorem 2.

Theorem 2. *Let $\{\mathbf{w}[n]\}_n \triangleq \{(\mathbf{w}_i[n])_{i=1}^I\}_n$ be the sequence generated by Algorithm 1, and let $\{\overline{\mathbf{w}}[n]\}_n \triangleq \{(1/I)\sum_{i=1}^I \mathbf{w}_i[n]\}_n$ be its average. Suppose that i) Assumptions A and B hold; and ii) the step-size sequence $\{\alpha[n]\}_n$ is chosen so that $\alpha[n] \in (0,1]$, for all n,*

$$\textstyle\sum_{n=0}^{\infty}\alpha[n] = \infty \quad \textit{and} \quad \sum_{n=0}^{\infty}\alpha[n]^2 < \infty. \qquad (12)$$

If sequence $\{\overline{\mathbf{w}}[n]\}_n$ is bounded, then (a) all its limit points are stationary solutions of (1); (b) all the sequences $\{\mathbf{w}_i[n]\}_n$ asymptotically agree, i.e., $\|\mathbf{w}_i[n] - \overline{\mathbf{w}}[n]\| \underset{n\to\infty}{\longrightarrow} 0$, for all i.

Proof. See [10]. $\square$

```
Algorithm 1: In-Network SCA
Data : $\mathbf{w}_i[0]$, $\mathbf{y}_i[0] = \nabla g_i[0]$, $\widetilde{\boldsymbol{\pi}}_i[0] = I\mathbf{y}_i[0] - \nabla g_i[0]$, $\forall i = 1, \ldots, I$, and $\{\boldsymbol{C}[n]\}_n$. Set $n = 0$.
(S.1) If $\mathbf{w}[n]$ satisfies a termination criterion: STOP;
(S.2) Local Optimization: Each agent
    (a) computes $\widetilde{\mathbf{w}}_i[n]$ as:
        $\widetilde{\mathbf{w}}_i[n] = \underset{\mathbf{w}_i}{\arg\min}\ \widetilde{U}_i(\mathbf{w}_i; \mathbf{w}_i[n], \widetilde{\boldsymbol{\pi}}_i[n])$    (11)
    (b) updates its local variable $\mathbf{z}_i[n]$:
        $\mathbf{z}_i[n] = \mathbf{w}_i[n] + \alpha[n](\widetilde{\mathbf{w}}_i[n] - \mathbf{w}_i[n])$
(S.3) Consensus update: Each agent $i$ collects data from its current neighbors and updates the estimates :
    (a) $\mathbf{w}_i[n+1] = \sum_{j=1}^{I} c_{ij}[n]\,\mathbf{z}_j[n]$
    (b) $\mathbf{y}_i[n+1] = \sum_{j=1}^{I} c_{ij}[n]\,\mathbf{y}_j[n] + (\nabla g_i[n+1] - \nabla g_i[n])$
    (c) $\widetilde{\boldsymbol{\pi}}_i[n+1] = I \cdot \mathbf{y}_i[n+1] - \nabla g_i[n+1]$
(S.4) $n \leftarrow n+1$, and go to (S.1).
```

On the choice of the surrogate functions $\widetilde{g}_i$: Algorithm 1 represents a family of *distributed SCA* methods for Problem (1). It encompasses a gamut of algorithms, each corresponding to various forms of the surrogate $\widetilde{g}_i$, the weight matrices $\boldsymbol{C}[n]$, and the step-size sequence $\alpha[n]$. These degrees of freedom offer a lot of flexibility to control iteration complexity, communication overhead, and convergence speed. Due to lack of space, here we focus only on the choice of the surrogate functions $\widetilde{g}_i$. In particular, from Proposition 1, we know that they must be chosen in order to satisfy F1-F3. A useful observation is that $g_i(\mathbf{w})$ in (2) is given by a composition of functions, where the exterior function l is convex (see Assumption A2). Thus, a possible choice is to preserve the convexity of $l(\cdot)$, while linearizing the interior function f around the current iterate $\mathbf{w}_i[n]$. The resulting surrogate writes as:

$$\widetilde{g}_i(\mathbf{w}_i; \mathbf{w}_i[n]) = \sum_{m \in \mathcal{S}_i} l(\mathbf{d}_{i,m}, \widetilde{f}(\mathbf{w}_i; \mathbf{w}_i[n], \mathbf{x}_{i,m})) + \frac{\tau_i}{2}\|\mathbf{w}_i - \mathbf{w}_i[n]\|^2, \tag{13}$$

where $\tau_i \geq 0$, and $\widetilde{f}(\mathbf{w}_i; \mathbf{w}_i[n], \mathbf{x}_{i,m}) = f(\mathbf{w}_i[n], \mathbf{x}_{i,m}) + \nabla_{\mathbf{w}} f(\mathbf{w}_i[n]; \mathbf{x}_{i,m})^T(\mathbf{w}_i - \mathbf{w}_i[n])$. If l is strongly convex, τ_i can also be set to zero.

