Information Sciences 301 (2015) 271–284

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Distributed learning for Random Vector Functional-Link networks

Simone Scardapane [a,*], Dianhui Wang [b], Massimo Panella [a], Aurelio Uncini [a]

[a] Department of Information Engineering, Electronics and Telecommunications (DIET), "Sapienza" University of Rome, Via Eudossiana 18, 00184 Rome, Italy
[b] Department of Computer Science and Computer Engineering, La Trobe University, Melbourne, VIC 3086, Australia

ARTICLE INFO

Article history:
Received 28 October 2014
Received in revised form 9 December 2014
Accepted 8 January 2015
Available online 13 January 2015

Keywords:
Random Vector Functional-Link
Distributed learning
Consensus
Distributed Optimization

ABSTRACT

This paper aims to develop distributed learning algorithms for Random Vector Functional-Link (RVFL) networks, where training data is distributed under a decentralized information structure. Two algorithms are proposed by using Decentralized Average Consensus (DAC) and Alternating Direction Method of Multipliers (ADMM) strategies, respectively. These algorithms work in a fully distributed fashion and have no requirement on coordination from a central agent during the learning process. For distributed learning, the goal is to build a common learner model which optimizes the system performance over the whole set of local data. In this work, it is assumed that all stations know the initial weights of the input layer, the output weights of local RVFL networks can be shared through communication channels among neighboring nodes only, and local datasets are blocked strictly. The proposed learning algorithms are evaluated over five benchmark datasets. Experimental results with comparisons show that the DAC-based learning algorithm performs favorably in terms of effectiveness, efficiency and computational complexity, followed by the ADMM-based learning algorithm with promising accuracy but higher computational burden.

1. Introduction

Over the past decades, supervised learning techniques have been well developed with theoretical analyses and empirical studies [15]. The ICT world, however, is being rapidly reshaped by emerging trends such as big data [22], pervasive computing [33], commodity computing [11], Internet of things [2], and several others. All these frameworks have a similar common theme underlying them: computing power is now a widespread feature surrounding us, and the same can be said about data. Consequently, supervised learning is expected to face major technological and theoretical challenges, since in many situations the overall training data cannot be assumed to lie at a single location, nor is it realistic to have a centralized authority for collecting and processing it. The previous trends also put forth the challenge of analyzing *structured* and *heterogeneous* data [4], however, we are not concerned with this issue in this paper.

As a prototypical example, consider solving a music classification task (e.g., genre classification [35]) over a peer-to-peer (P2P) network of computers, each node possessing its own labeled database of songs. It is reasonable to assume that, to obtain good performance, no single database may be sufficient, and there is the need of leveraging over the data of *all* users.

* Corresponding author. Tel.: +39 06 44585495; fax: +39 06 4873300.
E-mail addresses: simone.scardapane@uniroma1.it (S. Scardapane), dh.wang@latrobe.edu.au (D. Wang), massimo.panella@uniroma1.it (M. Panella), aurelio.uncini@uniroma1.it (A. Uncini).

http://dx.doi.org/10.1016/j.ins.2015.01.007
0020-0255/

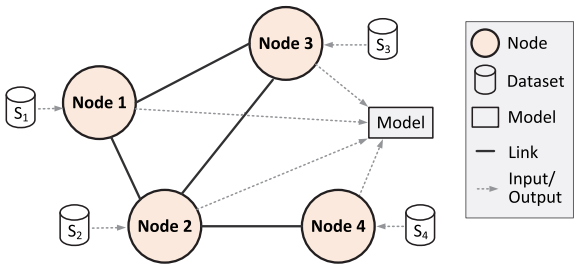

Fig. 1. Supervised learning in a network of agents: data is distributed throughout the nodes, and all of them must converge to a single learning model. For readability, we assume undirected connections between agents.

However, in a P2P network no centralized authority exists, hence the nodes need a *distributed* training protocol to solve the classification task. In fact, it is known that a fully decentralized training algorithm can be useful even in situations where having a master node is technologically feasible [16]. In particular, such a distributed algorithm would remove the risks of having a single point of failure, or a communication bottleneck towards the central node. Similar situations are also widespread in Wireless Sensor Networks (WSN), where additional power concerns arise [5]. Finally, it may happen that data simply cannot be moved across the network: either for being large (in term of number of examples or dimensionality of each pattern), or because fundamental privacy concerns are present [40]. The general setting, which we will call 'data-distributed learning', is graphically depicted in Fig. 1.

So far, a large body of research has gone into developing fully distributed, decentralized learning algorithms, including works on diffusion adaptation [21,34], learning by consensus [16], distributed learning on commodity clusters architectures [8], adaptation on WSNs [5,32], distributed online learning [13], distributed optimization [6,14,18,38], ad-hoc learning algorithms for specific architectures [12,26], distributed databases [20], and others. Despite this, many important research questions remain open [31], and in particular several well-known learning models, originally formulated in the centralized setting, have not yet been generalized to the fully decentralized setting.

In this paper, we propose two distributed learning algorithms for a yet-unexplored model, that is Random Vector Functional-Link (RVFL) networks [1,10,30,37]. As illustrated successively, RVFLs can be viewed as feedforward neural networks with a single hidden layer, resulting in a linear combination of a (fixed) number of non-linear expansions of the original input. A remarkable characteristics of such a learner model lies in the way of parameter assignment, that is, the input weights and biases are randomly chosen and fixed in advance before training. Despite this simplification, RVFLs can be shown to possess universal approximation capabilities, provided a sufficiently large set of expansions [17]. This grant them with a number of peculiar characteristics, making them particularly suited in a distributed environment. In particular, RVFL models are linear in the parameters, thus optimal parameters can be found with a standard linear regression routine, which can be implemented efficiently even in low-cost hardware, such as sensors or mobile devices [30]. In fact, the optimum of the training problem can be formulated in a closed form, involving only matrix inversions and multiplications, making the model efficient even when confronted with large amounts of data. Finally, the same formulation can be used equivalently in the classification and in the regression setting. In this paper, we focus on *batch* learning scheme development, however the proposed algorithms can be further extended for sequential learning with the use of standard gradient-descent procedures [10], whose decentralized formulation have been only partially investigated in the literature [34].

The key idea behind the proposed algorithms is to let all nodes train a local model (simultaneously) using the subset of training data, followed by finding the common output weights of the master learner model. Two effective approaches for defining the common output weights are adopted in this study. One is the Decentralized Average Consensus (DAC) strategy [28], and another is the well-known Alternating Direction Method of Multipliers (ADMM) algorithm [6]. DAC is an efficient protocol to compute averages over very general networks, with two main characteristics. Firstly, it does not require a centralized authority coordinating the overall process, and secondly, it can be easily implemented even on the most simple networks [16]. These characteristics have made DAC an attractive method in many distributed learning algorithms, particularly in the 'learning by consensus' theory outlined in [16]. From a theoretical viewpoint, the DAC-based algorithm is similar to a bagged ensemble of linear predictors [7], and despite its simplicity and non-optimal nature, our experimental simulations show that it results in highly competitive performance. The second strategy (ADMM) is the most widely employed distributed optimization algorithm in machine learning (e.g. for LASSO [6] and Support Vector Machines [14]), making it a natural candidate for the current research. This second strategy is more computational demanding than the DAC-based one, but it has high theoretical guarantees in term of convergence, speed and accuracy. Our simulation results obtained from both algorithms are quite promising and comparable to a centralized model exploiting the overall dataset. Moreover, the consensus strategy is extremely competitive on a large number of realistic network topologies.

The remainder of the paper is organized as follows. Section 2 briefly reviews RVFLs and its learning algorithm, and introduces the DAC algorithm. Section 3 describes the data-distributed learning framework, and proposes two training algorithms for RVFLs models. Sections 4 and 5 detail the experimental setup and the numerical results on four realistic datasets, plus an additional experiment on a large-scale image classification task, respectively. Section 6 concludes this paper with some discussions and future possible researches.

2. Preliminaries

This section provides some supportive results that will be used in the subsequent sections. We start from formulating some basic concepts related to RVFL networks with a least-square solution as its learning algorithm (Section 2.1). Then, we briefly introduce the DAC algorithm, for evaluating global averages under a decentralized information structure (Section 2.2).

2.1. Random-vector functional links

Let us consider first a regression problem with one-dimensional scalar outputs $y \in \mathbb{R}$. A Functional Link Artificial Neural Network (FLANN) with a single output neuron can be described as a weighted sum of B non-linear transformations of the input [30]:

$$f(\mathbf{x}) = \sum_{m=1}^{B} \beta_m h_m(\mathbf{x}; \mathbf{w}_m) = \boldsymbol{\beta}^T \mathbf{h}(\mathbf{x}; \mathbf{w}_1, \ldots, \mathbf{w}_B), \tag{1}$$

where the mth transformation is parametrized by the vector $\mathbf{w}_m$, and the input is a d-dimensional real vector $\mathbf{x} \in \mathbb{R}^d$. Each $h : \mathcal{X} \rightarrow \mathbb{R}$ is called a *base*, *hidden* function, or *functional link*. In particular, in our simulations we will use sigmoid basis functions given by:

$$h(\mathbf{x}; \mathbf{w}, b) = \frac{1}{1 + \exp\{-\mathbf{w}^T\mathbf{x} + b\}}. \tag{2}$$

The parameters $\mathbf{w}_1, \ldots, \mathbf{w}_B$ are chosen in the beginning of the learning process, independently of the training data. In particular, in a RVFL neural network the parameters of the hidden functions are chosen randomly from a predefined probability distribution [1,17,37]. A schematic depiction of a RVFL with two inputs, three hidden nodes, and one output is given in Fig. 2.

Under moderate assumptions on the smoothness of the underlying function, universal approximation capability of RVFLs is guaranteed provided a sufficiently large number of hidden functions, i.e., provided that B is large enough [17]. In this way, the problem of learning a model of the form in (1) is reduced to a linear regression over the coefficients $\boldsymbol{\beta} = [\beta_1, \ldots, \beta_B]^T$. Suppose we are provided with a set of N samples of the desired function, called the training set, that is $S = \{\mathbf{x}_i, y_i\}, i = 1 \ldots N$. Denote by $\mathbf{H}$ the *hidden* matrix:

$$\mathbf{H} = \begin{pmatrix} h_1(\mathbf{x}_1) & \cdots & h_B(\mathbf{x}_1) \\ \vdots & \ddots & \vdots \\ h_1(\mathbf{x}_N) & \cdots & h_B(\mathbf{x}_N) \end{pmatrix}, \tag{3}$$

where we have dropped the parametrization with respect to the $\mathbf{w}_m$ parameters for readability. Choosing the optimal $\boldsymbol{\beta}$ can be formulated as a standard regularized least-square problem:

$$\boldsymbol{\beta}^* = \arg\min_{\boldsymbol{\beta} \in \mathbb{R}^B} \frac{1}{2}\|\mathbf{H}\boldsymbol{\beta} - \mathbf{Y}\|_2^2 + \frac{\lambda}{2}\|\boldsymbol{\beta}\|_2^2, \tag{4}$$

where $\mathbf{Y}$ is a column vector whose ith element is the output sample y_i associated with the pattern $\mathbf{x}_i, i = 1 \ldots N$, of the training set. The problem in (4) is strictly convex, so its solution can be found by setting the gradient of $J(\boldsymbol{\beta})$ to 0:

$$\frac{\partial J}{\partial \boldsymbol{\beta}} = \mathbf{H}^T\mathbf{H}\boldsymbol{\beta} - \mathbf{H}^T\mathbf{Y} + \lambda\boldsymbol{\beta} = 0, \tag{5}$$

from which we obtain the well-known solution:

$$\boldsymbol{\beta}^* = \left(\mathbf{H}^T\mathbf{H} + \lambda\mathbf{I}\right)^{-1}\mathbf{H}^T\mathbf{Y}, \tag{6}$$

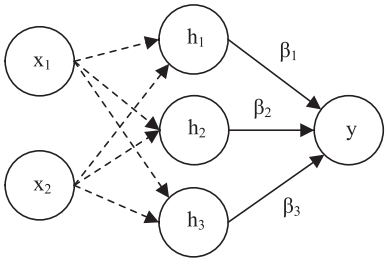

Fig. 2. Schematic depiction of a RVFL architecture with two inputs, three hidden nodes, and one output. Fixed connections are shown as dashed lines, while trainable connections as fixed lines.

where $\mathbf{I}$ is the identity matrix of suitable dimensionality. The computational complexity of RVFL training is mostly influenced by the $B \times B$ matrix inversion in (6). In cases where $N \ll B$, we can simplify it using the fact that, for any $\lambda > 0$, we have that:

$$\left(\mathbf{H}^T\mathbf{H} + \lambda\mathbf{I}\right)^{-1}\mathbf{H}^T = \mathbf{H}^T\left(\mathbf{H}\mathbf{H}^T + \lambda\mathbf{I}\right)^{-1}. \tag{7}$$

Combining Eqs. (6) and (7) we obtain an alternative formulation for the optimal weight vector, where the matrix to be inverted is reduced to dimensionality $N \times N$:

$$\boldsymbol{\beta}^* = \mathbf{H}^T\left(\mathbf{H}\mathbf{H}^T + \lambda\mathbf{I}\right)^{-1}\mathbf{Y}. \tag{8}$$

The derivation of optimal parameters in the case of binary classification or M-dimensional outputs (as for multi-valued regression or multiclass classification) proceeds identically, with the difference in the latter case that $\boldsymbol{\beta}$ becomes a $B \times M$ matrix and $\mathbf{Y}$ is a $N \times M$ matrix, where the ith row of $\mathbf{Y}$ will correspond to the M-dimensional output $\mathbf{y}_i^T$ of the training set. Moreover, the L_2-norm on vectors in (4) must be replaced with a suitable matrix norm (i.e., L_2 or Frobenius norm). In the case of binary classification, we have that $y \in \{-1, +1\}$, and we can retrieve the actual class of the pattern from the RVFL output as:

$$\text{Class of } \mathbf{x} = \operatorname{sgn} f(\mathbf{x}). \tag{9}$$

Similarly, in the case of multiclass classification, the output can take values in a set of M different labels $\{1, 2, \ldots, M\}$, where M is the number of classes. In this case, we consider the common encoding associating to a pattern $\mathbf{x}_i$ a single output vector $\mathbf{y}_i$ of M bits, where if its elements are $y_{ij} = 1$ and $y_{ik} = 0, k \neq j$, then the corresponding pattern is of class j. Also in this case, it is common to assume $\mathcal{Y} \subseteq \mathbb{R}^M$ and $f(\mathbf{x}) \in \mathbb{R}^M$, hence we can retrieve the actual class as:

$$\text{Class of } \mathbf{x} = \arg\max_{j=1\ldots M} f_j(\mathbf{x}), \tag{10}$$

where $f_j(\mathbf{x})$ is the jth element of the M-dimensional output $f(\mathbf{x})$.

2.2. Distributed average consensus

DAC is an iterative decentralized algorithm for computing the global average of a parameter vector over a network of nodes, which can be implemented even in low-cost devices. Assume a network of L interconnected nodes, whose connectivity is known a priori and can be formalized in the form of an adjacency matrix $\mathbf{A}$, where $A_{kj} = 1$ if node k is connected to node j, and 0 otherwise [27]. More generally, we may consider a real-valued connectivity matrix $\mathbf{C}$, where the element $C_{kj} \neq 0$ whenever two nodes are connected, and the magnitude of the values is proportional to the strength of the connection. A possible way to choose the elements of the matrix $\mathbf{C}$ will be discussed successively. For simplicity, we assume undirected connections (i.e., a symmetrical adjacency matrix), and a connected graph. In case of directed networks, convergence of the DAC procedure depends on the actual graph topology [5, Section III-A]. Finally, we denote by $\mathcal{N}_k$ the inclusive neighborhood of node k, i.e., the set of indexes of nodes connected to node k, including k itself.