The approximation in (13) preserves the hidden convexity in (2), but it might lead to local optimization problems in (11) that do not directly admit a closed form solution. An interesting tradeoff between performance and complexity can be achieved by selecting $\widetilde{g}_i$ as the linearization of g_i at $\mathbf{w}_i[n]$:

$$\widetilde{g}_i(\mathbf{w}_i; \mathbf{w}_i[n]) = g_i(\mathbf{w}_i[n]) + \nabla g_i(\mathbf{w}_i[n])^T(\mathbf{w}_i - \mathbf{w}_i[n]) + \frac{\tau_i}{2}\|\mathbf{w}_i - \mathbf{w}_i[n]\|^2, \tag{14}$$

where τ_i is any positive constant. The proximal regularization in (14) guarantees that $\widetilde{g}_i$ is strongly convex. It is easy to see how both surrogates (13) and (14) satisfy conditions F1-F3 given in Proposition 1.

Parallel computing of (11): When each node is equipped with a multi-core architecture or a cluster computer (e.g., each node is a cloud), each subproblem (11) can also be parallelized across the cores. Let us assume that there are C cores available at each node i, and partition $\mathbf{w}_i = (\mathbf{w}_{i,c})_{c=1}^{C}$ in C nonoverlapping blocks. Assume, also, that the regularization function r is block separable, i.e., $r(\mathbf{w}) = \sum_{c=1}^{C} r_{i,c}(\mathbf{w}_{i,c})$; an example of such a r is the ℓ_1-norm or the ℓ_2-block norm. Then, choose $\widetilde{g}_i$ as additively separable in the blocks $\mathbf{w}_i = (\mathbf{w}_{i,c})_{c=1}^{C}$, i.e., $\widetilde{g}_i(\mathbf{w}_i; \mathbf{w}_i[n]) = \sum_{c=1}^{C} \widetilde{g}_{i,c}(\mathbf{w}_{i,c}; \mathbf{w}_{i,-c}[n])$, where each $\widetilde{g}_{i,c}(\bullet; \mathbf{w}_{i,-c}[n])$ is any surrogate function satisfying F1-F3 in the variable $\mathbf{w}_{i,c}$, and $\mathbf{w}_{i,-c}[n] \triangleq (\mathbf{w}_{i,p}[n])_{1=p\neq c}^{C}$ denotes the tuple of all blocks excepts the c-th one. With the above choices, problem (11) decomposes in C separate strongly convex subproblems

$$\begin{aligned}\widetilde{\mathbf{w}}_{i,c}[n] &= \underset{\mathbf{w}_{i,c}}{\arg\min}\ \widetilde{U}_{i,c}(\mathbf{w}_{i,c}; \mathbf{w}_i[n], \widetilde{\boldsymbol{\pi}}_{i,c}[n]) \\ &= \underset{\mathbf{w}_{i,c}}{\arg\min}\ \widetilde{g}_i(\mathbf{w}_{i,c}; \mathbf{w}_{i,-c}[n]) + \widetilde{\boldsymbol{\pi}}_{i,c}[n]^T(\mathbf{w}_{i,c} - \mathbf{w}_{i,c}[n]) \\ &\qquad + r(\mathbf{w}_{i,c}),\end{aligned} \tag{15}$$

for $c = 1, \ldots, C$, where $\widetilde{\boldsymbol{\pi}}_{i,c}[n]$ denotes the c-th bock of $\widetilde{\boldsymbol{\pi}}_i[n]$. Each subproblem (15) can be now solved independently by a different core. Under F1-F3, the fixed points of the mapping $(\widetilde{\mathbf{w}}_{i,c}[n])_{c=1}^{C}$ in (15) coincide with the fixed points of $\widetilde{\mathbf{w}}_i[n]$ in (11), i.e. with the stationary solutions of (1).