Every node k has a measurement vector denoted by $\boldsymbol{\theta}_k, k = 1 \ldots L$, and the task is for *all* nodes to converge on the global average:

$$\hat{\boldsymbol{\theta}} = \frac{1}{L}\sum_{k=1}^{L}\boldsymbol{\theta}_k. \tag{11}$$

For generality, however, we allow every node to communicate only with its direct neighbors. DAC is an iterative procedure to compute Eq. (11) requiring only local communications. Denote by $\boldsymbol{\theta}_k[n]$ the average's estimate of node k at iteration n; estimates are initialized as $\boldsymbol{\theta}_k[0] = \boldsymbol{\theta}_k$. Then, for a generic time-instant n, every node computes a weighted sum of the current estimates of its neighborhood:

$$\boldsymbol{\theta}_k[n] = \sum_{j \in \mathcal{N}_k} C_{kj}\boldsymbol{\theta}_j[n-1], \tag{12}$$

where the values C_{kj} are the real-valued entries of the connectivity matrix $\mathbf{C}$. Consensus converges to the global average in (11), irrespective of the initial states, provided that the network is undirected, connected, and that the connectivity matrix $\mathbf{C}$ respects the following properties:

$$\mathbf{C} \cdot \mathbf{1} = \mathbf{1}, \tag{13}$$

$$\rho\left(\mathbf{C} - \frac{\mathbf{1} \cdot \mathbf{1}^T}{L}\right) < 1, \tag{14}$$

where $\mathbf{1}$ is a column vector containing only ones, and $\rho(\cdot)$ denotes the spectral radius of a matrix, given by:

$$\rho(\mathbf{M}) = \max_r\{|\lambda_r(\mathbf{M})|\}, \tag{15}$$

where $\lambda_r(\mathbf{M})$ is the rth eigenvector of a generic matrix $\mathbf{M}$. If these conditions are met, then, for $n \to \infty$ we have convergence to the global average at every node, i.e. $\boldsymbol{\theta}_k[n] \to \hat{\boldsymbol{\theta}}, k = 1 \dots L$. A simple choice for ensuring convergence is given by choosing the so-called 'max-degree' weights for the connectivity matrix $\mathbf{C}$ [28]:

$$C_{kj} = \begin{cases} \frac{1}{d+1} & \text{if } k \in \mathcal{N}_j \setminus k \\ 1 - \frac{d_k}{d+1} & \text{if } k = j \\ 0 & \text{otherwise} \end{cases}, \tag{16}$$

where $\mathcal{N}_j \setminus k$ is the set of neighbors of the kth node excluding k itself, d_k is the degree of node k and d is the maximum degree of the graph.[1] More sophisticated choices, such as the 'definite consensus' [16], are also possible. We refer to [42] and citations therein for a brief overview on recent advances on the topic of fast convergence in DAC algorithms.

As a stopping criterion, consensus stops if a maximum number of iterations is reached, or whenever the local update (in norm) at every node is less than an user-specified threshold $\delta > 0$:

$$\|\boldsymbol{\theta}_k[n] - \boldsymbol{\theta}_k[n-1]\|_2^2 < \delta, \; k \in \{1, 2, \dots, L\}. \tag{17}$$

3. Distributed learning for RVFLs

3.1. Problem formulation

In the distributed learning setting, we consider a network of nodes as detailed in Section 2.2, and we suppose that the kth node, $k = 1 \dots L$, has access to its own training set given by $S_k = \{\mathbf{x}_{k,i}, \mathbf{y}_{k,i}\}, i = 1 \dots N_k$. Note that we identify each example with a double subscript (k, i), meaning the ith example of the kth node. Moreover, we assume that node k has N_k examples available for training. In this case, extending Eq. (4), the global optimization problem can be stated as:

$$\boldsymbol{\beta}^* = \arg\min_{\boldsymbol{\beta} \in \mathbb{R}^B} \frac{1}{2} \left(\sum_{k=1}^{L} \|\mathbf{H}_k \boldsymbol{\beta} - \mathbf{Y}_k\|_2^2 \right) + \frac{\lambda}{2} \|\boldsymbol{\beta}\|_2^2, \tag{18}$$

where $\mathbf{H}_k$ and $\mathbf{Y}_k$ are the hidden matrix and output vector [2] computed over the dataset S_k. The task is for all the nodes to agree on a single parameter vector $\boldsymbol{\beta}^*$, whose generalization capability should approximate sufficiently well the optimal solution to problem (18). Remember from Section 1 that, in order to formulate algorithms with wide applicability, we impose three additional constraints that our decentralized training algorithms must respect. First, no node is allowed to coordinate the overall training process. Secondly, we do not allow any pair of nodes to exchange their local dataset S_k, or a given number of patterns between them. Finally, we only consider local communication between neighboring nodes. Next, we describe two decentralized algorithms fulfilling these properties.

3.2. Consensus-based RVFL

The first strategy that we investigate for training an RVFL network in a fully decentralized way is simple, yet it results in a highly efficient training algorithm. It is composed of three steps:

1. **Initialization**: Parameters $\mathbf{w}_1, \dots, \mathbf{w}_B$ of the activation functions are agreed between nodes. For example, one node can draw these parameters from an uniform distribution and broadcast them to the rest of the network. This can be achieved easily in a decentralized way using a basic leader election strategy [3].
2. **Local training**: Each node solves the training problem in (4), considering only its own training set. Solution is given by Eq. (6) or Eq. (8), obtaining a local set of output weights $\boldsymbol{\beta}_k^*, k = 1 \dots L$.
3. **Averaging**: Local parameters vectors are averaged using a DAC strategy, as explained in the previous section. After running DAC, the final weight vector at every node is given by:

$$\boldsymbol{\beta}_{\text{CONS}}^* = \frac{1}{L} \sum_{k=1}^{L} \boldsymbol{\beta}_k^*. \tag{19}$$

Despite its simplicity, consensus-based RVFL results in an interesting algorithm. It is extremely easy to implement, even on low-cost hardware [28]; it requires low training times (i.e., local training and a short set of consensus iterations); moreover, our results, supporting also the findings of Georgopoulos and Hasler [16] in the case of neural network learning, show that it achieves a very low error, in many cases comparable to that of the centralized problem. From a theoretical standpoint, this algorithm can be seen as an ensemble of multiple linear predictors defined over the feature space induced by the mapping $\mathbf{h}(\cdot)$.

[1] The degree of a node is the number of nodes that are connected to it, i.e. $d_k = |\mathcal{N}_k \setminus k|$, where $|\cdot|$ denotes the cardinality of a set.

[2] Once again, $\mathbf{Y}_k$ is either a column vector in case of scalar samples $y_{k,i} \in \mathbb{R}$ or a $N \times M$ matrix in case of M-dimensional outputs $\mathbf{y}_{k,i} \in \mathbb{R}^M$.

Remark 1. For a smaller sized model, an optimal solution can be obtained by executing two sequential consensus steps [41]: the first on the matrices $\left(\mathbf{H}_k^T\mathbf{H}_k + \lambda\mathbf{I}\right)$, and the second on $\mathbf{H}_k^T\mathbf{Y}_k$. In fact, it is easy to observe that, in this case, the final solution at every node is given by:

$$\boldsymbol{\beta}^*_{\text{CONS}} = \left(\frac{\sum_{k=1}^{L}\left(\mathbf{H}_k^T\mathbf{H}_k + \lambda\mathbf{I}\right)}{L}\right)^{-1}\frac{\sum_{k=1}^{L}\mathbf{H}_k^T\mathbf{Y}_k}{L} = \left(\mathbf{H}^T\mathbf{H} + L\lambda\mathbf{I}\right)^{-1}\mathbf{H}^T\mathbf{Y}, \tag{20}$$

which is equivalent to the centralized solution apart from a rescaling of λ. In most realistic situations, however, this is impractical to implement due to the very large dimensionality of the matrices, requiring $B \times (B+1)$ scalars to be sent at every communication between two nodes. Since we are interested into algorithms that can be implemented in generic situations, we will not consider this variant in our experiments.

3.3. ADMM-based RVFL

Another strategy for training in a decentralized way a RVFL network is to optimize directly the global problem in (18) in a distributed fashion. Although potentially more demanding in computational time, this would ensure convergence to the global optimum. We can obtain a fully decentralized solution to problem in (18) using the well-known ADMM.

3.3.1. Derivation of the training algorithm

First, we reformulate the problem in the so-called 'global consensus' form,[3] by introducing local variables $\boldsymbol{\beta}_k$ for every node, and forcing them to be equal at convergence. Hence, we rephrase the optimization problem as:

$$\begin{aligned}\boldsymbol{\beta}^*_{\text{ADMM}} = \underset{\mathbf{z},\boldsymbol{\beta}_1,\ldots,\boldsymbol{\beta}_L\in\mathbb{R}^B}{\text{minimize}} & \frac{1}{2}\left(\sum_{k=1}^{L}\|\mathbf{H}_k\boldsymbol{\beta}_k - \mathbf{Y}_k\|_2^2\right) + \frac{\lambda}{2}\|\mathbf{z}\|_2^2 \\ \text{subject to } & \boldsymbol{\beta}_k = \mathbf{z},\ k = 1\ldots L.\end{aligned} \tag{21}$$

Then, we construct the augmented Lagrangian:

$$\mathcal{L} = \frac{1}{2}\left(\sum_{k=1}^{L}\|\mathbf{H}_k\boldsymbol{\beta}_k - \mathbf{Y}_k\|_2^2\right) + \frac{\lambda}{2}\|\mathbf{z}\|_2^2 + \sum_{k=1}^{L}\mathbf{t}_k^T(\boldsymbol{\beta}_k - \mathbf{z}) + \frac{\gamma}{2}\sum_{k=1}^{L}\|\boldsymbol{\beta}_k - \mathbf{z}\|_2^2, \tag{22}$$

where $\mathcal{L} = \mathcal{L}(\mathbf{z}, \boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_L, \mathbf{t}_1, \ldots, \mathbf{t}_L)$, the vectors $\mathbf{t}_k, k = 1\ldots L$, are the Lagrange multipliers, $\gamma > 0$ is a penalty parameter, and the last term is introduced to ensure differentiability and convergence [6]. ADMM solves problems of this form using an iterative procedure, where at each step we optimize separately for $\boldsymbol{\beta}_k, \mathbf{z}$, and we update the Lagrangian multipliers using a steepest-descent approach:

$$\boldsymbol{\beta}_k[n+1] = \underset{\boldsymbol{\beta}_k\in\mathbb{R}^B}{\arg\min}\ \mathcal{L}(\mathbf{z}[n], \boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_L, \mathbf{t}_1[n], \ldots, \mathbf{t}_L[n]), \tag{23}$$

$$\mathbf{z}[n+1] = \underset{\mathbf{z}\in\mathbb{R}^B}{\arg\min}\ \mathcal{L}(\mathbf{z}, \boldsymbol{\beta}_1[n+1], \ldots, \boldsymbol{\beta}_L[n+1], \mathbf{t}_1[n], \ldots, \mathbf{t}_L[n]), \tag{24}$$

$$\mathbf{t}_k[n+1] = \mathbf{t}_k[n] + \gamma(\boldsymbol{\beta}_k[n+1] - \mathbf{z}[n+1]). \tag{25}$$

In our case, the updates for $\boldsymbol{\beta}_k[n+1]$ and $\mathbf{z}[n+1]$ can be computed in a closed form:

$$\boldsymbol{\beta}_k[n+1] = \left(\mathbf{H}_k^T\mathbf{H}_k + \gamma\mathbf{I}\right)^{-1}\left(\mathbf{H}_k^T\mathbf{Y}_k - \mathbf{t}_k[n] + \gamma\mathbf{z}[n]\right), \tag{26}$$

$$\mathbf{z}[n+1] = \frac{\gamma\hat{\boldsymbol{\beta}} + \hat{\mathbf{t}}}{\lambda/L + \gamma}, \tag{27}$$

where we introduced the averages $\hat{\boldsymbol{\beta}} = \frac{1}{L}\sum_{k=1}^{L}\boldsymbol{\beta}_k[n+1]$ and $\hat{\mathbf{t}} = \frac{1}{L}\sum_{k=1}^{L}\mathbf{t}_k[n]$. These averages can be computed in a decentralized fashion using a DAC step, as detailed in the previous section. We refer to [6] for a proof of the asymptotic convergence of ADMM.

Remark 2. In cases where, on a node, $N_k \ll B$, we can exploit the matrix inversion lemma to obtain a more convenient matrix inversion step [23]:

$$\left(\mathbf{H}_k^T\mathbf{H}_k + \gamma\mathbf{I}\right)^{-1} = \gamma^{-1}\left[\mathbf{I} - \mathbf{H}_k^T\left(\gamma\mathbf{I} + \mathbf{H}_k\mathbf{H}_k^T\right)\mathbf{H}_k\right]. \tag{28}$$

[3] Not to be confused with the decentralized consensus described earlier.

Moreover, with respect to the training complexity, we note that the matrix inversion and the term $\mathbf{H}_k^T\mathbf{Y}_k$ in (26) can be precomputed at the beginning and stored into memory; hence, time complexity is mostly related to the DAC step required in (27). Roughly speaking, if we allow ADMM to run for T iterations (see next subsection), the ADMM-based strategy is approximately T times slower than the consensus-based one.

3.3.2. Stopping criterion

Convergence of the algorithm at the kth node can be tracked by computing the 'primal residual' $\mathbf{r}_k[n]$ and 'dual residual' $\mathbf{s}[n]$, which are defined as:

$$\mathbf{r}_k[n] = \boldsymbol{\beta}_k[n] - \mathbf{z}[n], \tag{29}$$

$$\mathbf{s}[n] = -\gamma(\mathbf{z}[n] - \mathbf{z}[n-1]). \tag{30}$$

A possible stopping criterion is that both residuals should be less (in norm) than two thresholds:

$$\|\mathbf{r}_k[n]\|_2 < \epsilon_{\text{primal}}, \tag{31}$$

$$\|\mathbf{s}[n]\|_2 < \epsilon_{\text{dual}}. \tag{32}$$

A possible way of choosing the thresholds is given by [6]:

$$\epsilon_{\text{primal}} = \sqrt{L}\epsilon_{\text{abs}} + \epsilon_{\text{rel}} \max\left\{\|\boldsymbol{\beta}_k[n]\|_2, \|\mathbf{z}[n]\|_2\right\}, \tag{33}$$

$$\epsilon_{\text{dual}} = \sqrt{L}\epsilon_{\text{abs}} + \epsilon_{\text{rel}}\|\mathbf{t}_k[n]\|_2, \tag{34}$$

where ϵ_{abs} and ϵ_{rel} are user-specified absolute and relative tolerances, respectively. Alternatively, as in the previous case, the algorithm can be stopped after a maximum number of iterations is reached. The pseudocode for the overall algorithm at a single node is given in Table 1.

Algorithm 1. Local update steps for ADMM-based RVFL at kth node

Input: Training set S_k, number of nodes L (global), regularization factors λ, γ (global), maximum number of iterations T (global)
Output: Optimal vector $\boldsymbol{\beta}_k^*$

```
Select parameters w_1, ..., w_B, in agreement with the other L − 1 nodes.
Compute H_k and Y_k from S_k.
Initialize t_k[0] = 0, z[0] = 0.
for n from 0 to T do
    Compute β_k[n + 1] according to Eq. (26).
    Compute averages β̂ and t̂ with consensus with other nodes over the network.
    Compute z[n + 1] according to Eq. (27).
    Update t_k[n] according to Eq. (25).
    Check termination with residuals.
end for
return z[n]
```

4. Experimental setup

4.1. Description of the datasets

We tested our algorithms on four publicly available datasets, whose characteristics are summarized in Table 1. We have chosen them to represent different applicative domains of our algorithms, and to provide enough diversity in term of size, number of features, and imbalance of the classes:

- *Garageband* is a music classification problem [25], where the task is to discern among 9 different genres. As we stated in the introductory section, in the distributed case we can assume that the songs are present over different computers, and we can use our strategies as a way of leveraging over the entire dataset without a centralized controller.
- *Skills* is a regression dataset taken from the UCI repository [39]. The task is to assess the skill level of a video game user, based on a set of recordings of its actions in the video game itself. This is useful for letting the game adapt to the user's characteristics. In this case, data is distributed by definition throughout the different players in the network. By employing our strategy several computers, each playing their own version of the game, can learn to adapt better by exploiting collective data.