A practical example of parallel/distributed NN training: We consider the case where the local function in (2) is built from a squared loss $l(\cdot,\cdot) = (\mathbf{d}_{i,m} - f(\mathbf{w}; \mathbf{x}_{i,m}))^2$, and an ℓ_2 norm regularization $r(\mathbf{w}) = \|\mathbf{w}\|_2^2$, which is extremely common when training NNs. For convenience, we define:

$$\mathbf{A}_i[n] = \sum_{m=1}^{M} \mathbf{J}_{i,m}^T[n]\mathbf{J}_{i,m}[n] + \lambda\mathbf{I}, \tag{16}$$

$$\mathbf{b}_i[n] = \sum_{m=1}^{M} \mathbf{r}_{i,m}^T[n]\mathbf{J}_{i,m}[n]. \tag{17}$$

with

$$[\mathbf{J}_{i,m}[n]]_{kl} = \frac{\partial f_k(\mathbf{w}_i[n]; \mathbf{x}_{i,m})}{\partial \mathrm{w}_l}. \tag{18}$$

$$\mathbf{r}_{i,m}[n] = \mathbf{d}_{i,m} - f(\mathbf{w}_i[n]; \mathbf{x}_{i,m}) + \mathbf{J}_{i,m}[n]\mathbf{w}_i[n], \tag{19}$$

$m = 1, \ldots, M$, $i = 1, \ldots, I$, $k = 1, \ldots, O$, and $l = 1, \ldots, Q$. Partitioning $\mathbf{w}_i = (\mathbf{w}_{i,c})_{c=1}^{C}$ in C nonoverlapping blocks, using (13) (with $\tau_i = 0$), after some algebra, the cost function of problem (11), at agent i and core c, can be cast as:

$$\widetilde{U}_{i,c}(\mathbf{w}_{i,c}; \mathbf{w}_i[n], \widetilde{\boldsymbol{\pi}}_{i,c}[n]) = \mathbf{w}_{i,c}^T \mathbf{A}_{i,c,c}[n] \mathbf{w}_{i,c} - 2(\mathbf{b}_{i,c}[n] + \mathbf{A}_{i,c,-c}[n]\mathbf{w}_{i,-c}[n] - 0.5 \cdot \widetilde{\boldsymbol{\pi}}_{i,c}[n])^T \mathbf{w}_i \quad (20)$$

where $\mathbf{A}_{i,c,c}[n]$ is the block (rows and columns) of the matrix $\mathbf{A}_i[n]$ in (16) corresponding to the c-th partition, whereas $\mathbf{A}_{i,c,-c}[n]$ takes the rows corresponding to the c-th partition and all the columns not associated to c. Then, from (20), the solution is given in closed form as:

$$\widetilde{\mathbf{w}}_{i,c}[n] = \mathbf{A}_{i,c}^{-1}[n](\mathbf{b}_{i,c}[n] + \mathbf{A}_{i,c,-c}[n]\mathbf{w}_{i,-c}[n] - 0.5 \cdot \widetilde{\boldsymbol{\pi}}_{i,c}[n]) \,, \quad (21)$$

$c = 1, \ldots, C$. Simplifications in terms of complexity can be achieved by using (14) (with $\tau_i > 0$) as a surrogate function for the local function g_i. In this case, the cost function of problem (11), at agent i and core c, can be cast as:

$$\widetilde{U}_{i,c}(\mathbf{w}_{i,c}; \mathbf{w}_i[n], \widetilde{\boldsymbol{\pi}}_{i,c}[n]) = (0.5 \cdot \tau + \lambda) \left\| \mathbf{w}_{i,c} \right\|^2 - (\tau_i \mathbf{w}_{i,c}[n] - \nabla_c \, g_i(\mathbf{w}_i[n]) - \widetilde{\boldsymbol{\pi}}_{i,c}[n])^T \mathbf{w}_{i,c}, \quad (22)$$

thus leading to the closed form solution:

$$\widetilde{\mathbf{w}}_{i,c}[n] = \left(\frac{2}{\tau + 2\lambda} \right) (\tau_i \mathbf{w}_{i,c}[n] - \nabla_c \, g_i(\mathbf{w}_i[n]) - \widetilde{\boldsymbol{\pi}}_{i,c}[n]) \quad (23)$$

$i = 1, \ldots, I$, $c = 1, \ldots, C$, where ∇_c denotes the gradient along the components of the c-th partition. The computational cost of (23) is much lower with respect to (21). This gain is paid in terms of a slower convergence speed, as we will see in the numerical simulations.