Table 1
General description of the datasets.

Dataset name	Features	Instances	Desired output	Task type
G50C	50	550	Gaussian of origin	Classification (2 classes)
Garageband	44	1856	Genre recognition	Classification (9 classes)
Skills	18	3338	User's level	Regression
Sylva	216	14,394	Forest Type	Classification (2 classes)

Table 2
Optimal parameters found by the grid-search procedure.

Dataset	Hidden nodes	λ
G50C	500	2^{3}
Garageband	200	2^{-3}
Skills	400	2^{-2}
Sylva	450	2^{-5}

- *Sylva* is a binary classification task for distinguishing classes of trees (Ponderosa pine vs. everything else).[4] It is an interesting dataset since it has a large imbalance between the positive and negative examples (approximately 15:1), and a large subset of the features are not informative from a classification point of view. In the distributed case, we can imagine that data is collected by different sensors.
- *G50C*, differently from the others, is an artificial dataset [24], whose main interest is given by the fact that the optimal error rate is designed so as to be equal to exactly 5%.

In all cases, input variables are normalized between 0 and 1, and missing values are replaced with the average computed over the rest of the dataset. For all the models, testing accuracy and training times is computed by executing a 5-fold cross-validation over the available data. This 5-fold procedure is then repeated 15 times by varying the topology of the agents and the initial weights of the RVFL net. Final misclassification error and training time is then collected for all the $15 \times 5 = 75$ repetitions, and the average values and standard deviations are computed.

4.2. Algorithms and software implementation

We compare the following algorithms:

- **Centralized RVFL** (C-RVFL): this is a RVFL trained with all the available training data. It is equivalent to a centralized node collecting all the data, and it can be used as a baseline for the other approaches.
- **Local RVFL** (L-RVFL): in this case, training data is distributed evenly across the nodes. Every node trains a standard RVFL with its own local dataset, but no communication is performed. Testing error is averaged throughout the nodes.
- **Consensus-based RVFL** (CONS-RVFL): as before, data is evenly distributed in the network, and the consensus strategy explained in Section 3.2 is executed. We set a maximum of 300 iterations and $\delta = 10^{-3}$.
- **ADMM-based RVFL** (ADMM-RVFL): similar to before, but we employ the ADMM-based strategy described in Section 3.3. In this case, we set a maximum of 300 iterations, $\epsilon_{\text{rel}} = \epsilon_{\text{abs}} = 10^{-3}$ and $\gamma = 1$.

In all cases, we use sigmoid hidden functions given by (2), where parameters $\mathbf{w}$ and b in (2) are extracted randomly from an uniform distribution over the interval $[-1, +1]$. To compute the optimal number of hidden nodes and the regularization parameter λ, we execute an inner 3-fold cross-validation on the training data only for C-RVFL. In particular, we search the uniform interval $\{50, 100, 150, \ldots, 1000\}$ for the number of hidden nodes, and the exponential interval $2^{j}, j \in \{-10, -9, \ldots, 9, 10\}$ for λ. The step size of 50 in the hidden nodes interval was found to provide a good compromise between final accuracy and the computational cost of the grid-search procedure. These parameters are then shared with the three remaining models. We experimented with a separate fine-tuning for each model, but no improvement in performance was found. Optimal parameters averaged over the runs are shown in Table 2.

We have implemented CONS-RVFL and ADMM-RVFL in the open-source Lynx MATLAB toolbox[5]; configuration files and instructions for repeating the experiments in the following section are also available on the Web.[6] In this paper we are not concerned with the analysis of communication overhead over a realistic channel, hence we employ a serial version of the code where the network is simulated artificially. However, in the aforementioned toolbox we also provide a fully parallel version, able to work on a cluster architecture, in order to test the accuracy of the system over a realistic setting.

[4] http://www.causality.inf.ethz.ch/al_data/SYLVA.html.
[5] http://ispac.ing.uniroma1.it/scardapane/software/lynx/.
[6] http://ispac.ing.uniroma1.it/scardapane/software/lynx/dist-learning/.

5. Results and discussion

5.1. Accuracy and training times

The first set of experiments is to show that both algorithms that we propose are able to approximate very closely the centralized solution, irrespective of the number of nodes in the network. The topology of the network in these experiments is constructed according to the so-called 'Erdős–Rényi model' [27], i.e., once we have selected a number L of nodes, we randomly construct an adjacency matrix such that every edge has a probability p of appearing, with p specified a priori. For the moment, we set $p = 0.2$; an example of such a network for $L = 8$ is shown in Fig. 3.

To test the accuracy of the algorithms, we vary L from 5 to 50 by steps of 5. Results are presented in Fig. 4 (a)–(d). For the three classification datasets, we show the averaged misclassification error. For the Skills dataset, instead, we show the Normalized Root Mean-Squared Error (NRMSE), defined for a test set $\mathcal{T}$ as:

$$\text{NRMSE}(\mathcal{T}) = \sqrt{\frac{\sum_{(\mathbf{x}_i, y_i) \in \mathcal{T}} [f(\mathbf{x}_i) - y_i]^2}{|\mathcal{T}| \hat{\sigma}_y}}, \tag{35}$$

where $|\mathcal{T}|$ denotes the cardinality of the set $\mathcal{T}$ and $\hat{\sigma}_y$ is an empirical estimate of the variance of the output samples $y_i, i = 1, \ldots, |\mathcal{T}|$. For every fold, the misclassification error of L-RVFL is obtained by averaging the error over the L different nodes. While this is a common practice, it can introduce a small bias with respect to the other curves. Nonetheless, we stress that it does not influence the following discussion, which mostly focuses on the comparison of the other three algorithms.

The first thing to observe is that L-RVFL has a steady decrease in performance in all situations, ranging from a small decrease in the Skills dataset with 5 nodes, to more than 20% of classification accuracy when considering networks of 50 agents in the G50C and Garageband datasets. Despite being obvious, because of the decrease of available data at each node, it is an experimental confirmation of the importance of leveraging over *all* possible data in term of accuracy. It is also interesting to note that the gap between L-RVFL and C-RVFL does not always increase monotonically with respect to the size of the network, as shown by Fig. 4(d). A possible explanation of this fact is that, by keeping fixed the λ parameter, the effect of the regularization factor in Eq. (18) is proportionally higher when decreasing the amount of training data. Due to its imbalance, the Sylva dataset is sensitive to this change [29]. A similar peculiar behavior for feature selection was also observed in [43]. Since it is not directly related to the topic of the paper, we do not explore this behavior further.

The second important aspect is that CONS-RVFL and ADMM-RVFL are *both* able to match very closely the performance of C-RVFL, irrespective of the network's size. In particular, they have the same performance on the G50C and Skills datasets, while a small gap is present in the Garageband and Sylva cases, although it is not significant.

Next, let us analyze the training times of the distributed algorithm, shown in Fig. 5 for CONS-RVFL and Fig. 6 for ADMM-RVFL, respectively. In particular, we show the average training time spent at a single node. Generally speaking, CONS-RVFL is approximately one order of magnitude faster than ADMM-RVFL, which requires multiple iterations of consensus. In both cases, the average training time spent at a single node is monotonically decreasing with respect to the overall number of nodes. Hence, the computational time of the matrix inversion is predominant compared to the overhead introduced by the DAC and ADMM procedures.

5.2. Effect of network topology

Now that we have ascertained the convergence properties of both algorithms, we analyze an interesting aspect: how does the topology of the network influence the convergence time? Clearly, as long as the network stays connected, the accuracy is

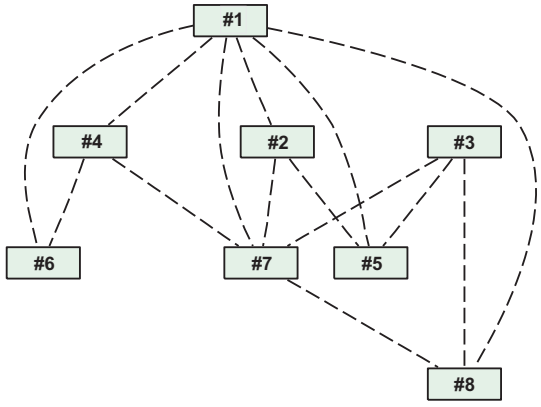

Fig. 3. Example of network used in the experiments with 8 nodes. Connectivity is generated at random, with a 20% probability for each link of being present.

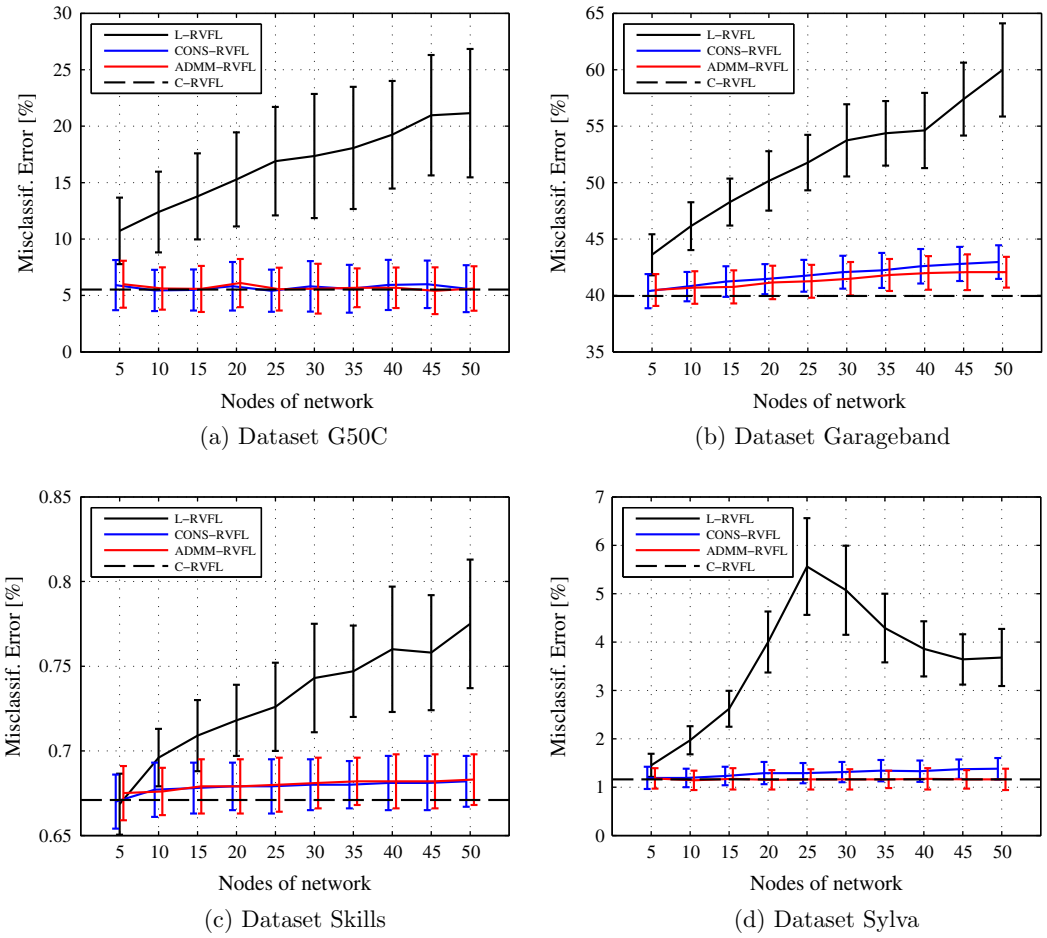

Fig. 4. Average error and standard deviation of the models on the four datasets, when varying the number of nodes in the network from 5 to 50. For G50C, Garageband, and Sylva we show the misclassification error, while for Skills we show the NRMSE. Lines for CONS-RVFL and ADMM-RVFL are slightly separated for better readability. Vertical bars represent the standard deviation from the average result.

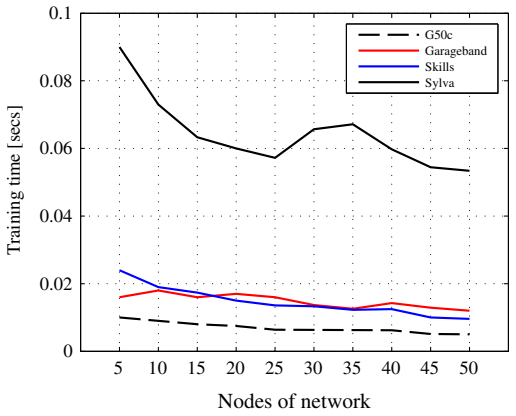

Fig. 5. Average training time for CONS-RVFL on a single node.

not influenced. However, the time required for the consensus to achieve convergence is dependent on how the nodes are interconnected. At the extreme, in a fully connected network, two iterations are always sufficient to achieve convergence at any desired level of accuracy. More in general, the time will be roughly proportional to the average distance between any two nodes. To test this, we compute the iterations needed to reach consensus for several topologies of networks composed of 50 nodes:

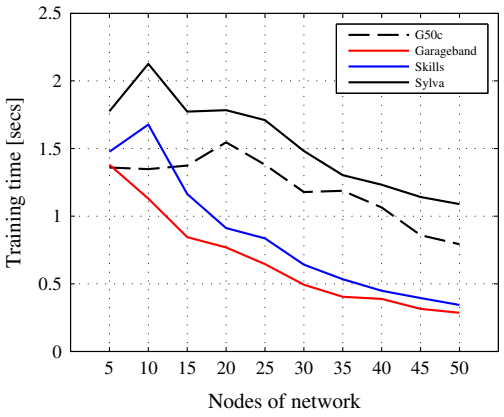

Fig. 6. Average training time for ADMM-RVFL on a single node.

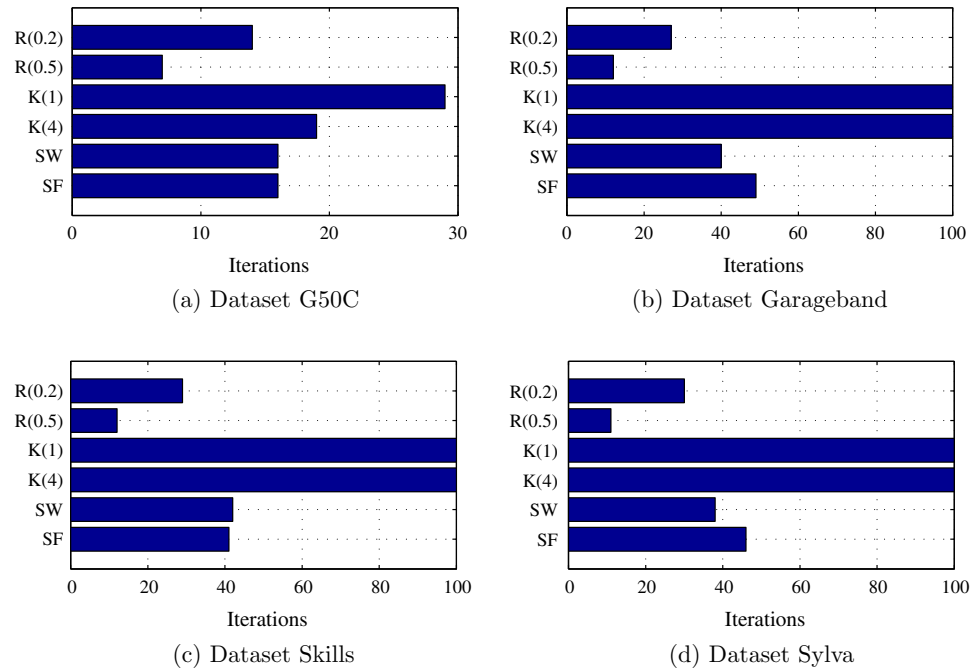

Fig. 7. Consensus iterations needed to reach convergence when varying the network topology. For the explanation of the topologies see Section 5.2. The number of iterations is truncated at 100 for better readability.