4. EXPERIMENTAL EVALUATION

4.1. Experimental setup

Experiments are performed by considering a standard NN with a single hidden layer, sigmoid nonlinearities, and adaptable bias terms. To evaluate the accuracy, we perform a 3-fold cross-validation on the available data, and for every fold we partition uniformly the training data among a predefined number of $I = 5$ agents with random connectivity, such that every edge has a fixed probability $p = 0.2$ to be in present in the graph. For simplicity, we consider time-invariant, connected, undirected communication graphs. The weights of the NN are initialized at each agent from a Gaussian distribution with zero mean and standard deviation 0.01. The overall cross-validation process is repeated 15 times, and results are averaged for each of the partitions. In all cases, the step-sizes satisfy (12), and are chosen according to the rule:

$$\alpha[n] = \alpha[n-1](1 - \mu\alpha[n-1]), \quad n \geq 1, \quad (24)$$

where α_0 and μ are chosen empirically with a process of hand-tuning, to provide the best practical convergence speed. Albeit not optimal, we have found this strategy provides a reliable behavior in most cases, while the two parameters are generally easy to tune. With respect to the mixing weights in (3), we consider the popular strategy denoted as "Metropolis" weights [2]. The distributed scenario is simulated on a single machine, using MATLAB R2015a on an Intel i7-3820 @3.6 GHz and 32 GB of memory. An open-source library to replicate the experiments is available on the web, together with a partial porting in the popular Theano Python framework.[1]

4.2. Experimental results

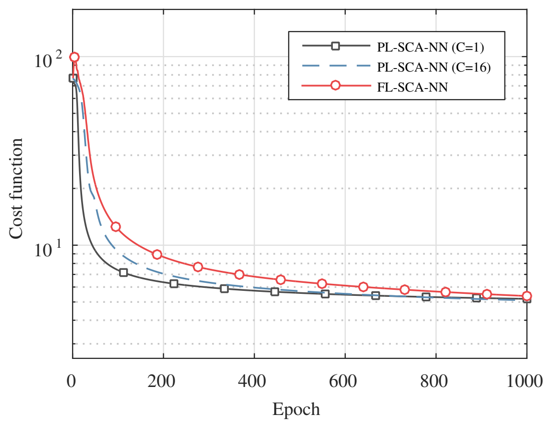
We consider the well-known Wisconsin breast cancer database (WDBC) as a simulated distributed medical setting. All features are normalized in $[0.1, 0.9]$, while the output of the network is binarized to obtain the class. We consider a NN with 20 hidden nodes, for a total of $Q = 641$ free parameters, using a small regularization factor $\lambda = 10^{-3}$. We vary the available (per-node) number of processors in the exponential range $C = 2^j$, with $j = 0, 1, \ldots, 6$. The step-sizes are selected for each choice of C according to the following procedure: we begin by a default setting of $\alpha_0 = \varepsilon = \tau = 0.1$ for $P = 0$, then, every time we increase the amount of processors, we steadily decrease α_0 by steps equal to 5%.

We consider two different variants of our algorithm, either with partial linearization (i.e., (21)) of the error term, and with full linearization (i.e., (23)). The two are denoted as PL-SCA-NN and FL-SCA-NN, respectively. All algorithms converged to equivalent solutions in terms of testing accuracy, obtaining an average misclassification error of 4%, equivalent to that of a purely centralized implementation. A representative set of evolutions of the cost function (evaluated at $\overline{\mathbf{w}}[n]$) is shown in Fig. 1a, where we report the behaviors of the PL-SCA-NN algorithm (for $C = 1$ and $C = 16$) and the FL-SCA-NN algorithm. We can notice how, increasing the number of processors available at each node, PL-SCA-NN tends to slightly decrease the convergence behavior, albeit this is always faster than FL-SCA-NN. Also, due to the specific nature of (23), the convergence behavior of FL-SCA-NN is independent of C, which explains why it is shown once in the figure. The availability of parallel core architectures allows to greatly reduce the training time of PL-SCA-NN, as shown in Fig. 1b. In particular, we have obtained a (per-node) reduction in training time of $\approx 40\%$ with 2 processors, of $\approx 52\%$ with just 4 processors, and up to $\approx 62\%$ for $C = 32$. Finally, we also report the vanishing behavior of the average disagreement among agents in Fig. 2, which confirms the theoretical results.