- **Random network**: this is the network constructed according to the Erdős–Rényi model described in the previous subsection. We experiment with $p = 0.2$ and $p = 0.5$, and denote the corresponding graphs as $R(0.2)$ and $R(0.5)$ respectively.
- **Linear network**: in this network the nodes are ordered, and each node in the sequence is connected to its most immediate K successors, with K specified a priori, except the last $K - 1$ nodes, which are connected only to the remaining ones. We experiment with $K = 1$ and $K = 4$, and denote the networks as $K(1)$ and $K(4)$ respectively.
- **Small world**: this is a network constructed according to the well-known 'Watts–Strogazt' mechanism [27]. First, a cyclic topology is constructed, i.e., nodes are ordered in a circular sequence, and every node is connected to K nodes to its left and K to its right. Then, every link is 'rewired' with probability set by a parameter $\alpha \in [0, 1]$. In our case, we have $K = 6$ and $\alpha = 0.15$, and denote the resulting topology as *SW*.
- **Scale-free**: this is another topology that tries to reflect realistic networks, in this case exhibiting a power law with respect to the degree distribution. We construct it according to the 'Barabási–Albert' model of preferential attachment [27], and denote the resulting topology as *SF*.

Results are presented in Fig. 7 (a)–(d). We see that the algorithm has very similar results on all four datasets. In particular, as we expected, consensus is extremely slow in reaching agreement when considering linear topologies, where information

takes several iterations to reach one end of the graph from the other. At the other extreme, it takes a very limited number of iterations in the case of a highly connected graph, as in the case of $R(0.5)$. In between, we can see that consensus is extremely robust to a change in topology and its performances are not affected when considering small-world or scale-free graphs.

5.3. Early stopping for ADMM

Next, we explore a peculiar difference between CONS-RVFL and ADMM-RVFL. In the case of CONS-RVFL, no agreement is reached between the different nodes until the consensus procedure is completed. Differently from it, an intermediate solution is available at every iteration in ADMM-RVFL, given by the vector $\mathbf{z}[n]$. This allows for the use of an *early stopping* procedure, i.e., the possibility of stopping the optimization process before actual convergence, by fixing in advance a predefined (small) number of iterations. In fact, several experimental findings support the idea that ADMM can achieve a reasonable degree of accuracy in the initial stages of optimization [6]. To test this, we experiment early stopping for ADMM-RVFL at $\{5, 10, 15, 25, 50, 100, 200\}$ iterations for the three classification datasets. In Fig. 8 we plot the relative decrease in performance with respect to L-RVFL. We can see that 10–15 iterations are generally enough to reach a good performance, while the remaining iterations are proportionally less useful. As a concrete example, misclassification error of ADMM-RVFL for G50C is 16.55% after only 5 iterations, 12.44% after 10, 7.89% after 25, while the remaining 175 iterations are used to decrease the error only by an additional 2 percentage points.

5.4. Experiment on large-scale data

As a final experimental validation, we analyze the behavior of CONS-RVFL and ADMM-RVFL on a realistic large-scale dataset, the well-known CIFAR-10 image classification database [19]. It is composed of 50,000 images labeled in 10 different classes, along with a standard testing set of additional 10,000 images. Each image is composed of exactly 32×32 pixels, and each pixel is further represented by 3 integer values in the interval $[1, 255]$, one for each color channel in the RGB color space. Classes are equally distributed between the training patterns, i.e., every class is represented by exactly 5000 images. Since we are mostly interested into the relative difference in performance between the algorithms, and not in achieving the lowest possible classification error, we preprocess the images using the relatively simple procedure detailed in [9]. In particular, we extract 1600 significant patches from the original images, and represent each image using their similarity with respect to each of the patches. We refer to [9] for more details on the overall workflow. In this experiment, we use $B = 3000$ and $\lambda = 10$, a setting which was found to work consistently on all situations. Moreover, we use the $R(0.2)$ graph explained before, but we experiment with lower number of nodes in the network, which we vary from 2 to 12 by steps of 2. All the other parameters are set as in the previous experiments. Although the test set is fixed in this case, we repeat each experiment 15 times to average out the effect of randomness in the RVFL and connectivity initializations.

The average misclassification error of the four models is shown in Fig. 9. In this case, the effect of splitting data is extremely pronounced, and the average misclassification error of L-RVFL goes from 39% with 2 nodes, up to 62.4% with 12 nodes. Both CONS-RVFL and ADMM-RVFL are able to track very efficiently the centralized solution, although there is a small gap in performance between the two (of approximately 1%), when distributing over more than 8 nodes. This is more than counterbalanced, however, by considering the advantage of CONS-RVFL with respect to the required training time. To show this, we present the average training time (averaged over the different nodes) in Fig. 10. In this case, due to the very large expansion block, time required to perform the multiple consensus iterations in ADMM-RVFL prevails over the rest, and the average training time tends to increase when increasing the size of the network. This is not true of CONS-RVFL, however, which obtains an extremely low training time with respect to C-RVFL, up to an order of magnitude for sufficiently large networks.

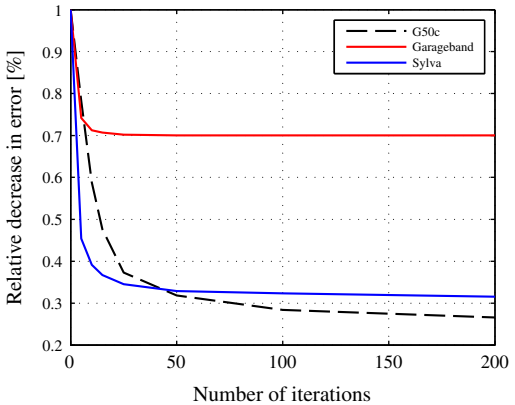

Fig. 8. Relative decrease in error of ADMM-RVFL with respect to L-RVFL, when using an early stopping procedure at different iterations.

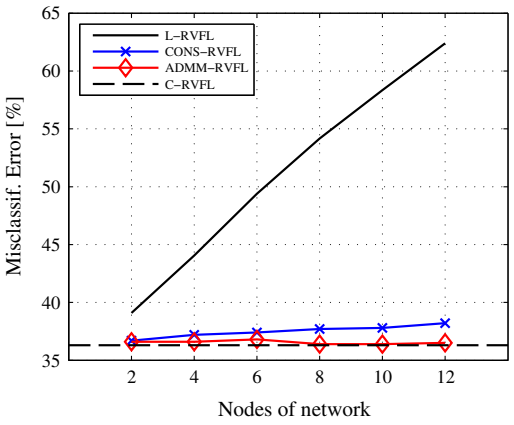

Fig. 9. Average misclassification error on the CIFAR-10 dataset, when varying the nodes of the network from 2 to 12.

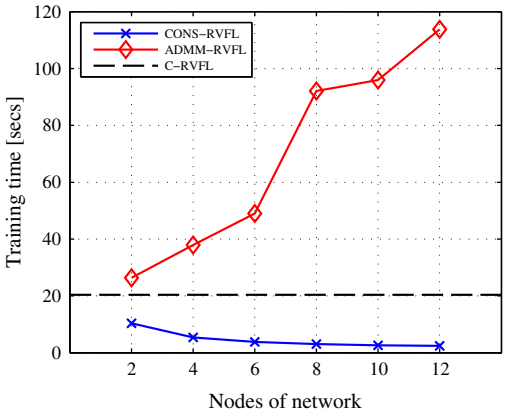

Fig. 10. Average training time on the CIFAR-10 dataset, when varying the nodes of the network from 2 to 12.

Hence, we can say that CONS-RVFL can also be an efficient way of computing an approximate solution to a standard RVFL, with good accuracy, by distributing the computation over multiple machines.

6. Conclusions

Distributed learning has received considerable attention over the past years due to its broad real-world applications. It is common nowadays that data must be collected, stored locally and data exchange is not allowed for some reasons. In such a circumstance, it is necessary and useful to build a master learner model effectively and efficiently. In this paper, we have presented two distributed learning algorithms for training RVFL networks through interconnected nodes. These algorithms allow the nodes to agree on a single model, whose testing performance is similar to that obtained by a centralized model. Experimental results demonstrate that the DAC-based learning algorithm can achieve comparable performances to the centralized learning method in terms of both accuracy and efficiency.

This work can be regarded as a part of machine learning for big data processing. Further researches along this direction can be both algorithmic developments and applications where no centralized solution is feasible. Moreover, extensions of the present algorithms to online learning for RVFL and to the case of a recurrent hidden layer (e.g. Echo State Networks [36]) are being expected.

Acknowledgment

The authors wish to thank Roberto Fierimonte, M.Sc., for his helpful comments and discussions.

References

[1] M. Alhamdoosh, D. Wang, Fast decorrelated neural network ensembles with random weights, Inf. Sci. 264 (2014) 104–117.
[2] L. Atzori, A. Iera, G. Morabito, The internet of things: a survey, Comput. Networks 54 (15) (2010) 2787–2805.

[3] B. Awerbuch, Optimal distributed algorithms for minimum weight spanning tree, counting, leader election, and related problems, in: Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing, ACM, 1987, pp. 230–240.
[4] G. Bakir, Predicting Structured Data, MIT press, 2007.
[5] S. Barbarossa, S. Sardellitti, P. Di Lorenzo, Distributed detection and estimation in wireless sensor networks, in: R. Chellapa, S. Theodoridis (Eds.), E-Reference Signal Processing, Elsevier, 2013, pp. 329–408.
[6] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, Distributed optimization and statistical learning via the alternating direction method of multipliers, Found. Trends® Mach. Learn. 3 (1) (2011) 1–122.
[7] L. Breiman, Bagging predictors, Mach. Learn. 24 (2) (1996) 123–140.
[8] C.-T. Chu, S.K. Kim, Y.A. Lin, Y.Y. Yu, Map-reduce for machine learning on multicore, Adv. Neural Inf. Process. Syst. (2007) 281–288
[9] A. Coates, A.Y. Ng, H. Lee, An analysis of single-layer networks in unsupervised feature learning, in: 14th International Conference on Artificial Intelligence and Statistics, 2011, pp. 215–223.
[10] D. Comminiello, M. Scarpiniti, L.A. Azpicueta-Ruiz, J. Arenas-Garcia, A. Uncini, Functional link adaptive filters for nonlinear acoustic echo cancellation, IEEE Trans. Audio Speech Lang. Process. 21 (7) (2013) 1502–1512.
[11] J. Dean, S. Ghemawat, Mapreduce: simplified data processing on large clusters, Commun. ACM 51 (1) (2008) 107–113.
[12] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, Q.V.L. Le, A.Y. Ng, Large scale distributed deep networks, Adv. Neural Inf. Process. Syst. (2012) 1223–1231
[13] O. Dekel, R. Gilad-Bachrach, O. Shamir, L. Xiao, Optimal distributed online prediction using mini-batches, J. Mach. Learn. Res. 13 (2012) 165–202.
[14] P.A. Forero, A. Cano, G.B. Giannakis, Consensus-based distributed support vector machines, J. Mach. Learn. Res. 11 (2010) 1663–1707.
[15] J. Friedman, T. Hastie, R. Tibshirani, The Elements of Statistical Learning, second ed., Springer, 2009.
[16] L. Georgopoulos, M. Hasler, Distributed machine learning in networks by consensus, Neurocomputing 124 (2014) 2–12.
[17] B. Igelnik, Y.-H. Pao, Stochastic choice of basis functions in adaptive function approximation and the functional-link net, IEEE Trans. Neural Networks 6 (6) (1995) 1320–1329.
[18] D. Jakovetic, J. Xavier, J.M.F. Moura, Fast distributed gradient methods, IEEE Trans. Autom. Control 59 (5) (2014) 1131–1146.
[19] A. Krizhevsky, G. Hinton, Learning multiple layers of features from tiny images, Computer Science Department, University of Toronto, Tech. Rep., 2009.
[20] A. Lazarevic, Z. Obradovic, The distributed boosting algorithm, in: Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2001, pp. 311–316.
[21] P. Lorenzo, A.H. Sayed, Sparse distributed learning based on diffusion adaptation, IEEE Trans. Signal Process. 61 (6) (2013) 1419–1433.
[22] C. Lynch, Big data: how do your data grow?, Nature 455 (7209) (2008) 28–29
[23] G. Mateos, J.A. Bazerque, G.B. Giannakis, Distributed sparse linear regression, IEEE Trans. Signal Process. 58 (10) (2010) 5262–5276.
[24] S. Melacci, M. Belkin, Laplacian support vector machines trained in the primal, J. Mach. Learn. Res. 12 (2011) 1149–1184.
[25] I. Mierswa, K. Morik, Automatic feature extraction for classifying audio data, Mach. Learn. 58 (2–3) (2005) 127–149.
[26] A. Navia-Vázquez, D. Gutierrez-Gonzalez, E. Parrado-Hernández, J.J. Navarro-Abellan, Distributed support vector machines, IEEE Trans. Neural Networks 17 (4) (2006) 1091–1097.
[27] M. Newman, Networks: An Introduction, Oxford University Press, 2010.
[28] R. Olfati-Saber, J.A. Fax, R.M. Murray, Consensus and cooperation in networked multi-agent systems, Proc. IEEE 95 (1) (2007) 215–233.
[29] Y.-Y. Ou, H.-G. Hung, Y.-J. Oyang, A study of supervised learning with multivariate analysis on unbalanced datasets, in: 2006 International Joint Conference on Neural Networks, IEEE, 2006, pp. 2201–2205.
[30] Y.-H. Pao, Y. Takefuji, Functional-link net computing: theory, system architecture, and functionalities, Computer 25 (5) (1992) 76–79.
[31] J.B. Predd, S.R. Kulkarni, H.V. Poor, Consistency in models for distributed learning under communication constraints, IEEE Trans. Inf. Theory 52 (1) (2006) 52–63.
[32] J.B. Predd, S.R. Kulkarni, H.V. Poor, Distributed learning in wireless sensor networks, IEEE Signal Process. Mag. (2007) 56–69
[33] D. Saha, A. Mukherjee, Pervasive computing: a paradigm for the 21st century, Computer 36 (3) (2003) 25–31.
[34] A.H. Sayed, Adaptive networks, Proc. IEEE 102 (4) (2014) 460–497.
[35] S. Scardapane, D. Comminiello, M. Scarpiniti, A. Uncini, Music classification using extreme learning machines, in: 2013 8th International Symposium on Image and Signal Processing and Analysis (ISPA), IEEE, 2013, pp. 377–381.
[36] S. Scardapane, G. Nocco, D. Comminiello, M. Scarpiniti, A. Uncini, An effective criterion for pruning reservoir's connections in echo state networks, in: 2014 International Joint Conference on Neural Networks, IEEE, 2014, pp. 1205–1212.
[37] W.F. Schmidt, M.A. Kraaijveld, R.P.W. Duin, Feedforward neural networks with random weights, in: 11th IAPR International Conference on Pattern Recognition, 1992, IEEE, 1992, pp. 1–4.
[38] K. Slavakis, G. Giannakis, G. Mateos, Modeling and optimization for big data analytics: (statistical) learning tools for our era of data deluge, IEEE Signal Process. Mag. 31 (5) (2014) 18–31.
[39] J.J. Thompson, M.R. Blair, L. Chen, A.J. Henrey, Video game telemetry as a critical tool in the study of complex skill learning, PLoS One 8 (9) (2013) e75129.
[40] V.S. Verykios, E. Bertino, I.N. Fovino, L.P. Provenza, Y. Saygin, Y. Theodoridis, State-of-the-art in privacy preserving data mining, ACM Sigmod Rec. 33 (1) (2004) 50–57.
[41] L. Xiao, S. Boyd, S. Lall, A scheme for robust distributed sensor fusion based on average consensus, in: Fourth International Symposium on Information Processing in Sensor Networks, IEEE, 2005, pp. 63–70.
[42] H. Zhang, Z. Chen, Consensus acceleration in a class of predictive networks, IEEE Trans. Neural Networks Learn. Syst. 25 (10) (2014) 1921–1927.
[43] Y. Zhang, S. Li, T. Wang, Z. Zhang, Divergence-based feature selection for separate classes, Neurocomputing 101 (2013) 32–42.