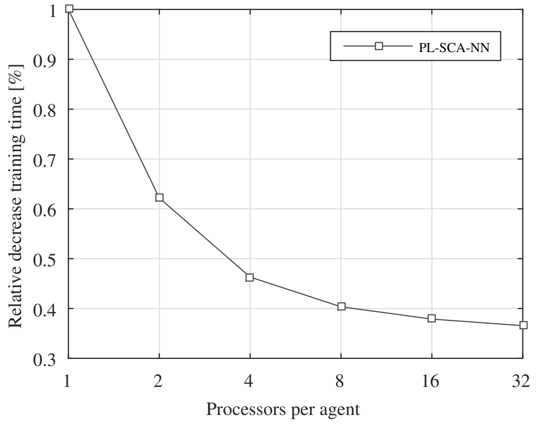
5. CONCLUSIONS

In this paper we have proposed a novel framework for parallel and distributed training of neural networks, where training

[1] `https://bitbucket.org/ispamm/parallel-and-distributed-neural-networks/`



(a) Cost function



(b) Training time

Fig. 1. (a) Cost function's evolution for FL-SCA-NN and PL-SCA-NN. (b) Relative decrease in training time (with respect to the case $C = 1$) obtained when varying the number of processors.

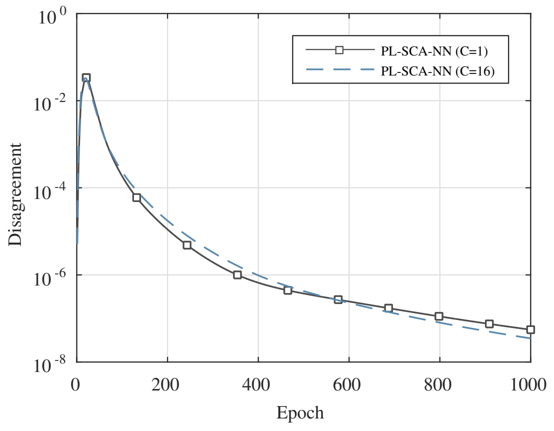


Fig. 2. Behavior of average disagreement $D[n]$, versus number of local communication exchanges.

data is distributed over a set of agents that are interconnected through a sparse network topology. The proposed method hinges on a (primal) successive convex approximation framework, and leverages dynamic consensus to propagate the information over the network. To the best of our knowledge, the proposed method is the first available in the literature to solve non-convex distributed learning problems with provable theoretical guarantees.

6. REFERENCES

[1] J. B. Predd, S. R. Kulkarni, and H. V. Poor, "Distributed learning in wireless sensor networks," *IEEE Signal Process. Mag.*, pp. 56–69, 2007.

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

[3] S. Barbarossa, S. Sardellitti, and P. Di Lorenzo, "Communicating while Computing: Distributed Mobile Cloud Computing over 5G Heterogeneous Networks," *IEEE Signal Process. Mag.*, vol. 31, no. 6, pp. 45–55, 2014.

[4] M. Zinkevich, M. Weimer, L. Li, and A. J. Smola, "Parallelized stochastic gradient descent," in *Advances in Neural Information Processing Systems*, 2010, pp. 2595–2603.

[5] P. A. Forero, A. Cano, and G. B. Giannakis, "Consensus-based distributed support vector machines," *J. of Machine Learning Research*, vol. 11, pp. 1663–1707, 2010.

[6] P. Di Lorenzo and A. H. Sayed, "Sparse distributed learning based on diffusion adaptation," *IEEE Trans. Signal Process.*, vol. 61, no. 6, pp. 1419–1433, 2013.

[7] S. Scardapane, D. Wang, M. Panella, and A. Uncini, "Distributed Learning with Random Vector Functional-Link Networks," *Inform. Sciences*, vol. 301, pp. 271–284, 2015.

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

[9] P. Bianchi and J. Jakubowicz, "Convergence of a multi-agent projected stochastic gradient algorithm for non-convex optimization," *IEEE Trans. Autom. Control*, vol. 58, no. 2, pp. 391–405, 2013.

[10] P. Di Lorenzo and G. Scutari, "NEXT: In-Network Nonconvex Optimization," *IEEE Trans. on Signal and Inform. Process. over Networks*, vol. 2, no. 2, pp. 120–136, 2016.

[11] D. Povey, X. Zhang, and S. Khudanpur, "Parallel training of Deep Neural Networks with Natural Gradient and Parameter Averaging," in *2015 Int. Conf. on Learning Representations*, 2015.

[12] F. Facchinei, G. Scutari, and S. Sagratella, "Parallel Selective Algorithms for Nonconvex Big Data Optimization," *IEEE Trans. on Signal Process.*, vol. 63, no. 7, pp. 1874–1889, 2015.

[13] M. Zhu and S. Martínez, "Discrete-time Dynamic Average Consensus," *Automatica*, vol. 46, no. 2, pp. 322–329, 2010.